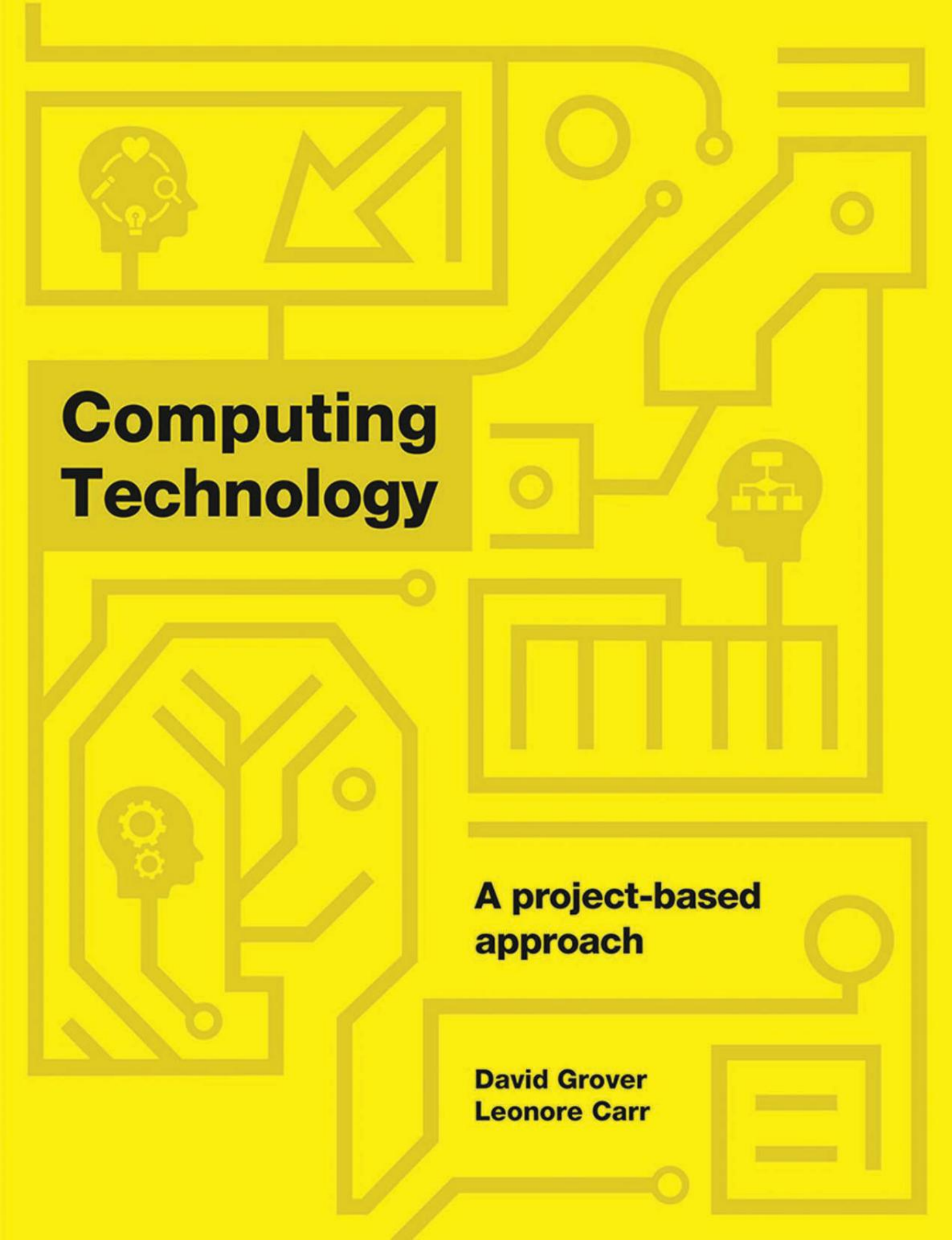


Computing Technology

A project-based approach

David Grover
Leonore Carr

CENGAGE Learning™

COPYRIGHT NOTICE

Copyright in this work is owned by Cengage Learning Australia ("the work"). A condition of purchase of this electronic version of the work is that you agree to respect the copyright in the work, abide by the Copyright Act 1968 and specifically agree not to transfer, sell, assign, misuse, copy or transmit an electronic or other version of the work to any third party.

Please note: This product is accompanied by a licence (single user, network or adoption) governing the terms and conditions of its use.

This is a legal agreement between the you, (the "Customer") and Cengage Learning Australia Pty Limited (ABN 14 058 280 149) (the "Licensor") which provides the terms and conditions of this non-exclusive licence and the limited warranty for the Product. Use of the Product indicates an acknowledgement that the Customer has read and agreed to be bound by the terms and conditions of this Agreement. If you do not agree to these terms and conditions, return the Product to the place of purchase within 15 days of the date of purchase (with proof of purchase) for a full refund

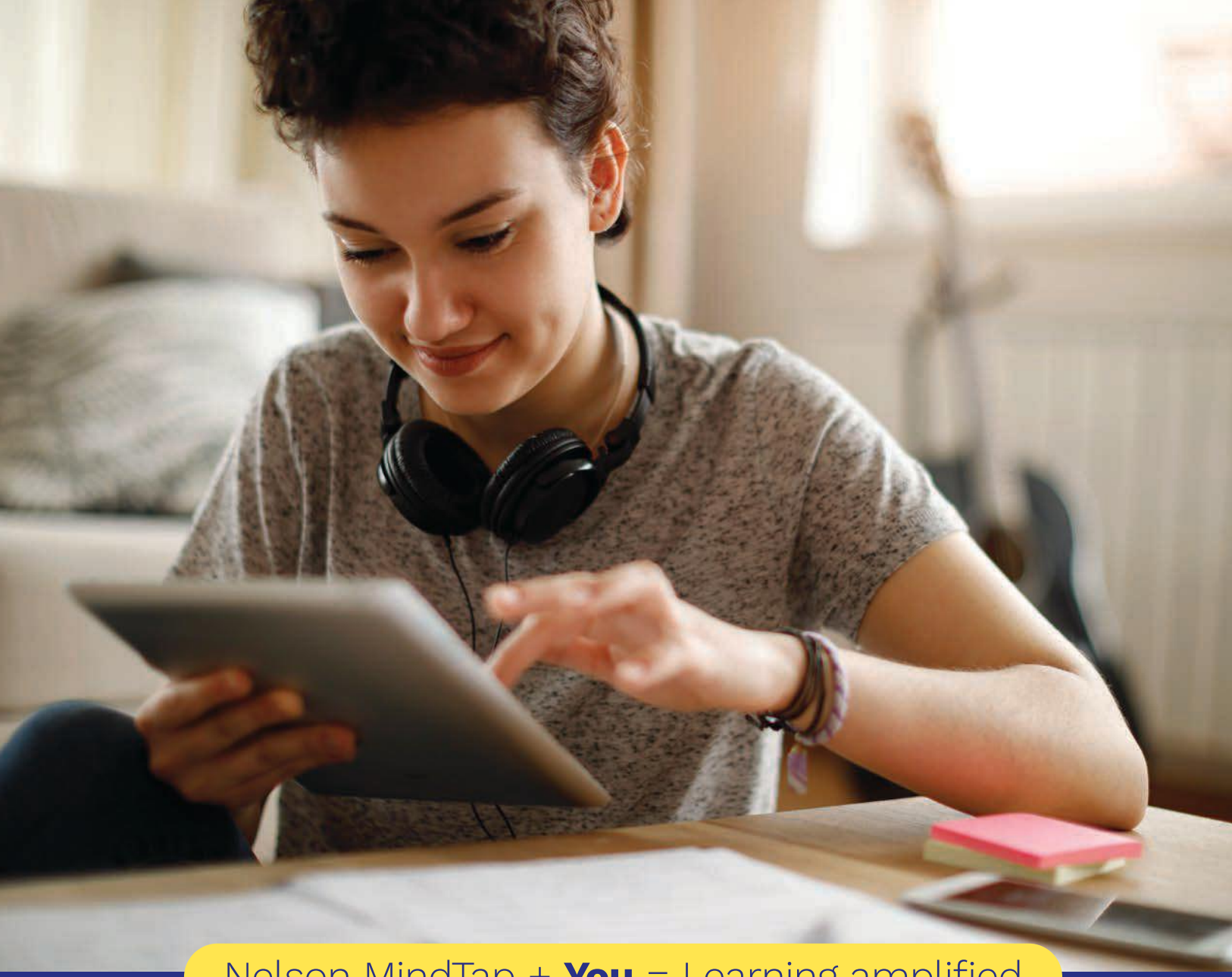
1. **Licence Grant**
   You do not receive title to the Product. Copyright in the Product (which includes all images, photographs, video, animations, audio, music and text incorporated in the Product, including all of the accompanying printed material) is owned by the Licensor and/or its suppliers and is protected by Australian copyright laws. The Licensor grants you a non-exclusive licence to use the Product subject to the restrictions and terms set out in this Agreement.

2. **A Licence allows you to:**
   Use the Product on your computer. The Customer represents that they shall in no way place the Product in the public domain or in any way compromise our copyright in the Material. You agree to take reasonable steps to protect our copyright.

3. **You may not:**
   Alter, modify, translate, reverse engineer, decompile, or adapt the software or create derivative works based on the Product. Make further copies by any means technological, electronic, digital whatsoever without the written permission of the Licensor. Rent or transfer all or any part of your rights under this Agreement. Remove or alter any copyright or other proprietary notice or label attached to the software.

4. **Termination**
   Any failure to comply with the terms and conditions of this agreement will result in the automatic termination of this licence. Upon termination of this licence for any reason, the Customer must destroy or return to the Licensor all copies of the software and accompanying documentation.

5. **Warranties**
   To the extent permitted by law, the Licensor's liability for any breach of the warranty or any term implied by law into this licence is limited to the lowest cost of replacing the goods, acquiring equivalent goods or having the goods repaired.



Nelson MindTap + **You** = Learning amplified

"I love that everything is interconnected, relevant and that there is a clear learning sequence. I have the tools to create a learning experience that meets the needs of all my students and can easily see how they're progressing."

— **Sarah,** Secondary School Teacher

Nelson MindTap

Computing Technology: A Project-Based Approach
1st Edition
David Grover
Leonore Carr
ISBN 9780170442787

Publisher: Melissa O'Shea
Editors: Sue Jamieson, Lauren McGregor
Contributors: Heather Knights, Eamon Gormley
Cover designers: Regine Abos Design + Illustration, James Steer
Text designer: Jenki
Permissions researcher: Catherine Kerstjens
Production controller: Karen Young
Typeset by: Lumina Datamatics

Any URLs contained in this publication were checked for currency during the production process. Note, however, that the publisher cannot vouch for the ongoing currency of URLs.

© 2023 David Grover, Leonore Carr, Eamon Gormley, Heather Knights

Copyright Notice
This Work is copyright. No part of this Work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without prior written permission of the Publisher. Except as permitted under the *Copyright Act 1968,* for example any fair dealing for the purposes of private study, research, criticism or review, subject to certain limitations. These limitations include: Restricting the copying to a maximum of one chapter or 10% of this book, whichever is greater; providing an appropriate notice and warning with the copies of the Work disseminated; taking all reasonable steps to limit access to these copies to people authorised to receive these copies; ensuring you hold the appropriate Licences issued by the Copyright Agency Limited ("CAL"), supply a remuneration notice to CAL and pay any required fees. For details of CAL licences and remuneration notices please contact CAL at Level 11, 66 Goulburn Street, Sydney NSW 2000,
Tel: (02) 9394 7600, Fax: (02) 9394 7601
Email: info@copyright.com.au
Website: www.copyright.com.au

For product information and technology assistance,
in Australia call **1300 790 853**;
in New Zealand call **0800 449 725**

For permission to use material from this text or product, please email **aust.permissions@cengage.com**

National Library of Australia Cataloguing-in-Publication Data
A catalogue record for this book is available from the National Library of Australia.

Cengage Learning Australia
Level 5, 80 Dorcas Street
Southbank VIC 3006 Australia

For learning solutions, visit **cengage.com.au**

Printed in China by 1010 Printing International Limited.
1 2 3 4 5 6 7 28 27 26 25 24

Contents

To students viii
About this book ix
How to use this book x
About the authors xii

INTRODUCTORY CHAPTERS

1 An introduction to project management 2
1.1 Approaches to project management 4
1.2 Identifying and defining 6
1.3 Researching and planning 8
1.4 Producing and implementing 12
1.5 Testing and evaluating 14
1.6 Project management considerations 16

2 An introduction to thinking skills 22
2.1 Design thinking 26
2.2 Systems thinking 28
2.3 Computational thinking 30

3 An introduction to enterprise information systems 34
3.1 What are information systems? 36
3.2 Hardware 42
3.3 Software 46
3.4 Data and information 49
3.5 People and processes 54
3.6 Considerations and cybersecurity principles 57

4 An introduction to software development 62
4.1 Understanding programming 64
4.2 Understanding software types 67
4.3 Understanding algorithms 70
4.4 Two types of computing languages 73
4.5 Understanding structured programming 76
4.6 Understanding the basic control structures 78
4.7 Anatomy of a computer program 84
4.8 Primitive data types 85
4.9 Parts of a program 87
4.10 Python 3 reference table 94

ENTERPRISE INFORMATION SYSTEMS

FOCUS AREA: Modelling networks and social connections 96

5 Modelling social networks 96
5.1 Introduction to social networks 99
5.2 Graph theory 101
5.3 Social network examples 108
5.4 Social network data 113
5.5 Social, ethical and legal considerations 120
5.6 Project: New social network pitch 125

6 Modelling computer networks 130
6.1 Introduction to computer networks 132
6.2 Network hardware 138
6.3 Transmission media 142
6.4 Topologies and navigation maps 152
6.5 Data processing and transmission 159
6.6 Network security 166
6.7 Internet of Things (IoT) 171
6.8 Project: Create a model of a smart home or smart garden 180

FOCUS AREA: Designing for user experience 184

7 Designing the user interface 184
7.1 The user experience 186
7.2 The purpose of the user interface 187
7.3 Types of user interfaces 191
7.4 Evolution of user interfaces 194
7.5 Planning a user interface 197
7.6 UI design principles 202
7.7 Audience and analytics 208
7.8 Inclusive design 209
7.9 Common errors in interface design 211
7.10 Guided project 1: Building an app with a functional UI 216
7.11 Project 2: Designing, building and prototyping a mobile travel app 224

8 Designing for user experience in AR and VR 232
8.1 Extended reality user experiences 235
8.2 Augmented reality (AR) 236
8.3 Virtual reality (VR) 245
8.4 Ethical issues in VR 251
8.5 The evolution of the VR user experience 252
8.6 Project: Create your own interactive 3D world using Unity 259

FOCUS AREA: Analysing data 272

9 Analysing data using spreadsheet systems 272
9.1 Introduction to spreadsheet systems 274
9.2 Organising data in spreadsheet systems 278
9.3 Formulas and functions 282
9.4 Advanced functions 286
9.5 Visual communication and charts 290
9.6 Modelling, simulating and predicting 294
9.7 Pivot tables 297
9.8 Data dashboards 302
9.9 System integration 307
9.10 Project: Large data analysis and visualisation 311

10 Analysing data using database management systems 314
10.1 Introduction to database systems 316
10.2 Building a database 319
10.3 Collecting and checking data 327
10.4 Analysing data 335
10.5 Presenting information 344
10.6 Integration 351
10.7 Project: HPE event management 357

SOFTWARE DEVELOPMENT

FOCUS AREA: Building mechatronic and automated systems 360

11 Building mechatronic systems 360
11.1 What is mechatronics? 362
11.2 Basic components of a mechatronic system 363
11.3 Sensors 365
11.4 Accelerometers 369
11.5 Data logging 377
11.6 Distance sensors 379
11.7 Actuators 383
11.8 Microprocessors 391
11.9 Control systems 392
11.10 Automated control systems 395
11.11 Project 1: An autonomous remote-sensing irrigation system 399
11.12 Project 2: The amazing all-knowing wizard 404

12 Building robotic systems 410
12.1 What is the difference between a robot and an automated system? 412
12.2 Robotics 412
12.3 Types of robots 415
12.4 The purpose of robots 420
12.5 Robots in action 422
12.6 Robot movement 429
12.7 Software programs 431
12.8 Ethical issues in robotics 436
12.9 Project 1: Build a rescue robot 438
12.10 Project 2: Build a maze-solving robot 440

FOCUS AREA: Creating games and simulations 448

13 Creating games 448
13.1 The importance of games 450
13.2 A brief history of computer games 451
13.3 Computer games and AI 453
13.4 Issues in gaming 455
13.5 What makes a good game? 458
13.6 What makes a computer game? 459
13.7 Game requirements 460
13.8 Designing a game 462
13.9 Tools for creating games 463
13.10 Project: Creating a handheld game 466

14 Creating simulations 474
14.1 Modelling and simulations 476
14.2 The spreadsheet as a simulator 479
14.3 Games and simulations 482
14.4 Guided project 1: A virtual pet 488
14.5 Designing games using OOP 496
14.6 Project 2: Building an OOP-based adventure game 499

FOCUS AREA: Developing apps and web software 508

15 Developing apps and OOP 508
15.1 Designing and writing software programs 509
15.2 Functions 510
15.3 Programming languages 513
15.4 Complex data structures 515
15.5 Error detection 520
15.6 Correcting errors and program documentation 522
15.7 Guided project 1: A menu-driven game collection 524
15.8 Object-oriented programming 532
15.9 Classes, attributes and methods 534
15.10 Parent classes, child classes and inheritance 539
15.11 Three key OOP concepts 540
15.12 Project 2: OOP-based dice game simulation 545

16 Developing apps for the web 556
16.1 Apps for the web: HTML, CSS and JavaScript work together 557
16.2 What is HTML? 557
16.3 CSS essentials 562
16.4 JavaScript essentials 564
16.5 Canvas essentials 569
16.6 Guided project 1: Building a web-based quiz in JavaScript 573
16.7 Project 2: A web-based retro snake game 583

To students

The world of computing technology you will explore using this student book is an amazingly creative, exciting, challenging and rapidly changing one.

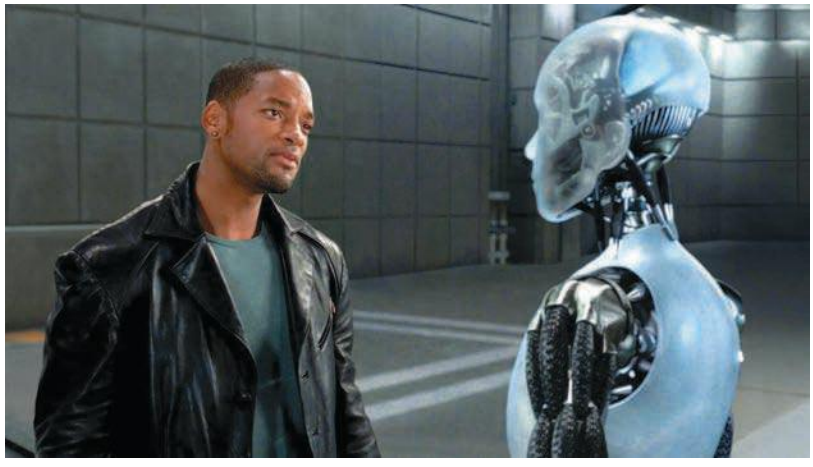
It is a world that offers something for everyone. If you are bursting with creative energy, you can explore digital media and multimedia. If you are a born organiser, you can contribute to this revolution by building a website or a database of your own. If you like to find out how things work, then topics such as artificial intelligence, networking and operating systems will fascinate you. If you love to be in control, robotics will give you a chance to design and program your own robots. Finally, the topic of programming offers to hand control over to you as you learn to write software yourself.

This book emphasises a project-based approach to computing technology. Topics are built around practical projects where you design, produce and evaluate your own solution to a given problem, either working on your own or as part of a team.

Most students know enough about computers to complete basic daily tasks, such as word processing, browsing the Internet catching up on social media or playing a computer game. However, as you learn about computing technology you will learn far more than this.

You will discover why the computer has been called the most complex and versatile device humans have ever built, and you will learn about a digital revolution that is only just beginning.

Source: Forth Interact

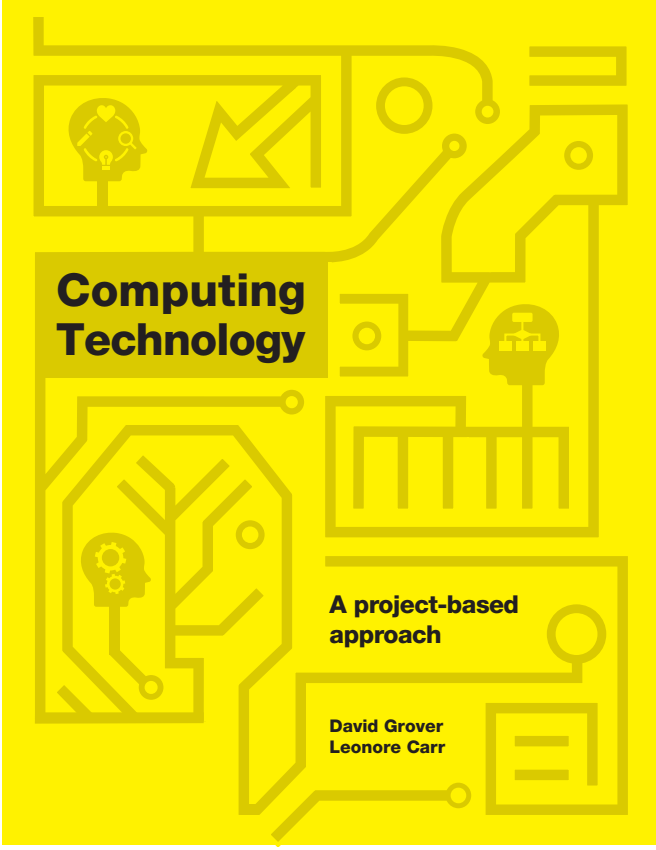


Alamy Stock Photo/Pictorial Press Ltd

```
answer = input('What is the meaning of life? ')
count = 1
while answer != '42':
    count += 1
    print('Incorrect.')
    answer = input('What is the meaning of life? ')
got it right it after', count, 'tries.')
```

About this book

Student Book



This edition of *Computing Technology: A project-based approach* comprehensively covers emerging technologies and practices.

The student book provides a clear project development focus in line with the spirit of the syllabus. The double-page spread format allows for easy navigation through the topics, while the activities at the end of each section and the wealth of questions throughout ensure student engagement.

Key features

- A clear, logical design makes the book easy to navigate.
- The inclusion of current computing technology advances ensures students are engaging in the most up-to-date technological landscape.
- Chapter-opening pages outline the syllabus outcomes and key terms used in that chapter.
- In action sections present real-life applications of the chapter content.
- Case studies present students with real-life material and highlight relevant examples of the theory.
- Skill builders develop essential practical skills.
- Knowledge builders develop an in-depth understanding of essential theory.
- Projects are provided for each Focus area. These implement skills and knowledge through extended hands-on practical work.
- Infobits provide snippets of interesting information to stimulate curiosity.
- People sections focus on profiling individuals working with technology.
- Engaging, hands-on group and individual tasks and projects illustrate theory and put it into practice.

Nelson MindTap

The student version includes:

- concept review quizzes
- supplementary resources
- datasets
- weblinks
- glossary

The teacher version includes:

- everything contained in the student version
- syllabus map
- student book solutions
- masterclass video presentations
- teaching guides for projects and project summaries
- teaching program

To access resources above, visit **cengage.com.au/nelsonmindtap**

Nelson MindTap

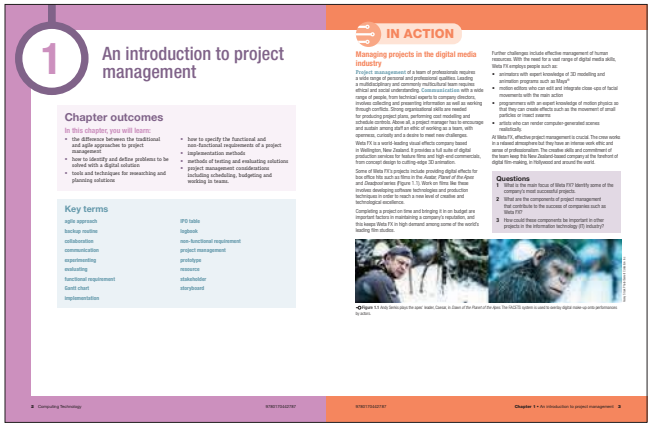
How to use this book

Computing Technology: A project-based approach, 1st edition, addresses the requirements of the New South Wales Years 7–10 Computing technology syllabus.

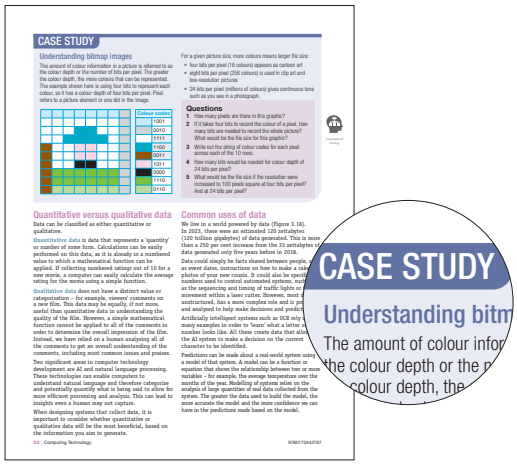
The book is divided into three colour-coded sections corresponding to the introductory content, the enterprise information systems content and the software development content.

There are 16 chapters in this book, with the following features included throughout.

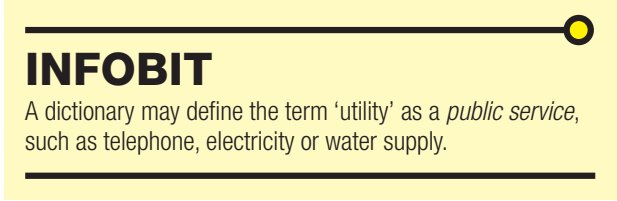
- **Chapter-opening pages** outline the syllabus outcomes and key terms used in the chapter. The chapter begins with an 'In action' section that presents a real-life application of the chapter content.



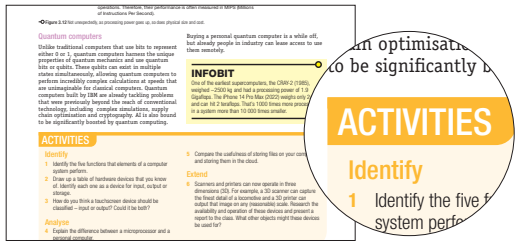
- **Case studies** highlight relevant, real-life examples of the theory.



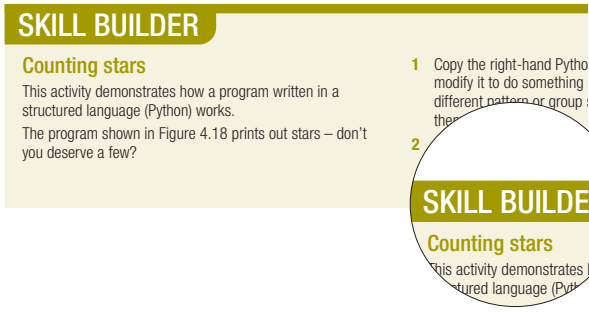
- **Infobits** provide snippets of interesting information to stimulate curiosity.



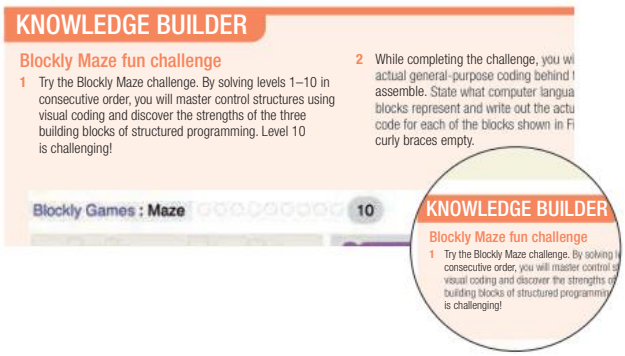
- **Activities** at the end of each section in the chapters are broken down under the following headings:
  - Identify: comprehension and recall questions that check understanding
  - Analyse: extension questions where students are required to apply their knowledge
  - Extend: higher order activities requiring more critical thinking.



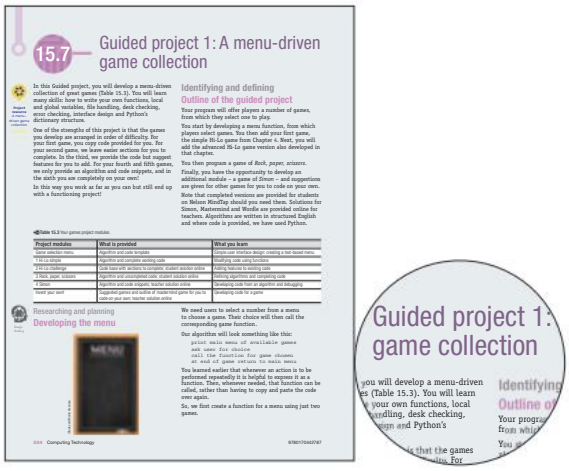
- **Skill builders** provide hands-on activities that develop essential practical skills and support the skills needed to complete the projects.



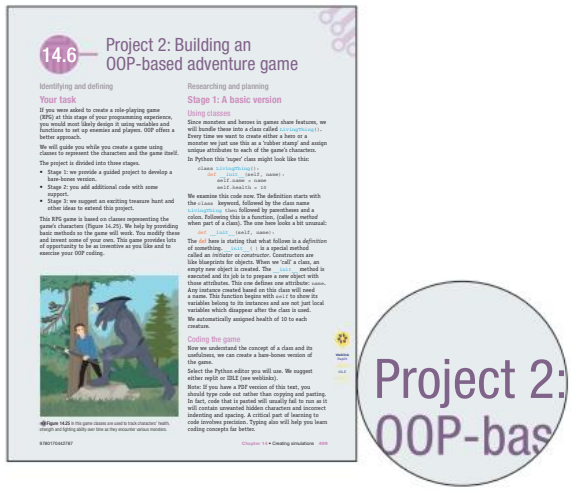
- **Knowledge builders** provide in-depth understanding of essential theory and develop the mastery needed to complete the projects.



- **Guided projects** provide detailed step-by-step instructions to support students at all levels to complete substantial projects and develop the essential practical skills needed for the standalone projects. Most projects address outcomes in other focus areas. A teacher guide details each project, its outcomes and its level of difficulty.



- **Projects** are a feature of this text. These are provided for each focus area and implement skills and knowledge through extended hands-on practical work. They can be conducted individually or as part of a group. Projects also contain suggestions for those who wish to develop projects further.



About the authors

David Grover

David Grover was Head Teacher of Computing at Chatswood High School, New South Wales, and is the author of a number of texts and resources in secondary Computing. He conducts many popular workshops for teachers of Computing and has been a senior marker for HSC Computing. He has many years' experience in the training of Computing teachers at a number of tertiary institutions.

David has conducted research in the educational applications of augmented reality (AR) for Macquarie University, trains teachers in the creative use of technology and at present is enjoying building and programming various mechatronic devices! He loves sharing his excitement for emerging technologies with his students, many of whom have gone on to enjoy successful careers in animation, games design, robotics and programming. He has established a reputation for expertise in interactive digital education, been recognised in various teaching awards and received a NSW Premier's Scholarship during which he studied new and emerging technologies around the globe.

Leonore Carr

Leonore Carr has been a Computing teacher in New South Wales since 2010. She is the Head Teacher of Computing at Chatswood High School, leading staff in the development and improvement of programs and resources. She is also the Head Teacher of Teaching and Learning, supporting teachers across the school in their professional development. Leonore previously lead the Cross Curricular STEM Project at NESA, working with teachers and students across the state to create programs for K–12 that integrated Science, Technology, Engineering and Maths outcomes in engaging activities and projects. Her enthusiasm and fresh perspective make her an extremely valuable new addition to the team.

1 An introduction to project management

Chapter outcomes

In this chapter, you will learn:

- the difference between the traditional and agile approaches to project management
- how to identify and define problems to be solved with a digital solution
- tools and techniques for researching and planning solutions
- how to specify the functional and non-functional requirements of a project
- implementation methods
- methods of testing and evaluating solutions
- project management considerations including scheduling, budgeting and working in teams.

Key terms

agile approach
backup routine
collaboration
communication
experimenting
evaluating
functional requirement
Gantt chart
implementation
IPO table
logbook
non-functional requirement
project management
prototype
resource
stakeholder
storyboard

IN ACTION

Managing projects in the digital media industry

Project management of a team of professionals requires a wide range of personal and professional qualities. Leading a multidisciplinary and commonly multicultural team requires ethical and social understanding. **Communication** with a wide range of people, from technical experts to company directors, involves collecting and presenting information as well as working through conflicts. Strong organisational skills are needed for producing project plans, performing cost modelling and schedule controls. Above all, a project manager has to encourage and sustain among staff an ethic of working as a team, with openness, curiosity and a desire to meet new challenges.

Weta FX is a world-leading visual effects company based in Wellington, New Zealand. It provides a full suite of digital production services for feature films and high-end commercials, from concept design to cutting-edge 3D animation.

Some of Weta FX's projects include providing digital effects for box office hits such as films in the *Avatar, Planet of the Apes* and *Deadpool* series (Figure 1.1). Work on films like these involves developing software technologies and production techniques in order to reach a new level of creative and technological excellence.

Completing a project on time and bringing it in on budget are important factors in maintaining a company's reputation, and this keeps Weta FX in high demand among some of the world's leading film studios.

Further challenges include effective management of human resources. With the need for a vast range of digital media skills, Weta FX employs people such as:

- animators with expert knowledge of 3D modelling and animation programs such as Maya®
- motion editors who can edit and integrate close-ups of facial movements with the main action
- programmers with an expert knowledge of motion physics so that they can create effects such as the movement of small particles or insect swarms
- artists who can render computer-generated scenes realistically.

At Weta FX, effective project management is crucial. The crew works in a relaxed atmosphere but they have an intense work ethic and sense of professionalism. The creative skills and commitment of the team keep this New Zealand-based company at the forefront of digital film-making, in Hollywood and around the world.

Questions

1. What is the main focus of Weta FX? Identify some of the company's most successful projects.
2. What are the components of project management that contribute to the success of companies such as Weta FX?
3. How could these components be important in other projects in the information technology (IT) industry?

Alamy Stock Photo/Everett Collection Inc

Figure 1.1 Andy Serkis plays the apes' leader, Caesar, in *Dawn of the Planet of the Apes*. The FACETS system is used to overlay digital make-up onto performances by actors.

1.1 Approaches to project management

In all walks of life, from the classroom to national and international affairs, people are involved in projects. Some are tasks that can be completed in a matter of days, while others may take several years. However, a common thread is that all projects are designed to address a specific issue or problem. The degree of success of the project will largely depend on how well the process has been managed.

During the past 50 years, there have been many developments in the way project management is approached. The traditional 'waterfall' approach to project management breaks down a project into a linear series of stages in which each stage depends on the deliverables of the previous stage. Most project management approaches used today, such as the prototyping approach and the agile approach, include the iteration of stages. Throughout this course, you will work through the main stages of project management: identifying and defining, researching and planning, producing and implementing, and testing and **evaluating**.

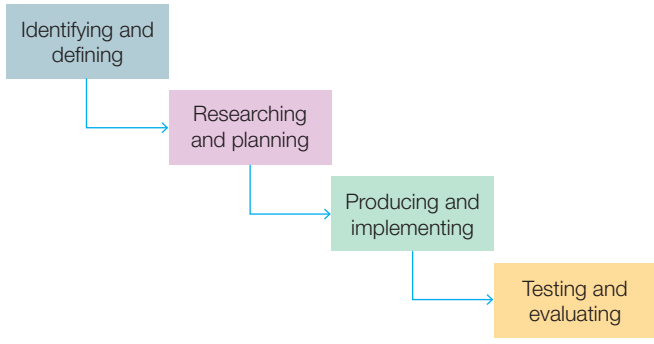


Figure 1.2 The traditional waterfall approach is commonly represented as these distinct stages flowing down from one another, illustrating that each stage feeds into the next and suggesting that there is no going back 'up' to previous stages.

Traditional waterfall approach

The traditional waterfall approach (Figure 1.2), or waterfall development model, was originally used in the manufacturing and construction industries. In these industries, where there is physical production of objects, mistakes made in the early stages could be very expensive if only discovered during the production stage. For this reason, this approach demands each stage of the process must be completed and documented before the next stage can begin.

The IT industry widely adopted this traditional approach, due to the benefit of the strict stages and thorough documentation requirements, particularly in large-scale software applications. However, given the non-physical nature of many IT projects, there were many limitations in not being able to go back to a previous stage or run stages in parallel.

Agile approach

The **agile approach** (Figure 1.3) is now a commonly used method in software development and was born out of the need for more efficient and flexible project development approaches for IT projects.

Rather than defining strict stages, it provides a more flexible scaffold to guide project teams through iterative phases of planning, producing and feedback. This allows for regular checks that the product is meeting the needs of the client and establishes a development environment that easily adapts to changing requirements.

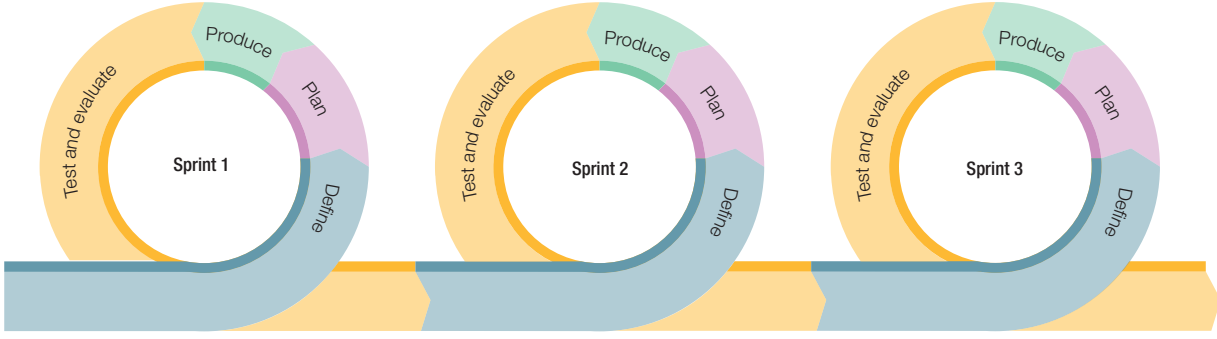


Figure 1.3 The agile approach encourages the early production and testing of a solution before repeating the process, further developing and improving it each iteration.

Stages of project management

When you start the process of addressing a specific issue or problem, the first thing you must do is make sure you clearly identify what issue needs to be solved. Only then will you begin to define the key features the solution aims to solved (Figure 1.4). A problem that is not clearly understood will be hard to solve.

Once the problem has been defined and clarified with the appropriate **stakeholders** (e.g. clients), you can start the phase of researching and planning towards a solution. This phase could include consideration and planning for factors that could impact your problem solving, generation of possible solution ideas, and applying design techniques to communicate or try out ideas (e.g. a **storyboard**).

Following your researching and planning, the solution can begin to be produced using identified IT (hardware and software).

The last phase, testing and evaluating, involves checking whether the solution is working as expected and is meeting the needs of the users.

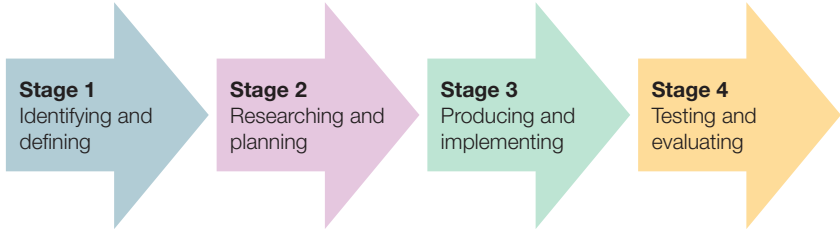


Figure 1.4 The four stages of project management include distinct steps and deliverables, which may be used in different ways depending on the overall development approach.

KNOWLEDGE BUILDER

Comparing the waterfall and agile approaches

1 Do some further online research into the waterfall and agile approaches to find out more on the differences and similarities between the two.

2 Summarise your findings in a table like the one below. Feel free to add more rows for other features to compare the approaches.

	Waterfall	Agile
Stages		
Team sizes		
Documentation		
Best use		
Advantages		

ACTIVITIES

Identify

1 Identify the main phases of project work. If you are not yet creating a product in the first two phases, why are these still very important?

Analyse

2 A skateboard company is developing a website for its brand. Which approach would you recommend that the company use for the production of its website? Why?

3 Think of a project you have recently completed. What researching and planning techniques did you use and did you find them useful?

Extend

4 Investigate two other project management approaches commonly used today. What are some of their key features and what type of projects might each suit? Represent each approach diagrammatically, using Figures 1.2 and 1.3 as examples.

1.2 Identifying and defining

This initial stage of project development is more important than it may first appear to be. Although you will not be making anything or even sketching any ideas, this stage is the springboard for the rest of the project, so it must be approached with an open mind and commitment. Without properly identifying and defining what you are trying to do, you could go down a path that does not solve the problem and wastes a lot of time and money.

Identifying the problem

One of the first tasks that must be completed before we can solve any problem is to work out exactly what the problem is or what need we are trying to address.

We might also question whether in fact the problem needs to be solved at all. If your friend rang you one evening asking for help because she could not do her Maths homework, it would help to know whether it was because she didn't understand Maths or because she was going to the movies and didn't have time to do it. You need to identify the problem. You could then decide whether to tell her you are busy or find out more about her problem.

You will already be familiar with the completion of projects in many areas of your study at school. At higher levels of education and in the workforce, these projects become more demanding in terms of originality and documentation.

The processes that you go through – both physical and intellectual – will be an education in themselves. Learning to solve problems is a valuable lesson in any walk of life.

It is useful to keep written records of what was done and when, so that everyone knows how the project is developing. You will find that documentation is essential in large business projects. Even in smaller, individual projects documentation can be a useful tool to reflect on your processes and to communicate key features and processes to others, such as the teacher reviewing your work.

Read the scenario in the next case study, which we will follow throughout this chapter. Take some time to consider the problem and discuss your ideas within your class groups.

Functional and non-functional requirements

From the first stage it is important to start outlining the requirements of the system to ensure it meets the needs of its users and is a success. These requirements define what a system should do and how it should do it, referred to as the functional and non-functional requirements.

Functional requirements describe the specific features and functionalities that a system must provide in order to fulfill its intended purpose. They are concerned with what the system should do, and typically relate to the system's behaviour and outputs. Examples of functional requirements might include the ability to log in, search for products or add items to a shopping cart.

Non-functional requirements are concerned with how the system should perform and are typically related to the system's quality attributes, such as usability, reliability, performance and security. These requirements commonly describe system characteristics that are not visible to end-users, but are important for the system's overall success. Examples of non-functional requirements might include the system's response time, availability, scalability and maintainability.

As a system is developed, requirements should be assessed by applying use cases and developing test cases of inputs and expected outputs.

Factors that impact on problem solving

In any project, there are various factors that might impact on any planned solution. These fall into five categories: time, technical, operational, financial and ethical.

- Time factors are crucial and can affect the success of a project significantly. If the project is not finished in time, there can be severe financial consequences and other related projects may be negatively affected as well.
- Technical factors might include whether the available hardware or software is adequate for the task. Is there other hardware that could be used to input information into the system?

- Operational factors need to be considered to determine how the new plans would fit into the existing system or workplace. Will the staff be comfortable using a new system? Should training opportunities be offered?
- Financial factors will influence the budget. How much will the whole project cost to set up and maintain? Will it save money and increase profits eventually?
- Ethical factors influence ideas about data access and marketing practices. New systems commonly have unintended effects that should be assessed. What information should be stored on the system and who will be allowed access to it?

CASE STUDY

Implementing a computer system – Part 1

Identifying and defining the problem

Denis is the owner of a local hardware shop (Figure 1.5) and has been running his business the same way for 20 years. Recently, customers have complained about being incorrectly charged for items and his stock regularly being out of date. He also suspects that he has a large stock of items that have been sitting in the storeroom for years. Perhaps it is time he sold the shop and retired, or maybe he should replace his assistant Lisa, who has difficulties using the computer system properly.

Susan is a business colleague of Denis. She asks if perhaps the computer system he is using is a little outdated. More modern hardware and software might be easier to use, so mistakes would occur less often and some stock analysis could be implemented. It should be possible to solve the problem with just a little thought and planning.

Denis and Susan begin to define the problem. They write down exactly what the problem is and several ideas on how it can be solved. They soon realise that there are a few factors (Figure 1.6) that might affect some of their plans. For example, they know they will need some new software, and probably hardware. How much money can they afford to spend on the project and how will Lisa, the assistant, react to a new computer system?

Case study continues in next section …

Alamy Stock Photo/adrian arbib

Figure 1.5 Organising stock in a hardware shop.

Questions

1. What is the problem that Denis is having with his hardware shop?
2. What are some of the solutions Denis has thought of? Do you think they are reasonable?
3. Which of the five factors listed in the text are considered in this scenario by Denis and Susan?
4. What other factors should be considered? Why are these important?

Figure 1.6 Factors that influence problem solving.

INFOBIT

A study by a leading project management consultancy in the United Kingdom found that only 20 per cent of projects were considered to be truly successful and the reason they were in each case was effective project management.

ACTIVITIES

Identify

1 Outline what needs to be done in the first stage of project development.

2 What is the difference between functional and non-functional requirements? Give an example of each.

Analyse

3 What projects have you been involved in at school? Do you feel you approached them in an organised way? What were the most important factors you needed to consider?

4 What would be the most important factors to consider when working on a large project?

Extend

5 Investigate some large projects that have been completed around the world – for example, building venues for sports events, installing the National Broadband Network (NBN) throughout Australia, or developing online health systems. Choose one project and prepare a report that describes the nature of the project and its success or otherwise in terms of completion times and budget.

1.3 Researching and planning

In the researching and planning phase, we create several possible solutions for the project. It is important to write down the criteria that the solutions must meet in order to keep the ideas on track. Think about the target audience and the time frames and **resources** available.

Generating ideas

Generating ideas can be a creative and spontaneous pursuit. It is useful to document ideas so that you can revisit them later or share them with other people. You might choose to use a concept map (Figure 1.7). There are computer programs, such as Inspiration®, and online tools, such as Bubbl.us, that can help you to draw these diagrams. A concept map helps to formulate your ideas in a logical and connected way.

Research techniques

For thorough research that provides a strong foundation for developing a quality solution, a variety of research techniques should be selected and applied. For all research sources, make sure you write down where you found your ideas. You will need to acknowledge these sources of information formally at the end of your project. Acknowledgement of sources is important to prevent plagiarism and provides a record of your sources to revisit later.

Surveys and interviews

Surveys and interviews can be used to collect data from your intended clients or users. Surveys can be a very efficient way to gather data from lots of people. Survey questions need to be designed so that users can easily and accurately provide their responses and you can easily quantify and analyse responses. Interviews allow you to design and deliver questions to individuals with the aim of finding out their needs and wants on a more detailed level. They have the advantage of allowing follow-up questions if the interviewee responds in a way that requires further explanation or exploration.

Observations and case studies

This is probably your 'go-to' method for research. What better way to get ideas for projects and development techniques than to see what others have done before you? Your teacher may show you an example project from a previous year or you may do some online research into similar projects. The important part of observing other's work is your analysis and evaluation. These reflective processes allow you to understand how the project was completed and which features you think are applicable to your project. Through this research you may also learn about what was not successful so you can avoid these mistakes in your own project.

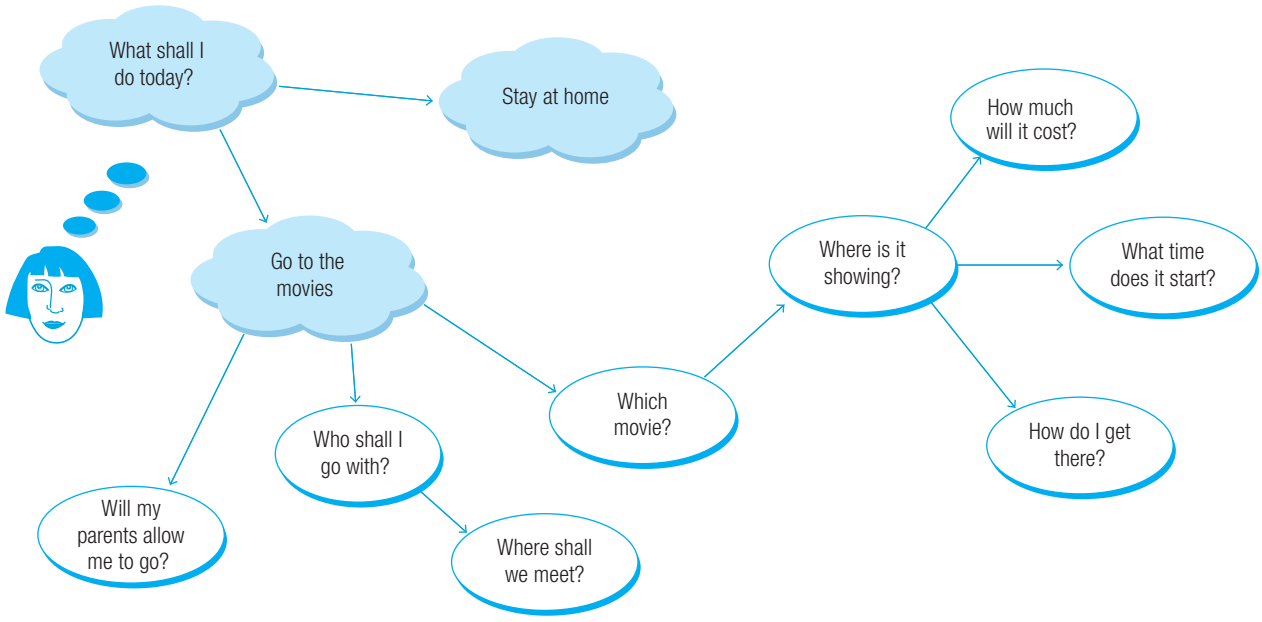


Figure 1.7 Planning a trip to the movies.

Experimentation

Once you have some initial ideas for a project, **experimenting** with them will give you some insight into whether they are suitable. For example, you may be thinking of using a particular piece of software so use this to create a simple product, or even just follow a tutorial to get used to the tools and techniques. Experimenting may also involve creating **prototypes** and testing them out on users.

Planning tools and techniques

Planning involves looking at the what, when, who and how, with a range of tools and techniques available to help you produce the necessary documentation to support your successful project development.

One planning tool you will use in all of your project work is a time plan. It is important to create a schedule of tasks once you have a good idea of your solution. A popular type of time plan diagram is a **Gantt chart**. To create a Gantt chart, draw up a table of tasks that need to be done and allocate a realistic time frame to complete them. You can also use arrows to show the dependencies of tasks being completed before others can begin. Figure 1.8 shows an example of a Gantt chart. You will need to list specific tasks in your time schedule. When collaborating on a project, the time plan can also show who is responsible for what task. This tool can be a 'live' online document, allowing it to be updated to reflect changes in ideas and when work was actually achieved.

Other important resources that project developers in industry need to consider are money and the technical skills and knowledge of team members. These will be covered in section 1.6.

Planning of the product involves using tools and techniques that allow you to try ideas to see how they work. Depending on the type of project you are working on, you may want to build a prototype or draw up a storyboard to help you decide on the overall look of the project solution.

Software developers working on a new project commonly use a prototype to test their ideas. A prototype is a simple version of the final product. It will not have every detail working, but it can be used to gain feedback from typical users. For example, if a program was needed to produce an online quiz for students, the initial prototype could be used to judge whether instructions to the user were clear, or if all the modules connected with each other as planned. You could receive useful feedback by giving the user a list of questions to rate their opinions, or just by observing how they use the quiz.

Software developers may also use an input, processing, output (IPO) table to document their plans. An **IPO table** shows how data input to the program is processed to become output (Table 1.1). Using the same example of our student online quiz, it is important to identify exactly what the program is designed to produce (output), the data it needs to create that result (input) and also what work needs to be done in between (processing).

Resource
Gantt chart template

In many areas of a project, it is good practice to generate multiple plans so the features of each can be compared against requirements in order to evaluate them and select the most appropriate design. For example, a website developer could sketch a few layout designs for a client's new website homepage. All of these designs can be presented to the client for feedback on what they think works best. A tool such as the plus, minus, interesting (PMI) strategy may be helpful in structuring the critical evaluation of the design alternatives.

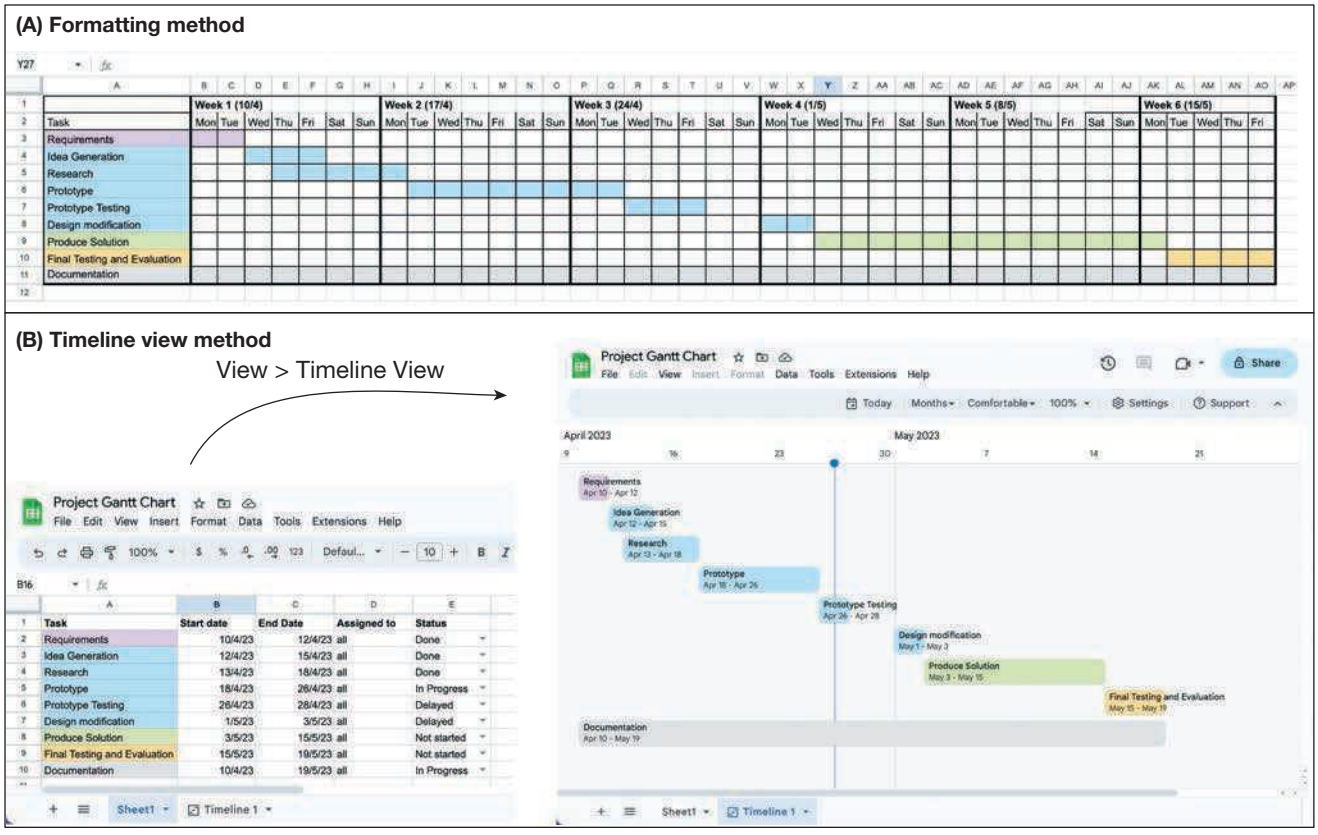


Figure 1.8 A Gantt chart used to help manage time spent on tasks to be completed in a typical project. (A) An easy way to create a simple Gantt chart is by organising tasks and dates in a spreadsheet table, then colouring cells to show time allocation. (B) Google Sheets also has a timeline feature that will generate a time plan for you from a table of tasks, and start and end dates.

Table 1.1 An IPO table for an online quiz.

Input	Processing	Output
• Username • User responses to questions	• Find highest scorer • Compare to correct answer • Increase score for each correct answer	• List of highest scorers • User score • Answers to questions that were answered incorrectly

Hardware and software

During the researching and planning stage, it is important to research and plan the technology you will need to produce and implement your solution. For your project's context, you should investigate what common hardware and software are used, their benefits and limitations, and how they can be used for your product. For example, if creating a game you could consider coding in a development environment or using a gaming engine such as Unity. For your game graphics you could use a paint application like Adobe Photoshop or vector-based applications like Adobe Illustrator and Adobe Animate. Alternatively, if your game is 3D, Blender may be a better choice for graphics creation.

The platform the game will be played on is an important consideration at this stage as not all development software will compile to all platforms. For example, will it be run in a browser or for a particular operating system such as a mobile application? Hardware requirements, such as CPU power, RAM, storage and display resolution, also need to be considered, both for the device used to create the product and for the intended users' devices. For school projects you will often be limited by the hardware and software already available to you. However, it can be interesting to research ideal technology, preparing budgets and investigating what resources are available that could help you learn how to use the technology.

CASE STUDY

Implementing a computer system – Part 2

Researching and planning

Our friends in the hardware shop write down some clear ideas on the time frame, budget and requirements for the new system. They start to experiment with a database program to see how the information about stock and suppliers can be organised. Susan has seen a website that might give them some ideas for giving customers online access to information. She suggests that they could also collect details about their regular customers and introduce some marketing ideas such as loyalty reward schemes and special discounts.

A storyboard is a suitable means for designing many projects, such as an animation, a movie or a complete website. Simple sketches of screen layouts will help to keep graphic design elements and navigation controls consistent (Figure 1.9).

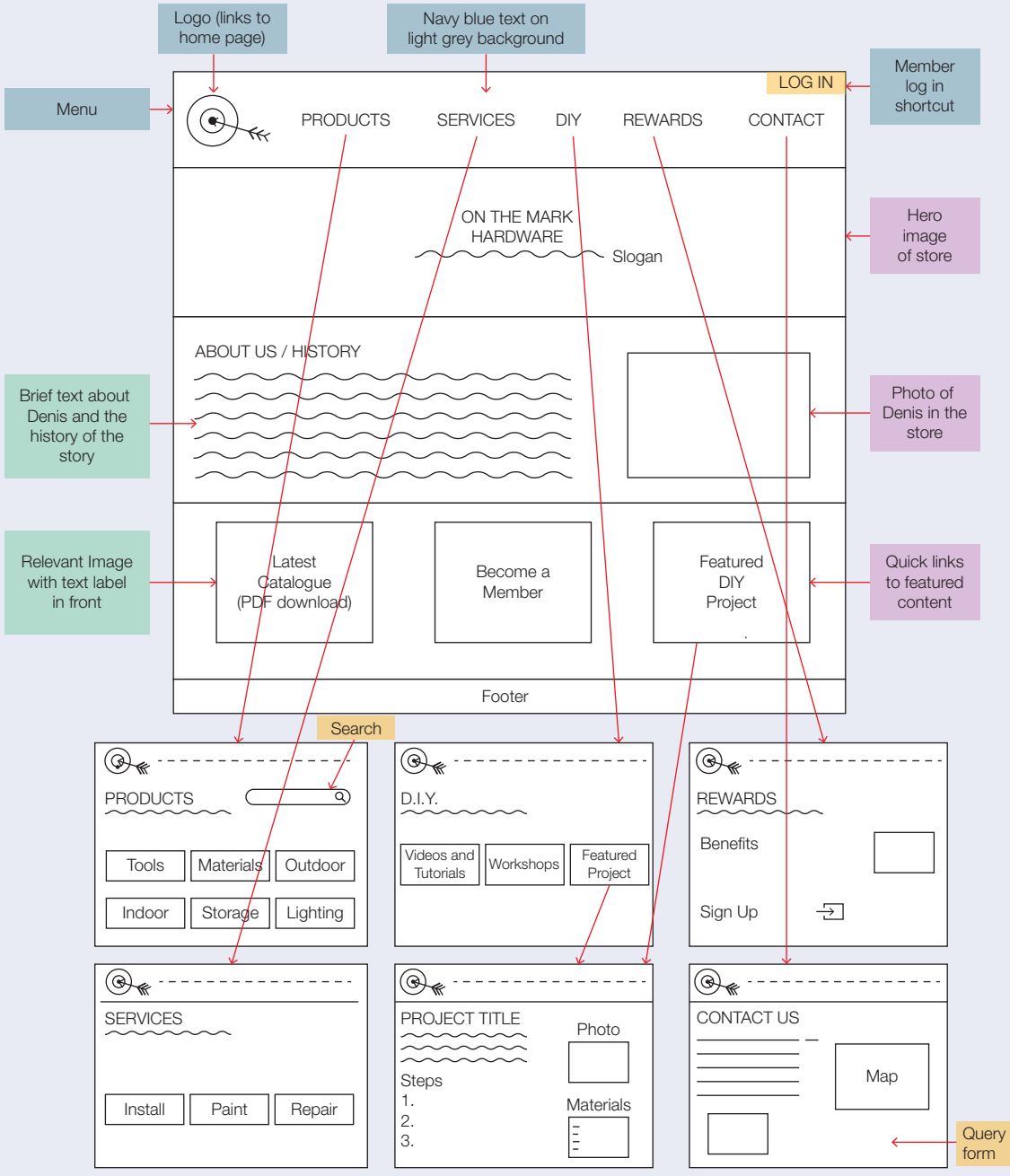


Figure 1.9 A storyboard for a hardware store website. Consistency in design means layout, colour and fonts are similar on all slides.

For example, in an interactive presentation it is important to ensure there is a link to the main menu on all pages, and that company logos and user instructions are placed appropriately on each page.

Storyboards allow members of a team to understand the various parts of the production. During the design stage, a storyboard makes it easy to rearrange scenes, change descriptions and modify, add or delete parts while creating the complete story. Susan sketches out a storyboard to document her ideas about the website design.

Case study continues in next section …

Questions

1 What research techniques have Denis and Susan used to generate ideas? What else could they do?

2 What other activities should Denis and Susan be considering at this stage?

ACTIVITIES

Identify

1 What is a prototype? What is its main purpose?

Analyse

2 You want to include a video clip in a presentation you are making to your local sports club. What are the possible ways you could do this and what might be some of the problems? Brainstorm some ideas and draw up a concept map of possible solutions.

3 Draw up an IPO table for the purpose of calculating a budget for a holiday you are planning.

Extend

4 Access a website that incorporates an online quiz. Sketch the storyboard for the site. Does it have clear instructions and navigation links?

1.4 Producing and implementing

Having spent a significant amount of time generating and documenting ideas, the next stage of your project will be to produce a solution and implement it. There are guidelines that must be followed while you produce your solution to ensure your product will meet standards set by the developers, industry and the clients. If these guidelines and standards are clearly documented, it can help you achieve your goals.

Producing solutions

The producing stage of project development is where all the planning and preparation start to come together. With your requirements defined, designs for your solution documented and the technology selected, you can begin to make your idea a reality. It is essential you document this stage carefully and follow some basic rules to maximise your project's chance of success.

Record of project development

A simple way to keep a record of your project development is through a **logbook**. Each time you make significant progress on your project, aim to note down what you did with sufficient detail. Reflect on what went well and what did not, and how you might resolve this when you next work on the project or face this problem. Ensuring you regularly evaluate your product and process leads to regular iterations of your design, giving it plenty of opportunity to improve and better meet the requirements. Capturing a visual record of the project, such as a screenshot or photo, is also useful evidence.

Basic rules and innovation

During this course, you will use many different software applications to produce some exciting projects. There will be frequent opportunities to think outside the square and investigate different methods of development. It is important, however, to remember some basic rules. It is better to have a simple product that works, rather than a complex and sophisticated program that is full of bugs. Also, losing your work at a crucial moment can be frustrating, so ensure you develop and follow a **backup routine** to regularly and securely save your files. Keep an open mind and ask for

constructive feedback from your peers. Discovering new techniques can be the best part of project work and can encourage innovative solutions.

Implementing

A product may appear to function as intended on the platform or in the environment where it was developed; however, this does not guarantee it will work correctly within its intended context. Implementing a solution is putting the solution into action. This **implementation** stage needs careful planning.

When planning implementation, there are many things to consider, including who will be using the solution and what their current knowledge and skills are, what other systems the solution will need to interact with, and what additional hardware or software needs to be purchased.

How implementation occurs needs to also be carefully considered (Figure 1.10). The simplest but most risky form of implementation is a *direct cutover* or change. This involves the new system completely replacing the old system in one event. Common implementation methods that are more forgiving of unexpected problems allow for testing gradually. This may be a *phased implementation*, where functionality from the new solution is added a bit at a time, or *pilot implementation*, where a selected group of users or a selected business location uses the solution before complete implementation. Another even more forgiving, although resource-inefficient method, is *parallel implementation*, where the new and old solutions run at the same time. If there is a significant problem, the old system acts as a backup.

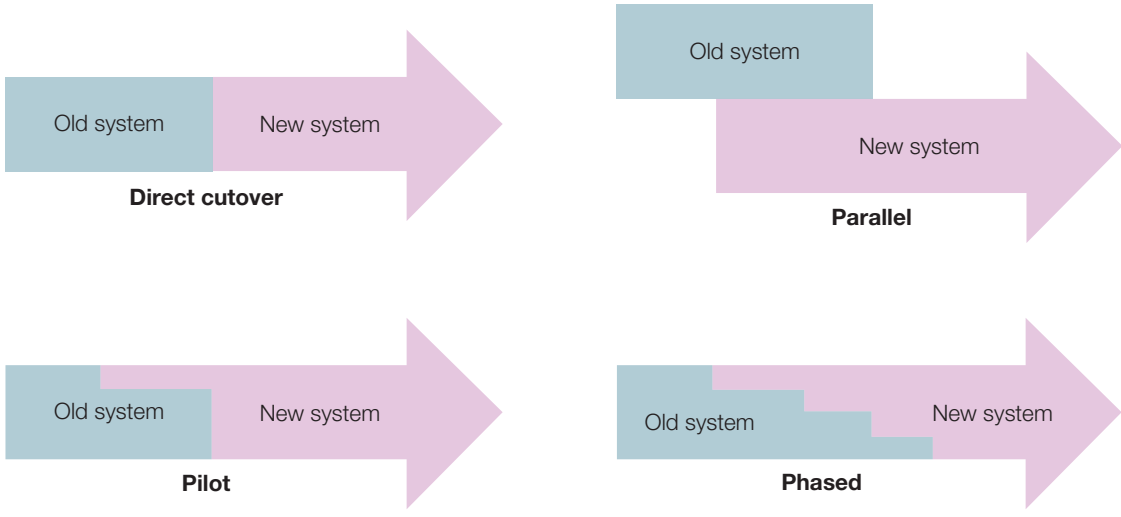


Figure 1.10 An implementation method is selected based on the complexity of the system, available resources (for example, time, money) and the impact if the system fails.

CASE STUDY

Implementing a computer system – Part 3

Producing a solution

Denis and Susan gave the prototype database that they had developed to the assistant, Lisa, to see if it worked successfully. Although she was quite an experienced computer user and keen to use the new system, Lisa seemed to have a problem understanding how it all worked. There was obviously a need for some training and online user help. In light of Lisa's comments, Denis and Susan decided they would be better off hiring a computer professional to manage the whole project. The time plan and financial analysis that they had carried out earlier confirmed this decision.

Denis and Susan hired Andrew, a computer project manager, to complete the solution for the hardware shop. They gave him all the documentation they had created, and Andrew discussed with them some other factors that they had not yet considered, such as data security and backup. Andrew was also able to advise them on the legal implications of collecting and storing information about people.

Weblink
Australian Privacy Principles

The Office of the Australian Information Commissioner website has information about privacy issues you must consider when collecting data about people.

Finally, Andrew suggested a direct cutover to the new database system for stock and suppliers, then a phased implementation for the website, initially starting with just information about the store and gradually adding functionality, such as customer login and a loyalty program.

Case study continues in next section …

Questions

1. Why do you think Denis and Susan decided to use a computer professional to complete the project?
2. What factors would have been important in making this decision?
3. What other issues do you think should be considered at this stage?
4. Find out about the Australian Privacy Principles and the laws about collecting information about people.
5. What are the benefits and risks of using a direct cutover method of implementation for the new database system? Why do you think Andrew suggested a phased implementation for the system?

ACTIVITIES

1. What is the purpose of a project logbook and what should it contain?
2. Why is it important to find the right balance between following basic rules and being innovative when producing your solution?
3. What are the different methods of implementing a new solution? What kind of project would use the direct cutover method of implementation?
4. Consider the following projects and identify what method(s) of implementation were used.
   - a. Opal travel card system: First used by school students, before full implementation on trains, then implemented on buses and ferries.
   - b. Digital driver's licence: First used by selected people living in the Eastern Suburbs of Sydney. In the first year, it was recommended that people still carry their physical licence with them as personal identification for entry into establishments such as hotels.
   - c. NBN installation: Initially implemented and tested in rural areas, then gradually implemented across the country over many years.

1.5 Testing and evaluating

Testing

Testing is the process of carefully checking your solution's functionality and goals against the requirements and existing standards. Testing should occur throughout the project development. For example, when building a website you will check that each component is working as expected throughout its development. Does an image display at the right size and location and do links navigate to the correct pages?

Evaluating

Evaluating a project includes looking carefully at the functional and non-functional requirements that were originally specified for the solution, explaining how they have been met and making a judgement of the overall success.

If, for example, you have created a digital media presentation for an advertising agency, it may be more important to use quality graphic design principles than to include lots of busy animations and sound effects. Similarly, file sizes may be a critical issue for a website design.

Functionality and quality

A common mistake made by developers is to forget they are creating a product for users and not for themselves. Evaluating the predefined functionality

requirements allows you to decide whether the application works in the way users expect. Thorough testing needs to be carried out to ensure there is no potential for misunderstanding by the user. For example, in a multimedia presentation, if an icon needs to be clicked to play a movie clip, it is a good idea to prompt the user to do it.

Quality of information in any project is a key non-functional requirement. Information must be accurate, otherwise errors can be made in processing data or making decisions. Also, it must be relevant to the project, otherwise it is useless and will take up valuable space. It may also be confusing to the user.

Integrity of information is a measure of its reliability, so information should be collected only from a trusted source. Finding information in a newspaper or on the internet does not guarantee that it is true. Information can also become out of date. For example, clients can move house or change jobs.

Scalability of a product relates to how it handles significant increases in the amount of data and processing involved. Ensuring a system can handle increases while maintaining efficiency is an important non-functional requirement related to quality.

Social impacts, ethics and the environment

Social and ethical considerations will have also helped to define other important non-functional requirements for evaluating projects. For example, there is a reasonable expectation that clients' personal details will be protected from hackers through the application of up-to-date cybersecurity practices and that any statistics produced from stored data will be presented in a fair and honest manner.

In addition, we must all take responsibility for protecting our environment, both close to home and across the world. Excessive use of paper and inconsiderate disposal of old hardware can have a major impact on our environment (Figure 1.11). Issues such as the placement of communication towers and clean methods of delivering power to our computers are further examples of such environmental concern.

Methods of evaluation

We have looked at a range of requirements that should be used to evaluate our projects. Here again, the users of the project are most important as we want to evaluate the user experience. We can also use different groups of users to test those requirements. Initially, of course, we must take time to step back and inspect the usability of our own work.

Observing a peer or teacher using the system and asking for feedback is a valuable experience. Try to reflect on any comments so that you can modify your work to meet the requirements more effectively, and also to improve on performance in future projects. Where a project is targeted at a particular group of people or a specific end user, it would make sense to gain regular feedback from these people during the development of the project. It might be useful to write down some specific questions for them to answer, to elicit specific comments on all aspects of your work.

Getty Images/Uriel Sinai

Figure 1.11 Recycling old hardware can reduce the impact of technology on the environment.

CASE STUDY

Implementing a computer system – Part 4

Testing

Throughout the development of the database system, Andrew, the project manager, performed regular testing on individual modules and the system as a whole. This mainly involved running procedures with the system to check that the output was as expected, such as generating sales and stock reports at the end of the day. Andrew also got Denis, Susan and Lisa to try the system as part of user testing, observing their interaction with the interface and interviewing them to identify any difficulties or concerns they had as well as any additional functionality they would find useful. A series of final tests were also performed and documented so Denis and Susan would be confident in their system before the direct cutover.

Evaluating

Part of this final testing involved reflecting on the requirements and whether the system had achieved the intended goals.

Through their user testing and from Andrew's clear documentation, Denis and Susan felt the new system was going to make their ordering, sales and stock checking procedures much more efficient and accurate. They could also see how this database, when integrated with their intended website, would allow them to access more clients and develop marketing strategies to improve their business success.

Case study continues in next section …

Questions

1. Why is it important to do ongoing testing when developing a solution like this?
2. What is the difference between testing and evaluating?
3. Do you think a waterfall approach or an agile approach was used in this project's development? Why do you think this approach was used?

ACTIVITIES

1. Why is it important to check each component of a website throughout its development? Provide an example to illustrate your answer.
2. How could you evaluate the usability features of a website, such as ease of navigation and user prompts?
3. Why is the quality of information an important non-functional requirement in project evaluation? Discuss the potential consequences of inaccurate or irrelevant information.
4. How are social and ethical considerations important in project evaluation? Provide an example of a social or ethical non-functional requirement that should be considered.

1.6 Project management considerations

We have seen how to use specific processes, tools and requirements to develop successful projects and we will look now at other factors that contribute to a satisfactory outcome. The bigger the project, the more important these considerations will be.

It is estimated that there is an 80 per cent failure rate on projects in the information and communication industry due to lack of project management. This can become very expensive and frustrating for the workers and the client. Project managers have specialised jobs in the IT field. They are responsible for the overall success of projects and must therefore carefully plan the use of resources such as time, finances and people.

Time

Having a clear schedule is essential for completing a project on time. Remember that problems may arise while you are developing your solution, so it is a good idea to work well ahead of deadlines. Resources such as people, printers and computers all have the potential to let you down at the most inconvenient moments. Also, making secure backups of your work will pay dividends and save you time.

Finances

Money is always an important issue, no matter how big the project. Trying to predict how much a project will cost is a difficult task, but estimates and limits can be determined by examining other similar projects. Keeping track of how much has been spent, and on what, is an easy exercise using a spreadsheet (Figure 1.12). When formulae are used in the spreadsheet, it is easy to adjust the budget as changes occur and instantly see how this impacts the overall costs. Other complex analysis and predictions can be carried out, and graphs can be used for comparisons and trends in spending.

Organising a team of people calls for special talents. A big project may need to be serviced by a variety of people, each with their own area of expertise. Their availability and correct understanding of needs will have to be coordinated by the project manager. Their costs and work location are issues that need to be negotiated. Perhaps some tasks could be outsourced (sent to people outside the company for completion) so that expertise and office accommodation costs can be optimised.

People

Very few complex projects are developed solely by individuals. In any area of business, groups of people are brought together to work on projects so that there is a range of expertise within the team. People find themselves working with people they don't know very well and who may have different views on a number of work issues. The challenge for any team leader is to encourage **collaboration** so that each member of the team can contribute positively to the project. Every situation can provide opportunities to develop organisational, communication and negotiation skills. Luckily there are some highly functional project management software available to help with these processes (Figure 1.13).

Roles and responsibilities

Organising a list of the roles and responsibilities for the team is important at an early stage in the project planning. Individuals can then plan each task so that it fits with everyone else's schedule.

Sharing time and experience is important in developing group dynamics, and each team member should be prepared to do this. At group meetings, take turns in expressing ideas and reporting problems. You can learn a lot from listening carefully to other people's problem-solving techniques. Take the opportunity to understand other people's strengths and weaknesses.

When working as a group, it is important to let everyone have the opportunity to express their opinions in a positive and comfortable atmosphere. Write down ideas and decisions so there are no misunderstandings by any individual. Describing a problem and its solution can help to prevent similar problems being repeated in the future. If you are using email, make sure everyone receives a copy of each message. Accept comments from your partners as positive feedback and consider how any conflict might be resolved in a calm and friendly way.

Collaboration

Collaboration means working together to produce quality solutions. This is not always easy, as people work differently from each other.

It is essential to document ideas and decisions so that everyone in the group understands their responsibilities. Plan regular meetings so that team members will work to deadlines and can report their progress. Phone calls and emails are effective for reminders, but all members of the team need to be notified of any discussion that took place.

	A	B	C	D
1	Budget			
2	Train fare	$3.80		
3	Movie ticket	$13.00		
4	Snacks	$8.00		
5	Total:	$24.80		
6				
7				
8				
9				
10				

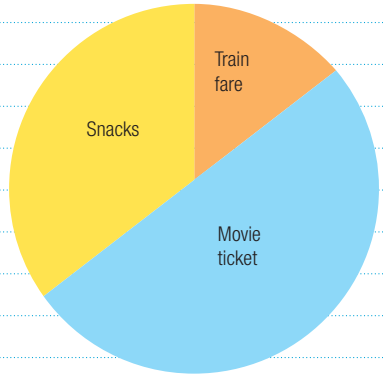
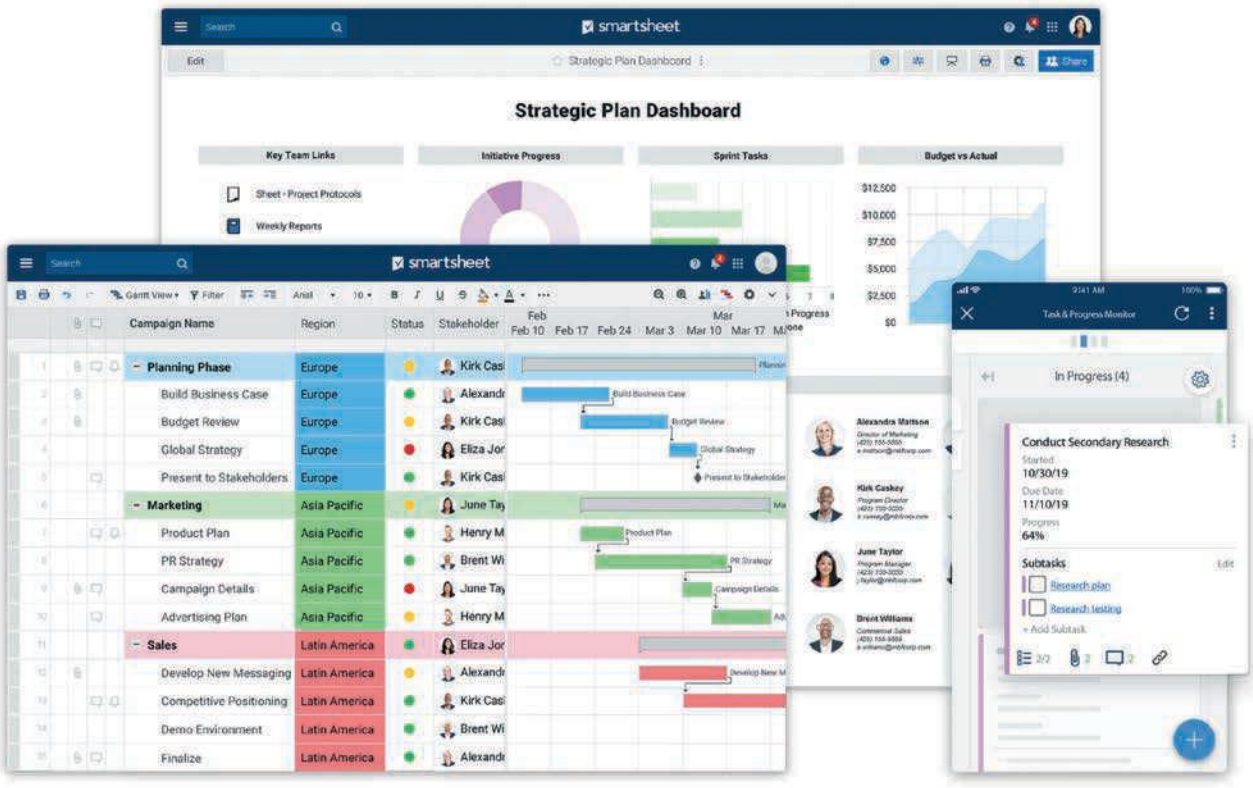


Figure 1.12 A simple spreadsheet shows the budget for a trip to the movies. Include formulas to automatically calculate totals.

Figure 1.14 illustrates how effective videoconferencing can be. With increasing bandwidth, and user-friendly tools such as Zoom and Microsoft Teams, videoconferencing can bring together people who live many kilometres apart. This has the advantage of:

- including a wider range of people with different knowledge and skills
- saving time
- saving on expensive travel and accommodation
- giving visual cues for speaking and agreeing.



Source: Smartsheet Inc

Figure 1.13 Large projects use professional project management software like Smartsheet to organise tasks, people and deadlines.

Shutterstock.com/DGLimages

Figure 1.14 Videoconferences allow people to work together even though they may be many kilometres apart, bringing significant benefits to a project.

Many software projects have been completed entirely online without anyone meeting physically. Online collaboration on documents has also become possible with the development of free and sophisticated documentation software. For example, blogs and wikis allow a large team to develop ongoing ideas and documentation.

Allow time at meetings for each member of the group to comment on the development of the project as a whole. Agree on strategies for assessment of individual contributions. If anyone seems to be falling behind the agreed schedule, it is important that the problem be discussed and a solution negotiated.

In the final evaluation of a project, you should reflect on the challenges that collaboration presented and the advantages you found from working as part of a group.

Communication

Effective communication within a team or with a client is an essential factor in the success of a project (Figure 1.15). There are various ways to communicate information and to present projects: verbally, in written format, and with or without graphical and visual aids.

Verbal

Verbal communication can be difficult if you are in an uncomfortable situation. If you need to express something clearly, take time beforehand to make a note of the main idea you wish to convey. Remember that the more you practise this skill, the easier it will become. Oral presentations of projects bring their own challenges. If you are presenting to an audience, make sure all equipment is working as you would expect. Practise the presentation using hardware in the actual location you will use on the day. Any visual aids must use text or images large enough for all your audience to see. A detailed animation will be difficult if your audience is too far away. Graphics should be clear and appropriate. Try not to read your script too much and never insult (or bore!) your audience by reading word for word from a screen.

Written

Written documentation is an essential part of any project. It is a good idea to write a diary of events so that you can reflect on the progress you have made. This is even more important for larger projects. Working collaboratively using online documents can be very effective. Copies of documentation and research findings should be clearly named and dated, and submitted with the final project so that they are easily accessed. Always acknowledge your sources of information and anyone who helped you with the project.

Graphical

Throughout this course, you will have the opportunity to incorporate many graphical images and visual aids into your projects. These can add impact and capture audience interest. However, you must ensure they are appropriate to the theme of the project and that issues of copyright are not violated.

Another option for presenting your project might be to publish it on the internet or your school's intranet. Your audience would have the flexibility of accessing your presentation in their own time, and you could ask for feedback from them by email or questionnaire.

INFOBIT

Project team members often use professional project management and communication tools, such as Slack, Smartsheet, Wrike and Trello, to manage and keep up to date with the progress of tasks. Social networking tools, blogs and wikis are also widely used, as well as Facebook, Twitter and Yammer.

INFOBIT

Project experts suggest that the maximum number of people in a team is eight. If the team gets any larger, team members can spend too much time communicating instead of applying their skills and knowledge to their part of the task.

Chapter review quiz

Figure 1.15 Important elements in group work.

CASE STUDY

Implementing a computer system – Part 5

Project management considerations

Andrew is an experienced project manager who has worked in the computer industry for many years. However, every project that he undertakes allows him the opportunity to think creatively and to apply some new ideas. He documents his ideas for Denis and Susan, and is confident he will keep within the scheduled time and cost estimates. He knows he will need to subcontract some of the work to other professionals he has worked with in the past. This will enhance the effectiveness of the solution and allow him to deliver the solution in the shortest possible time frame.

Denis and Susan have every confidence that Andrew will produce an effective solution to the problem in the shortest possible time. It is still important, however, for all of them to keep in touch and receive regular feedback as the project progresses. Andrew has been careful to make notes of all discussions and has kept the documents containing their original ideas. He will continue to do this throughout the project, and to date and sign all decisions that they make together. In turn, Andrew will need to pass on specific tasks and time schedules to the members of his team.

Questions

1. What are the major advantages in employing a project manager in this scenario?
2. Summarise the main types of communication used in project management.
3. What are the specific roles and responsibilities of the people working on this project?
4. Look back at the unfolding situation presented throughout this case study. Identify each step in effective project management.

ACTIVITIES

Identify

1 Identify at least three ways to communicate information and discuss the advantages of each.

Analyse

2 Consider a group project that you have completed for another subject. What strategies did you use to ensure the project was completed on time? What were the areas of most concern? How did you negotiate through difficult situations? What strengths do you think you bring to a project team?

Extend

3 Your school has just gone through a significant rebuild. You have been asked to create an interactive virtual tour using 360-degree cameras so parents and members of the community can see what the new school is like and how to navigate through it. List all the tasks that will need to be done before you actually start work on capturing the content for the tour. Create a storyboard and a Gantt chart. Estimate the cost of the project using a spreadsheet. Where will you get information from? Who will you include in your team to help you complete the project?

Weblink:

- Australian Privacy Principles

Resource:

- Gantt chart template

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

2 An introduction to thinking skills

Chapter outcomes

In this chapter, you will learn:

- the importance of the three thinking skills
- what design thinking is and how it is used
- what systems thinking is and how it is used
- what computational thinking is and how it is used
- how to use the stages of design thinking in your projects
- how to use the stages of systems thinking in your projects
- how to use the stages of computational thinking in your projects.

Key terms

abstraction
algorithm design
computational thinking
data analysis
decomposition
define
design thinking
empathise
ideation
modify
pattern recognition
prototype
systems thinking

IN ACTION

Three thinking skills and the Sydney Opera House

When we try to solve a problem, we use a variety of approaches, often without realising it. There are three important thinking skills that feature strongly throughout this course. They are **design thinking**, **systems thinking** and **computational thinking**.

Thinking skills are the mental processes we use to solve problems and to create things. They involve asking questions, developing ideas, making decisions, organising and evaluating.

A great example of this was the innovative and complex design and construction of the Sydney Opera House. This project took 15 years and provides an example of the way these three different thinking skills were used together to solve problems in a real-life project.

Getty Images/SCANPIX DENMARK/AFP

Figure 2.1 Jørn Utzon.

Design thinking in action

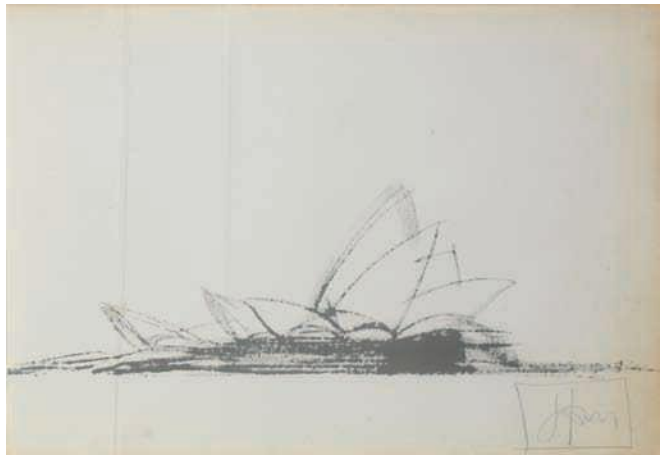
In 1957, the Danish architect of the Sydney Opera House, Jørn Utzon (Figure 2.1), won Australia's worldwide Opera House design competition based on the creative and original drawings he submitted.

Utzon said his greatest design challenge was how to construct the main feature of his original design: the unique roof shell shapes (Figure 2.2). No one had ever constructed such shapes from concrete before. Even creating shapes for the concrete moulds was challenging. This challenge occupied the building's engineers for years.

DESIGN THINKING

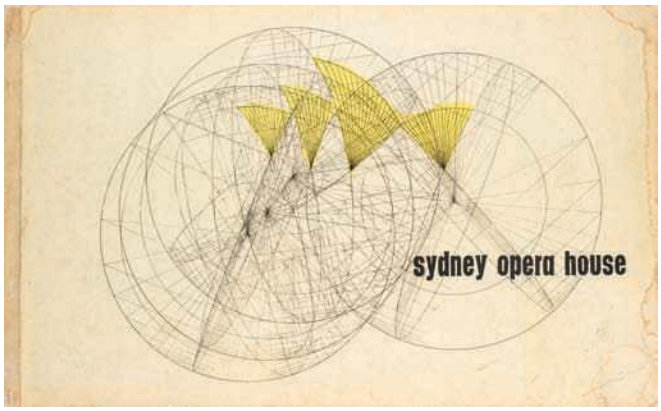
Design thinking involves a process where a need or opportunity is identified and a design solution is developed. The icon shown here identifies opportunities to apply design thinking throughout this text.

Design thinking

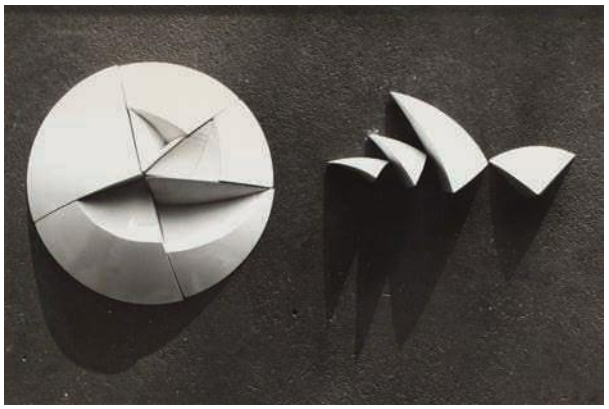


State Library of New South Wales, FL1027641

Figure 2.2 A sketch of Utzon's original rough design for the iconic Sydney Opera House sails.



State Library of New South Wales, FL967383



Arne Magnussen Helsingør

Figure 2.3 The design for the shape of the roof shells was inspired by a wet ball in a bath.

Utzon said that he noticed one day that when he pushed a ball belonging to his son into a bathtub of water, the water changed its surface colour and these coloured shapes closely matched the shell shapes of his roof design (Figure 2.3). He and the engineers had been trying out many geometric shapes, including parabolas. He now realised the sides of all the shells could be thought of as parts of a sphere.

He had used *design thinking* to achieve an overall solution to his problem, where just one shape could be used to mould concrete ribs for all the shells.

Systems thinking in action

Constructing the 'sails' was the most complex building task in constructing the Opera House. Many different components had to be used together to achieve the result. To succeed, *systems thinking* was applied.

The building firm invented a system of arches for assembling the roof 'sails', specialists developed spherical moulds to cast the concrete for the arch sections, and materials engineers developed a system for sealing their ribs between them and an ingenious system for pre-stressing them. Finally, a completely original system was developed for tiling the sails by placing them in the base of a mould and pouring concrete on top.

Each of these systems was necessary for a successful outcome and none of them could have been developed without understanding how they interacted with each other.

Systems thinking

SYSTEMS THINKING

Systems thinking is understanding how related objects or components interact. The icon shown here identifies opportunities to apply systems thinking throughout this text.

Computational thinking in action

The Opera House was commenced in 1959 and opened in 1973, when most engineers and architects still used slide rules for calculations (Figure 2.4). Many engineers believed the shells would be impossible to build as the calculations would be too difficult to complete.

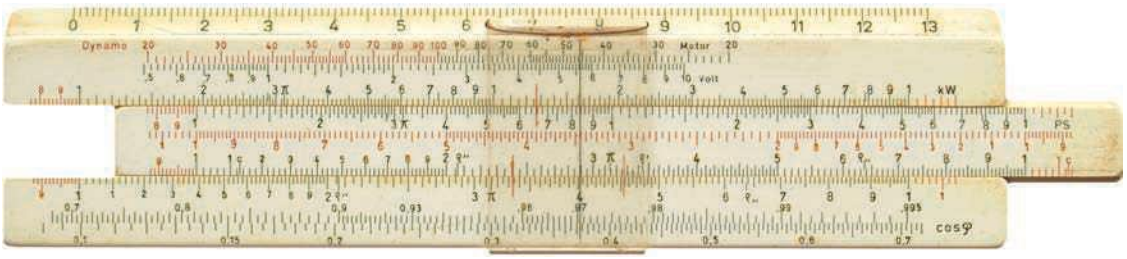
Every time a design change was made, the thousands of calculations had to be redone. Utzon's engineering firm used one of the earliest computers, the Pegasus, to help (Figures 2.5 and 2.6). Specialist programmers wrote custom software programs. They were using *computational thinking* to analyse the problem of solving these calculations.

The Pegasus took between 12 and 14 hours to run its programs, but, if this had been done manually, engineers believe the calculations would have taken at least 10 years. The computer code they developed solved the structural design calculations for the roof, as well as later helping with concrete fabrication and construction.

Computational thinking

COMPUTATIONAL THINKING

Computational thinking is a process where a problem is analysed and solved so that a human, machine or computer can effectively implement the solution. The icon shown here identifies opportunities to apply computational thinking throughout this text.

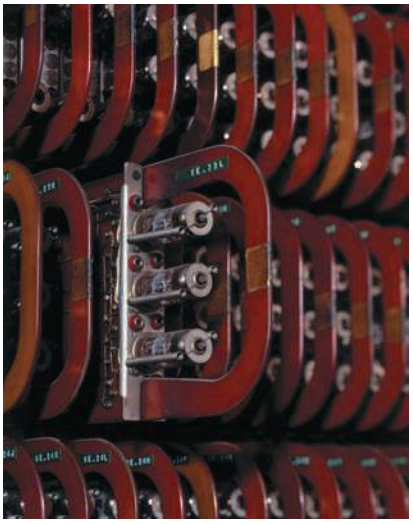


iStock.com/claudiodivizia

Figure 2.4 Slide rules were widely used before computers and calculators became available.

Alamy Stock Photo/Chronicle

Figure 2.5 The calculations performed by the Pegasus computer in 12–14 hours would have taken engineers at least 10 years.



Science & Society Picture Library/Science Museum

Figure 2.6 Further design thinking is illustrated by these Pegasus computer memory modules, which were plug-in modules about the size of a paperback book. Users stated this made it very easy to replace burnt-out valves in the computer.

Thinking skills

In completing projects in this course you will use each of these critical thinking skills at different times: design thinking, systems thinking and computational thinking. Different projects you undertake may feature one thinking skill more than others.

Each thinking skill can be broken down into stages, but you will not always need to think about these stages as you apply their principles. At all times, however, it is important to focus on the needs of the users as well as the problem needing to be solved.

2.1 Design thinking

Design thinking is a problem-solving process where a need or opportunity is identified and a solution is developed.

Design thinking must consider the economic, environmental and social impacts of any design proposed. As you learnt in the previous In action feature, each of these aspects proved important in the design process for the Sydney Opera House, as mounting costs became critical.

Design thinking can be divided into five stages (see Figure 2.7):

- **Empathise**: learn about the audience for whom you are designing.
- **Define**: decide main requirements based on user needs.
- Generate ideas: brainstorm and come up with lots of creative solutions.
- **Prototype**: build a rough model of one or more of your ideas to show to others.
- Test and **modify**: test your ideas and use feedback to modify them.

Empathise

In this stage of design thinking, the designer develops an understanding of the problem they need to solve with a focus on the needs of users. They research similar problems that have been solved, research the needs of users and talk to experts.

Architect Jørn Utzon had never visited Australia before he entered the design competition for the Sydney Opera House (Figure 2.8). However, from his home in Denmark he spent many days watching films and looking at photographs of Sydney Harbour and Bennelong Point where the Sydney Opera House would be located. He said his design was inspired by nature: by bird wings, by sails, by the shape and form of clouds, shells and palm trees. He was *empathising* not only with the natural location for the building but with the way onlookers would see it and users would experience it.

Mitchell Library, State Library of New South Wales, Australian Photographic Agency - 03802, FL1625440

Figure 2.8 Danish architect Jørn Utzon with NSW Premier Joe Cahill on seeing the location of the Sydney Opera House for the first time.

Define

During this stage of design thinking, information gathered during the previous step is collected and analysed.

Here, you (and your group, if you have one) decide which aspects of the problem are important. By the end of this stage, you should have an understanding of the problem and be able to *define* it clearly.

In the following stages, you begin to develop possible solutions.

Generate ideas

This third design thinking stage involves generating lots of ideas. At the beginning, you go for quantity, not quality – accept all suggestions! This stage is sometimes also known as **ideation**.

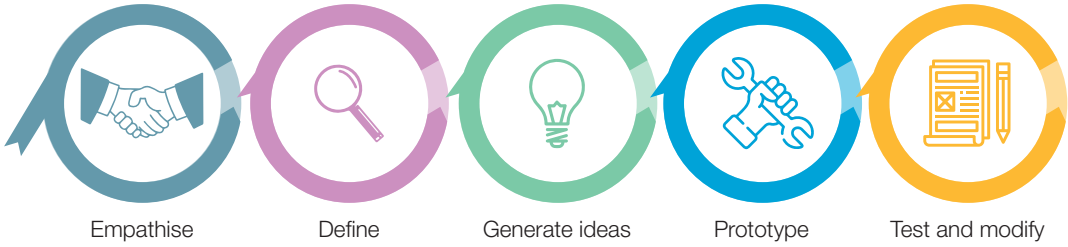


Figure 2.7 The five stages in design thinking.

Use information from previous stages to come up with ideas. Explore unusual or different solutions using earlier stages as your springboard.

Prototype

The fourth stage in design thinking is prototyping. A prototype is a rough working model of a solution. It doesn't matter if these are unfinished or fairly rough, as they are only meant to help in deciding which ideas are worth keeping.

A number of prototypes can be created to address different issues, even if there is to be only one final outcome. Jørn Utzon created many rough sketches and doodles before he arrived at his final design for the competition.

Test and modify

The final stage of design thinking is testing and modifying. During this stage, you test your prototypes to see how well they solve the problem you defined in the first two stages.

The whole design process should not be thought of as linear – one step followed by the next. You can jump backward and forward as often as you wish. At this point, for example, you may decide you need to create more prototypes.

You will then be able to decide what you can achieve and what might be too difficult. This may mean redefining the problem and working through the design process again.

KNOWLEDGE BUILDER

The paperclip

Shutterstock.com/Jakub Krechowicz

To understand better the process of ideation, all members of the class are issued with a paperclip. The challenge is to think of as many uses as possible for it, other than the obvious one of holding sheets of paper together.

Everyone is instructed to create something useful, using only their metal paperclip, by bending it. After 2–3 minutes, everyone takes turns explaining their invention.

Activity

Search internet images to check out the amazing variety of designs for paperclips that have been created since the first paperclip was invented in the late nineteenth century.

SKILL BUILDER

Design thinking: impossible flap design challenge

Can you create from a single square of paper the 'impossible' flap shown in Figure 2.9, using just straight scissor cuts (or tearing by hand)? No gluing, no sticking! The whole square is used, nothing is removed.

The solution is actually very simple, but we find it difficult because we have been conditioned to think about it in predictable ways. To solve it, you need to think differently! Generating lots of ideas is key.

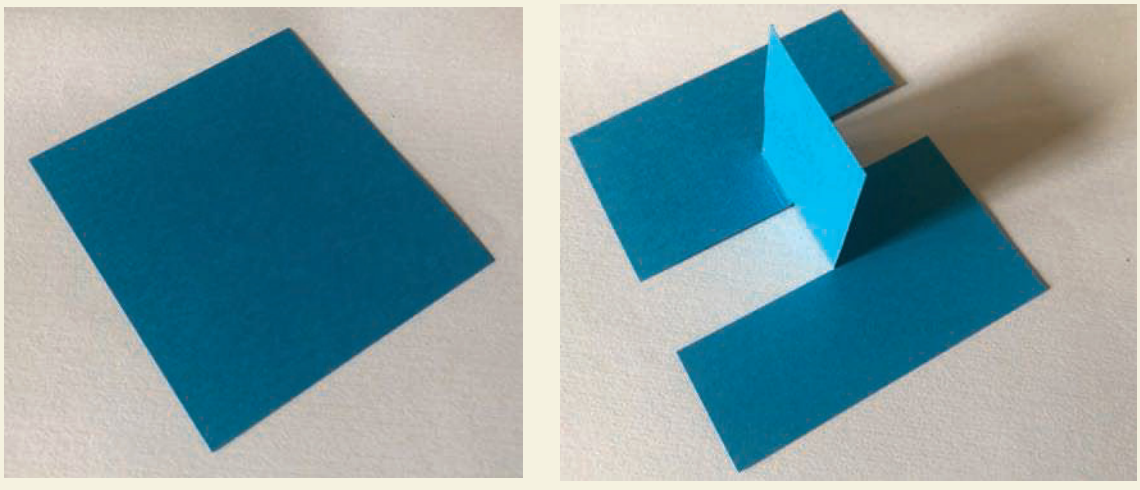


Figure 2.9 Exercise your design thinking – can you create this 'impossible' flap (b) from the paper square (a)?

2.2 Systems thinking

Systems thinking

Systems thinking considers how different components of a system interact with each other. It involves looking at the connections and interactions between all the separate parts that make up the system (Figure 2.10).

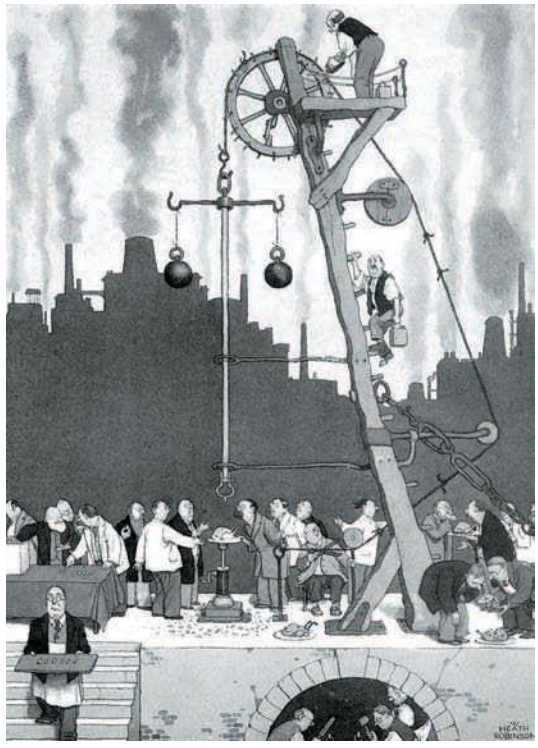
Because systems can be very complex, it is helpful to think of a large system as being made of many interconnected smaller ones (Figure 2.11).

An understanding of the interdependence of components is needed to solve many types of complex problems. We saw earlier how systems thinking was needed to solve the technical problems of the Sydney Opera House roof shells. It can also be used to solve economic and social problems. However, it is especially important when developing digital solutions.

It is no use being a systems or software engineer if you are unable to break down large problems into separate smaller systems. In programming, this is known as top-down or stepwise design.

The advantage of separating large systems into separate smaller connected ones is that different teams can be given responsibility for each subsystem.

You will often apply this approach in this course.



Alamy Stock Photo/Chronicle

Figure 2.10 Heath Robinson, and after him Rube Goldberg in the United States, were famous for their humorous drawings of overly complex interacting systems that solved simple tasks.

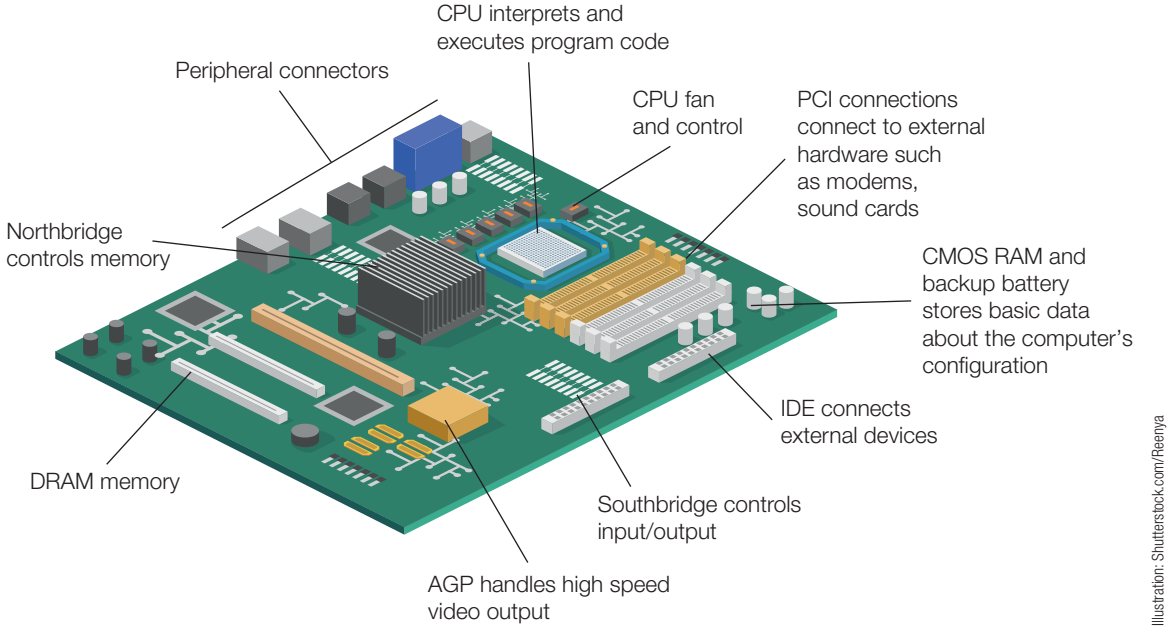


Illustration: Shutterstock.com/Reenya

Figure 2.11 The computer system sitting on your desk is made up of many subsystems, including the CPU itself, an input/output system, a graphics handling system, data storage, fan speed control and others. Computer engineers needed to apply systems thinking principles to create the computer systems we use today.

Systems thinking stages

Stages in systems thinking can be thought of as:

- Overall analysis: expected inputs, processes and required outputs of a complex system are identified.
- Component analysis: separate parts needed to create the more complex system are identified.
- Interactions: relationships and interconnections between the separate components are identified.
- Overall system: separate components are combined into a single functioning system.
- Impacts: once a system is developed, its effects on the people who use and engage with the system must also be considered. It may introduce unexpected costs or benefits on users or on wider society.

SKILL BUILDER

Systems thinking: building a self-driving car

One of the most challenging problems occupying systems engineers at present is building a reliable autonomous (self-driving) vehicle. Many separate subsystems need to work together and be completely reliable.

Data from at least three sensor subsystems (cameras, radar and LiDAR) must be combined to produce a 'map' of all obstacles and their location relative to the car (Figure 2.12).

Cameras use parallax methods from multiple images to calculate distances to various objects. They identify traffic lights and stop signs, as well as help recognise stationary and moving objects such as other cars, pedestrians and cyclists.

LiDAR (light detection and ranging) spin constantly, using laser beams to create a 360-degree image of the surroundings.

Radar (radio detection and ranging) also measure distance from the car to obstacles.

These three subsystems send data to the car's central computer system, which compares this to stored digital maps of the local road system, constantly producing updated representations of the surroundings. This data is then used by the car's driving subsystems: its acceleration, braking and steering systems. Other systems detect lane markings and other physical triggers.

Activity

Draw a diagram of an autonomous vehicle, showing connections between the various systems described here.

Source: Google

Alamy Stock Photo/Scharfsinn

Figure 2.12 The various sensor subsystems of self-driving cars work together to produce constant updates of the surroundings, identifying and distinguishing traffic lights, vehicles and pedestrians, and using this data to control the vehicle's mechanical subsystems.

2.3 Computational thinking

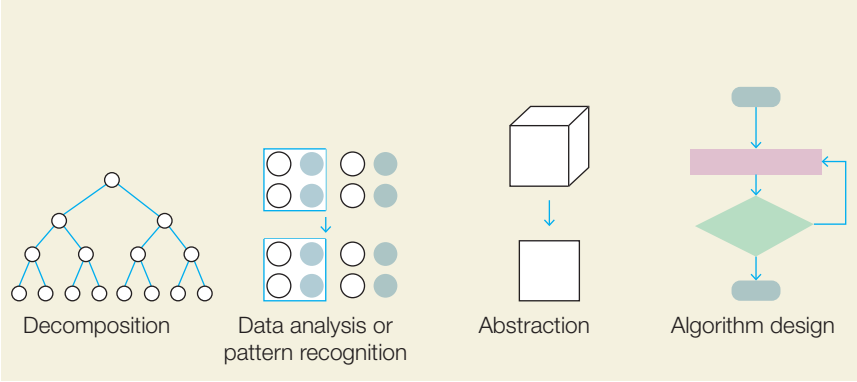


Figure 2.13 The stages of computational thinking.

Computational thinking

Computational thinking uses strategies that include organising data logically, breaking down problems into parts, defining abstract concepts and designing and using algorithms, patterns and models. It is a problem-solving process typically used to develop solutions that can be implemented by a digital system such as a computer.

Computational thinking can be divided into five stages:

- **Decomposition**: break big problems down into many smaller ones.
- **Data analysis** and **pattern recognition**: collect data and look for patterns.
- **Abstraction**: discover an overarching idea and hide the detail.
- **Algorithm design**: develop a rule that solves the problem.
- Transferring: use algorithms to help with other problems.

Decomposition

It is very difficult, and sometimes impossible, to solve highly complex problems. Very few software applications or mobile apps are now developed by just one person. The work is too complex. However, by breaking a big problem down into smaller simpler problems, a solution can often be found.

Decomposition is used to describe breaking a problem down into smaller and more manageable parts.

Data analysis and pattern recognition

The second stage of a computational thinking approach is collecting data and finding patterns.

The aim in computational thinking is to develop an algorithm or rule to solve a problem. To do this, we need to find patterns in the data we collect. By analysing data, patterns can emerge that trigger ideas in creating these algorithms.

Abstraction

The third stage of computational thinking is abstraction. Abstraction turns something complex into something simpler by removing detail.

The word 'abstraction' can be confusing, as we often use it to describe a painting that doesn't look realistic or something that exists only as an idea but not in real life. The original Latin means 'to draw off' or 'draw away', and this is its meaning here. We are simplifying a problem by discovering a general principle or main idea.

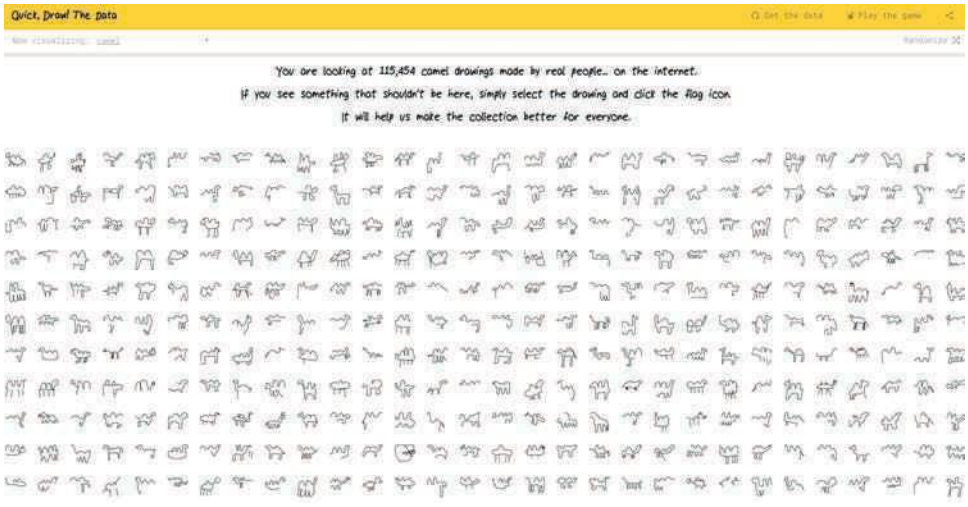
In our earlier example, Utzon discovered the central unifying geometry of a sphere could be used to create all the shapes for the shells of the Sydney Opera House roof. This abstraction came from his recognition of shape patterns, which allowed algorithms to be developed and then eventually the code to solve highly complex engineering problems.

Players of Pictionary use this principle of abstraction – they sketch a drawing that communicates an object to another player as quickly as possible by mentally extracting and then drawing only the essential features of the object, avoiding any details.

Have some fun with abstraction and play Google's Quick, Draw! (See weblink.)

Weblink
Quick, Draw!

9780170442787



Source: https://quickdraw.withgoogle.com/data/camel

Figure 2.14 Abstraction in action! What is the simplest drawing of a camel possible? Google's Quick, Draw! website uses artificial intelligence (AI) and machine learning (ML) to learn to recognise objects drawn by humans. Here we see a few of over 150 000 drawings of a camel that it was able to recognise. The computer's AI algorithms store features that are unique to a camel.

Designing and transferring algorithms

The final stage in computational thinking is designing a series of ordered steps to solve the problem and refining these for automation by a computer.

Once the algorithm is worked out, it is usually straightforward to encode it in a suitable computer language. This solution can then be applied to solve similar problems. This ability to transfer knowledge and apply it to new problems is one of the biggest strengths of computational thinking.

KNOWLEDGE BUILDER

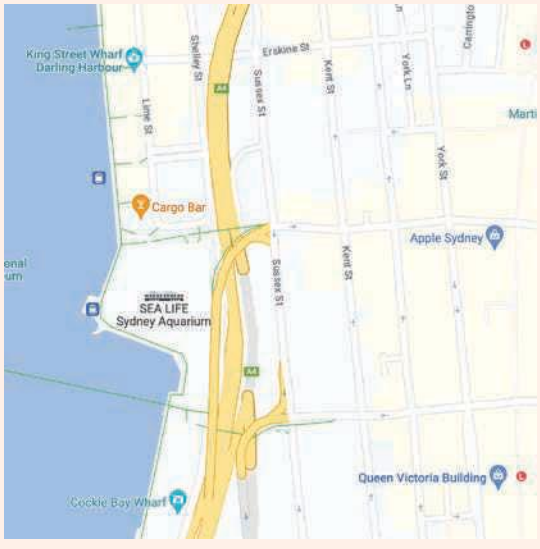
One type of problem that has been solved using computational thinking is automated navigation. Google Maps is an example. Software engineers faced the challenge of directing owners of mobile devices from any point to their chosen destination using turn-by-turn spoken instructions.

Decomposition meant first mapping smaller geographic regions, then using data analysis to determine road characteristics. Abstraction meant streets were represented using numerical data rather than drawn maps and vehicle locations were determined by satellite. Finally, algorithms were written to identify the most efficient route based on traffic and route length and turned into spoken directions.

Activity

View Google's video Exploring computational thinking.

Weblink
Exploring computational thinking



Map data ©2023 Google

Figure 2.15 Abstraction is used to develop turn-by-turn navigational systems. An aerial photograph is used to abstract a map of the road system, which is then further abstracted as numerical data sets.

SKILL BUILDER

Understanding computational thinking: the game of Nim

As well as being fun, this game will help you understand the principles of computational thinking.

The game: Arrange 10 coins in a row between two players. Each player takes a turn removing *exactly* one, two or three coins at a time. The person who takes the last coin loses.

The challenge: can we develop a computer algorithm to intelligently compete against a human player?

1. Play the game five times, taking turns.
2. Suggest how you could apply the concept of decomposition to develop your algorithm.

Decomposition

3. Play games using fewer coins.
   - **a** Set up a game using just five coins and make the first move. If you take one coin (leaving four), your opponent will take three – you will lose.

     For example, say there are five coins and Person A has the first move.

     Person A takes one coin.

     Person B takes three coins.

     Person A loses, because they have been forced to take the last coin.

   - **b** Now play using four coins, then three, then two. Does this reveal anything?

Coin image: Shutterstock.com/hddigital

Pattern recognition

4. What should you take if faced with six, seven or eight coins? Analyse what happens if there are nine coins on your turn.
5. Imagine it is your turn. Copy the table below and record your best move when faced with different numbers of coins. If there is no best move, just enter an 'X'. Complete all remaining rows in this table.

Abstraction

6. Refer to the table you completed in the last stage. Which numbers caused you to lose? You therefore need to force your opponent to face these numbers.
7. Can you discover a pattern or rule for the number of coins you should take each time to win on your move? (Hint: What is common to all the moves marked with X?)

It's your turn and there are this many coins:	You should take:	This leaves your opponent:	Do you win or lose?	Outcome
1	X (1 is your only move)	0	You lose	You are forced to take the last coin
2	1	1	You win	You force your opponent to take the last coin
3	2	1	You win	You force your opponent to take the last coin
4	3	1	You win	You force your opponent to take the last coin
5	X (there are no good moves for you)	X	You lose	You are forced to take the last coin
6	1	5	You win	You force *your opponent* to face five coins on their turn, which you found in the last row means they will lose

9780170442787

8 Play again with 10 coins, but this time arrange them in groups of four. Does this suggest a rule or pattern that can be used by a player no matter what number of coins?

9 Can you express this rule:

- **a** in words?
- **b** mathematically?

Algorithm design and coding

10 Can you write this abstracted rule in the form of structured English or flowchart?

11 (Challenge) Can you express this rule in your chosen computing language?

12 (Further challenge) Can you write a program for a user to play Nim against a computer? Can you add a feature to allow a player to select who plays first – computer or themselves?

In a later chapter you will be able to use your preparation here in developing a working program to play this game against a computer!

Once a problem is solved and expressed as an algorithm, it is possible for the solution to be used again to solve other problems. This is called transference. Here our algorithm for Nim with 10 coins can be applied to games with any number of coins.

Another example is a sorting algorithm. Once developed, this can be coded and used to sort alphabetically, numerically, by date and more.

Skill builder resource
The game of Nim

Weblink
Quick, Draw!

Chapter review quiz

ACTIVITIES

Identify

1 Name the three types of thinking skills that feature in this course.

2 Identify the five stages of design thinking.

3 Identify the five stages of systems thinking.

4 Identify the five stages of computational thinking.

Analyse

5 After playing a few games of Quick, Draw!, select one object and suggest what features are being abstracted.

6 Like Jørn Utzon, designers often use pen and paper during the initial design thinking stages. Why do you think they do this? Evaluate strengths and weaknesses of computer-based alternatives.

Extend

7 Select an example of a computer-based solution to a problem and identify evidence of design, systems and computational thinking.

Weblinks:
- Quick, Draw!
- Exploring computational thinking

Skill builder resource:
- The game of Nim

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

3 An introduction to enterprise information systems

Chapter outcomes

In this chapter, you will learn:

- the purpose, components and categories of information systems
- the functions and use of common hardware used in information systems
- the functions and use of common software used in information systems
- how data is stored and processed in digital systems, transforming it into information
- how structured and unstructured data, and quantitative and qualitative data are processed
- about the different people involved in information systems and the information processes they are involved with
- what should be considered when developing information systems, including cybersecurity principles.

Key terms

central processing unit (CPU)
cybersecurity
data
data analysis
data type
enterprise information system
hardware
information
information processor
information system
microprocessor
network
qualitative data
quantitative data
software
structured data
unstructured data
user

9780170442787

IN ACTION

City2Surf information system

Every year, more than 80 000 people register to participate in the world's largest fun run, the Sydney City2Surf (Figure 3.1). Although this event is a huge feat of physical strength and endurance, the extensive data processing going on behind the scenes is just as vital for the event's success.

When a participant registers and pays for the City2Surf online through the website, an account is set up that stores key information about the person and their entry, including their name, date of birth, gender and the group they are racing in. A bib number is also generated, which is used to uniquely identify every participant. As the City2Surf is also a fundraising event, each participant has a fundraising web page set up to which people can be directed to donate money.

Systems thinking

Finally, it is recommended participants download the City2Surf app to help them get 'race-ready' (Figure 3.2). This app includes much of the functionality on the website as well as suggested training programs.

Before the event, participants receive or collect their race bibs, which include a special piece of technology that allows their race time to be accurately recorded. Embedded in the plastic bib is a radio-frequency identification (RFID) tag. At the start, end and at various checkpoints along the race route, the participants run over timing mats that contain antennas that receive radio signals from the tags. The sensitivity of the technology allows the system to keep track of all 80 000+ participants. These signals are then processed into time stamps, which are instantaneously transmitted to a centralised database and stored against each participant's record. During the event, photographers take photos of the participants. When the photos are uploaded to the City2Surf website, optical character recognition (OCR) software is used to determine the bib number in the photo and link it to an account.

After the event, participants are able to search for their results in the public access database or log into their accounts to see more detail about their results on their dashboard. Low-resolution versions of the action photos are linked to their accounts with the option for purchase of the full-quality images. SMS results are also sent to those who paid for that option at registration.

Getty Images/Brendon Thorne

Figure 3.1 All of the 80 000+ runners and walkers have registered for the event online and will have their results added to online accounts at the end of the race.

Map data ©2019 Google

Figure 3.2 Interactive course map showing the location and distance of key features along the route, including checkpoints, toilets, medical support, water stations and entertainment.

Questions

1. What is RFID? What other information systems use RFID technology?
2. Suggest three reasons why it is better to use an automatic tracking system rather than manually recording times.
3. Why would an app be more useful to a participant than the website on race day?
4. What is another technology the participants could use to keep track of their progress around the course?
5. What is OCR? What other information systems use OCR technology?

3.1 What are information systems?

In the most general sense, a system is a group of connected things that work together to achieve a goal or outcome.

Consider our respiratory system. It is made up of many organs and muscles in our body, including the mouth, nose, trachea, lungs, rib muscles and diaphragm. These all work together to allow us to respire, or breathe, providing oxygen to, and removing waste from, our blood. Our solar system is composed of the sun, planets, moons, asteroids, comets and meteorites, all moving based on the gravitational forces acting on them.

A digital system includes **hardware** and **software** components integrated to perform a multitude of possible functions. Many digital systems today also involve **networks**, such as the internet, allowing the functionality to occur across many locations and sometimes huge physical distances. A digital system allows you to video chat with a friend on the other side of the world or control the temperature of your house from your mobile phone.

When we consider a digital system as including **data**, people and processes, it is referred to as an **information system** (Figure 3.3).

Purpose of an information system

As the name suggests, the purpose of an information system is to produce **information**, commonly to solve a problem or identified need.

In this course, you will be developing a range of information systems, however, there will be an emphasis on **enterprise information systems** in early chapters. You will be applying knowledge and skills on how to use different hardware and software with people and processes in order to transform collected data into useful information.

Components of an information system

Hardware

Hardware is the physical components of an information system. A typical information system will include:

- one or more computers
- external or peripheral input and output devices, such as mice, cameras, printers and speakers
- networking infrastructure, such as a server, router and network cables.

Consider the City2Surf information system outlined at the beginning of the chapter. The hardware components of this system include the RFID tags and scanners, the computers used by the people who run the system to check on the data, the server used to store all the results and to host the website, and all the networking cabling and hardware used to connect these components together. Other hardware components of this system include computers used by participants to access the website and mobile devices used with the app. Cameras also form an important part of this system. Even the laser cutter used to engrave individual nameplates that fit into medals is a component of the system.

Software

Software is the instructions that allow hardware to process data. These instructions are initially written in one of many programming languages then transformed into raw binary instructions for the computer, called machine code. The software you would be most familiar with is the application software that allows computers, generally controlled by **users**, to perform a wide range

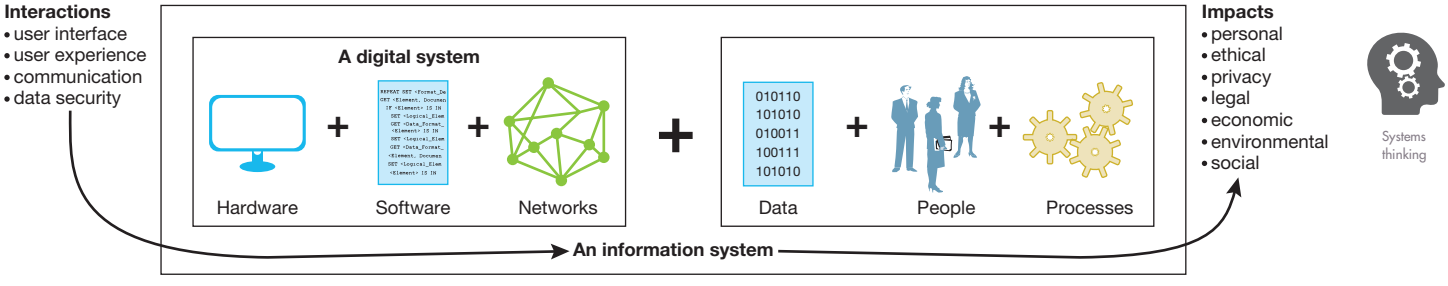


Figure 3.3 Information systems are created when a digital system becomes part of a larger system, along with data, people and processes.

of processing tasks, such as word processing, image editing and animation creation. Other software that goes on behind the scenes includes operating systems and utility software.

In the City2Surf information system described earlier, all computers would be using operating systems and utility software for the back-end control of the hardware. Many software applications are also involved, including the internet browser used to access the website, the database management software used to store and process user accounts and results, and the mobile application (Figure 3.4).

Networks

Networks are formed when two or more computers are connected together to allow the sharing of data. Connections are made using physical media, such as ethernet cables and optical fibre, or wireless technology, such as Bluetooth and Wi-Fi. Many networks rely on the biggest network of all, the internet, while other simpler networks allow more direct transfer of data, such as the use of Bluetooth to upload data from a smartwatch to your phone, or sending a print job to your printer on your home Wi-Fi network. There are lots of specialised networking hardware and software involved in establishing a fast, secure and reliable network for an information system.

In the City2Surf information system, each checkpoint would have included one or more computers that were all connected to a centralised server computer where data was transmitted for processing. The internet is also a key feature in this system for users, allowing them to create accounts, use the app and collect results (Figure 3.4).

Data and information

Data is the raw facts collected, stored and processed by an information system. When data is processed in a way that gives it meaning for humans, it is called information. Data is collected, stored and processed, and the resulting information is displayed by the hardware and software of an information system.

Consider again the City2Surf information system. The raw data collected by this system via the website includes name, date of birth, gender and address. One of the first processes performed on this data is to determine the category of participant defined by their age (calculated from their date of birth) and gender. During the race, the raw data collected from the RFID tags is timestamps at each location checkpoint. The data is processed to determine the final and section times, as well as the runner's pace. This data creates information that is further processed, along with other data, to determine total times for teams and ranks for participants and teams.

People and processes

People include the system's users and those involved in the operation and maintenance of the system. Although advances in artificial intelligence (AI)

Figure 3.4 Software, hardware, networks, data, people and processes are all involved in the City2Surf information system.

are supplanting many tasks previously requiring human involvement, many systems still require the participation of people, with the success of the system commonly relying on their roles and responsibilities.

Processes are the steps and rules followed to allow for an information system to perform successfully. It may be helpful to think: 'Processes are to people what software is to hardware'.

In the City2Surf information system, there are many people interacting with the system at different levels of control. Apart from the people participating in the event, there is a person in control of starting the official clock at the start of each wave, which will be used to determine final times, and the photographers who take photos and upload them to the centralised storage platform. There are also numerous people involved with checking and maintaining the system to ensure any issues or errors are corrected.

Structured vs unstructured data

Data that we would like to process into information may be structured or unstructured, impacting the information systems and processes we use to collect, store and analyse the data.

Structured data

Structured data is generally quantitative data that fits into predefined formats or models. It is commonly stored in databases with records for each entity and event, and fields for the attributes of these entities and events. For example, in the City2Surf, the structured data includes all the attributes for the participant entities including their names, dates of birth and bib numbers as well as the attributes for the events that occur as they run the race such as passing a checkpoint (time attribute). When data is structured, it is easier to store and analyse.

Unstructured data

Unstructured data does not fit into predefined models and can be very complex. Photographs are a common example of unstructured data. In the City2Surf, for example, the photographic data are able to be analysed but require other technology such as OCR. The analysis of unstructured data may require machine learning.

Three Vs of data

Data should be evaluated before processing to ensure the most efficient and accurate methods are used. Considering the three Vs of data is one way to do this. *Variety* refers to how structured, unstructured or semi-structured the data is as described above. The second V is *volume* and refers to how much data we have and from how many different sources it is acquired. *Velocity* is the final V, referring to the speed at which the data is generated.

Evolution of data analysis

Advances in AI have allowed for analysis of unstructured data and the links between previously siloed collections of structured data. Machine learning does not require predefined models, instead it looks for patterns, some not seen before, to bring structure and meaning to the data. In 2020, the City2Surf organisers wanted to analyse all of the past results to determine some key information about participants over the years including demographics and returning participants. Early years' data was in the form of scanned newspapers. AI algorithms were used to extract names and race times from these images, which were then stored with other data in a cloud-based data repository. Other machine learning algorithms were then used to find relationships between all the data and to provide key information such as the City2Surf 'legends', who had participated in all 49 events since it first began in 1971.

INFOBIT

Pokemon Go is a game that relies on a range of modern **data analysis** techniques for structured and unstructured data. In this game, players use their smartphone cameras to capture images of real-world locations, which are then analysed by a classifier to identify and overlay virtual Pokemon on the camera view. The classifier uses data analysis and machine learning techniques to identify features in the captured images including terrain and environments that correspond to specific types of Pokemon. For example, water-type Pokemon are more likely to appear near bodies of water, while grass-type Pokemon are more likely to appear in parks and other natural environments. This involves analysing the colour, shape and texture of the images, as well as identifying landmarks and other features that can be used to pinpoint the location of the virtual Pokemon. The image recognition system used in *Pokemon Go* is a good example of how classifiers can be used in video games to create immersive and interactive experiences.

Categories of current information systems

There are numerous types of information systems, some with very specific purposes and others designed to meet a range of needs. Following are some systems widely used today in organisations.

Management information systems

Management information systems involve the processing of data for organisations. They aim to increase an organisation's value by improving efficiency through coordination, control and automation of data. Analysis and visualisation of data helps support

decision making. Management information systems commonly include a range of other information systems, such as transaction processing systems.

Transaction processing systems

Whenever a purchase is made at a register or online, a transaction processing system is at work. In general, a transaction is whenever an exchange occurs or business is done between two parties. This mostly involves the buying and selling of products, but also relates to the payment of employees and when business occurs that does not necessarily involve money, such as a reservation or a loan. Transaction processing systems collect, process and store data about transactions. If an error occurs at any stage during the processing, such as a buyer having insufficient funds in their account, the transaction fails. Data can be processed in real time (for example, when you make a contactless payment using a debit card) or it can be done in batches, as in the payment of employees.

Decision support systems

Decision support systems provide guidance for individuals and organisations attempting to make a decision about a situation that cannot be clearly defined by known variables. These situations are called unstructured or semi-structured. For example, if we were deciding on what type of dog to get as a family pet, there are some clear variables that can help in the decision process, such as the size of yard and time to spend on grooming. However, there are many other more subjective or little-known variables that could also be useful to consider. A decision support system built using a knowledge base of all the possible variables, their weightings of importance, interactions and known outcomes could be used to help you make your decision of the most appropriate dog type. It is important to remember that decision support systems just *support* the decision process. In the end, it is the user that makes the final decision.

Expert systems

Expert systems were one of the first forms of AI. Their purpose is to replace the role of an expert in a particular field, allowing users to query the system to find answers to questions or to find evidence to support their theories.

An expert system is based on a knowledge base of facts and rules, which are defined using knowledge from an expert, and an inference engine that processes the queries from the user using the knowledge base to form a conclusion.

Expert systems are commonly used in identification and decision making (Figure 3.5). However, they are being increasingly replaced with AI identification systems that use neural networks to learn and improve each use.

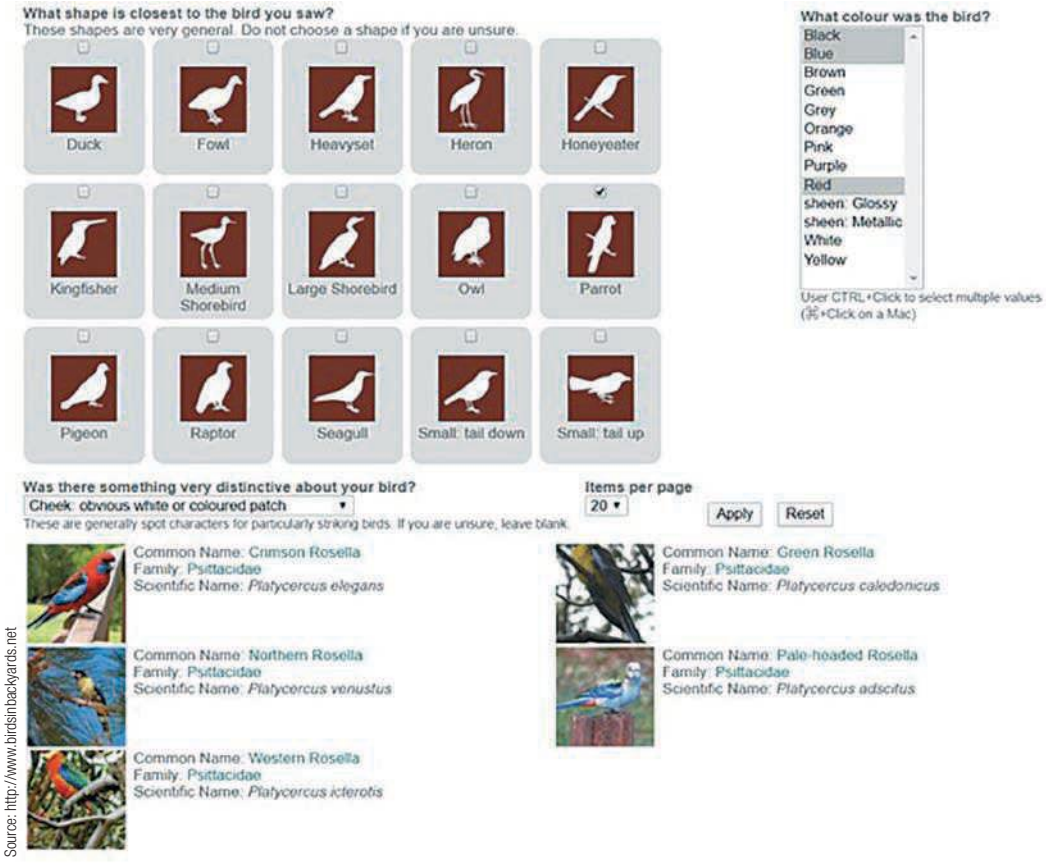


Figure 3.5 The website Birds in Backyards' finder application is an example of an expert system. This expert system allows a user to identify a bird they may have seen in their backyard based on observable features such as size, shape and colours.

Geographic information systems

Geographic information systems (GIS) layer many data sets over one another on a geographic plane. This provides location-based analyses and visualisation, which can aid in communication of data and reveal hidden trends and patterns. Google Maps is a comprehensive GIS that includes data on streets, businesses and satellite imagery as well as real-time data on traffic, speed cameras and location (Figure 3.6).

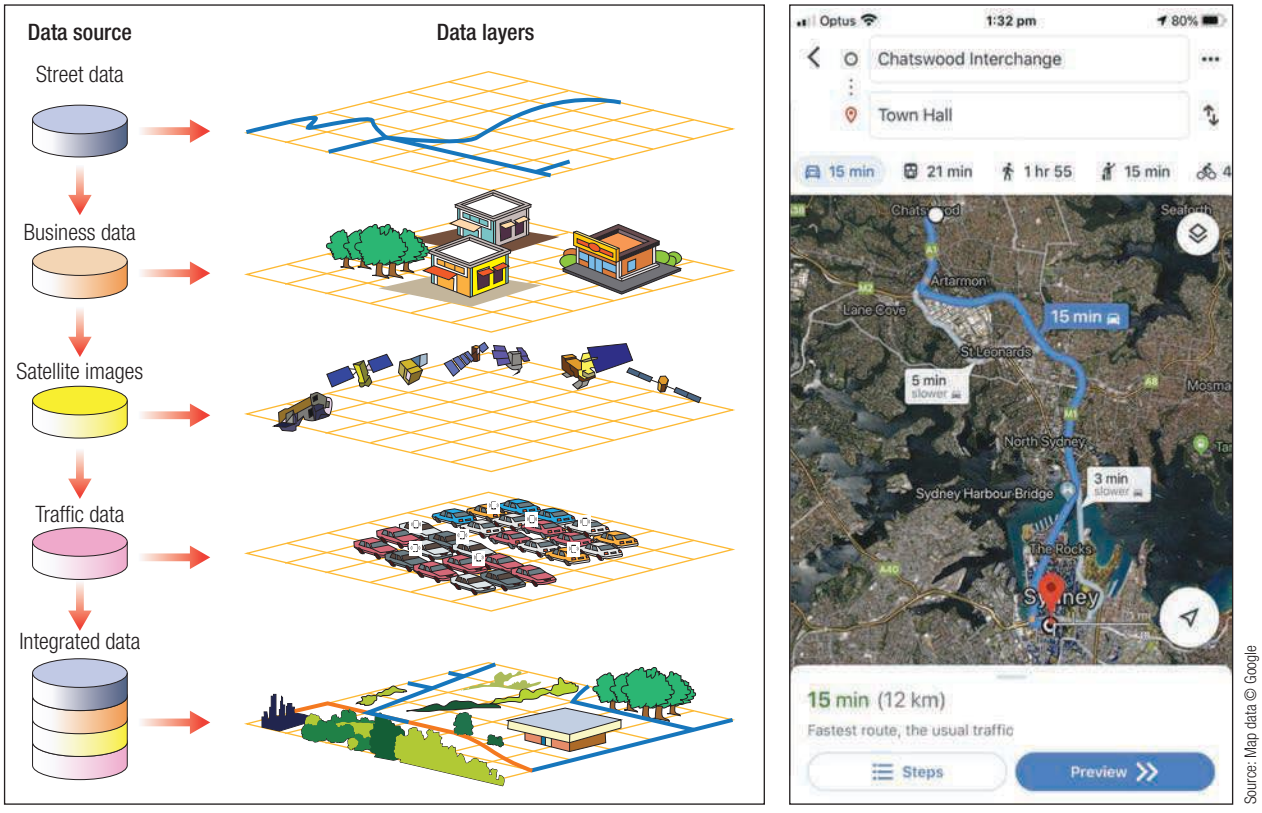


Figure 3.6 Some of the different data sources integrated into Google Maps.

Enterprise information systems

Enterprise information systems are the ultimate management information systems and are used in organisations such as businesses, government, schools, charities and clubs (Figure 3.7).

The functions of an enterprise package could include online payment processing and automated billing systems, security, content management, customer relationship management, business intelligence, project management, collaboration and human resource management.

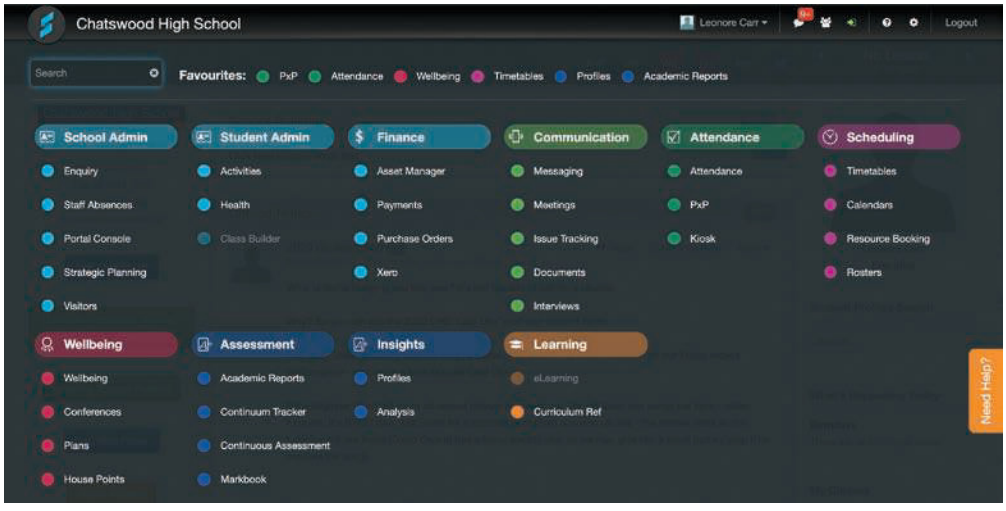


Figure 3.7 Many schools use enterprise systems such as this to allow them to manage attendance, reports, timetables, student payments, wellbeing documentation, staff communication and interview scheduling.

The early chapters will introduce you to technologies and processes similar to those used in enterprise information systems. Modelling the same principles of enterprise information systems within the context of your smaller scale experiences will help in your knowledge and skill development.

Learning management systems

A learning management system (LMS) is designed to manage and deliver educational courses, training programs, or learning and development activities. LMSs provide an online platform where educators can create, manage and deliver learning materials, track learners' progress and performance, and facilitate communication between learners and educators. Some popular examples of LMSs include Canvas and Google Classroom.

Multimedia systems

A multimedia system allows users to interact with and manipulate multimedia content, which can include a combination of text, graphics, audio, video and animations. Multimedia systems are designed to provide an immersive and engaging user experience by presenting information in a variety of formats, and can be used for a wide range of purposes, including entertainment, education, communication and business. Multimedia systems typically consist of a combination of hardware and software components, such as a computer, a monitor or display, input devices such as a keyboard, mouse or touch screen, and multimedia authoring tools or applications. Examples of multimedia systems include video game consoles, streaming media players and interactive kiosks.

INFOBIT

Keeping Culture is an online platform designed to store and safeguard Aboriginal and Torres Strait Islander Peoples cultural knowledge and media. The system organises the media, including photos, documents, movies and sounds, into archive records, and the knowledge into profile records, such as people and places. These are all linked to create a network of knowledge. This system is called a knowledge management system and incorporates elements of geographic information, enterprise information and multimedia systems. To find out more, go to the Keeping Culture website.

Tools for information systems

There are a number of common tools or key features integrated into many current information systems. You will learn about these in more detail in later chapters. In sections 3.2 and 3.3 a range of hardware and software and their functions in different information systems are introduced.

Databases and data warehouses

Organisations collect data from a wide range of sources relating to their operations and management. This data is organised using databases and processed using database management systems. By aggregating these databases and creating data warehouses, organisations can use specialised tools and processes such as data mining to gain useful, possibly hidden, information to guide future management decisions.

Combining separate databases can raise significant ethical issues, as users are generally unaware of the new information that can now be deduced about them.

Spreadsheet software

Quantitative data analysis is commonly built into commercial enterprise software. However, organisations or individuals may see the need to develop their own custom software systems for a particular purpose. Spreadsheet software, such as Microsoft Excel (Figure 3.8), can form the basis of many of these internally developed systems for data analysis, such as stock tracking and budgeting. Spreadsheet software can also be used to develop database management systems when the organisation of data stored is not overly complex.

Shutterstock.com/Rawpixel.com

Figure 3.8 Microsoft Excel is the most widely used spreadsheet software, in industry, education and for personal use. Google Sheets is a strong competitor to Microsoft Excel as it has the advantage of being freely available, a straightforward interface and is web-based. Both offer very similar features, so users can easily transition between them.

Weblink
Keeping Culture

Artificial neural networks

For true automated insight that operates and grows along with the organisation, an artificial neural network can be implemented as part of an information system.

Artificial neural networks are programs that learn. They provide many data analytics and decision-support functions much like other information systems; however, they also involve machine learning to improve their performance with each experience. For example, an artificial neural network may be used in an automated online chat system on a business's website. After each exchange that a chatbot has with a customer, it will learn what terms and phrases are associated with the services the customer is seeking.

ACTIVITIES

Identify

1 Identify the components of a digital system.

2 What is the difference between a digital system and an information system?

3 List examples of data and information in the City2Surf information system.

Analyse

4 Explain the phrase 'Processes are to people what software is to hardware'.

5 Both expert systems and artificial neural networks can help identify and support decisions. What are the key differences between how they do this?

Extend

6 Use an expert system to help you select a suitable dog breed for you and your family (such as the one at the link provided). Explain how you think the expert system came up with the results.

Weblink
Expert system

Computational thinking

3.2 Hardware

A computer can be defined as an **information processor** – we supply it with data to be changed into an appropriate format, which can then be stored or displayed for a specific purpose. Hardware includes a range of physical devices – items that you can see and touch. Figure 3.9 shows a picture of a typical computer hardware system. The hardware components work together to complete the overall task.

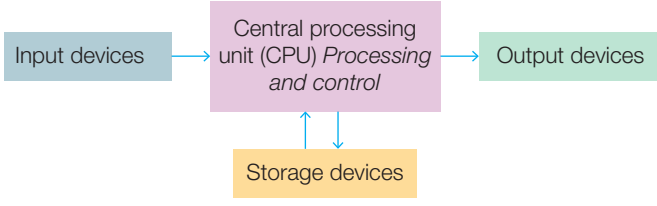


Figure 3.9 Hardware components of a computer system and the data flow between them. Most of the devices connected to the computer are under the control of the CPU. Data and information flow into, around and out of the system.

Hardware functions

The components of a computer system, such as the one in Figure 3.10, are classified according to the type of function they perform – input, output, storage, processing and control. The hardware of a computer system has a part to play in all of these functions.

Input

An input device collects data from the outside world and changes it into a form that the computer can work with. The keyboard, mouse and touchpad are probably the most commonly used input devices. The keyboard will convert the press of a key into an electrical signal to be transmitted either wirelessly or along a cable to the system box. Different signals will indicate which key has been pressed. This data is stored in memory until it can be processed by the **central processing unit (CPU)**.

Microphones, scanners, digital cameras and graphics tablets are also common input devices. Some systems have sensors that collect data such as temperature, light, noise and movement that are sent to a central unit for processing into appropriate information.

Output

An output device communicates information from the computer to the user. Information may be output as a display on a monitor or printer, or it may be output as sounds from a speaker. The monitor converts digital electrical signals from the system into patterns of light on the screen, which can be interpreted by the human brain as text or images. Head-up displays, 3D printers and laser cutters can also function as output devices. Speakers take electrical signals output from a computer and convert them into sound waves, perhaps speech, music or sound effects.

Storage

There are two types of storage in a computer system: primary and secondary.

Primary storage is short-term, *volatile* memory, meaning it is only stored while the computer is running. Two main examples of primary storage are random-access memory (RAM) and cache.

There are three main technologies associated with secondary storage: magnetic, optical and solid state (or flash) memory. Not that long ago it was common for people to use portable hard disk drives (magnetic)

to back up their computers, DVDs or CDs (optical) to share relatively small collections of files. This has now been largely superseded by flash memory alternatives, including flash hard drives and USB sticks, which offer large storage capacity, fast transfer speeds and good portability, all with ever-reducing costs.

You will already understand the need to save your files for future use. Storage devices can be local or remote. Local external drives are being replaced in many cases by online storage using remote servers, known as cloud storage. Networked and online file servers are designed to allow secure storage of data and are generally accessible across the internet. This reduces the risks involved in physically carrying around portable devices, such as accidental loss or damage, and also offers the advantage that the data may be accessed by many people from various locations.

Processing and control

The CPU is widely referred to as the brain of the computer. It can comprise a single integrated circuit (chip) or a number of chips working in parallel. It is responsible for the processing and control of all the data flowing into, out of and around the system.

Computer chips are made up of millions of tiny electronic components called transistors. A transistor is really just a switch. By combining these switches into complex circuits, **microprocessors** are able to perform a variety of complex tasks. Some of these chips (for example, RAM chips) are classed as storage devices, as they store programs and data that need to be instantly accessible to the CPU. Other chips act as processors to carry out routine tasks, such as preparing audio and video data for output.

RAM is volatile, which means it loses all its data when the power is switched off. That is why you need to save your files to secondary storage devices such as your hard disk. Read-only memory (ROM) chips hold information that the CPU needs on a regular basis, such as when the computer is starting up. This data cannot be changed by the user. These ROM chips hold their content even after the power is switched off, otherwise your computer would not be able to start or perform other essential operations.

Graphics processing unit (GPU) chips were designed primarily to manage and boost the performance of video and graphics processing tasks. While the CPU handles most of the general computational workload, the GPU takes care of graphics-intensive tasks, such as rendering videos, playing video games and running graphical applications. In recent years, GPUs have found use beyond graphics processing. Their ability to handle parallel tasks efficiently has made them useful for other data-intensive tasks, such as cryptocurrency mining, scientific computing and artificial intelligence applications.

Figure 3.10 Most parts of the computer system perform a specific task. These tasks can be classified as input, output, storage, processing and control. Can you identify and classify each hardware item illustrated?

Classification of hardware systems

Computers are classified according to how powerful they are, and this generally determines what type of job they do. System classification is used to give people a general idea of what the hardware can do. These classifications are generally a little blurred because technology changes so rapidly. For example, mobile devices such as phones and tablets now have the technical power and functionality that only desktop personal computers had previously. Figure 3.11 shows examples of hardware systems and Figure 3.12 shows a comparison between them in terms of time, cost and processing power.

Microprocessors

Microprocessors are tiny electrical circuits made up of millions of transistor switches, mounted on a silicon chip, that control the flow of electrical signals. Much of what a computer does is dependent on these electronic switches routing data in different ways, according to the software instructions. The CPU inside the system

box of a personal computer is commonly referred to as the processor, or microprocessor, as it is very small and controls the main operations of data processing.

As well as computers, microprocessors are found in many different electronic devices. Their job is to store and process data – that is, to change data into a form that is useful to humans or another device. These devices include cameras, watches and monitoring devices.

Smartphones

Smartphones provide many more functions than just phone services, and their wide use and acceptance are evidence of their extensive functionality. As well as being portable, they also boast many more sensors than are found in personal computers.

Most of their data processing is done on the servers they connect to, although the high speed at which they transmit and receive data tends to obscure this.

Personal computers

A personal computer is any computer that can perform all of its input, processing, output and storage activities by itself. It is small enough to sit on a desktop, although the system box may be positioned on the floor. A workstation is a networked desktop computer that can be used in combination with a server or other computers as part of a network for a wide range of tasks, including complex calculations and graphics processing. A notebook or laptop computer is a light and portable personal computer, yet still powerful enough to meet most needs of the individual worker.

Servers

A server can be any networked computer located anywhere that has appropriate server software installed to meet its purpose.

In larger organisations, servers are generally dedicated machines that control access to software, files and printers. They may be required to support thousands of connected users at one time. These host computers can be used on the internet to act as web servers.

Mainframe computers

Mainframe computers can cost several millions of dollars. They can store vast amounts of data and are accessed with terminals or personal computers. Banks and large government organisations use mainframe computers to process data. Users communicate with these computers using terminals.

Supercomputers

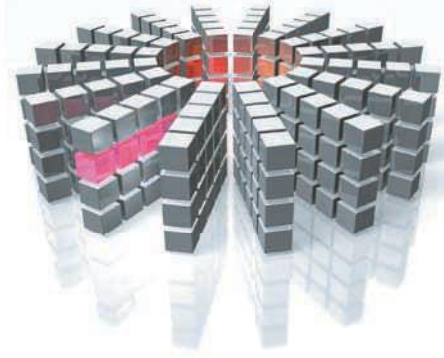
A supercomputer is the fastest and most expensive of all computers. Supercomputers will contain thousands of processors that can share jobs and work in parallel. This means that they are capable of processing many trillions of instructions in one second and can be used for complex mathematical calculations such as weather forecasting or scientific research. A supercomputer might cover the area of two basketball courts. They are used to make weather predictions, making sense of huge amounts of data.

Shutterstock.com/Kaspars Grinvalds

A smartphone is now more like a computer than a phone.

Shutterstock.com/Den Rozhnovsky

A personal computer might be small enough to sit on your lap, yet powerful enough to act as a server on a network.



Shutterstock.com/cyclone project

Supercomputers are used for weather forecasting and scientific research.

Shutterstock.com/dotshock

Network servers are housed in cabinets and are easily removed and replaced.

Mainframe computers are used by banks to process millions of transactions per day.

Shutterstock.com/Timofeev Vladimir

Figure 3.11 Portability and power are two factors that must be considered when designing new hardware systems.

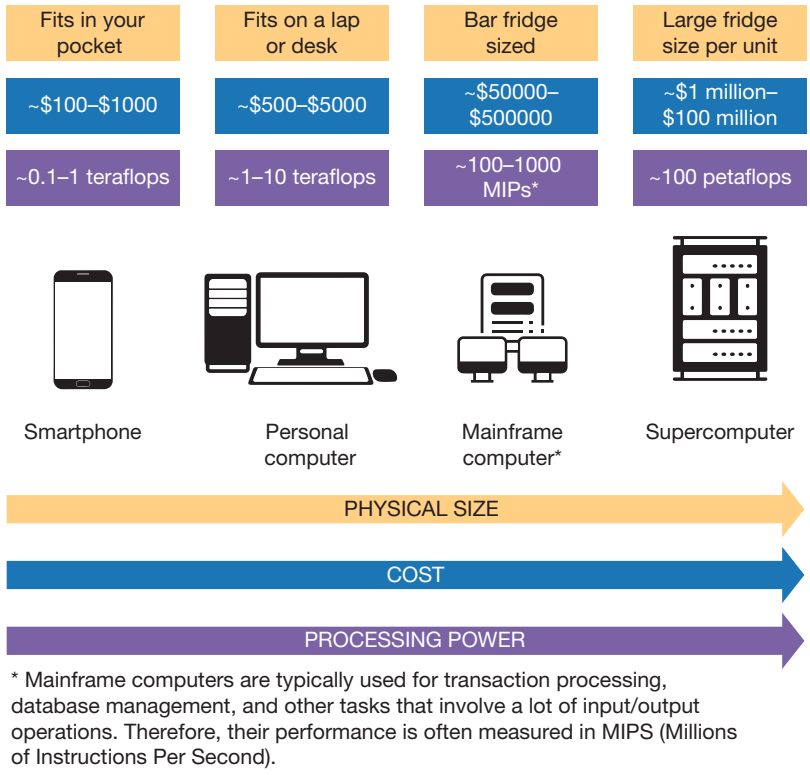


Figure 3.12 Not unexpectedly, as processing power goes up, so does physical size and cost.

Quantum computers

Unlike traditional computers that use bits to represent either 0 or 1, quantum computers harness the unique properties of quantum mechanics and use quantum bits or qubits. These qubits can exist in multiple states simultaneously, allowing quantum computers to perform incredibly complex calculations at speeds that are unimaginable for classical computers. Quantum computers built by IBM are already tackling problems that were previously beyond the reach of conventional technology, including complex simulations, supply chain optimisation and cryptography. AI is also bound to be significantly boosted by quantum computing.

Buying a personal quantum computer is a while off, but already people in industry can lease access to use them remotely.

INFOBIT

One of the earliest supercomputers, the CRAY-2 (1985), weighed ~2500 kg and had a processing power of 1.9 Gigaflops. The iPhone 14 Pro Max (2022) weighs only 240g and can hit 2 teraflops. That's 1000 times more processing in a system more than 10 000 times smaller.

ACTIVITIES

Identify

1 Identify the five functions that elements of a computer system perform.

2 Draw up a table of hardware devices that you know of. Identify each one as a device for input, output or storage.

3 How do you think a touchscreen device should be classified – input or output? Could it be both?

Analyse

4 Explain the difference between a microprocessor and a personal computer.

5 Compare the usefulness of storing files on your computer and storing them in the cloud.

Extend

6 Scanners and printers can now operate in three dimensions (3D). For example, a 3D scanner can capture the finest detail of a locomotive and a 3D printer can output that image on any (reasonable) scale. Research the availability and operation of these devices and present a report to the class. What other objects might these devices be used for?

3.3 Software

Software systems enable hardware to do a wide variety of tasks. The software is a list of instructions that enables documents and data to be processed by the hardware. This software is created in a range of programming languages and can be written using graphical icons or English expressions.

Application software

Today's computers and digital devices have a range of uses, and each of them will need to have a particular type of software. Most people are familiar with application software such as word processors, graphics editors and web browsers that are able to perform a range of tasks in creating, editing, formatting and displaying documents (Figure 3.14). Increasingly, people are creating their own applications (apps) for mobile devices. These small applications can perform a single or limited task and can be created using open-source (free), user-friendly software.

Some devices, such as automatic teller machines (ATMs), routers and traffic-light controllers, perform just one or two, sometimes complex, tasks. Other hardware, such as a microwave oven, has software embedded in it so the devices are fast and reliable.

In general, application software is designed to allow the user to process data – for example, to create and edit a file, or to display and transmit information across a network. Consider the following categories of application software.

Communication software

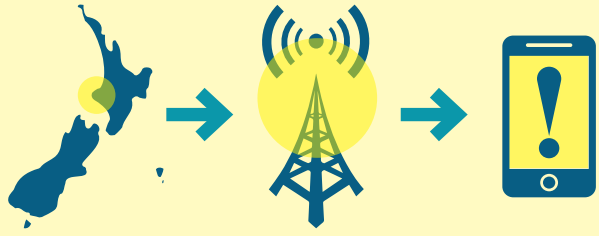
Due to the ever-increasing accessibility of fast internet, people can easily communicate using a range of systems.

- Internet browser software is used to download a wide variety of internet and intranet documents. Browsers also enable users to interact with web applications such as online banking and shopping.
- Instant messaging software enables real-time communication between two or more people.
- File transfer programs allow users to upload files over the internet – for example, to web servers and wikis.
- Increasing numbers of companies are moving to internet protocol (IP) telephony or voice over internet protocol (VoIP), which enables workers to make lower-cost phone calls using their computer. Skype™ is a popular example of this.
- Videoconferencing software enables video and audio communication between users over the internet. These programs can provide other tools, such as chat and multi-user document editing.

INFOBIT

New Zealand is a geologically active country with regularly occurring earthquakes and risks of other natural disasters such as volcanic eruptions and tsunamis. New Zealand uses an Emergency Mobile Alert system that broadcasts warning messages to people's mobile phones within range of targeted mobile towers when there is serious threat to safety (Figure 3.13).

How does Emergency Mobile Alert work?



1. Authorised agency determines affected area and creates alert.
2. Alert is broadcast from relevant cell towers.
3. Capable mobile phones in the affected area receive the alert.

Source: National Emergency Management Agency (NZ)

Figure 3.13 A diagram from a fact sheet detailing the new mobile communication system that can alert people of an emergency in New Zealand

Business software

Business organisations depend on a variety of software tools to manage their workload effectively. Word processing and email software are essential elements in today's modern business. In addition, intranets or knowledge management systems enable information to be posted for reference by all or specific employees.

Project management software is useful for scheduling tasks and resources in large team projects. Resource planning software is used to manage resources such as staff and finances. This helps to keep all branches of a business fully informed of essential business information, especially in large companies that may be developing rapidly or located in different parts of the world.

Businesses use computer models to help them make decisions – for example, by identifying market trends. These can range from a simple spreadsheet to a powerful analytical tool custom-built by software specialists.

Software for schools

All students in New South Wales are required to meet specific standards when using computing technology. Consequently, most schools will have a list of standard software installed on their computers for students to use. Students who choose computing technology as an elective will learn to use more sophisticated software and probably in a more innovative manner than their peers. In addition, school administrators use software to create and publish reports, and to store and submit data to educational authorities. Most of this work is database-driven.

Science and research

Most areas of science and research have their own specially developed software to analyse large volumes of information and help them in their quest for new knowledge. Expert systems and AI are used and have potential for further development. Current applications include fingerprint matching by forensics teams and diagnosis of cancer cell malignancies.

Applications for mobile devices

Software applications on mobile devices are commonly referred to as 'apps'. People develop games and other apps for smartphones and tablet computers. User-friendly software is available to help with this, so that you do not have to be a specialist **programmer** to develop your own apps.

Operating systems

On a personal computer, an operating system is needed to work with application software to control the computer's hardware. Microsoft® Windows® and macOS are the most widely used operating systems around the world.

The operating system is the first thing we see on the screen when we start up a computer. It provides the interface that asks us to log on or it can display the desktop and its icons immediately. Similarly, when we close down the computer, it tells us that it is logging off and closes all the files in a systematic order before it actually powers down the hardware.

System requirements

When we want to install new software on our computers, we will probably be notified of the system requirements – that is, the minimum requirements in terms of hardware and operating system versions for the software to work properly (Table 3.1). Currently, manufacturers are producing 64-bit computers, which means the CPU, operating system and the application all have to be capable of processing 8 bytes of data in one instruction or data item.

The CPU

If a software package defines a 3 gigahertz (GHz) CPU and 4 gigabytes (GB) of RAM as its minimum hardware requirements, the package may fail to run properly on anything less than this, or it may run so slowly that you would be disappointed with the performance.

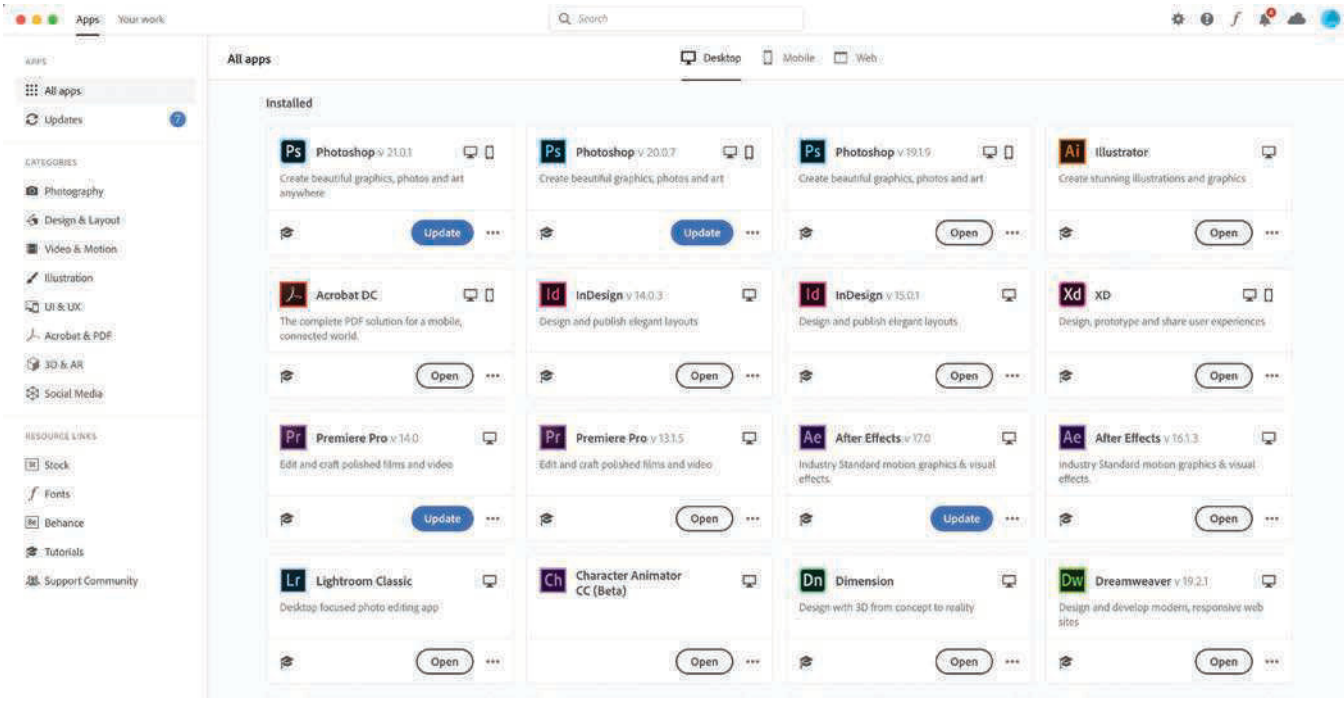


Figure 3.14 Adobe® Creative Cloud Suite® is a market-leading collection of applications available on desktop, mobile and web-based platforms.

Some programs use the CPU much more than other programs. Graphics programs that manipulate photographs and computer-aided design (CAD) systems generally use the CPU far more than word processing programs, because of the very large number and complexity of the calculations required to manipulate graphics. Specialised graphics cards containing a GPU are required for efficient operation of such applications. Businesses that need to work with the latest versions of software will need to regularly upgrade their hardware, as well as the software.

Demands on memory

RAM in the computer is used up by:

- the operating system of the computer,
- any programs currently running, and
- the data those programs use.

If a program uses a lot of memory to run, and it works with large files in memory, a computer system needs to have more than the minimum memory required by the operating system or a single program.

When programs cannot find enough memory, the operating system may use part of the hard disk to store some of the data in 'swap files', and this will slow down processing. This is referred to as virtual memory, where the computer looks at RAM for areas that have not been used recently and copies them onto the hard disk. This frees up space in RAM to work with the currently used applications.

Table 3.1 It is very important to check the system requirements for software before installing them. If the computer cannot meet the requirements, the application may not work as expected or at all.

Minimum system requirements for Adobe After Effects (Windows)	
Processor	Intel or AMD quad-core processor
Operating system	Microsoft Windows 10 (64 bit) versions V20H2 or later
RAM	16 GB minimum (32 GB recommended)
GPU	2 GB of GPU VRAM
Hard disk space	15 GB of available hard disk space; additional free space required during installation (cannot install on removable flash storage drives) Additional disk space for disk cache (10 GB recommended)
Monitor resolution	1920 × 1080 or greater display resolution
Internet	Internet connection and registration are necessary for required software activation, validation of subscriptions, and access to online services

Source: Adobe, 'After Effects (2023) system requirements'

Monitors

Size, resolution and speed of your monitor can impact significantly on the effectiveness of your application. Monitor size should be decided on before you buy your computer system. Resolution and colour depth can be set in the properties window of your display. Computers used for gaming and other processes with a highly dynamic display need monitors with a high refresh rate, such as 144 Hz, which means the image refreshes 144 times per second.

Storage

Secondary storage is essential as a non-volatile means of saving your documents. It also stores the software applications and the operating system, and loads a copy of them into RAM as required. However, as software applications become more powerful and sophisticated, they will become larger and so require more space on the hard drive. Similarly, so will the documents they create. Many people use online storage such as that provided by Dropbox or Google Drive instead of purchasing their own external drives, although online storage may involve an annual fee.

ACTIVITIES

Identify

1 List application software you use frequently.

2 Identify each category of application software and provide an example of each.

3 When installing new software on a computer, what system requirements need to be considered?

Analyse

4 A company opts to use videoconferencing software for a meeting between employees in offices at different locations. What are the advantages and disadvantages of using videoconferencing instead of face-to-face meetings?

5 Open a range of application software and compare the system requirements. Summarise your findings in a table.

Extend

6 Interview an adult friend or family member about software they use in their place of work. Outline the processes involved in some key functionality of the software.

3.4 Data and information

Data and information are all around us. As human beings, we collect, process and store information all the time. The functioning of our daily lives relies on our access to information, but our brains can only cope with so much. For this reason, most people and companies rely on digital technology to do the collecting, processing, storing and transferring for them. The difference between us and a computer is that we can make sense of what we hear and see even when it is presented in many different ways. Computers require data in digital form and they need to be told exactly what to do and how to do it.

In its simplest form, **digital data** is either a zero or a one – a '**bit**' of data. Computers store and perform calculations on this data in very large amounts, very quickly. This gives us access to huge volumes of information that we can process and present again in a variety of forms for a variety of purposes.

In this section, we consider how data is entered and stored by a computer system, and the different forms it has to take to allow the user to perform a particular task. We will look at where and how data is used to produce information and how this in turn can form data for further analysis.

Data versus information

Many people confuse the terms 'data' and 'information'. Imagine a list of numbers written on a piece of paper. This might be called data because it has no real meaning for anyone. However, if someone told you that the numbers were telephone numbers, then they would have some meaning. The numbers have now become information.

Think about a sentence typed into a computer from a keyboard. The individual letters of each word are entered by pressing the keys. The letters are items of data for the computer to process. A word processor is used to store the letters and to display them on the screen. The output will make sense to a person, but for the computer, the display is just an arrangement of coloured dots on the screen (Figure 3.15).

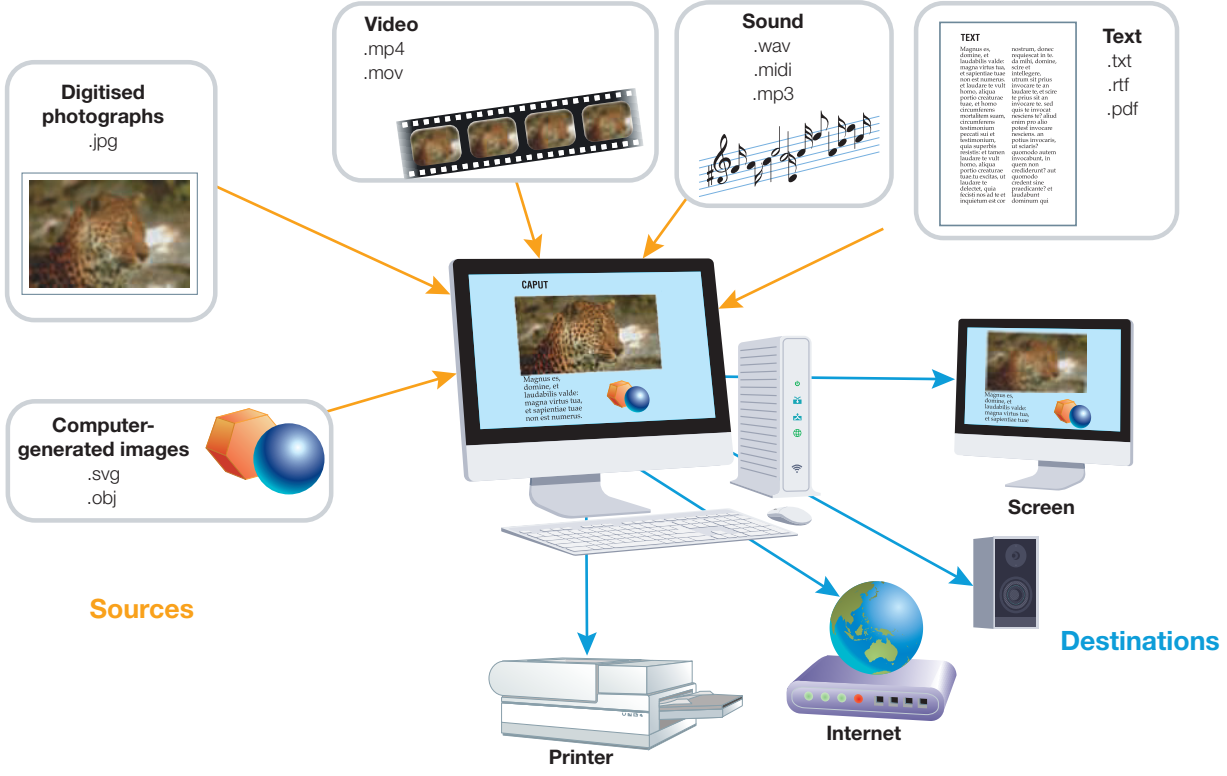


Figure 3.15 Information becomes data when we enter it into the computer for processing. It can come from a variety of sources and in many different forms, all represented using the binary system. This data becomes information again when it is displayed and interpreted by people.

As you work at the keyboard, the information in your head becomes data in the computer system. As you recognise the dots on the screen as words, so the data changes back into information.

Data can be letters or numbers, a pattern of dots, or a series of electrical or light signals. Data becomes information when it is given meaning by people.

Data forms

Most of the information around us comes in variable quantities – for example, the volume of music playing or the speed of a passing car. If the values of these quantities can vary along a smooth line or curve, we refer to it as **analog data**. Some information, however, comes only in fixed quantities – counted values, for example. This is called **digital data**. The values on a digital watch do not represent time passing in a smooth and continuous way, they only display the precise number of hours and minutes that have passed since midnight.

Digital data

Modern computers are designed to work with digital data. This means that all the data that enters the system has to be digital or has to be digitised before the computer can store or process it. Most input devices, such as a keyboard or scanner, are designed to provide the system with digital data. They input data to the system in digital form. However, a microphone inputs an analog electrical signal that is digitised by the sound card inside the computer.

Data coding

While a digital signal is allowed to take several distinct predetermined values, most computer systems work with switches or signals that can either be 'on' (1) or 'off' (0). Since the computer works with just these two values, we call them binary (meaning two) digits or bits. This arrangement allows computers to work very quickly and accurately.

We may find it difficult to work with binary numbers, but with a little practice it can be fun. We can use a series of binary digits, or bits, to represent bigger numbers and many patterns of data. The computer's work is performed using streams of zeros and ones (Figure 3.16).

So what does the series of bits 01100110 mean to a computer? It depends on how these bits are being used. It is very important to understand this clearly. If it is a signal coming from a keyboard, it might represent a character. If it is going to a monitor, it might be the colour of one of the pixels, or it might be part of a program instruction in the CPU. Remember that each bit represents a unit of data, so a series of eight bits, or a byte, can represent 256 or 28 patterns of data, including zero. That is enough for 256 colours on the screen or characters on the keyboard.

Computational thinking

Counting in decimal

In the decimal system, each column value is 10 times more than the one on its right. We count up to 9, then we have to put a 1 in the tens column, and so on.

Hundreds	Tens	Ones
		0
		1
		2
		3
		4
		5
		6
		7
		8
		and so on.

Counting in binary

In binary counting, each column value is two times more than the one on its right. We count 0, then 1, then we have to put a 1 in the twos column, and so on.

Eights	Fours	Twos	Ones
			0
			1
		1	0
		1	1
	1	0	0
	1	0	1
	1	1	0
	1	1	1
1	0	0	0
			and so on.

To convert the binary number 1101 to decimal, we just read off the column values where there's a 1, and add the decimal values together.

	Eights	Fours	Twos	Ones	
Binary	1	1	0	1	
Decimal	8	+ 4	+ 0	+ 1	= 13

To convert back to binary from decimal, just share out the biggest one. For example, 43 in decimal is one lot of 32 plus one lot of 8 plus one lot of 2 plus 1.

32s	16s	8s	4s	2s	1s
1	0	1	0	1	1

Figure 3.16 Counting in the decimal and binary systems.

INFOBIT

Analog computers were practically completely superseded by the more precise and repeatable processing of digital computers. In recent years, with advances in AI and quantum computing, analog computers may be making a comeback.

Data types

It is amazing to think that the almost perfect recording of your favourite band or the beautiful rendering within the 3D world of a game you are playing is defined simply by zeros and ones. But how does the computer know how to distinguish one set of binary data as a 3D animation and others as audio, graphics, text and so on? A file format defines how these zeros and ones are organised and coded so they can be interpreted for the intended display. For example, the sequence of bits 01001101 could be interpreted as: the letter 'M', the decimal number 77, the red value in the RGB code for a pixel of colour 'cornflower blue', the amplitude of one of the millions of the audio samples in a song, and so on. Figure 4.28 in Chapter 4 explains this example in more detail.

Most of the common file formats fall under one of the following **data type** categories: text and hypertext, graphics, audio, animation and video.

Text and hypertext

Text is data that consists of strings of printable characters separated by white space and other non-printing characters. Hypertext is simply text that acts as a link to another document or another place in the same document.

Graphics

A graphic is a picture or diagram used to convey information easily to the user. There are many forms of graphics that can be used by the computer, and they can be incorporated into many types of applications. Graphics files fall into two main types: bitmaps or vectors. Bitmaps define the colour of each pixel on the screen, whereas vector graphics define an image using mathematical objects such as lines, squares and circles.

Audio

An audio file is digitised sound, which can be speech, music or sound effects. The amplitude and frequency of the sound are represented by a series of ones and zeros (Figure 3.17). As with graphics files, audio files can become very large, but there are compression techniques that can be used to reduce the size.

Good-quality speakers or earphones are needed to listen to long audio files.

Animation

Animation is a series of graphic images that are shown rapidly in succession so that we see it as smooth motion. These graphics are mostly computer-generated or assembled and displayed using a software package. Morphing and warping images can be a lot of fun. Morphing software will change one image into another, whereas warping will create a distortion of an image.

Video

Video is now a common form of digital data. A video file, like animation, is also a series of graphic images. The images are commonly of the real world, such as people or events. Video files can have an audio element included in the same file. Video cameras and internet downloads are easily accessible. Video file sizes can be a problem, but streaming (continual transmission of audio and video files while being played) can be used to speed up download and playback.

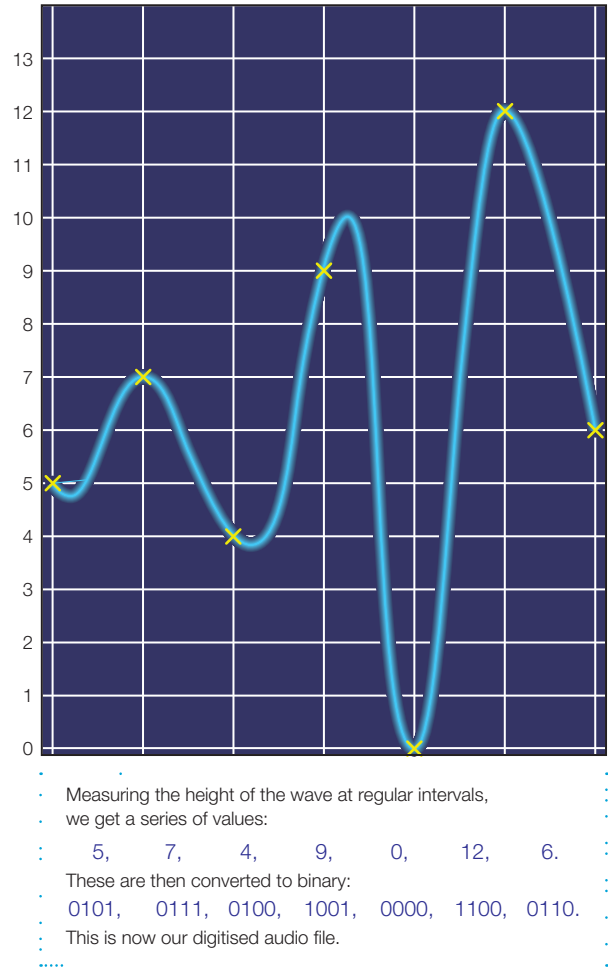
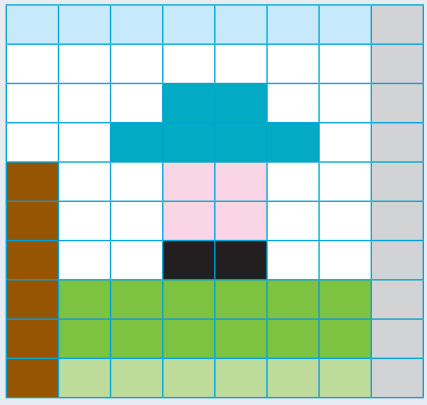


Figure 3.17 Converting analog sound to digital.

CASE STUDY

Understanding bitmap images

The amount of colour information in a picture is referred to as the colour depth or the number of bits per pixel. The greater the colour depth, the more colours that can be represented. The example shown here is using four bits to represent each colour, so it has a colour depth of four bits per pixel. Pixel refers to a picture element or one dot in the image.



Colour codes	
	1001
	0010
	1111
	1100
	0011
	1011
	0000
	1110
	0110

For a given picture size, more colours means larger file size:

- four bits per pixel (16 colours) appears as cartoon art
- eight bits per pixel (256 colours) is used in clip art and low-resolution pictures
- 24 bits per pixel (millions of colours) gives continuous tone such as you see in a photograph.

Questions

1. How many pixels are there in this graphic?
2. If it takes four bits to record the colour of a pixel, how many bits are needed to record the whole picture? What would be the file size for this graphic?
3. Write out the string of colour codes for each pixel across each of the 10 rows.
4. How many bits would be needed for colour depth of 24 bits per pixel?
5. What would be the file size if the resolution were increased to 100 pixels square at four bits per pixel? And at 24 bits per pixel?

Computational thinking

Quantitative versus qualitative data

Data can be classified as either quantitative or qualitative.

Quantitative data is data that represents a 'quantity' or number of some form. Calculations can be easily performed on this data, as it is already in a numbered value to which a mathematical function can be applied. If collecting numbered ratings out of 10 for a new movie, a computer can easily calculate the average rating for the movie using a simple function.

Qualitative data does not have a distinct value or categorisation – for example, viewers' comments on a new film. This data may be equally, if not more, useful than quantitative data in understanding the quality of the film. However, a simple mathematical function cannot be applied to all of the comments in order to determine the overall impression of the film. Instead, we have relied on a human analysing all of the comments to get an overall understanding of the comments, including most common issues and praises.

Two significant areas in computer technology development are AI and natural language processing. These technologies can enable computers to understand natural language and therefore categorise and potentially quantify what is being said to allow for more efficient processing and analysis. This can lead to insights even a human may not capture.

When designing systems that collect data, it is important to consider whether quantitative or qualitative data will be the most beneficial, based on the information you aim to generate.

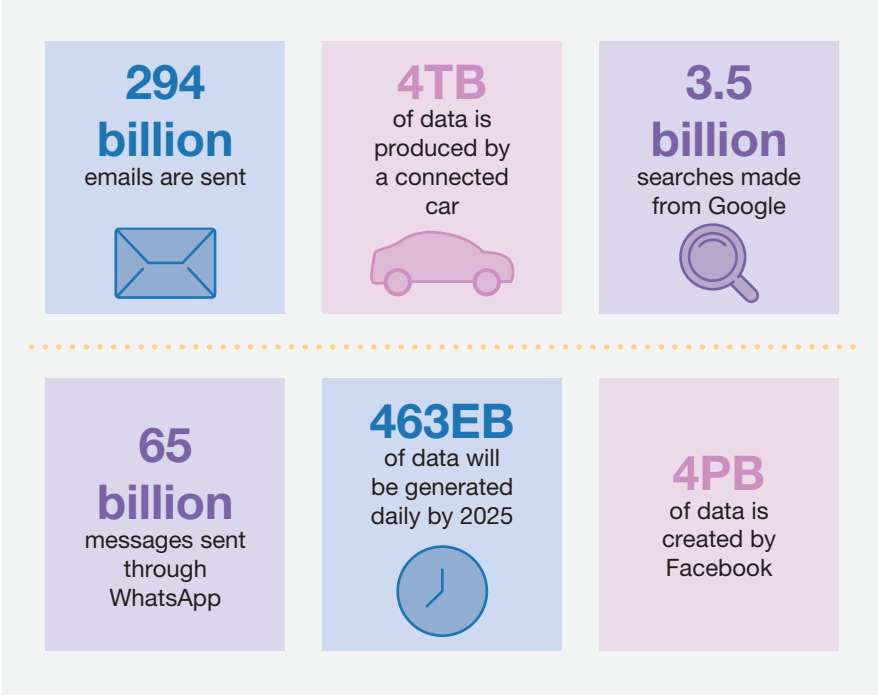
Common uses of data

We live in a world powered by data (Figure 3.18). In 2023, there were an estimated 120 zettabytes (120 trillion gigabytes) of data generated. This is more than a 250 per cent increase from the 33 zettabytes of data generated only five years before in 2018.

Data could simply be facts shared between people, such as event dates, instructions on how to make a cake or photos of your new cousin. It could also be specific numbers used to control automated systems, such as the sequencing and timing of traffic lights or the movement within a laser cutter. However, most data is unstructured, has a more complex role and is processed and analysed to help make decisions and predictions.

Artificially intelligent systems such as OCR rely on many examples in order to 'learn' what a letter or number looks like. All these create data that allows the AI system to make a decision on the current character to be identified.

Predictions can be made about a real-world system using a model of that system. A model can be a function or equation that shows the relationship between two or more variables – for example, the average temperature over the months of the year. Modelling of systems relies on the analysis of large quantities of real data collected from the system. The greater the data used to build the model, the more accurate the model and the more confidence we can have in the predictions made based on the model.



Icons: Shutterstock.com/matsabe

Figure 3.18 Infographic showing interesting facts about daily data usage.

INFOBIT

YouTube: another petabyte every minute

Every day, YouTube needs a petabyte (PB) of new hard disk space to store the approximately 500 hours of video uploaded every minute. But what is a petabyte? It depends whether you are thinking in the decimal or binary system. The prefixes kilo, mega, peta and so on are terms from the decimal system, hence they define values based on the power of 10. Therefore, the decimal value $10^3 = 1000$ bytes or 1 kilobyte (KB). However, since computers work with just 0 and 1, the International Electrotechnical Commission (IEC) defines these values in the binary system with powers of 2. The binary value of $2^{10} = 1024$ bytes is called a kibibyte (KiB) to avoid confusion. So, the decimal term 'petabyte' is $10^{15} = 1\,000\,000\,000\,000\,000$ bytes, while a pebibyte (PiB) is $2^{50} = 1\,125\,899\,906\,842\,624$ bytes. When comparing KB and KiB, the difference is only 2.4 per cent, but when we compare PB and PiB, the difference is 12.6 per cent. So, as data storage requirements continue to increase, it's important to know the difference to make sure we are getting value for money!

Computational thinking

Table 3.2 Decimal and binary values.

Prefix	Symbol	Power of 10 (SI unit)	Equivalent power of 2 (IEC Unit)	Percentage difference IEC/SI
Kilo-	k or K	$10^3 = 1000$ bytes (1 KB)	$2^{10} = 1024$ bytes (1 KiB)	2.4%
Mega-	M	$10^6 = 1\,000\,000$ bytes (1 MB)	$2^{20} = 1\,048\,576$ bytes (1 MiB)	4.9%
Giga-	G	10^9	2^{30}	7.4%
Tera-	T	10^{12}	2^{40}	10%
Peta-	P	10^{15}	2^{50}	12.6%

ACTIVITIES

Identify

1 What is the difference between data and information? Provide an example to support your explanation.

2 List the different data types of media commonly stored on our computers.

3 Draw a diagram to demonstrate the difference between analog and digital data.

4 Write your age and height in centimetres as a binary number.

Analyse

5 Think of a project you recently completed. If you were to design a survey to get feedback on your project from your peers, what quantitative and qualitative data would you collect? Why is it important to collect both quantitative and qualitative data when getting feedback like this?

Extend

6 Find out more about the comeback of analog computers. What key technologies are they using? What industries are they being used in? Will they replace digital computers in the future? Create a brief presentation for your class.

3.5 People and processes

There are two general categories of people involved in information systems: the users and the technical experts involved in the development, operation and management of the system. For the technical experts, the varied and complex nature of information systems means there are countless roles and responsibilities. Users are involved in the development, operation and management of the system. A career in information and software technology is obviously not for everyone, but for those interested in working in this area, there are many possibilities from which to choose (Figure 3.19). People with skills and knowledge working with information technology (IT) are highly sought after in many industries, including health care, education, defence, agriculture, engineering and entertainment.

Users

High-quality, successful information systems have been designed and developed with the users' needs always in mind. A system needs to have logical processes and an intuitive interface to make it user-friendly. The processes a user applies to the system should also be clearly outlined in documentation, such as a user manual.

Consider an information system you use regularly, such as an LMS like Google Classroom. Common processes you perform as a user include logging in, navigating, adding text, searching, displaying resources, uploading files, and so on. How easy is it to use the system? Are there any other processes you wish you could do in the system?

Developers

Computer scientists

Computer scientists design and develop hardware and software technologies. Specific tasks may include providing new approaches to security or devising improved methods for using computers in areas such as robotics, computer vision, medical research or digital forensics. Computer scientists generally require in-depth knowledge of the theoretical foundations of computer systems and the ability to construct and comprehend complex algorithms. Most professionals have tertiary-level computer engineering, software engineering or computer science qualifications.

Computer programmers

A programmer writes the code used to create, update and maintain software packages. A programmer also assists systems analysts and project managers to determine the type of software a user or client needs. The programmer translates the solution into a programming language and must also test and debug the code to ensure it is error-free. These professionals mostly have studied tertiary-level computer science.

Software engineers

Software engineers are involved in the design and development of software for operating systems and application programs. They use different programming languages, depending on the purpose of the program, and solve technical problems in existing software. A software engineer will have likely studied tertiary-level software engineering courses.

Systems analysts

Systems analysts play a critical role in the development and upgrading of information systems. The systems analyst's task is primarily to solve problems related to computer technology by designing and managing the development of information systems. Most analysts specialise in specific types of systems, such as accounting systems, financial systems or engineering systems. A systems analyst may have studied tertiary-level business IT courses in information systems.

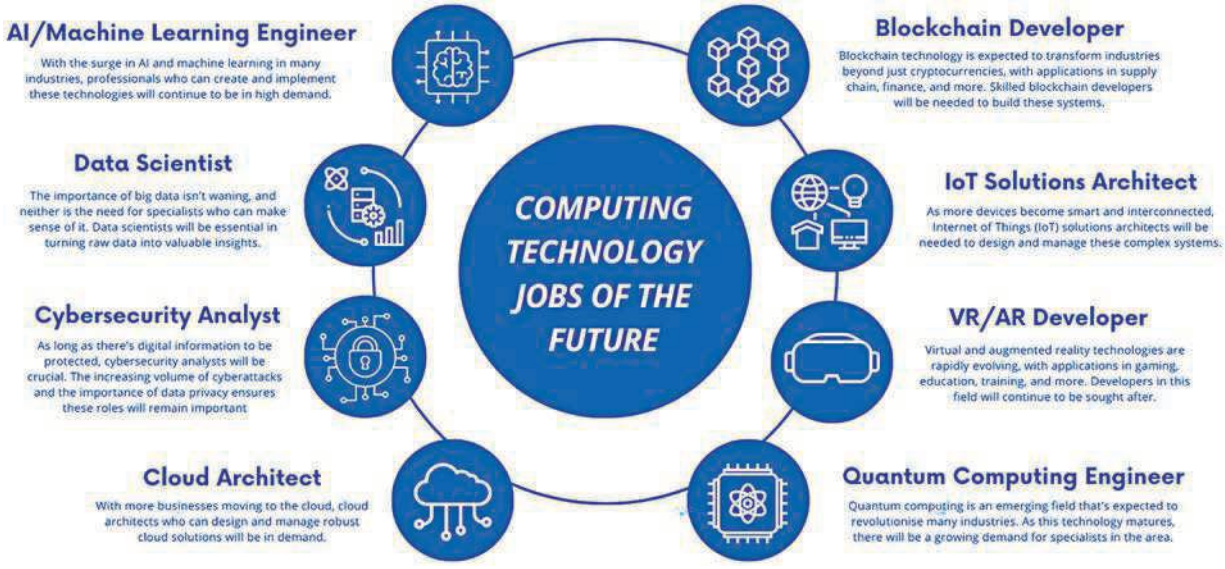


Figure 3.19 Computing technology future jobs.

Operators and administrators

Computer operator

The job of a computer operator is to keep powerful, essential computers, such as servers, operational. Responsibilities include keeping the software up to date, ensuring there is enough available storage and memory, checking the physical environment allows for healthy functioning of the computer, and identifying problems with the computers. The role of a computer operator also includes working with any specialised system software running on the computers.

Database administrator

Database administrators create and maintain databases used within an organisation. This may be on a small scale using database management systems or it may involve working with data warehouses. Databases are used across many processes within an organisation, so the administrator may be required to work with people such as systems analysts and programmers to help integrate database functions into new projects.

IT support staff

Support staff provide assistance for people who are having problems with their computer system. Most large organisations have support staff or help desks. A help desk is a department within an organisation that responds to users' technical questions. Support staff should be able to recognise and interpret the meaning of common error codes. They also need to be able to fix general problems associated with the daily use of hardware. More specialised hardware tasks, such as the installation, repair and maintenance of computers, servers and other devices, are mostly the responsibility of technicians.

Training specialists

The rapid spread of computers has placed more and more demands on people in terms of the technology skills they need to work, live and learn. Training specialists are educators who plan, coordinate and teach a range of activities related to information and software technology.

Management

Chief information officer

A chief information officer (CIO) is an increasingly common senior management role in many organisations, reflecting our increasing reliability on information systems. The CIO's role is to lead the planning and operation of information systems' function so that it aligns with the goals of the organisation. Key jobs include budgeting, scheduling and defining requirements for the information systems. Although knowledge and experience in an IT field is beneficial, it is more important for the CIO to understand the business and have good management skills, including effective communication.

Project managers

The principles of good project management apply across every industry and can be applied to many tasks, from making a dress to constructing an aeroplane. The field of information and software technology includes projects such as the creation of websites, video games, hardware devices, computer networks and operating systems. A **project manager** is required to keep the process on schedule and within budget by providing clear communication about goals, responsibilities, performance and expectations.

Information security officer

With nearly all organisations and their data connected to the internet, it is essential to have someone responsible for defining and implementing effective information security policies and protocols. The information security officer's role is to prevent external threats, such as viruses and hacking, and internal threats, such as the intentional or unintentional leaking of sensitive information. **Cybersecurity** is a rapidly growing field in the IT industry.

ACTIVITIES

Identify

1 Why are users important people to consider in an information system?

2 What processes are usually performed by operators and administrators?

Analyse

3 Compare the job requirements for a junior software developer and a senior project manager listed in Table 3.3. Suggest reasons for the differences in requirements.

Extend

4 Explore a new software application. Record and evaluate your experience as a user, including some key processes you are guided through. What made the application easy to use? Were there any features you found confusing or difficult?

Table 3.3 Job requirements for a junior software developer and a senior project manager.

Junior software developer	Senior project manager
Some experience in C# development	Tertiary qualifications and certifications in project management
Understanding of Agile development methodologies	Ability to manage complex issues and project deliverables
Experience in C++ development and WPF are a bonus	Expertise in best practice in project management methodologies
Friendly and easy to work with	Site mobilisation with systems and applications experience is a must
A willingness to learn	Enterprise ERP experience is highly desirable
	Excellent communication and stakeholder management skills
	Ability to manage complex issues and project deliverables
	Must be able to work in a fast-paced environment

9780170442787

3.6 Considerations and cybersecurity principles

One way to represent an information system is to show all the elements and the impacting environment using an 'information system in context' diagram. This diagram includes all of the elements covered so far in the chapter, wrapping it all up in the context of the environment in which the system will be implemented. No information system can be successfully designed and implemented unless everything that could impact the system and everything the system could impact has been considered and acted upon. Figure 3.20 is a template for creating a diagram for any system you are documenting.

Sociological considerations

When considering the social issues relating to the implementation of the system, it is important to consider both the positive and negative impacts it could have on individuals and the community.

For example, developers of the Opal travel card system would have considered the benefit to users of saving time and money by having an account and card that included multi-trip fare reductions and automatic top-up of credit. They would have also needed to consider the issue of people feeling their privacy was at risk and not wanting to have personal data being tracked and stored, so provided the option of still using a card without being linked to an account.

Information systems must also be designed with consideration of the perspectives of diverse groups including Aboriginal and Torres Strait Islander Peoples, culturally and linguistically diverse people, people of different ages and genders, and people with disabilities. For example, avoiding bias in data analysis, ensuring content is accessible to people with disabilities such as following World Wide Web Consortium (W3C) standards for machine-readable text, and respecting the Indigenous Cultural and Intellectual Property (ICIP) protocols relating to Aboriginal peoples' rights to their heritage, knowledge and cultural expressions.

INFOBIT

AI systems are commonly used to analyse medical imagery. In the area of chest radiography, it has been shown that models for pathology classification have a higher rate of underdiagnosis for underrepresented sub-populations due to sampling bias in the training data set. The use of these systems could therefore increase the probability of those patients being sent home without receiving the care they need.

Technical considerations

Technical considerations for an information system involve understanding how the system will integrate with existing hardware, software or networking and, if there are compatibility issues, how these issues will be resolved.

If data exported from one system needs to be imported into another system, the data needs to be in a suitable format that can be read by the new system.

It is also essential to consider other external digital systems to ensure protocols are followed to allow for this interaction and communication of data. For example, most organisations selling products online use an online payment system such as PayPal. Their website and payment system need to be able to accurately and securely communicate with PayPal.

Information system - <Title>
<Purpose>

Digital System

Components:

Hardware	Software	Networks

Key functions:

Data and information

People and processes

People are those that use the system or are involved more directly with the functioning of the system, such as a system's administrator.
Processes include anything related to the collection, organisation, editing, transmitting and receiving, analysing, storing and retrieving, and displaying of data and information.

Process	Person / People

Context / Environment

Social considerations	Technical considerations	Legal considerations
How will/does the system impact (positive or negative) individuals and the community?	*What existing hardware, software or networking will the system need to integrate with? Does the system need to interact with other external information systems?*	*Does the system abide by laws such as privacy and copyright? Does the system facilitate possible illegal action by users?*

Systems thinking

Figure 3.20 A template for documenting the key features of an information system in context.

Cybersecurity principles

With most systems existing online, cybersecurity should be a key focus in information system development. The following are some of the key cybersecurity principles that organisations and individuals should follow to protect themselves against cyber threats:

- Confidentiality: Sensitive information should be protected from unauthorised access. This can be achieved through measures such as encryption, access controls, and data classification.
- Integrity: Data should be accurate and not tampered with or modified without authorisation. This can be achieved through measures such as data backups, version control and digital signatures.
- Availability: Authorised users should have access to information and systems when needed. This can be achieved through measures such as redundancy, disaster recovery planning and access controls.
- Authentication: Only users who are authorised can access information or systems. This can be achieved through measures such as passwords, biometrics and multi-factor authentication.
- Authorisation: Users have access to only the resources and information that they are authorised to access. This can be achieved through measures such as role-based access controls and user permissions.
- Accountability: Users should be responsible for their actions and that there is a clear record of who has accessed what information and when. This can be achieved through measures such as audit logs, activity monitoring and incident response planning.

Overall, these principles form the foundation of a strong cybersecurity strategy, and organisations and individuals should implement them to better protect themselves against cyber threats.

Legal considerations

Many IT companies employee specialist legal advisers to ensure their systems meet legal requirements. Not meeting the requirements can be very costly if an incidence highlights a company's lack of consideration. Two common laws to which information systems should abide are privacy and copyright. Many information systems store personal information about users. To ensure this information is kept secure, the information system should use established processes such as secure passwords and encryption of data.

Another legal consideration is whether the system could be used to facilitate illegal action. For example, an online alcohol supplier should not sell products to people under the age of 18. Social media systems even have some level of responsibility to consider and address the illegal act of cyber bullying that could occur through their system.

CASE STUDY

Electronic bike and scooter sharing

Shutterstock.com/Cascade Creatives

Figure 3.21 Lime-E bikes for hire.

A transport trend spreading around the world is the sharing of electronic bikes and scooters. Companies such as Lime provide the bikes (Figure 3.21) and scooters, along with the complex information system controlling their sharing and maintenance.

Users download the mobile application (Figure 3.22), sign up for an account, then for a small fee can unlock the scooter using a Quick Response (QR) code on their phone, with further charges based on the distance travelled.

Although this concept has many merits, including low-cost, efficient travel and environmental benefits such as lower carbon emissions, in some cities there have been many issues that have limited the success of the system. This is because, like all information systems, the people using the system and the context of where the system is being implemented have a huge impact on its success.

Chapter review quiz

Systems thinking

One big issue relating to people is that users are not following the outlined processes correctly, including riding in locations that are not allowed, or parking the bike or scooter in places that are inappropriate. Brisbane trialled the e-scooters before fully implementing them. One procedure put in place in order to address past issues is users were required to upload a photo of how they parked their scooter, which was analysed to determine if it was satisfactory. If a user parked badly a few times, the company would suspend their account. In order to minimise this issue even further, the latest technology in development includes an indicator on the light-emitting diode (LED) screen on the scooter to tell the user if they have parked in the wrong place.

Legal issues are a key consideration relating to the context of the system's implementation. Many countries and cities have laws that make it illegal to ride electronic scooters in public areas either above a certain speed limit or at all. An automated feature used to address this is the use of a speed limiter. Companies have also gone down legal paths in some cities to have laws changed or exemptions allowed.

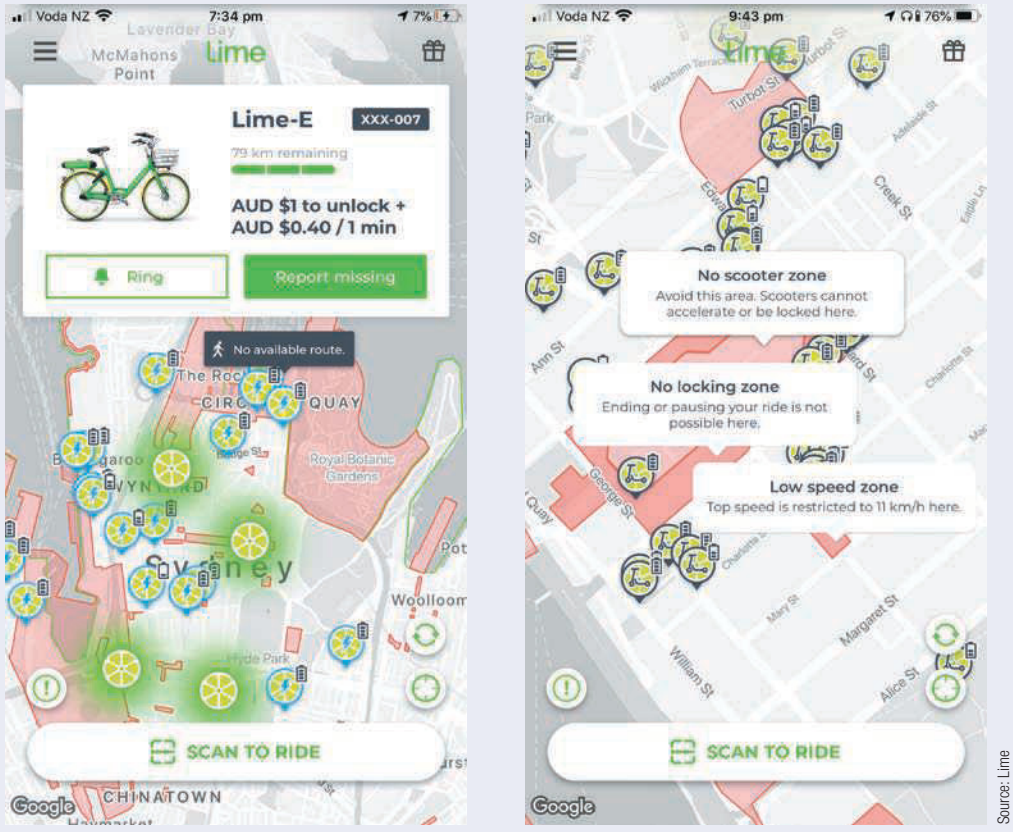


Figure 3.22 Screenshots from the Lime-E app showing key functionality of the map view, including locating available bikes or scooters and showing which areas have certain use restrictions.

Questions

1. Identify the purpose of this information system.
2. Outline some processes performed by users, highlighting any data or information involved.
3. This information system incorporates elements of a transaction processing system and a GIS. Describe the functionality in the elements relating to these categories of information systems.
4. Many information systems are trialled before they are fully implemented. Explain the benefits of trialling a new system.
5. Summarise the considerations and issues associated with this system. Find a recent news article on how a system like this may have had a negative impact on its environment.

ACTIVITIES

Identify

1 When creating an information system, what should a developer consider in addition to the components of hardware, software, networks, people, processes and data?

2 Identify five examples of the application of cybersecurity principles.

Analyse

3 Think of an information system you are familiar with. Suggest how a factor in an information system's environment could impact its functionality and provide an example of how the information system could impact a factor in its environment.

Extend

4 Copy and complete the information system in context template (Figure 3.20) for the electronic bike and scooter sharing system.

Weblinks:

- Keeping Culture
- Expert system

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

4 An introduction to software development

Chapter outcomes

In this chapter, you will learn:

- what software is
- basic types of software
- ways of representing algorithms
- about different computing languages
- basic programming concepts
- structured programming
- basic data structures
- anatomy of a computer program: variables, operators, expressions, assignment statements.

Key terms

algorithm
application software
assignment statement
compiler
control structure
desk check
flowchart
function
object code
operating system
software
source code
structured English (pseudocode)
syntax
utility software
variable

9780170442787

IN ACTION

Three computing languages in search for the *Titanic*

In the perpetual darkness more than two miles below the surface of the North Atlantic, a submersible sled slowly traced the alpine contours of the ocean bottom in the summer of 1985.

Named the *Argo*, after the ship in which the legendary Greek hero Jason sought the Golden Fleece, the 4.5-metre-long craft looked like a section of scaffolding flung on its side and stuffed with equipment. Far above, in front of a video screen in the control room of the US Navy research vessel *Knorr*, members of a joint French–American scientific expedition intently watched the images transmitted by the submersible as it was towed above a desolate landscape of canyons and mudslides. After 16 days of patient searching, a scattering of metallic debris appeared on the screen, followed by the unmistakable outline of a ship's boiler. An excited yell arose from the scientists.

The ocean liner *Titanic*, which sank 73 years earlier with more than 1500 of its 2200 passengers on board, had finally been found. This quest for the remains of the *Titanic* in the crushing depths of the sea was a remarkable application of computer programming.

Argo's collection of sonar, lights and cameras was controlled by an array of computers, each programmed in a different computer language. The computer on the unmanned *Argo* itself was programmed in FORTH, a tiny but versatile language originally designed to regulate movement of telescopes and also used to control devices and processes ranging from heart monitors to special-effects video cameras. The computer on the *Knorr* was programmed in C, a powerful and rather cryptic language capable of precisely describing computer operations. The telemetry system at either end of the finger-thick coaxial cable connecting the vessels (Figure 4.1), which allowed their computers to talk to each other, was programmed in a third, extremely basic language known as assembly language.

Programming languages are the sequences of words, letters, numerals and abbreviations used by people to communicate with their computers. Without them, computers and their equipment would be useless hardware. Computer languages are far more precise than human languages. They do not have multiple meanings or a sense of humour. For this reason, programmers must be precise when they use them.

Today there are thousands of computer languages (see a few in Figure 4.9). They allow their users to solve complex mathematical problems, crunch business statistics, create musical scores and draw computer graphics. No language is perfect for every situation. Usually, the language is known to the programmer, can be used on the available computer and is suited to the problem to be solved.

The three languages used on the *Titanic* expedition are a good example. For the computers aboard the surface ship *Knorr*, C was the preferred language because it provided more direct control of the computerised hardware. FORTH was the only high-level language that could be used on the submersible *Argo*'s computer. The precise timing required of the signals passed by cable between the vessels was best achieved using strict assembly language.

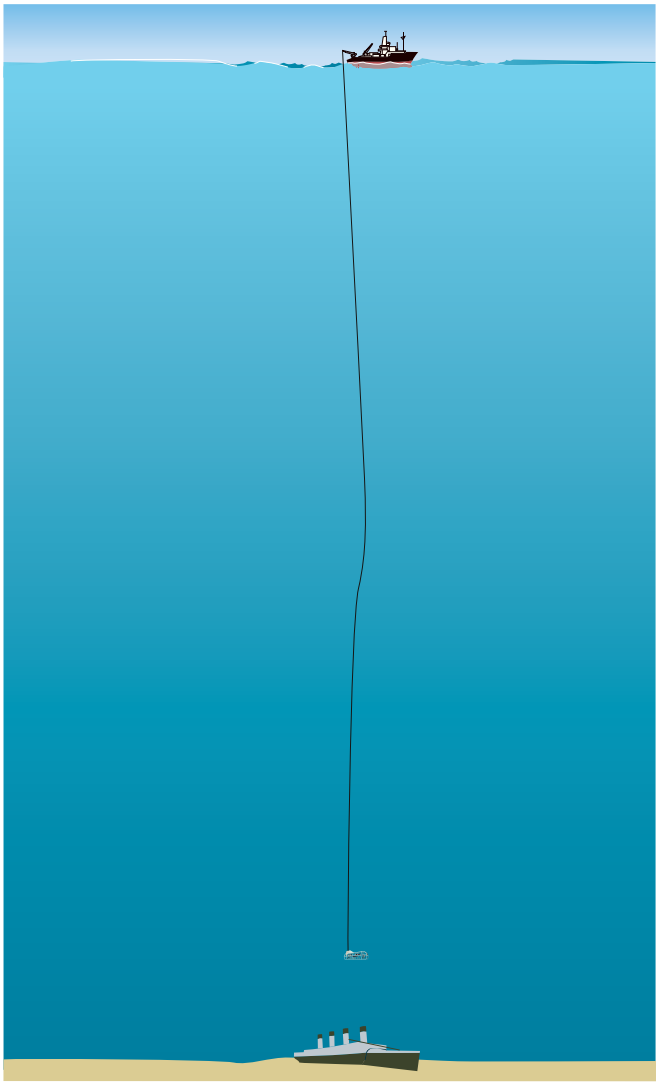


Figure 4.1 The *Argo*, with its ability to send real-time video images to the *Knorr*, was the key to finding the *Titanic*. This illustration, to scale, shows the relative size and depth of the *Titanic*, resting on the seabed.

Questions

1. List the three computer languages used to find the *Titanic*.
2. Describe a major strength of each of these languages.
3. What are the main similarities and differences between human languages and computer languages?

4.1 Understanding programming

This chapter provides an overview of basic concepts in **software** programming. These are essential for many of the focus areas in this course. By the end of this chapter you will be writing your first working code including coding a micro:bit. In later chapters you will need this knowledge to build many exciting programming projects, programming micro:bits to create physical toys, developing interactive websites, controlling robots as well as coding role-playing games using object-oriented programming.

Software is created using programming languages. In this course you will learn the principles common to most programming languages. We begin by looking briefly at the history of computer programming to gain an appreciation of the journey that has delivered the amazing tools we now use.

The first computers

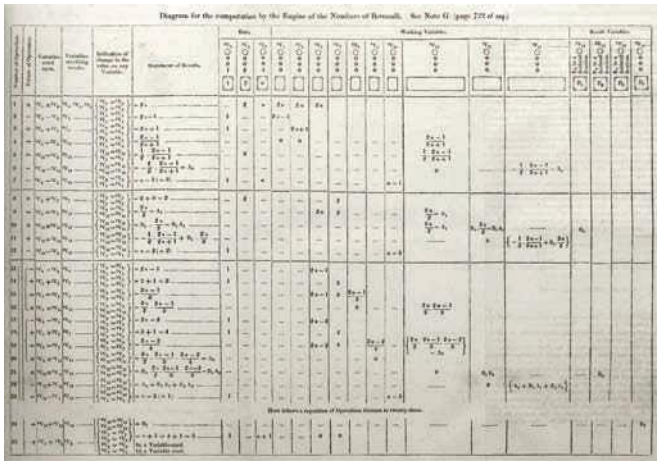
Systems thinking

Computational thinking

The computer is the most complex machine human beings have ever created, so you might think this means programming a computer would be incredibly difficult. However, all programming depends on the basic concepts covered in this chapter.

The first computer program was written by Ada Lovelace in 1842. It was a program for a calculating machine that her colleague Charles Babbage designed but never built (Figure 4.2).

Early computer programs (pre-1940s) had to be hard-wired into a computer. By arranging wires with plugs in a certain way and flipping switches, different programs could be run. Each new program needed a different arrangement (Figure 4.3).



Wikimedia Commons/Ada Lovelace/Public domain

Figure 4.2 The first computer program, written by Ada Lovelace in 1842 for a calculating machine that was never built

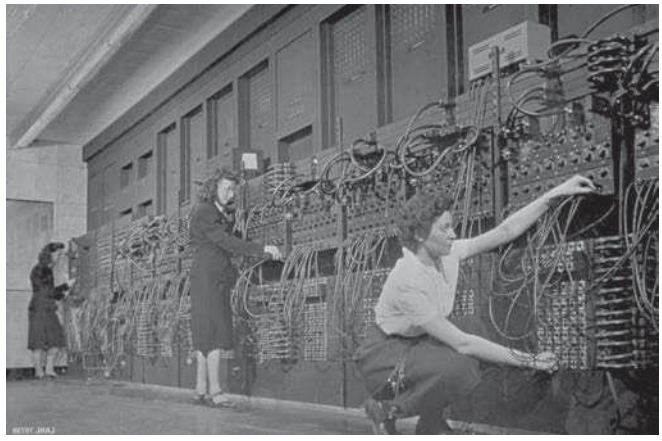
This was inconvenient for the humans operating the computers as well as being very tedious and time-consuming. In fact, it was the human operators, not the machines, who were called 'computers' in the 1940s!

Translating from human to computer language

Design thinking

Computers do not understand ordinary spoken languages such as English. Computers only understand *machine language*, which humans represent using just two binary symbols, '0' and '1'. If you could see an even lower level, you would observe machine language being translated into instructions called *microcode*, used to control the millions of transistors in the circuits of the computer's components.

As machine-friendly code is just strings of zeros and ones, it is very difficult for humans to read it. Other low-level languages called *assembly languages* were developed to help solve this problem. These languages use alphabetic codes with mnemonic features in place of binary instructions, which makes them easier for humans to understand. An example of assembly code is shown in Figure 4.9.



Getty Images/Corbis Historical

Figure 4.3 Early computer programs (pre-1940s) had to be hard-wired into a computer. Each new program needed a different arrangement, which was very inconvenient. The human operators were actually called 'computers'!

9780170442787

PEOPLE

Grace Hopper

Grace Hopper (1906–1992; Figure 4.4) argued programs should be written in languages closer to English, rather than in machine code or even in assembly language.

In 1952, she developed one of the earliest 'compilers', which converted regular English into machine code understood by computers. These high-level languages hid the details of the computer's operation and were far easier for a programmer to use. Programmers could finally concentrate on their code instead of the operation of the machine. This led to the development of high-level languages that were able to run on more than one type of computer, such as the business language COBOL.

Getty Images/Cynthia Johnson

Figure 4.4 In 1952, Grace Hopper led a team that invented the first compiler to convert English-like instructions into machine code.

Eventually, far more human-friendly languages (high-level languages) were developed, the kind you will use in this course. Software, known as **compilers**, translates this high-level human-friendly code into the assembly code, or other in-between code, then finally into machine code. All the applications we use, such as Microsoft Word, Adobe Photoshop, gaming software, browsers as well as **operating system** software, such as those used in Windows and Apple computers, must first be translated into machine language before a computer can use it.

For humans to communicate with computers, high-level computing languages, such as C++, JavaScript and Python, are essential.

Natural languages

Have you wondered why we do not use natural spoken languages for such programming?

Computers are not yet able to interpret natural English reliably. Recognising human speech (which they can do quite well) is much easier than turning speech into computer instructions.

The meanings of natural languages can be unclear. For example, the newspaper headline 'Giant waves down funnel!' (see Figure 4.5) can have more than 10 meanings (see if you can work out at least eight), so it would be easy to choose an incorrect one.

English is full of idioms and metaphors, such as 'I have been working like a dog', and insults such as 'she's one sandwich short of a picnic' or 'he's as thick as two planks' may be quite challenging for a computer! It is likely that artificial intelligence (AI) will some day solve this limitation.

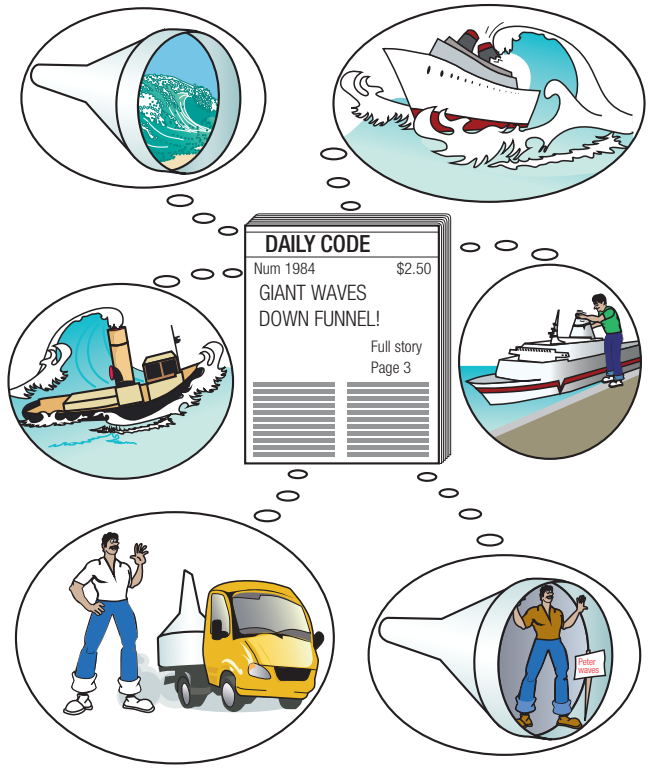


Figure 4.5 English language can be interpreted in many different ways.

Breaking down complex machines

We started this chapter by stating that the computer is the most complex machine that human beings have ever built. However, like most complex things, computers can be understood by breaking them into simpler parts.

A computer program is a set of instructions run by a computer, just as a storybook is a collection of sentences. These instructions are like the turn-by-turn instructions you might be given for directions: turn right at the intersection, walk two blocks, keep walking until the first set of traffic lights. The computer follows each instruction it is given.

Computational thinking

Computers add – very fast!

At their lowest level, computers just process two voltage levels: one high and one low, which we represent using the symbols 1 and 0. Computers perform calculations using circuits with up to 10 billion transistors.

In fact, it can be argued that computers really only do one thing – add – but they do it very, very fast. How could this be true? Think of the following basic mathematical operations:

- Subtraction is just adding the negative of a number.
- Multiplication is adding the same number a given number of times.
- Division is repeatedly subtracting a number (which we have said is just negative adding) until zero is reached.
- Raising a number to a power is multiplying a number (repeated addition) by itself a certain number of times.

When a user alters an image using photo editing software, binary code representing each coloured pixel is altered, involving addition. Even comparing two quantities uses addition: $x > 5$ is true only when $x - 5$ is a positive number (remember, subtraction is just negative addition).

The first computers were programmed simply like a cookbook. Ingredients (*input*) were put into the computer, the steps for the recipe were written out in order (*processing*) and finally the product emerged (*output*).

These early languages, such as the first versions of the popular language BASIC (Beginner's All-purpose Symbolic Instruction Code), were just lines of instructions written one after another, and each line was numbered. Often there were many jumps around the program code. This was messy, hard to follow and became known as 'spaghetti' code, and sometimes in Australia as 'kangaroo' code (see Figure 4.18). A better approach to coding needed to be found.

In computer programming, high-level languages are described as having a high level of *abstraction*. Abstraction is an important concept in computer science, and you will encounter it many times as it is an important part of computational thinking. Here abstraction means to hide details and concentrate on the bigger picture.

Source code and object code

High-level code used by humans is known as **source code**. Machine code used by the computer is known as **object code**. Machine code is the only language that can be understood by a computer. High-level computing languages such as C++, Java and Python are used by humans because they are easy to understand but they must be converted into computer machine code. Compilers are the software tools that perform this conversion (Figure 4.6).

Systems thinking

In the earliest computers, programmers had to write code using the language of computers (object code), making programming extremely challenging. From the 1940s and for the following 35 years, programs were input to computers using holes punched into cards or into paper tape (Figure 4.7). It is only in the last 50 years that it has been possible to input code directly into a computer using a keyboard.

INFOBIT

The fact that a 'byte' in binary code is formed by a sequence of eight bits was because that was the maximum number of holes that could fit in the width of the original paper tapes (as in Figure 4.7).

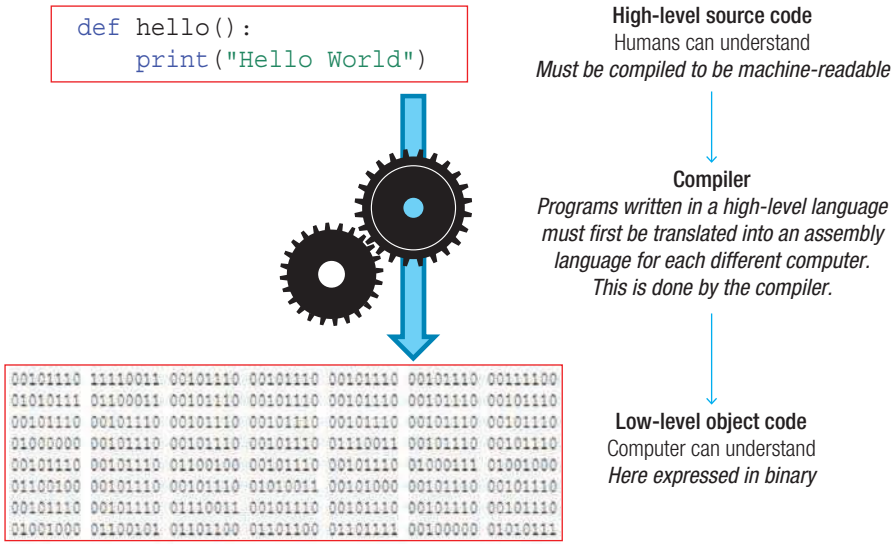
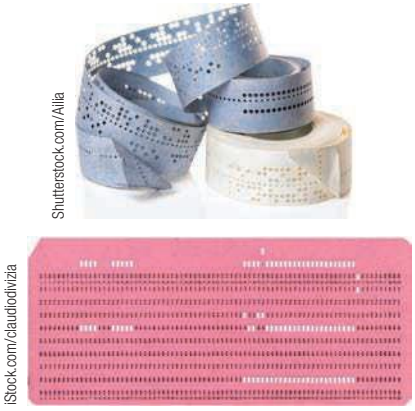


Figure 4.6 Machine code used by the computer is known as object code. High-level code, such as C++, Java and Python, is used by humans and is known as source code. A compiler translates source code into object code.



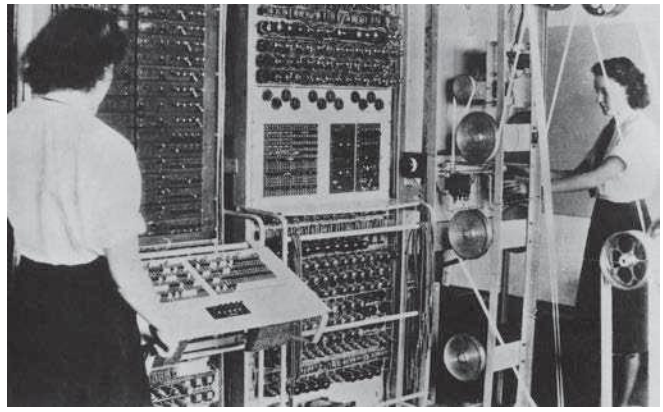


Figure 4.7 For many years, paper tape (top left) and, later, punched cards (bottom left) were the only ways to input code into a computer. On the right is Colossus, the world's first large-scale electronic computer, showing paper tape used for data input. Colossus was critical in Britain's successful wartime code-breaking at Bletchley Park.

4.2 Understanding software types

Systems thinking

The 'In action' section at the start of this chapter tells the real-life story of how three different software languages were used to explore the wreck of the *Titanic*. Different types of software are needed by computing systems and each performs different tasks.

Operating system software

On a personal computer, an operating system is needed to control the computer's hardware. Microsoft Windows and macOS are both widely used operating systems.

Application software communicates with operating system software and the operating system then communicates directly with hardware.

The operating system is the first thing we see on the screen when we start up a computer; however, it has many tasks apart from managing this opening interface. Its other major tasks include managing hardware, the central processing unit (CPU), memory, software applications, and input and output devices. Finally, when we shut down the computer, it tells us it is logging off and closes files in a systematic order before it powers down the hardware.

Application software

Most people are familiar with common **application software**, such as web browsers, word processors, spreadsheets, image editors like Adobe Photoshop and presentation software like Microsoft PowerPoint, which are able to perform a wide range of tasks creating, editing, formatting and displaying documents.

However, it is not only these big players who produce useful software applications. Individual software developers have written hundreds of thousands of applications (apps) for mobile devices. These apps perform single or limited tasks and can be created using open-source (free), user-friendly software.

Embedded systems

Some hardware equipment, such as microwave ovens, dishwashers and other home appliances, or even anti-lock braking systems on cars, are designed with both an operating system and an application embedded in a single circuit board – a type of mini-computer.

Embedded systems respond quickly and reliably, as the program is always running, awaiting an event such as the press of a button on the microwave oven to set a timer and commence heating a meal. Embedded systems perform well as small single-purpose computers.

Embedded systems are popular as prototyping tools and with hobbyists. Examples include Arduino, Raspberry Pi and micro:bit boards. In this course, you will have the chance to program your own boards.

Utility software

A third type of software, known as **utility software**, has specific tasks to perform, usually 'housekeeping' such as file management, print management or virus protection.

Utility software is usually part of system software, and works at the operating system level. It includes functions such as creating directory structures,

copying and deleting files, performing backups and altering system settings. Other utilities work at the application level and include virus scanners, file-sharing software and compression software (Figure 4.8).

Choosing a computing language

Every language has its strengths and weaknesses. Some have been designed for particular purposes; some are difficult to read but are fast; and some, such as COBOL, are widely used in banking and business. This was highlighted in the 'In action' story of the *Titanic* at the start of this chapter.

Weblink
The Hello World Collection

There are thousands of high-level computer languages, but popular ones today include Java, Python, JavaScript and C++. Popular languages in early computing included BASIC, Pascal and FORTRAN. Figure 4.9 shows a collection of languages, all with their version of the traditional first program students write when learning a new computer language. They all output the message 'Hello world!'.

Using Python

This chapter will introduce you to computer programming using structured English (pseudocode) and features the Python programming language. Your teacher may choose another language for your class; if so, you will discover how similar computer languages are. By learning one, you will be well positioned to learn others.

We have chosen Python for many of our examples in this text because it is open-source and free, is suitable for all platforms, and is a popular choice for teaching programming in schools and universities. It is also used by large organisations, including YouTube, the National Aeronautics and Space Administration (NASA), Google and the European Organization for Nuclear Research (CERN). It is close to structured English. It is also the language of choice for some programming competitions we recommend, such as the National Computer Summer School Competition. If your teacher chooses another language for your class, this will serve to further illustrate these principles.

INFOBIT

The creator of Python, Guido van Rossum, was employed by Google from 2005 until 2012. Most of the core search algorithms for Google are written in Python and C++. YouTube uses Python heavily, and machine learning, AI and robotics projects at Google are implemented using Python and C++.

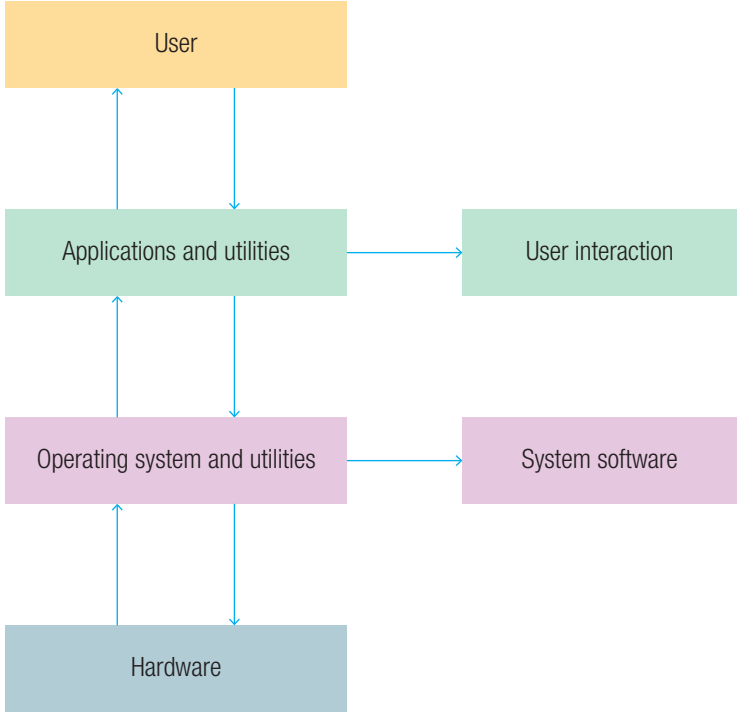


Figure 4.8 The three types of software – application, utility and operating system – are considered to operate like a stack, so that the application software or a utility communicates with the operating system. The operating system will then communicate directly with the hardware.

Figure 4.9 'Hello world!', a traditional first programming exercise in any programming language, shown in a number of different computing languages. Take a look at more than 200 examples by visiting the Hello World Collection website.

All programming languages use their own methods to define their structure. Structure is needed to make code more readable for humans by separating blocks of code from one another. You can see in Figure 4.9 many languages do this using curly braces, { }, while others use BEGIN and END statements. Python instead uses indentation to define its separate structures, and this enhances its readability. Many languages use a special character to mark the end of each instruction, usually a semicolon. Python instead simply uses the end of a line to show where instructions end.

We think you will agree the Python version of 'Hello world!' in Figure 4.9 is the easiest to understand and is the most efficient to write.

ACTIVITIES

Identify

1. Write a list of the most common things you use your computer for, at home and at school.
2. Which software applications would you use to complete the tasks you listed in question 1?
3. Identify the three types of software that make up the software system on most desktop computers. Give an example of each.

Analyse

4. Compilers are used to convert source code to object code. Explain the main processes involved.
5. What are the most common examples of utility software found on computers and what are they used for?

Extend

6. Research two strengths and two weaknesses of each of Windows, macOS and Linux operating systems. Draw up a table to summarise your research.

4.3 Understanding algorithms

Computer programs depend on **algorithms**. An algorithm is a step-by-step procedure to solve a problem.

If you offer to make a cup of tea for someone, you follow a set of steps in your head, including asking if your guest has sugar and/or milk and whether they prefer it strong or weak. You would be following an algorithm for making tea.

If a programmer knows a computer language well, the development of the algorithm usually takes far more time and effort than the actual coding, in the same way that writing a novel takes longer than accurately translating it into another language.

Ways of representing algorithms

Computer programmers have developed two main methods to represent these algorithms or rules: flowcharts and structured English (also called pseudocode).

Flowcharts

If you were asked to make a cup of tea for a stranger, you would probably do something similar to Figure 4.10. By following these steps, a person could make a cup of tea. The order in which you perform the actions is important. It would not be a good idea to put the milk in the water before the tea bag or boil the water after all the other steps! This diagrammatic way of representing an algorithm is called a **flowchart**.

Flowcharts are often used by computer programmers to help them think through the order in which steps in their code need to be written.

Flowcharts use five basic symbols, shown in Figure 4.11.

- Flowcharts always include BEGIN and END using rounded rectangular shapes.
- Any *action* or *process* that needs to be performed is drawn inside a plain rectangle.
- Any *information (input or output)* needed from a user of the program or any information given by the computer back to the user is drawn inside a parallelogram.
- Finally, any *decision* needing to be made is drawn inside a diamond. Decisions always have one *flow line* in and two flow lines out: one for True and one for False.

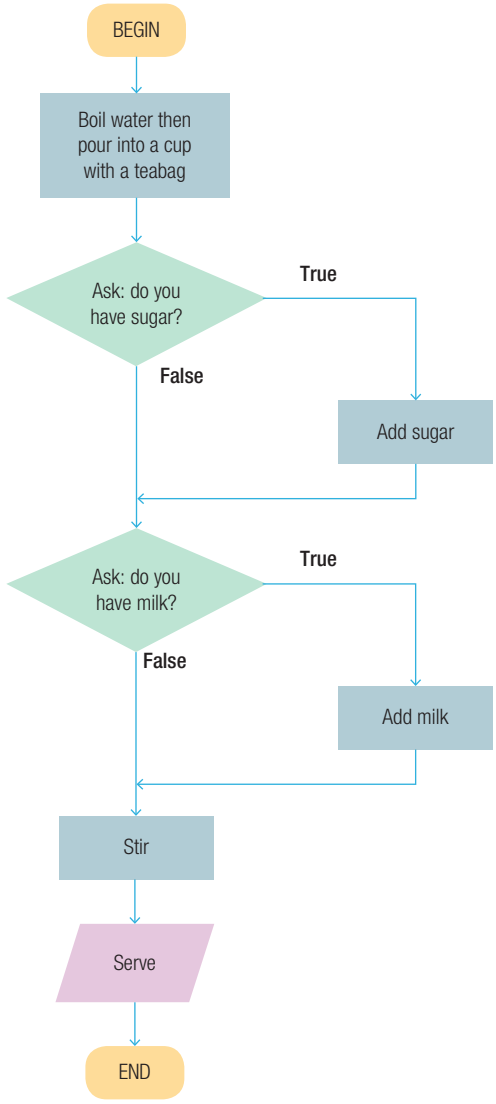


Figure 4.10 Possible flowchart for making a cup of tea.

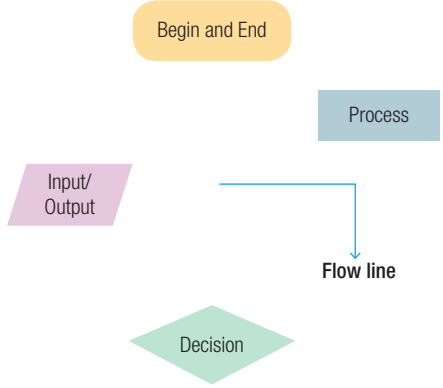


Figure 4.11 Flowcharts use the standard symbols shown here.

9780170442787

SKILL BUILDER

Developing algorithms using flowcharts and structured English (pseudocode)

Flowcharts and **structured English (pseudocode)** can be very helpful in developing algorithms, and eventually your own working code.

Flowcharts

1 Imagine a number guessing game being played against a computer. The computer has secretly chosen a number (under 100) that you have to guess in as few moves as possible, only being told 'Too high' or 'Too low' each time you guess. Typical gameplay (using 63 as the number to be guessed) is shown in Figure 4.12.

2 How would you program a computer to play this game?

3 Try drawing a flowchart for this game before you look at Figure 4.13.

4 Carefully check through your flowchart, by supposing `secretnum` (the secret number) is equal to 30 and working through it a few times using starting guesses of 1, 29, 30, 80 and 100. Trace what happens for each secret starting number. This process is known as desk checking. A complete **desk check** should cover every path through a program and every boundary value.

Your flowchart might look like Figure 4.13. Notice that the first test decision diamond captures a correct guess and the second checks for a guess that is 'Too high'. This leaves 'Too low' as the only possibility remaining.

Notice how this flowchart also tracks how many guesses you are having.

5 Now try drawing a flowchart for a human playing a computer in a single turn of a Rock Paper Scissors game. Use `comp_choice` for the computer's move, and separately test its logic against every possible move by a human opponent. Do this for each of the three items, tracing the paths using your finger.

Computational thinking

Structured English

As well as flowcharts, programmers have invented a text-based method to represent an algorithm before coding it. It is known as structured English (or pseudocode – which means 'not quite real code'). This is very much like plain English and describes the logic of a program. Programmers can then convert structured English into actual code.

Figure 4.12 Typical gameplay for the number guessing game. Here the computer is represented by thought bubbles on the right and the human guesser by speech bubbles on the left. The randomly generated secret number this time was 63.

Structured English uses standard keywords, which are always written in capitals. The main ones are `START, END, IF, ELSE, ELSE IF, ENDIF, WHILE and UNTIL.`

Its main feature is indenting, just as the Python language does, which makes it easy to identify separate parts of the algorithm. Here is our number guessing game, this time expressed using structured English:

```
BEGIN high low guessing game
   set secretnum to 30
   get guess
   WHILE guess is not equal to
     secretnum
      IF guess > secret THEN
         print 'Too high'
      ELSE
         print 'Too low'
      ENDIF
      get next guess
   ENDWHILE
   print 'Correct!'
END
```

6 Trace the logic through this pseudocode with your fingers by supposing `secretnum` (the secret number) is equal to 30. Do this several times using various guesses so that every possible path through the algorithm is tested.

7 If the guess is neither greater nor less than the secret number then it must be equal to it! Can you rewrite this flowchart to eliminate the diamond that checks for equal?

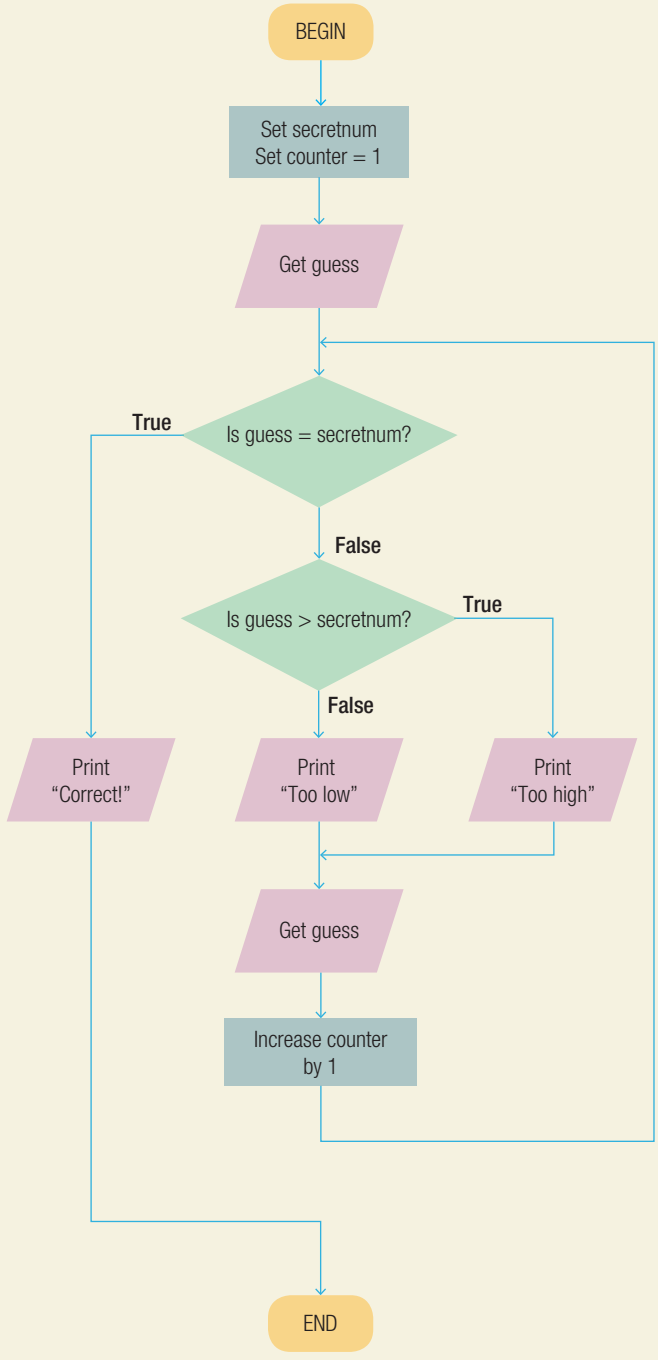


Figure 4.13 Our number guessing game represented as a flowchart

4.4 Two types of computing languages

There are many different styles of computing languages. We looked at two methods to represent algorithms. Now we will look at two ways to code them: visual programming and structured programming.

Visual programming

Many people imagine all computer programming languages are very difficult to understand. This is not true.

Blockly and Scratch are examples of *visual programming languages* that are very similar to flowcharts and are a great way to begin programming.

Perhaps you have already used Scratch in primary school or perhaps coded a micro:bit using Blockly. Visual programming languages are becoming more and more sophisticated. In the beginning, they were only used to introduce younger students to coding, but many are now capable of being used to create advanced code.

Visual programming languages use jigsaw-like shapes that can only be joined in correct ways. Try tracing through the simple example in Figure 4.14, noting how similar it is to the pseudocode and the flowchart in Figure 4.13.

SKILL BUILDER

Visual programming

Use the online version of Scratch to build the number guesser game shown in Figure 4.14.

If you have never used a visual programming language before, Scratch is a good one to start with. There are excellent introductory tutorials provided on the Scratch website. Click Create, then select Tutorials. You can locate any shape by matching its colour to identify its correct library in the Scripts panel.

1. First create two **variables**, and name these `secret` and `guess` using Scratch's Variables tab. To store user input from the keyboard, use the blue `ask` shape.
2. Insert the `answer` shape below it to assign the user `guess` to `answer`.
3. Click the green flag to run your program code.

Weblink
Scratch

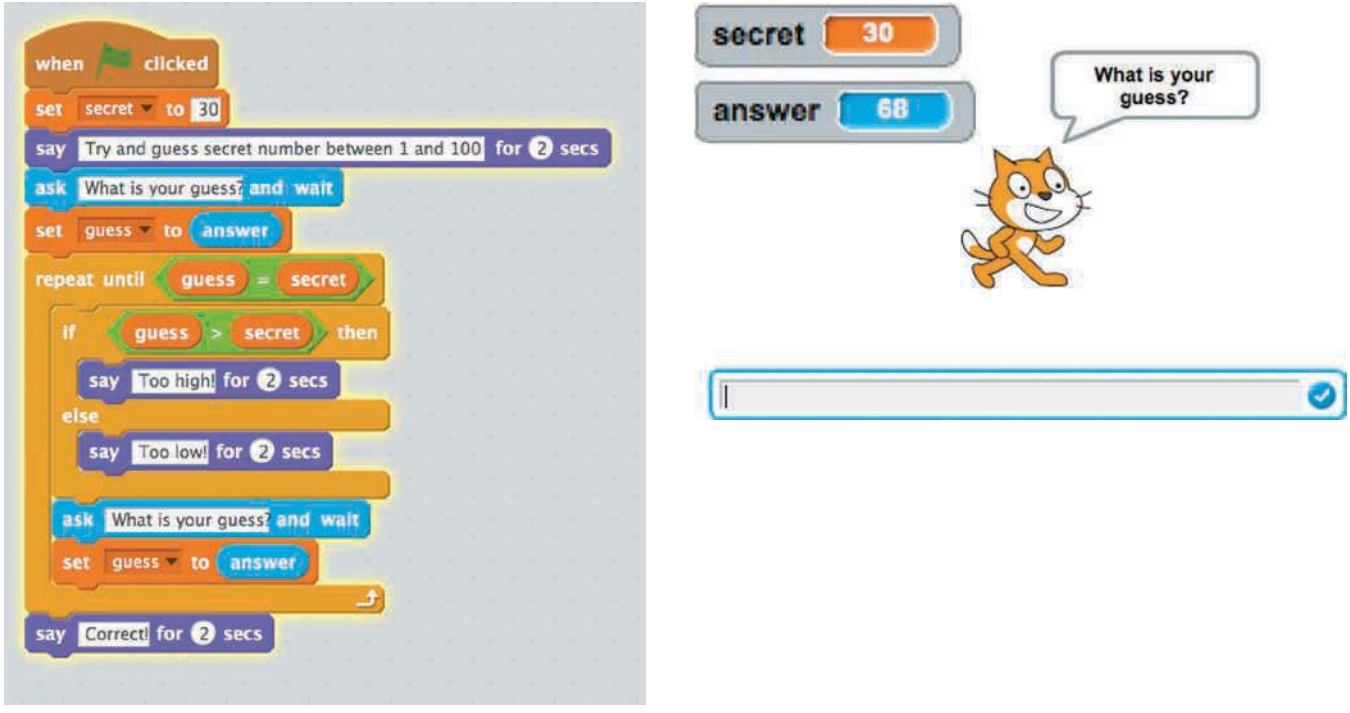


Figure 4.14 Our number guessing game using the visual programming language Scratch. Notice how close this is to the flowchart version in Figure 4.13.

General-purpose programming languages

In secondary school, you are expected to do the majority of your coding using a *general-purpose language*. We will now turn our attention to these. Here we take a first quick look at our number guessing game written in Python, a general-purpose programming language.

If you were to translate the number guesser pseudocode or flowchart (Figure 4.13) into actual code, without a counter, using the language Python (version 3), it would look like Figure 4.15. Compare these statements with the visual Scratch version in Figure 4.14. To help you understand what is happening in each line, comments have been added in red. In Python, comments are indicated by the symbol #. These are not part of the code and are ignored by the computer.

Here we are setting the secret number to be 30 to make testing easier, but later on we will generate a new random secret integer each time.

Can you see how the code uses the same steps as the flowchart and in the same order? Notice how the indented `while` block asks if the guess is *not* equal to the secret number. This whole indented block keeps repeating until the guess is equal to the secret number. When that happens, the program execution escapes the `while` block and moves to the last line and prints 'Correct!'.

Notice too how the indentation of the `while` block and the further indentation of the `if…else` blocks help you identify where each block begins and ends.

You will be adding to this code and improving it in a later chapter as you learn more.

Programming language syntax

Just as English sentences must use recognised vocabulary and obey rules dealing with their placement to make sense, for the same reasons computing languages must use recognised vocabulary (keywords) that must be placed in the correct order and with correct punctuation. In fact, it is even more important when coding, as humans can guess at meanings when rules of grammar are disobeyed, but computers can't … yet!

A **syntax** error will occur whenever a computer language's strict rules of grammar are disobeyed.

SKILL BUILDER

Syntax errors: identifying syntax errors in Python

Here are two examples where a print statement in Python is written using incorrect syntax. Can you pick the errors?

```
>>> print Hello world!
SyntaxError: Missing parentheses in
  call to 'print'
>>> print(Hello world!)
SyntaxError: invalid syntax
```

If you already know how to use your Python programming environment, then try correcting these errors to successfully print 'Hello world!'.

```
secret = 30                                  # set secret number to 30
guess = input("Your guess?: ")               # ask user for input
guess = int(guess)                           # change guess into integer
while guess != secret:                       # != means "does not equal"
    if guess > secret:
        print("Too high")
    else:                                    # arrives here only if guess < secret
        print("Too low")
    guess = input("Your guess?: ")           # next guess requested
    guess = int(guess)                       # change guess into integer, repeat while loop
print("Correct")                             # print "Correct!" only when guess is correct
```

Figure 4.15 A general-purpose programming language. This is our number guessing game written in Python. Comments for the programmer in red text are ignored by the computer.

KNOWLEDGE BUILDER

Tools for coding

Many free tools are available for writing and running code.

Your teacher might ask you to type the code shown in Figure 4.15 using a simple Python programming environment such as IDLE or in an online editor such as replit, as shown in Figure 4.16.

Such tools are known as integrated development environments (IDEs).

If you download Python from its official site, IDLE will come bundled with it. At the end of this chapter, a summary of Python essentials is available for you to reference along with excellent online resources for learning Python.

Try copying and successfully running this code for yourself to become familiar with your programming environment. Note that syntax errors are very common for beginners! Especially note that a single mistake in indentation can generate lots of errors all at once! Once you have the code working, try removing the indents on lines 9 and 10. Also note that you do not need to type all the comments, which begin with '#'.

In programming it is also important that quotation marks are vertical, not sloping. Your Python IDE will make sure that all quotation marks are vertical.

Weblinks
Python IDLE

Replit

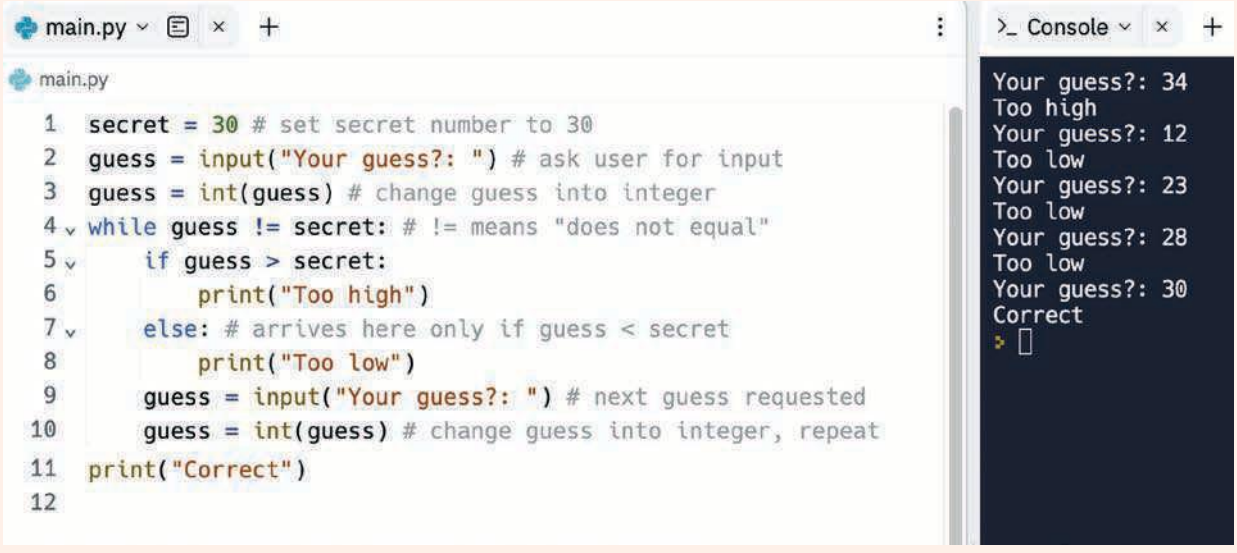


Figure 4.16 Showing the same Python code as in Figure 4.15 now using the online editing environment of replit. The result is shown in the right-hand panel.

INFOBIT

The creator of the Python programming language, Guido van Rossum, was a fan of the popular BBC TV skit show *Monty Python's Flying Circus* and he named Python after this and not after the snake! IDLE, the IDE tool for Python, was probably named after Eric Idle, an original cast member, although some say it stands for integrated development learning environment. This show was also the source for another familiar tech term: 'spam' for unwanted emails. Search online for this piece of unusual computing history.

4.5 Understanding structured programming

Systems thinking

In the 1960s, Dutch computer scientist Edsger Wybe Dijkstra disliked the way the order of execution of a program's code would jump around sequentially numbered lines from line to line (see Figure 4.18). Programmers coined the phrase 'spaghetti code' to describe the difficulties of following the logic in such code. Dijkstra popularised what became known as structured programming. He argued any computer program could be written using only three basic **control structures**: *sequence*, *selection* and *repetition* (or iteration).

Control structures are also called code *blocks*. Computer languages using this approach are called *structured languages*. This concept is similar to the way thousands of different structures can be built using just a few basic types of Lego bricks (see Figure 4.17).

In Figure 4.18, the old-style 'spaghetti code' we referred to earlier appears on the left. To the right is the same program written in Python's structured format. Note how much easier this is to understand. The output is shown in the figure and is identical in both cases.

Figure 4.17 In the same way as basic Lego brick shapes can be arranged to create thousands of different structures, the three building blocks of computer programming (sequence, selection and repetition) can be arranged to create any program.

Program to print stars written using early 'spaghetti' coding style in the BASIC programming language

```
10  INPUT "What is your name: ", U$
20  PRINT "Hello "; U$
30  INPUT "How many stars do you want: " N
40  S$ = ""
50  I = 1
55  IF I > N THEN GOTO 80
60  S$ = S$ + "*"
65  I=I+1
70  GOTO 55
80  PRINT S$
90  INPUT "Do you want more stars? ", A$
100 IF LEN(A$) = 0 THEN GOTO 90
110 A$ = LEFTS(A$, 1)
120 IF A$ = "Y" OR A$ = "y" THEN GOTO 30
130 PRINT "Goodbye "; U$
140 END
```

The same program written using the structured Python programming language

```
UserName=input("What is your name: ")
print("Hello ", UserName)
Answer = "Y"
NumStars = 1
while Answer == "Y" OR "y":
   NumStars = int(input("How many stars do you want: "))
   print(NumStars * "*")
   Answer = ""
   while Answer == "":
      Answer = input("Do you want more stars? ")
print("Goodbye", UserName)
```

When the integer '42' is input for a user 'Alan', the output from both programs is identical:

```
What is your name: Alan
Hello Alan
How many stars do you want: 42
******************************************
Do you want more stars? No thank you.
Goodbye Alan
>
```

Figure 4.18 (left) An example of line-numbered old-style 'spaghetti' code compared to the same program written in a structured language (right), in this case Python. Follow the steps in the Skill builder activity on the next page to copy and run the Python version to print any number of stars.

SKILL BUILDER

Counting stars

This activity demonstrates how a program written in a structured language (Python) works.

The program shown in Figure 4.18 prints out stars – don't you deserve a few?

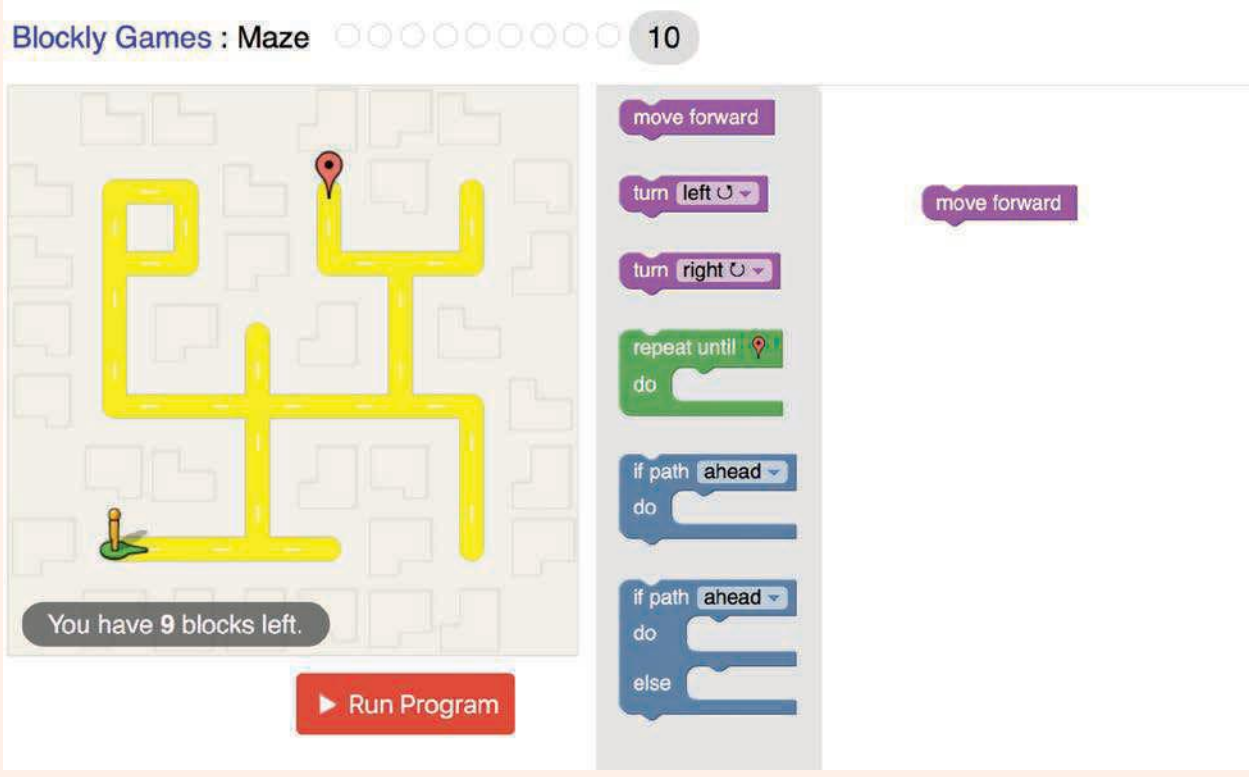
1. Copy the right-hand Python version. Run it and then modify it to do something different, perhaps to print a different pattern or group stars in blocks of five to make them easier to count.
2. Extra challenge: Can you use internet research to discover the connection between 'Alan' and 42 (see Figure 4.18)?

KNOWLEDGE BUILDER

Blockly Maze fun challenge

1. Try the Blockly Maze challenge. By solving levels 1–10 in consecutive order, you will master control structures using visual coding and discover the strengths of the three building blocks of structured programming. Level 10 is challenging!
2. While completing the challenge, you will encounter the actual general-purpose coding behind the blocks you assemble. State what computer language these visual blocks represent and write out the actual general-purpose code for each of the blocks shown in Figure 4.19. Leave curly braces empty.

Weblink
Blockly Maze challenge



Source: Blockly Games

Figure 4.19 Try the Blockly Maze challenge. By solving levels 1–10 in order, you will master control structures – the building blocks of structured programming.

4.6 Understanding the basic control structures

By completing the previous Knowledge builder, you encountered the three basic control structures (sequence, selection and repetition). All computer coding is based on these three basic control structures. We can represent them as flowcharts, in structured English or as the actual code in a particular computer language.

Sequence

The simplest control structure is sequence: lines of code follow in order one after the other. Most programs have lines written this way. Figure 4.20 shows the sequence structure in both the flowchart and pseudocode formats.

Selection

Few programs follow a simple sequence from beginning to end. Instead, they come to a branch where a decision is required. This means that the program reaches a point where it must decide what to do next. In order to make the decision, the program asks a question or tests a condition. This simple structure is called the selection control structure. The outcome of the test condition decides the next action.

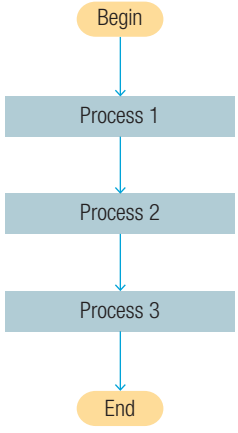
Program languages generally use two main selection structures: one allows a choice from two options, known as *binary selection*, and another is used when there are many choices, and is known as *multiple selection* or *multi-way selection*.

In a flowchart, binary selection is shown using a diamond-shaped box, as shown in Figure 4.21. Inside the diamond, a question that can only be answered with a 'Yes' or 'No' or, if you prefer, 'True' or 'False', is asked.

The equivalent pseudocode is written underneath.

SEQUENCE

Flowchart



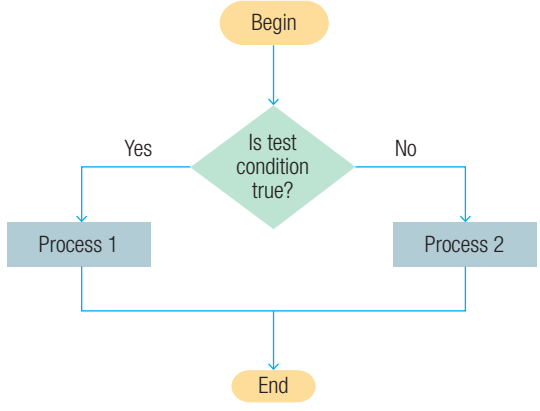
Pseudocode

```
BEGIN
  Process 1
  Process 2
  Process 3
END
```

Figure 4.20 The sequence control structure shown using both flowchart and structured English.

BINARY SELECTION

Flowchart



Pseudocode

```
IF <test conditions> = True THEN
 do Process 1
ELSE
 do Process 2
END IF
```

Figure 4.21 The binary selection control structure shown using both flowchart and structured English.

Repetition

Repetition (also called looping or iteration) is the most important of all the control structures as it allows a computer to repeat actions effortlessly many times.

The repetition structure does not have a special flowchart symbol. Instead, it uses a diamond-shaped decision (selection) box and a rectangular process box to show movement through a loop. Like branching, loops include a test condition. The test is used to decide whether the loop should keep going or terminate (stop repeating).

The type of loop we decide to use depends on how we want it to behave. Two categories of loops are:

- pre-test loops
- post-test loops.

Pre-test loop

A pre-test loop performs a test before starting on the loop. This type of loop might not execute the loop statements at all, depending on the outcome of the test condition.

If you run a finger around the logic flow lines in Figure 4.22, you can see that the loop tests the condition and only enters and executes the loop code if this condition evaluates to True, then tests again and repeats this cycle until the test condition evaluates to False, when it finally escapes the loop.

The equivalent pseudocode is written underneath.

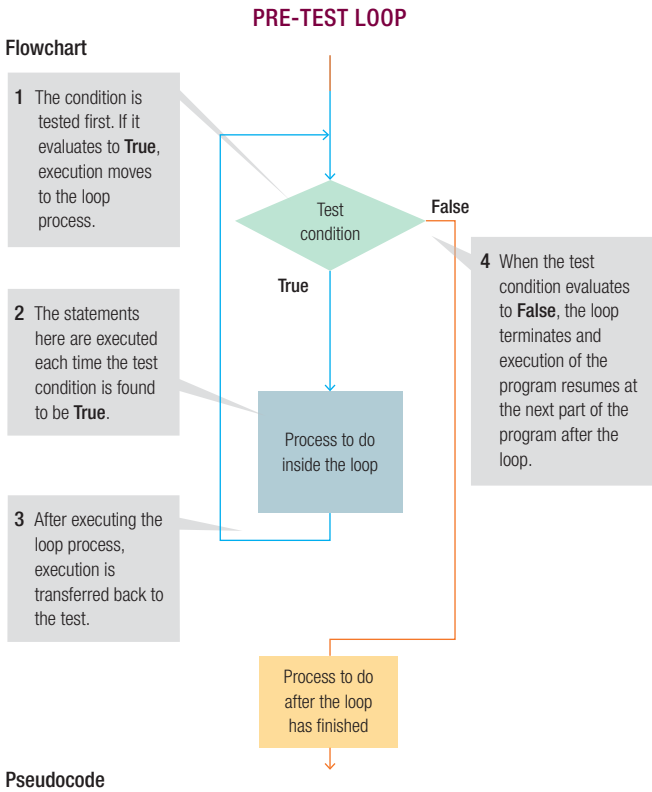


Figure 4.22 The logic of a pre-test loop is explained here, shown using both flowchart and structured English form. A pre-test loop performs a test before entering a loop.

Post-test loop

A post-test loop performs the test condition after each execution of the loop and keeps repeating while the condition evaluates to False. This type of loop must always perform the loop action at least once – the very first time – and possibly many times – until the loop test condition evaluates to True, which causes the loop to terminate.

If you run a finger around the logic flow lines in Figure 4.23, you can see that the loop first performs the process and only then tests the condition, here repeating the loop code again if the condition evaluates to False, or exiting when the test condition evaluates to True.

The equivalent pseudocode is written underneath.

Many languages support both pre-test and post-test loops. However, although Python has no post-test ('repeat') structure, a `while` loop combined with an `if` decision can be used instead.

Figure 4.24 illustrates the difference between pre-test and post-test loops using the example of eating a bag of chips.

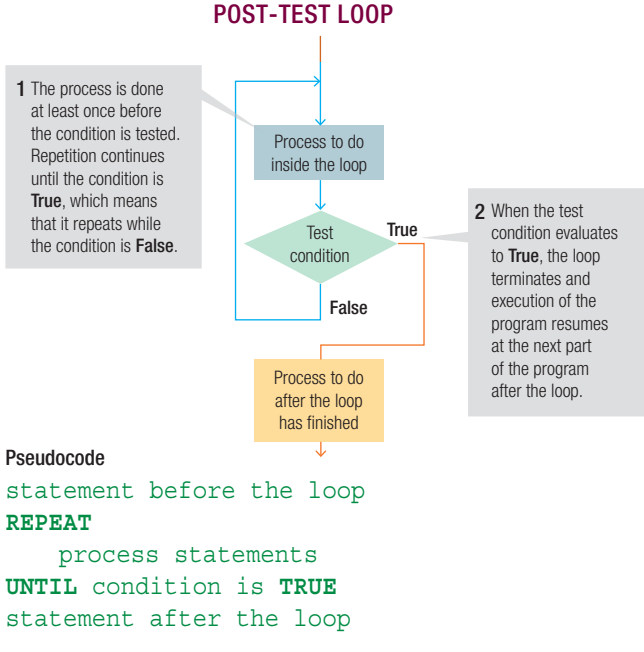
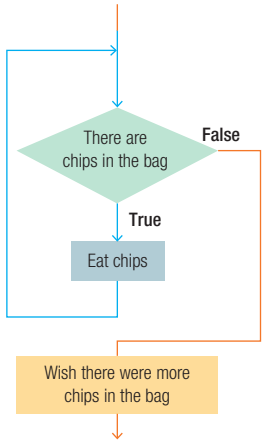


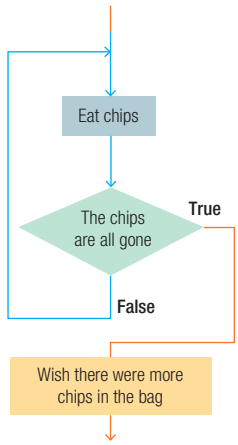
Figure 4.23 The logic of a post-test loop is explained here, in flowchart and pseudocode form. A post-test loop performs a test at the end of a loop.

PRE-TEST LOOP




```
WHILE There are chips in the bag
  Eat chips
END WHILE
Wish there were more chips in the bag...
```

POST-TEST LOOP




```
REPEAT
  Eat chips
UNTIL The chips are all gone
Wish there were more chips in the bag...
```

Figure 4.24 Some programs suit a pre-test loop and others a post-test loop. Compare the idea of eating all the chips in a bag using these two different looping approaches. If the chip bag was empty to start with, the post-test example will instruct you to do something that is not possible (eat chips). While this is not a disaster in real life, in a program it may cause an error. The error could not happen with a pre-test loop.

SKILL BUILDER

Using Python to apply the three control structures

In Python, statements that belong to a control structure block are indented. This indentation makes it clear that all statements in that block belong together. Blocks begin with a statement controlling the block, and the block itself is called the *body* of the control structure.

Selection and loop control structures in Python must also include a colon followed by an indented block.

Sequence

Copy and run the following Python program, which depends only on the sequence control structure.

```
firstname = input("What is your first name? ")
familyname = input("What is your family name? ")
print ("Hello", firstname, familyname)
print("Welcome to the sequence control structure!")
print("Next you will try selection.")
```

1 See if you can make a comma appear after `Hello` in the output.

2 See if you can output an exclamation mark following the family name in the output.

Selection

Copy and run the following Python program, which features the selection control structure.

```
firstname = input("What is your first name? ")
familyname = input("What is your family name? ")
print("Hello", firstname, familyname)
print("Welcome to the selection control structure!")
myage = int(input("How many years old are you to the nearest year? "))
if myage < 18:
    print("You are not yet old enough to vote!")
else:
    print("You are old enough to vote!")
print("Next you will try the repetition control structure.")
```

1 Make a comma appear after `Hello` in the output.

2 Output an exclamation mark following the family name in the output.

3 Can you explain the reason for `int` in line 5?

4 Try removing `int` and run the program again. Explain the error message that results.

Repetition

Copy and run the following Python program, which features the repetition or loop control structure.

```
count = 0
firstname = input("What is your first name? ")
familyname = input("What is your family name? ")
print("Hello", firstname, familyname)
print("Welcome to the selection control structure!")
stars = int(input("Do you want to meet some stars? How many? "))
print(firstname, familyname, "meets")
while count < stars:
    print("*")
    count = count + 1
print("Now you have experienced each of the 3 basic control structures!")
```

1 What does `count = count + 1` do?

2 What would happen if it was left out? Try it!

3 Python has a shortcut for `count = count + 1`, which is `count +=1`. Rewrite the increment line using this shortcut and rerun the program.

4 Can you explain the reason for `count = 0` in line 1? Try removing it and running the program. Explain the error message that results.

5 It is possible to write this program without using repetition at all. The clue is that Python allows you to multiply a string by an integer. See if you can do this.

6 The first version of this program printed each star on a new line. This is because Python by default inserts a new line instruction every time it prints. Try changing `print('*')` to `print('*', end=' ')` and see what happens. By default Python ends every print statement with a new line character. This replaces this new line character with an empty space. Modify the code to make the output neat.

A special loop structure: for loops

There is a special loop in Python that can be used if you know in advance the number of times you need to loop. This is called a *counted* loop, and most programming languages provide their own version.

This uses a very useful function in Python called the `range` function. `range(n)` returns a sequence of integers starting from `0` and stopping at `n-1`.

`range(5)` is the same as writing `[0, 1, 2, 3, 4]`.

`range` can be very useful as it can take three parameters:

`range[start_integer, stop_integer, step]`.

`range(2, 10, 2)` is the same as writing `[2, 4, 6, 8]`.

It creates a list starting at 2 and going up to *but not including* the second parameter 10, stepping up each time by 2 (the third parameter). If `start_integer` is omitted, it is assumed to be 0. Note that the value for `step` may be negative.

This special counted loop begins with the keyword `for` followed by a variable and ending with a colon. By combining it with the keyword `range`, we can control how many times loops are executed.

Both these versions of the `for` loop print integers 0 to 4 and their loop is executed five times:

```
for n in range(0, 5):
    print(n)
```

and

```
for n in range(5):
    print(n)
```

1 What do you think the following will do? Copy and run it to see if you are correct.

```
for n in range(10, 4, -2):
    print(n)
```

2 Rewrite your earlier star printing program using a `for` loop.

SKILL BUILDER

Using the control structures to program a micro:bit

In this Skill builder you will be giving life to a micro:bit microprocessor. However, you do not need to have an actual micro:bit to complete this as a realistic simulated one is provided.

You will use each of the control structures to code a micro:bit board using these three computer languages: Blockly, JavaScript and Python.

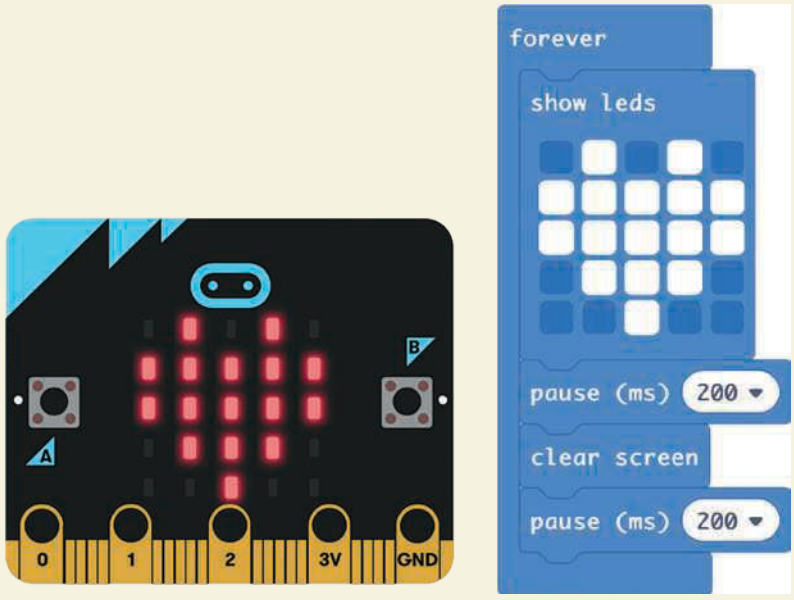
Your goal is to bring your micro:bit 'to life' by lighting up a pattern of its 25 light emitting diodes (LEDs) to create a pulsing heart! Get ready to perform your heart transplant!

Weblink
MakeCode for micro:bit

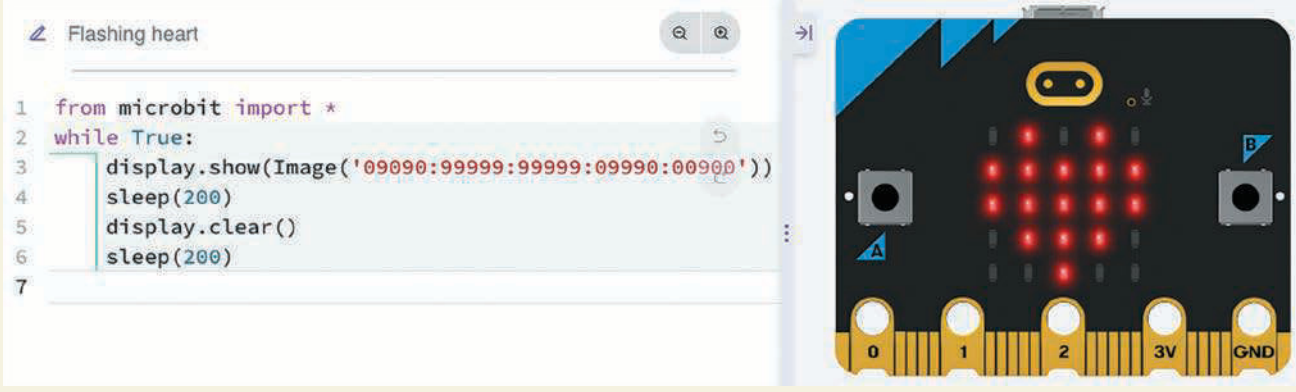
Flashing heart using the sequence structure

Here you create a simple flashing heart, first using visual code and then the general-purpose Python code. The code wraps a sequence structure inside an endless loop to keep the heart flashing.

1. Open the MakeCode website (see weblink) and create a new project.
2. Create the visual code shown in Figure 4.25. Check your code is working correctly by looking at the built-in micro:bit simulator.



Visual Blockly code using the MakeCode for micro:bit Visual Editor



General-purpose code in Python using the micro:bit Python Editor

Figure 4.25 Two computer languages, one visual and the other a text based general purpose language, producing the same result on a micro:bit: a flashing heart.

3 In the MakeCode editor, select the JavaScript tab and inspect its code.

4 Next, visit the micro:bit Python editor (see weblink). We won't use MakeCode's built-in Python editor as it uses a non-standard version of Python. Carefully type the Python code shown in Figure 4.25 using the micro:bit Python editor. Check your code works correctly using the built-in micro:bit simulator then download your code to an actual micro:bit to test it. Instructions on how to connect a real micro:bit and download code to it are given in Chapter 6.

5 Compare the different keywords used by the three languages for *pausing*. What fraction of a second is the pause in each?

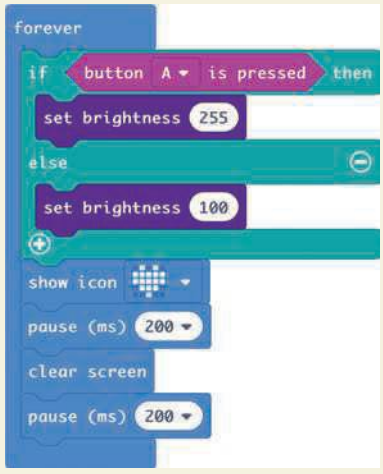
6 Compare the different methods used in the three languages for describing a heart pattern. Can you explain the Python method?

Weblink
micro:bit Python Editor

A pacemaker using the selection structure

Now we use the selection (branching) control structure to make our heart beat stronger when we press button A – as if it was acting as a heart pacemaker.

1 Create the code shown in the Visual Blockly left panel of Figure 4.26. Check your code is working correctly by holding down button A on the built-in micro:bit simulator.



Visual Blockly

```
from microbit import *

while True:
    if button_a.is_pressed():
        display.show(Image('09090:99999:99999:09990:00900'))
    else:
        display.show(Image('02020:22222:22222:02220:00200'))
    sleep(200)
    display.clear()
    sleep(200)
```

Python

Figure 4.26 Pressing button A causes your heart to beat more strongly!

2 In the MakeCode visual Block editor, select the JavaScript tab and inspect its code.

3 Use the micro:bit Python editor to code the MicroPython version. Note that indents must be accurate!

4 Compare the different ways these languages code the *selection control structure*.

5 How does the Python code make the heart fainter?

6 Become an instant heart surgeon! Modify the code to create your own unique heartbeat.

Pulsing heart using a repetition (loop) control structure

You have already been using the loop structure in the previous examples as you needed to keep your heart beating. Here you use the `for` loop (the specialised *counted* loop we introduced in the previous Skill builder) to make your heart's beat far more realistic!

1 Using your MicroPython editor, enter the code shown here. Before running the program describe what you think it will do.

Run the code, see below.

```
from microbit import *

heart = "0x0x0:xxxxx:xxxxx:0xxx0:00x00"

while True:
    for count in range(0, 10, 1):
        display.show(Image(heart.replace('x', str(count))))
        sleep(200)
    for count in range(9, -1, -1):
        display.show(Image(heart.replace('x', str(count))))
        sleep(200)
```

2 We first define an image for a heart with each pixel's brightness described using an 'x' instead of a number. We then use `for` loops introduced earlier in this chapter to change the brightness up then down. Can you describe how this program works from what you learned about the range function in the previous Skill builder?

3 (Spoiler alert!) The range function in Python works like this: `range(1, 10, 2)` counts *up* from 1 to 9 (Note! One less than the value given for 'stop') in steps of 2. If the step value was negative it would count *down* instead.

On your own:

- Modify the code to make the heart beat faster.
- Modify the code so that pressing button A makes the heart smaller. (Hint: define a smaller pattern.)
- Modify the code so that pressing button B makes the heart beat faster.

4.7 Anatomy of a computer program

The three basic control structures used in programming can be compared to the structures we use to write English; sequence, selection and repetition are structures, like the different types of sentences that construct a story. Strict coding rules in computing, called *syntax*, can be compared to the grammar rules of English.

English grammar also has names for the different types of parts of a sentence: nouns, verbs, adverbs, proper nouns, phrases and so on. It is no different for computer programming syntax. Different components of code have names and rules for their use.

All computing programs use *variables*, **assignment statements**, *operators* and ***functions***. We have already seen these in action. Examine the labelled diagram in Figure 4.27.

We have talked about the idea of programs receiving input, performing some processing and providing output.

Our programs used special keywords and built-in *functions* such as `print` and `input`.

Variables are a type of 'container' that holds values while the computer works with them.

We need to be able to compare variables. For this we use *operators*.

We need a way to calculate and assign new values held by these variables. These are called *assignment statements*.

Some blocks of code are used so often in a program we give them a name. This way we can write them just once and call them by name to save rewriting them in full. We call these reusable blocks of code *functions* or *procedures*. We now explain each of these components of basic programming code in more detail.

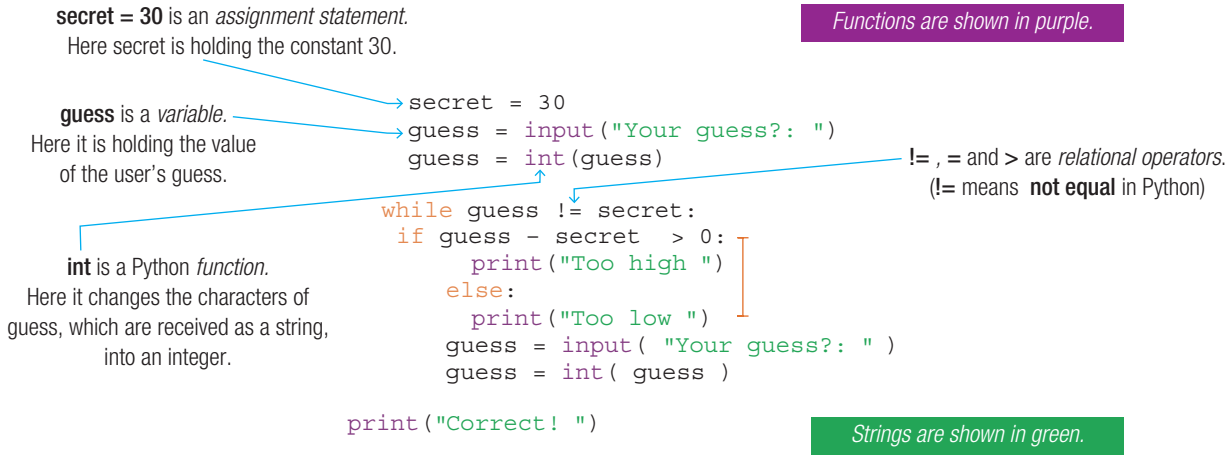


Figure 4.27 Our number guesser code showing variables, constants, assignment statements and functions.

4.8 Primitive data types

Binary data in a computer can be grouped into 8 **bi**nary digi**ts** (‘**bits**’) – just zeros and ones, called *bytes*. These bytes all look the same to the computer (and to us, if we could see them).

Every programming language supports different data types and has rules to tell the computer what it can do with each data type. For example, if data is defined as a string, then mathematical operations are not possible.

The short program in Figure 4.27 contains only two basic types of data: *integers* and *strings*. A computer needs to be told what data type it is using. Figure 4.28 explains in detail how this works.

Most of the data that programs work with can be classified using one of the following data types:

- character data – the building blocks of text
- string data – holds strings of characters, rather like strings of pearls
- integer data – used for whole numbers
- floating point data – used for decimals
- Boolean data – used to represent true or false
- binary data – holds large objects such as movies, sound or graphics.

We will encounter some more complex data structures in Chapter 11.

Next, we will consider some of these primitive data structures.

Characters

Characters are the building blocks of words (strings) of text – for example, letters of the alphabet and the numerals 0 to 9. Many of our programs will need to perform operations on individual characters contained in a string. Character data types exist in different forms in different programming languages.

In Python, single characters are considered to be just another string variable – of length 1.

Characters are represented by a code – either the American Standard Code for Information Interchange (ASCII) or the more extensive Unicode.

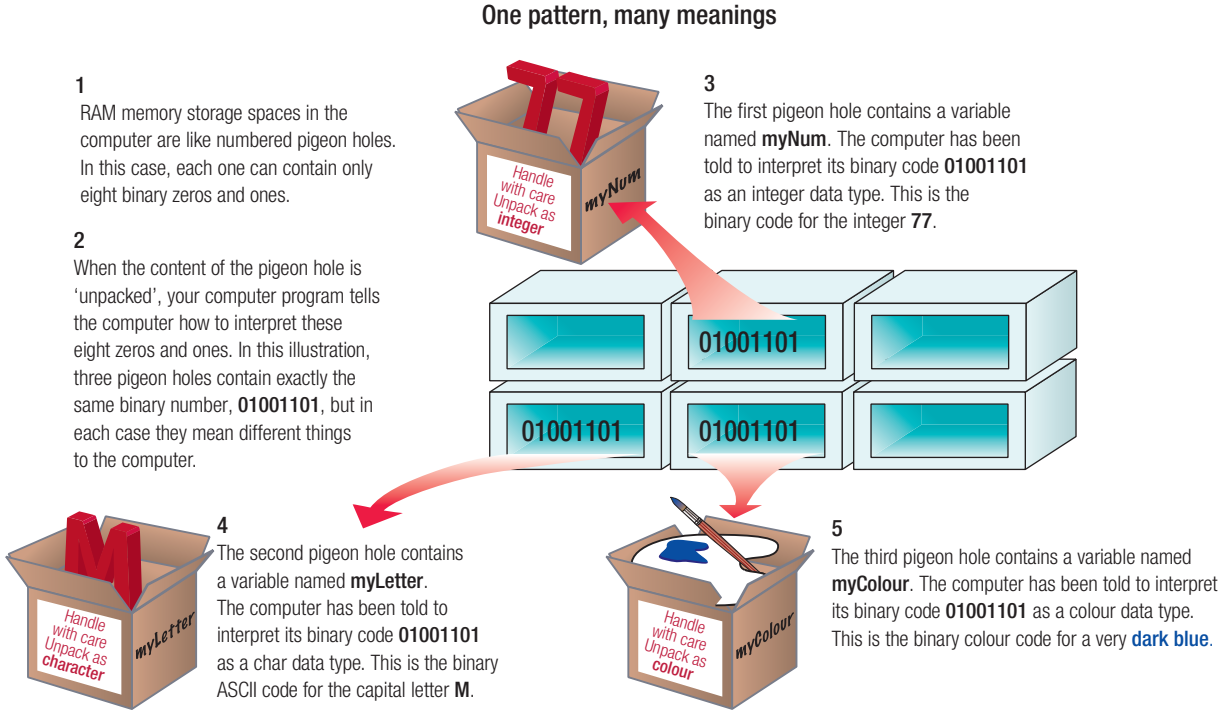


Figure 4.28 Exactly the same pattern of zeros and ones – in this case, 01001101 – can represent very different types of data. Here it can mean either the integer 77, the colour dark blue or the letter M. The computer program needs other information to tell it in which of these ways it should interpret the binary code. This is known as declaring a variable.

Strings

Strings are just sequences of characters: alphabetic, punctuation symbols or numerals. Computer programs use strings to hold data such as names, addresses or telephone numbers. Note that telephone numbers must be stored as string characters, not integers, as they often include spaces or other symbols.

Integers

Integers are whole numbers, either positive, negative or zero. The number of bytes used to represent an integer determines the maximum size of the integer possible. We have already learned that a byte contains eight bits (a 1 or 0). So integers represented by one byte can only hold 2^8, or 256, different integers. Two-byte integers hold 2^{16}, or 65 536, integers; four-byte variables can hold 2^{32}, or 4 294 967 296, different integers; and so on. Eight-byte variables can represent 18 446 744 073 709 600 000 different integers!

Floating point numbers

Real numbers, called floating point numbers in computing, are what we usually call decimals, and may be either positive or negative. There are two common data types supporting real numbers:

- single – meaning single precision, which usually has at least six significant digits after the decimal point
- double – meaning double precision, which usually has at least 10 significant digits after the decimal point.

Python

In Python, a variable is assumed to be a floating point number when it is assigned a value containing a decimal point (or the letter 'e' to indicate a power of 10). Thus, writing

```
myNum = 1.23
```

or

```
myNum = 12.3e5
```

both define `myNum` to be a floating point number.

In Python, the type belonging to a variable can be found using the function `type()`:

```
print(type(myNum))
```

C++

In C++, variable types must be declared. Thus,

```
float myNum = 1.23
```

defines `myNum` to be a number of the type floating point.

In many languages, it is possible to convert between data types using special functions.

Boolean

The Boolean data type, named after mathematician George Boole, is used to represent data that is either true or false. It is the simplest data type of them all. It is often used to test whether a condition is true, which would then allow a loop to be entered in a block of code.

Python

Writing

```
>>> x = True
```

means Python assumes the variable x is Boolean, and from then on can only take the values `True` or `False`. Note that initial capital letters are required for these keywords.

Date and time

Many programming languages have special data types to support dates and times. The date component is an integer representing days since January 1, 1970. The time component represents the fraction of a 24-hour period. For example, 6 a.m. would be represented as 0.250 000, as a quarter of the day has elapsed by that time.

INFOBIT

The Y2K date crisis

In the years and months leading up to the year 2000 (Y2K), analysts started to find that some software systems were showing faults in calculations that involved the year 2000.

Programmers realised that this was because many computer systems stored the year portion of dates using only two digits, in this case 00, and these dates were being interpreted as the year 1900 rather than 2000.

Throughout the world, many believed that computer systems would crash as the year ticked over to 2000. At the new millennium's New Year, fireworks exploded, crowds cheered, almost nothing failed; however, programmers had made a great deal of money throughout the year and the world had spent billions of dollars in disaster prevention.

ACTIVITIES

Identify

1 List the data types and variable names you might use in your programming language to hold the following data:
   - a first name
   - b birth date
   - c URL for a web page
   - d recommended retail price
   - e bank balance
   - f the number of printers in a network
   - g the selling price of a house.

Analyse

2 Think about different ways of representing dates. Invent two ways of representing dates in a computer's memory.

Extend

3 Search the web for examples of how different programming languages store integers and real numbers. Provide language examples for:
   - a integer
   - b long integer
   - c single-precision real numbers
   - d double-precision real numbers.

4.9 Parts of a program

Variables

Variables are like containers used to hold data. An important property of variables is that their values can be changed by a program. They are just containers and not the values themselves. Variables exist in special areas of memory, and each variable is given a name (an *identifier*) so that the program can find it. In Figure 4.27, *guess* is a variable.

Declaring variables

We *declare* variables before using them in order to let the computer know what data type each holds. This sets aside space in the computer's memory in preparation for holding the variable values.

The computer only 'sees' a string of zeros and ones (binary). In our program, *guess* is an integer (a whole number). However, as far as the computer is concerned, this string of zeros and ones might represent an integer, a colour, a sound, a decimal or the letter 'M'! (See Figure 4.28.)

We learned in the previous section that in Python and JavaScript, the data type of a variable is assumed from the value it is first assigned. In Python, the data type does not need to be *declared* (defined). In C++, however, the type must be declared.

If a variable called `MachineName` is given the value 10, Python will assume `MachineName` is storing integers and not characters or decimals.

For this reason, in Figure 4.27 `secret` is assumed to be an integer in the program, because it was first declared to be 30, which is an integer.

Unlike Python, program declaration statements in many other languages must be made deliberately, as in the following example using C++. The `int` keyword indicates that the program should reserve some space for a variable and to expect that it will be in the form of an integer.

```
// declaring variables:
int a, b;
int result;
```

These declaration statements are placed at the very beginning of programs. You will learn more about data types in Chapter 11.

Initialising variables

Most programmers initialise variables before they are used in a program. Initialisation involves setting an actual starting value for a variable. If this is omitted, the program may try to perform an operation for a value that does not exist, even though it might have been told what type of data it is expecting.

Initialisation statements are usually placed immediately after declaration statements in the program.

In Figure 4.27, we initialised `secret` to 30 by using an assignment statement `secret = 30`.

Initialisation might look like this:

```
a = 5
b = 2
```

Constants

As well as variables, programs may also use constants to represent fixed numbers and strings. Like variables, constants can be created in the program and use a reserved area of memory. They are given a name.

A constant cannot be changed once its value has been set in the program. In our Scratch number guessing program, we assigned the value 30 to the constant `secret`. In Python, unlike other languages, there is no special restriction that makes a variable into a constant and prevents it changing. So, in Python it is common practice to indicate a variable is a constant by using capital letters in the code: SECRET.

Operators

Programming languages enable us to perform a variety of operations on data using three types of operators – arithmetic, relational and logical operators.

Arithmetic operators

The main arithmetic operators in Python are:

Table 4.1 Arithmetic operators in Python.

Arithmetic operator	Symbol	Description	Example
Addition	+	Adds two numbers	`5 + 2 = 7`
Subtraction	–	The difference between two numbers	`5 - 2 = 3`
Multiplication	*	The product of two numbers	`5 * 2 = 10`
Division	/	Division of one number by another	`5 / 2 = 2.5`
Exponent	**	First number raised to the power of the second	`5 ** 2 = 25`
Modulus	%	The remainder when two numbers are divided	`5 % 2 = 1`
Whole number division	//	Whole number division without decimal, fraction or remainder	`5 // 2 = 2`

Relational operators

The main relational operators in Python are:

Table 4.2 Relational operators.

Relational operator	Description	Example
`==`	is equal to	`if num == secret:`
`>`	is greater than	`if num > secret:`
`<`	is less than	`if num < secret:`
`!=`	is not equal to	`if num != secret:`
`>=`	is greater than or equal to	`if num >= secret:`
`<=`	is less than or equal to	`if num <= secret:`

Our number guesser code uses != to indicate 'not equal to' (see Figure 4.27).

Note the difference between = and ==. We use = for assignment – that is, to give or assign a value to a variable. We use the double equals symbol == to *test* if values to the left and right are equal. Thus, we could use assignment to write

```
a = 4
b = 5
```

Now, if we write the expression

```
a == b
```

it will answer False as it does *not* mean 'a becomes the value of b', but is asking the question 'Is a equal to b?'.

Logical operators

Logical operators are used to test conditions.

Table 4.3 summarises these.

Table 4.3 Logical operators.

Logical operator	What it means	In Python	Example
`and`	True if *both* are true	`x and y`	`x > 10 and x < 20` would be true if x was 15 but not true if x was 25.
`or`	True if *at least* one is true	`x or y`	`x < 10 or x < 20` would be true if x was 15 but not true if x was 25.
`not`	True only if what follows is false	`not x`	`not(x < 10)` would be true if x was 15 but not true if x was 5.

Note *each* of the following would respond with True.

```
print(5 < 10 and 5 < 20)
print(15 < 10 or 15 < 20)
print(not(15 < 10))
```

KNOWLEDGE BUILDER

AND and OR

Here we will be using your micro:bit simulator in the Python Editor. The equivalent command to `print` when using the micro:bit LED display is `display.scroll` or `display.show`. Try this:

```
display.scroll(5 < 10 and 5 < 20)
```

and note the result.

A statement like this:

```
(SwimmerAge = EventAgeGrouping) AND
(SwimmerGender = EventGenderGroup)
```

will only be true if *both* statements in *each* pair of parentheses are true: the swimmer must be the right age *and* the right gender before being permitted to enter the event.

A statement like this:

```
(DiscountVoucher) OR
(CustomerMonthlyBorrowings > 5)
```

will be true if you have a discount voucher (i.e. it is `True`) *or* if you have borrowed more than five books in the last month, *or* both (note!).

A statement like this:

```
not DiscountVoucher >= 5
```

is logically identical to `DiscountVoucher < 5`.

Weblink
micro:bit Python Editor

Concatenation

This strange-looking word becomes very important when you work with data stored in string and character variables.

Concatenation is a special operator that joins strings of letters together. For example, if you want to compose a small message to a sporting winner congratulating them on their result, you might concatenate a string with winner Jessica Hull's name like this:

```
Message = 'Congratulations to ' + Winner
```

This would result in output like this:

```
Congratulations to Jessica Hull
```

We usually concatenate (link together) strings using either the ampersand symbol (&) or the addition symbol (+), depending on the programming language we are using.

Assignment statements

When data is received by a program procedure, it will usually belong to a variable. From that point, the variable contains the value it has been assigned. At any time that suits us, a variable's value may be changed by program code.

Assignment statements are used in program code to assign a value to a variable (Figure 4.29). In the following example, a variable called Area is assigned a value calculated on the right-hand side of the equals sign.

```
Area = Length * Width
```

Be careful not to confuse assignment with a check on mathematical equality. In coding, think of the '=' as meaning 'becomes'. Assignment means the left-hand side is being changed in value by the expression on the right-hand side. Always think about assignment as taking the value calculated on the right and placing it into the variable on the left.

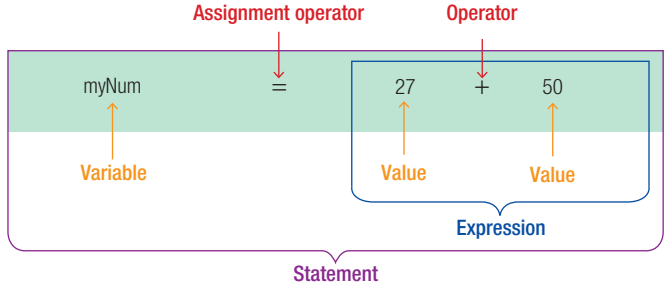


Figure 4.29 An assignment statement. Here our statement begins with a variable that is being changed by an expression. The expression is the sum of two integer values. This sum is assigned to `myNum` using the equal operator `=`.

Here is an example of assigning a specific value to a variable:

```
score = 50
```

We often need to increase a numeric counter variable, usually to keep count of how many times an action has happened. We describe this as incrementing the variable. The code to do this might be:

```
counter = counter + 1
```

This can also be written in shorthand in many languages as:

```
counter += 1
```

The program will execute this line, which adds one to the value in counter, using the following steps:

- read the value from `counter`
- add one to this value
- assign the new value to `counter`.

If `counter` had the value 5 just before this line of code, it would have the value of 6 immediately after the statement has executed.

Some languages also use `counter++` to increment by 1.

Note that when there are multiple calculations in one line of code, the order of operations is the same as in mathematics. Calculations are performed in this order:

- first: bracketed operations and numbers raised to a power, from left to right
- second: multiplication or division, in order, from left to right
- third: addition or subtraction, in order, from left to right.

Functions

Although you will later be naming and writing your own functions, programming languages have their own set of built-in functions, which saves us rewriting code every time we need to do the same thing. This also helps to collect related code in the one place. It is helpful to know these functions for your coding language.

You will be learning to write your own functions in Chapter 11. Here we look at a few of Python's built-in functions.

Functions are usually written using the function's name followed by parentheses – for example, `MyFunctionName()`. The parentheses indicate the function may receive some input value(s). In computing, as in mathematics, functions usually produce a single value of some type; in computing, functions often do not produce a value, but instead simply perform some action.

Our number guesser code shows built-in functions, such as `input`, `int` and `print`, by using purple code (see Figure 4.27).

Input and output functions

Functions are available to receive data input. To receive keyboard input from a user, Python provides the `input` function:

```
birthdate = input('Please enter your
  birthday in form dd/mm/yyyy')
```

To provide output on a screen, Python provides the `output` function:

```
print('Your birthdate is ' + birthDate)
```

Mathematical functions

Most languages have a variety of special mathematical functions for such things as:

- trigonometry (e.g. sine, cosine and tangent)
- random number generation
- logarithmic calculations
- squares and square roots
- rounding numbers
- finding statistical values (e.g. mean or average).

Functions that round numbers

The following examples show functions that round numbers.

Consider the following calculation:

```
AverageAge = SumOfAges / NumberOfPeople
```

Let us assume the variable `SumOfAges` contains a value assigned earlier in the program – the total of all of the ages for a group of people. The variable `NumberOfPeople` contains the number of people in the group. When this division takes place, it is unlikely to result in an integer, perhaps a value like 27.67.

In Python, the `round()` function rounds off to a given number of digits and if no number of digits is provided, it rounds to the nearest integer.

```
AverageAge = round(SumOfAges /
  NumberOfPeople)
```

would produce the value 28, and

```
AverageAge = round((SumOfAges /
  NumberOfPeople),1)
```

would produce the value 27.7. Note that this code must be typed on a single line.

In Python, the `trunc` function, part of Python's maths library of functions – chops off the decimal fraction part of the number and returns only the integer part.

```
AverageAge = trunc(SumOfAges /
  NumberOfPeople)
```

would produce the value 27.

String-handling functions

Special functions called methods are designed to work with certain objects.

In Python, string objects are defined as a class. You will learn more about classes in Chapter 12, which discusses object-oriented programming (OOP). Functions that are part of the definition of a class are called *methods*.

Table 4.4 shows examples of string methods you will find useful in Python, but they have equivalents in most languages. Notice methods are written using dot notation – that is, following the object name and separated by a dot. The method operates *on* the object *in front of it*.

Other functions

The built-in functions we have considered so far are only a few of the very large number of functions available, apart from ones you will later write yourself. Programming languages provide documentation to assist you to locate their built-in functions to use in your programs.

Table 4.4 Examples of string methods in Python. Note that these would need to be output to a print statement to view the result.

Method	Output	Example given `myText = 'Programming'`
`lower()`	Converts letters in a string to lowercase	`myText.lower()` results in `programming`
`upper()`	Converts letters in a string to uppercase	`myText.upper()` results in `PROGRAMMING`
`replace(old,new)`	Searches and replaces parts of strings	`myText.replace('ram','ewe')` results in `Progeweming`

SKILL BUILDER

Improving the number guessing game

The number guessing game introduced in section 4.4 is often given as a first programming exercise because it uses so many important features of programming: variables, user input, output, counters, loops, arithmetic and logical operators and functions. It is a great way to learn about coding.

There are two other chapters focusing on programming where you will undertake more challenging programming (see Chapters 11 and 12). However, as this is an introductory chapter, the following Skill builder is gentler!

Your task is to change the game so that instead of responding with `Too high` or `Too low`, it responds with `HOT, WARM, COLD` and `FREEZING`.

Read each stage carefully and use the flowchart and IPO table on the next page to assist you in making these, and other, improvements to the basic game.

Computational thinking

Your task

The player (user) has to guess a secret number chosen by the computer. The computer will respond each time with one of five words indicating how close the guess is, as in the party game of blindfold.

We will use a special function in Python to generate a secret number:

```
secret = random.randint(1,100)
```

Absolute value function

1 We need a way to test closeness of a player's guess to the secret number.

We could use (`guess - secret`) to measure this gap, but this will be negative when the guess is less than the number.

Python has a module called `abs()` that gives the absolute value or size of a number only. It is exactly what we need! It looks like this:

`abs(x)` returns the absolute value of a number, without a positive or negative sign.

Decide the words you want to use for guess ranges. Here are suggestions:

- **Hot** within 5 of the secret number
- **Warm** within 20 of the secret number
- **Cold** within 40 of the secret number
- **Freezing** greater than 40 away from the secret number
- **Correct!** exactly equal to the secret number.

Flowchart

2 Roughly sketch a flowchart outlining the logic you will use before reading further.

3 Look at Figure 4.30. Can you explain why the tests have been placed in this order? Could you test the big gap of 40 first?

Skill builder resource Improving the number guessing game

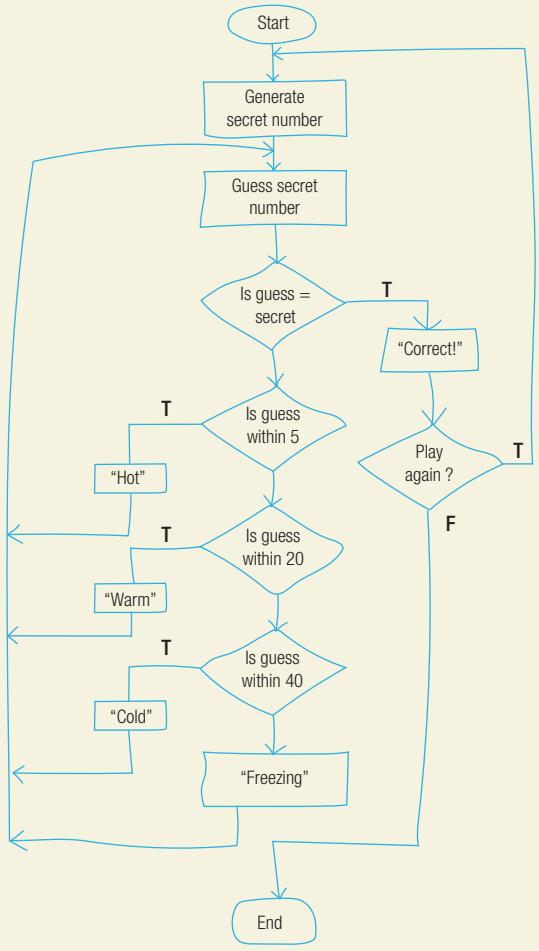


Figure 4.30 Flowchart sketch for an improved number guessing game.

Designing possible solutions

4 Develop structured English for the algorithm represented by the flowchart. The act of coding is made much simpler when the logic has already been carefully thought out.

5 Table 4.5 summarises input, processing and output for our game.

Choice of language

We will use Python for our solution, but the code can be easily adapted to other languages.

Table 4.5 IPO table for the improved number guessing game.

Input
• Our guesses
Processes
• Generate a secret random number • Compare secret number and guess • Count the number of guesses the player makes • Ask if the player wants to play again
Output
• Display the appropriate message: 'Freezing', 'Cold', 'Warm', 'Hot' or 'Correct!' • Display the number of turns

Producing and implementing

6 We need to give different messages depending on the user's guess.

First, write out a version of your algorithm in structured English. Before reading further, complete your algorithm.

We will carefully analyse this algorithm before commencing to code it. Recall earlier in the chapter we stated that designing an algorithm takes more time than coding in the case of experienced programmers.

```
BEGIN high low guessing game version 3
   set playAgain to true
   print 'I'm thinking of a number between 1 and 100'
   WHILE playAgain is true
      set counter to 0
      set guess to -1
      generate secret as a random number between 1 and 100
      WHILE guess is not equal to secret
         print 'What is your guess?'
         ask player for their guess
         increase counter by 1
         IF absolute value of guess-secret is 0:
            print 'Correct!'
            print 'You took ' + counter + ' guesses.'
         ELSEIF absolute value of guess-secret <= 5:
            print 'HOT'
         ELSEIF absolute value of guess-secret <= 20:
            print 'WARM'
         ELSEIF absolute value of guess-secret <= 40:
            print 'COLD'
         ELSE:
            print 'FREEZING'
      ENDWHILE
      print 'Would you like to play again?'
      get playAgain response
   ENDWHILE
   print 'Thanks for playing! '
END
```

Repeating guesses

7 To allow users to keep playing until they guess correctly, we use a `while` loop that is entered only if `secret` and `guess` are *not equal*. If they *are* equal the program loop will not be entered and the game ends. We also keep track of the number of guesses by using a variable called `counter`.

8 The player is not asked for their guess until they are inside the `while` loop, but a value for it is needed to allow it to be entered the first time. We therefore assign a dummy value for `guess` of, say, –1 before we ask the player for one. This is a common technique in coding. We also initialise our values for `counter` and `secret`.

Play again?

9 We want players to be able to play again without quitting and restarting. To do this, we wrap a second `while` loop around the inner one that is entered whenever the variable `playAgain` has the value 'Y'. Players only quit if they answer any other way. This creates two problems!

First, when the game is played for the first time, `playAgain` has no value yet, so the `while` loop is never entered. We must initialise `playAgain` to 'Y' or True. You can see this solution at the start of the structured English version.

10 The second problem is how to handle different ways users will respond to the question. Players may say 'Yes' or 'yes', which are not the same to a computer! We need to list the many ways players might say 'yes'.

11 Translate this algorithm into working code and get it running.

Testing and evaluating

12 Test your version thoroughly. In Chapter 11, you will learn more about testing techniques.

13 Once you have it working, suggest at least five improvements.

14 Ask people outside your class, such as family members, to suggest improvements. Try to perfect your code.

On your own

15 We have provided first and second Python versions of the game on MindTap.

However, your project is not yet complete. It's time to try some improvements on your own.

Here are some suggestions:

- What happens if the player types in a number outside the range 1 to 100?
- What happens if the player types in a word instead of a number?
- Improve 'play again' by writing it as a function.
- Inform players if they are below or above the secret number.
- Allow players to set a range for their secret number.
- Add more choices to the 'Freezing', 'Cold', 'Warm', 'Hot' section.
- Allow players a personal challenge of setting themselves a maximum number of guesses.
- Store highest score.

Can you suggest other features?

ACTIVITIES

In these activities, use the programming language you are using in class.

Identify

1 Identify common symbols used for 'not equal to'.

2 Give three examples of arithmetic operators.

3 Give three examples of relational operators.

4 Give three examples of logical operators.

5 What is concatenation? Give an example using strings.

6 Use pseudocode to give an example of an assignment statement.

7 What is the purpose of a counter?

8 What is the purpose of a function?

9 Give an example of an input function.

10 Give an example of an output function.

Analyse

11 Explain what the following line of code does:

```
SalePrice = RRP - (RRP * DiscountRate)
```

12 Explain what the following line of code does:

```
TimeTaken = TimeFinished - TimeStarted
```

Extend

13 Find out about five useful built-in functions used in your programming language that have not been covered in this chapter, and prepare a short table summarising how these functions can be used.

4.10 Python 3 reference table

Learning to code

There are many online resources to teach you to code. Here we will suggest a few excellent ones, but no matter which you choose, the best way to learn is to just start coding. As with learning any spoken language, coding is best mastered by practice.

Weblinks
Grok Academy
W3Schools
code.org

Grok Academy

Each year over 20 000 students participate in Grok's NCSS programming challenge. As well, the Grok website provides free tutorial modules for all ages on a wide variety of languages. We recommend you download the helpful free 'cheatsheet' for Python 3 available through Grok's website.

code.org

Over 65 million students have learned to code either during the annual Hour of Code or using the tutorials on code.org.

W3Schools

W3Schools has been around for a long time and offers interactive tutorials for a wide range of languages.

9780170442787

Python 3 reference table

How to	Python code example
Input	
Input `input` keyword containing a message to user. Here input is assigned to the variable called `name`	`name = input()`
Input with message Use `input` keyword containing a message to user. Here input is assigned to the variable called `name`	`name = input('What is your name?')`

Input string converted to integer
Convert a string to an integer and assign it to the variable called `mynum`
	`num = input('Enter a number: ')`
	`mynum = int(num)`
	or
	`mynum = int(input('Enter a number: '))`

Output	
Output a string Use `print` keyword to print a message and use single or double quotes (‘ or “) around strings	`print('Some text')`
Output both message and variable value Separate items using commas	`print ('Your number is', num)`
Output multiple variable values Separate items using commas	`print(a, b, c)`
Control structures	

Selection
`if` statement
with `elif` and `else` added when required.
NB: Double equal signs needed to check equality
	`if num == secret:`
	`  print('Well done')`
	`elif num < secret:`
	`  print('too low')`
	`else:`
	`  print('too high')`

Repetition `while` loop	`while password != 'python':` `  password = input('Enter your password: ')`
Range function	`range(10) # NOTE!: counts from 0 up to, but not including, 10` `range(0, 10, 2)  # counts from 0 to 9 in steps of 2`

Counted repetition
`for` loop
	`for n in range(0,10):`
	`  print(n)`
	`  print('python is cool')`

Other	
Number of items in an object This inbuilt function returns the number of items in an object. This expression would return the value 7.	`len('lengthy')`
Import modules Some functions such as `random` need to be imported before they can be used. This one generates a random integer 1 to 100 (inclusive).	`import random` `num = random.randint(1,100)`

Writing functions
Use the keyword `def` followed by function name, parentheses and colon then indent all code in this function. `return` keyword delivers result.
NB: Double asterisk for powers (`r` is squared here)
	`def circleArea(r):`
	`  area = 3.14*r**2`
	`  return area`

Calling functions This example calls the function `circleArea` (see above) and passes it the value of radius `r = 4.` The area is calculated to be 50.24. A print statement could then be used to view this value.	`area = circleArea(4)`

Weblinks:

- The Hello World Collection
- Scratch
- Python IDLE
- Replit
- Blockly Maze challenge
- MakeCode for micro:bit
- micro:bit Python Editor
- Grok Academy
- W3Schools
- code.org

Skill builder resource

- Improving the number guessing game

Chapter review quiz

Nelson MindTap

To access resources above, visit
cengage.com.au/nelsonmindtap

5 Modelling social networks

FOCUS AREA: Modelling networks and social connections

Chapter outcomes

In this chapter, you will learn:

- about the purpose and components of social networks, including entities, relationships and events
- the basics of graph theory and how it applies to the structure and analysis of social networks
- to identify and analyse real-world examples of social networks, ranging from online platforms to kinship networks
- how to work with social network data, including methods of collection, storage and analysis using graph databases and ethical considerations
- about social, ethical and legal considerations when participating in and developing social networks.

Key terms

accessibility
centrality
clustering
cybersecurity
degree
edge
entity
event
graph theory
kinship
node
personally identifiable information (PII)
relationship
social network
user agreements

9780170442787

IN ACTION

Buy Nothing Project

The Buy Nothing Project is a decentralised network of local gift economies, where members of the community share goods and services without using money (Figure 5.1). The network operates through individual Facebook groups that are organised geographically. Facebook groups are created and managed by a small number of volunteer administrators who guide the community on how to use the platform, including protocols on what and how to gift and ask.

The main processes involved in the functioning of the system are:

- Joining a group: Individuals can join their local Buy Nothing Group by finding the group on Facebook and requesting to join. Group membership is typically limited to people living within a certain geographic area, such as a specific suburb or city.
- Giving and receiving: Members of the group can post items or services that they want to give away or items or services that they are looking for. These posts can include anything from furniture and clothing to food and home repair services.
- Requests, offers and responses: When a member posts a request or an offer, other members of the group can respond if they have or need that item or can offer the requested service. The member who posted the request or offer can then choose who they want to receive the item or service from or gift it to.

Systems thinking

Key principles of a Buy Nothing Group:

- No buying or selling: One of the key principles of the Buy Nothing Project is that no money is exchanged. Members give and receive goods and services freely, without any expectation of payment.
- Building community: In addition to facilitating the exchange of goods and services, the Buy Nothing Project also helps build community by encouraging members to interact with one another and get to know their neighbours.
- The Buy Nothing Project is based on principles of sustainability, generosity and community building. By creating a local gift economy, the network encourages people to reduce their consumption and waste, while also building relationships and connections within their community.

The first Buy Nothing Group was established on Facebook in 2013 for a community in the state of Washington, USA. Since then, it has spread to 50 countries with over 500 Facebook groups in Australia. In 2021 a new dedicated platform was released for the Buy Nothing Project (Figure 5.2). This app functions in much the same way as the Facebook groups, but has the structure of how the network works built into it. However, in most communities where Buy Nothing Facebook groups were already well established, the app has not been widely used. Administrators and members of these groups attribute this to the disconnect from the sense of community and friendships already established on Facebook in these geographical areas. Positive reviews of the app mention how users can set their own community boundaries and that they don't need to join Facebook, and believe that once the word spreads about the app and more people join the network it will become a great tool for sharing.

Figure 5.1 The Buy Nothing Project network functions on the Facebook platform by users following protocols on how to gift, express interest and ask. Other functionality built into Facebook can also be used, such as polls.

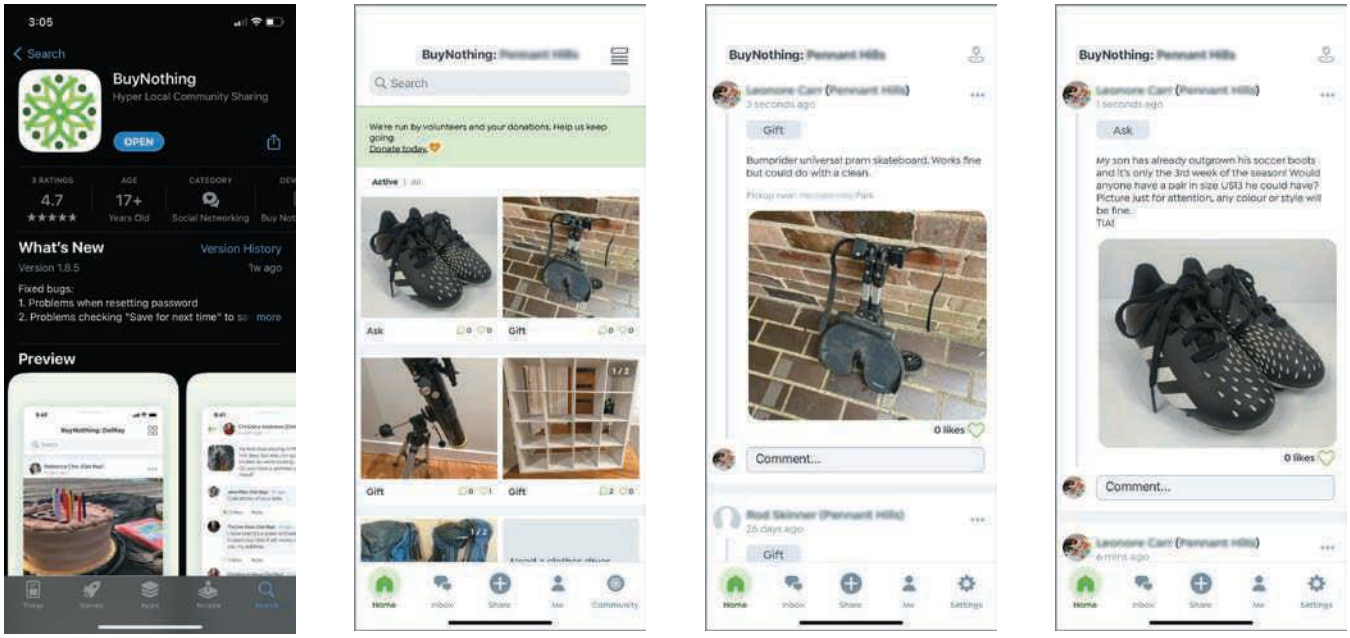


Figure 5.2 The Buy Nothing Project app is available for free download in the App Store and Google Play. Functionality previously defined by Facebook posting protocols has been coded into the app function, including viewing the community posts, and creating 'gifting' and 'asking' posts.

Questions

1. What is the purpose of the Buy Nothing Project network?
2. What is a decentralised network? How is the Buy Nothing Project decentralised?
3. In what ways did this digital social network promote physical social networks?
4. What were the benefits of using an existing social media platform as the base for this gift economy network?
5. Why do you think a dedicated app was developed to serve as the platform for the Buy Nothing Group, instead of Facebook?

5.1 Introduction to social networks

A network is a collection of interconnected entities or nodes that are linked together by some form of relationship or interaction. In this chapter we will be looking at **social networks**, where people are the main **entity**. We will explore non-computer examples, **graph theory** (the mathematics behind analysing the properties of networks), the information system features of digital social networks and how these have evolutionalised how social relationships are formed and maintained (Figure 5.3). Computer networks, including their design based on graph theory, will be explored in Chapter 6.

What is a social network?

A social network is a structure that allows people to connect and interact with each other. In this era, the term social network is almost exclusively used to describe the digital platforms that use the Internet and web to facilitate these connections. Examples of popular social networks include Facebook, Twitter, Instagram, Snapchat, YouTube, Reddit, LinkedIn and Discord. Social networks have become an integral part of everyday life for many people, providing an easy way to connect and communicate with friends, family and acquaintances. They also provide a platform to share photos and videos, as well as a way to stay up to date on the latest news and events. Social networks can be a great tool for staying in touch and staying connected, but they can also be a source of distraction and negative emotions if used improperly. It is important to use social networks in a responsible way, and to make sure to take breaks from them, especially if they are causing stress or anxiety. Being conscious of the personal data you are sharing online and how this may be used or misused is also a key step in staying safe online.

To get a true understanding of how a social network functions, both digitally and in the more physical form, we need to look at the main components: entities, relationships and events.

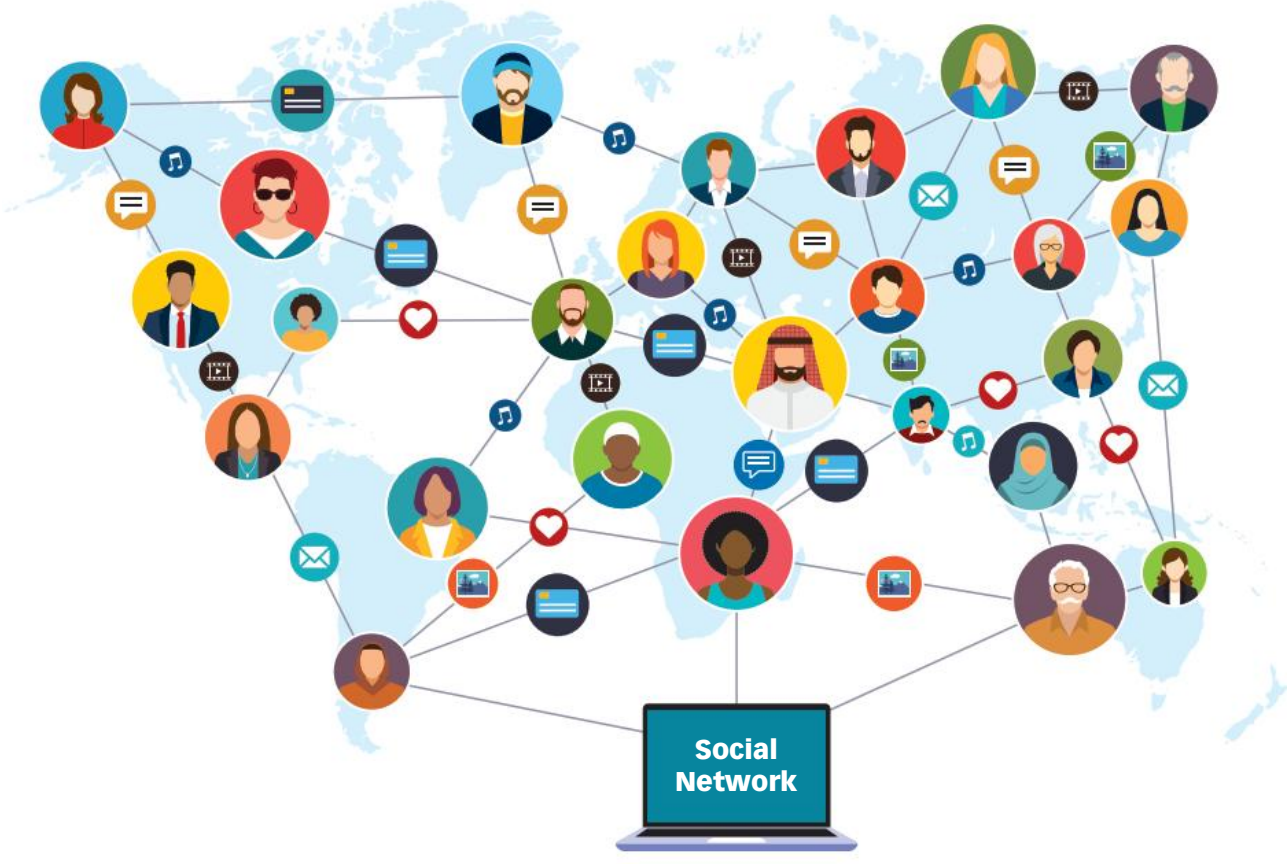


Figure 5.3 Digital social networks are used by people all over the world to communicate, share media, learn and keep up to date, make transactions, and to be entertained. How did people do this before the Internet? Does this make digital social networks better than physical social networks?

Entities

An entity (or a **node**) in a social network is typically a person, group or organisation that has a presence. In digital social networks entities are typically identified by their username, profile image and profile information. Entities can join groups, post content, comment on posts and interact with other entities within the network. They can also form relationships with other entities and follow or be followed by them. Entities are the fundamental building blocks of a social network and are essential to the overall user experience.

In the Facebook Buy Nothing network, an entity is a person within the community who intends to participate in the gifting economy as well as other related Buy Nothing groups, such as those in neighbouring communities (Figure 5.4).

Relationships

A **relationship** in a social network refers to a connection or association between two or more individuals within the network. This connection can be based on various factors such as shared interests, common affiliations, personal interactions or even online engagement. Relationships can be categorised in many ways, such as friends, family, acquaintances, followers or colleagues, depending on the nature and depth of the connection. In social network analysis, relationships are commonly represented as **edges** or links between nodes or entities in a network, and can be analysed to gain insights into social structures, dynamics and behaviours.

In the Facebook Buy Nothing network there are many relationships, including the existing Facebook friendships of people in the community and the relationship established when two people exchange an item.

Events

In a network, an **event** typically refers to a significant occurrence or activity that takes place within the network. An event can be initiated by an individual or other entity within the network and may involve interactions, changes or updates that affect the network or its members. Examples of events in a network can include a new member joining the network, a group discussion taking place, a message being sent or received, a post being shared, a comment being made, a like or dislike being registered or any other activity that involves engagement or participation within the network.

In the Facebook Buy Nothing network, events include request or offer posts and the expressions of interest comments on the post.

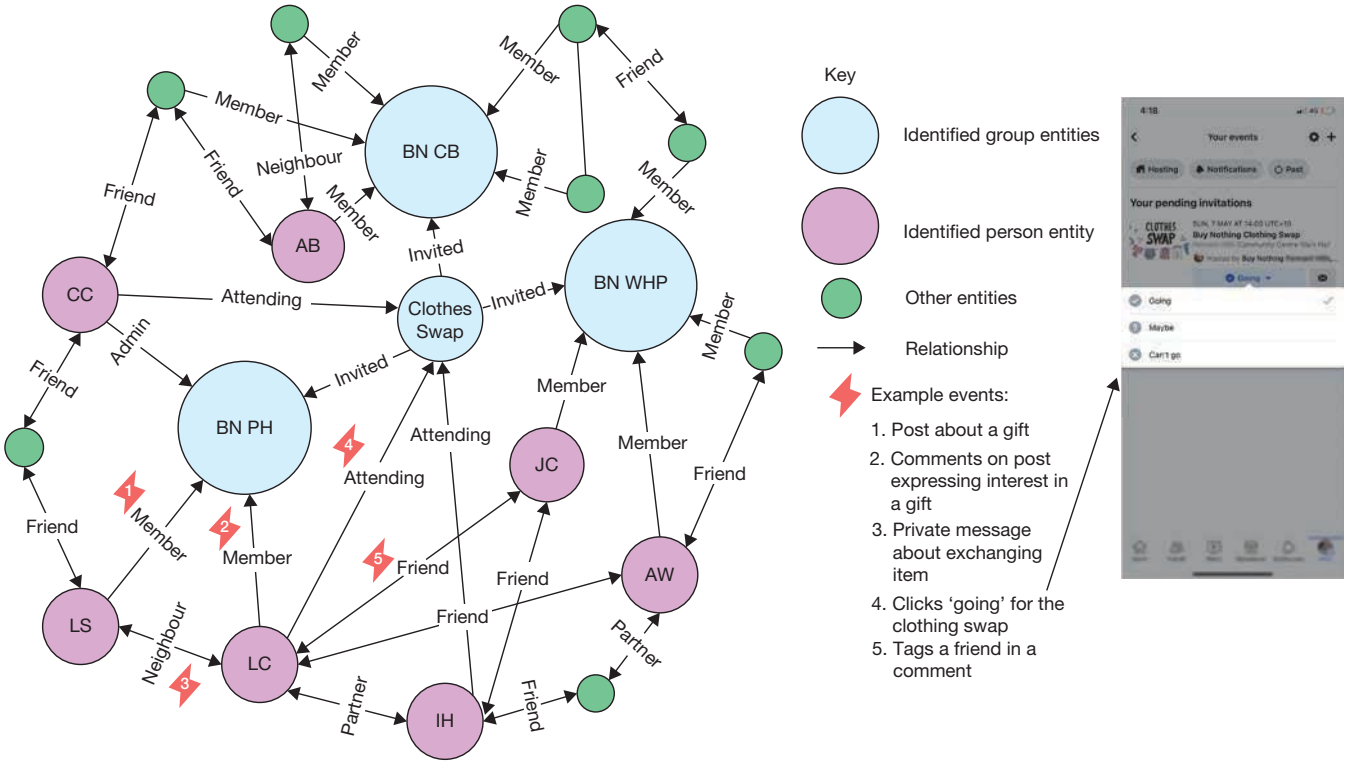


Figure 5.4 Sociogram of a subset of the Buy Nothing Project. The entities are in the circles and include people, groups and a meetup (clothing swap). The lines show the relationship between the entities. An event that could occur on this network is a member accepting an invite to attend the clothing swap.

ACTIVITIES

Identify

1 What are the main components of a social network?
2 How are entities and relationships represented on a network graph?
3 Other than gifting posts, what is another example of an event on the Facebook Buy Nothing network?

Analyse

4 How is a relationship different from an event?
5 How do people communicate on digital social networks? What benefits and issues are associated with this as opposed to physical social networks?

Extend

6 Create a diagram like the one in Figure 5.4 to show the features of a particular social media platform of your choice (e.g. YouTube).
7 Conduct an anonymous survey of your peers' use of social media, finding out the different platforms they use, the extent of their use and what they like and don't like about the network.

5.2 Graph theory

Insights into how social networks function have many uses. The structure of a network can be analysed using graph theory, a branch of mathematics that studies the properties and characteristics of graphs or networks. Some common network metrics include the **degree** of a node (the number of edges connected to a node), the diameter of the network (the shortest path between the two furthest nodes) and the **clustering** coefficient (the degree to which nodes in the network tend to cluster together). Networks can be visualised as graphs, with nodes represented as circles and edges represented as lines.

It started with abstract thinking

Konigsberg

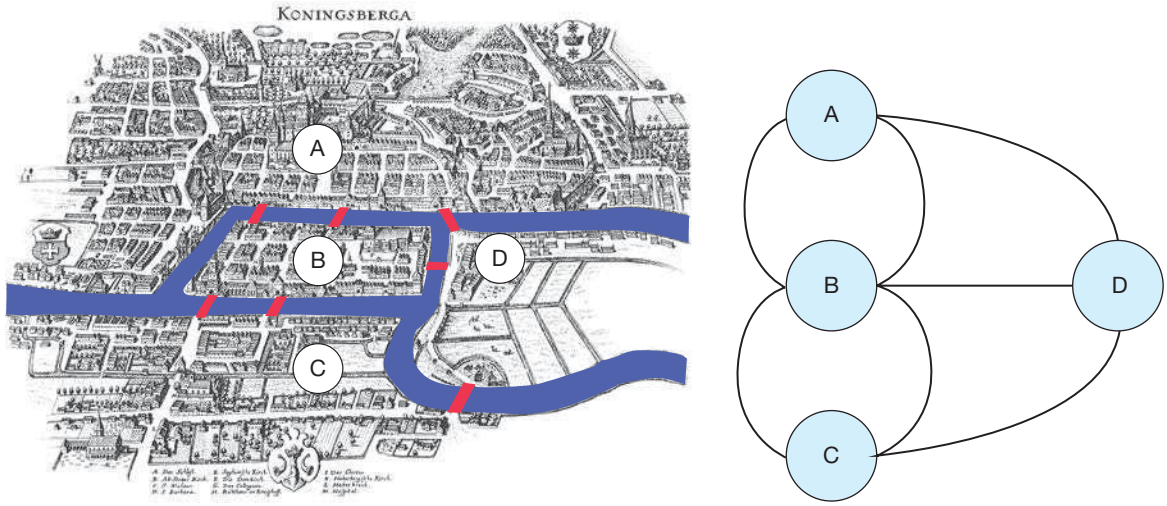
Konigsberg is a city in Russia (now Kaliningrad, Russia) that played a significant role in the history of graph theory. The city is famous for the Seven Bridges of Konigsberg problem, which is considered one of the earliest problems in graph theory.

The problem involved four land masses, separated by the Pregel River, which were connected by seven bridges. The challenge was to find a walk through the city that would cross each bridge exactly once before returning to the starting point (Figure 5.5).

Leonhard Euler, a Swiss mathematician, was the first to solve the problem using the principles of graph theory. He represented the land masses and bridges as nodes and edges in a graph and showed that it was impossible to find such a walk because there were more than two nodes with an odd degree (the number of edges connected to a node). This result led to the concept of Eulerian paths and Eulerian circuits in graph theory, which are paths and circuits that visit every edge exactly once and start and end at the same node, respectively.

The Seven Bridges of Konigsberg problem is considered a landmark problem in graph theory, and played a key role in the development of the field. It demonstrated the power of abstract thinking and the usefulness of representing problems in terms of graphs, which has led to the development of numerous graph-based algorithms and techniques that are widely used today in diverse fields such as logistics, physics and computer science, including the modelling and routing of computer networks.

Computational thinking



Alamy Stock Photo/Album

Figure 5.5 There are four land masses, or nodes, in Konigsberg, labelled A, B, C and D with the seven bridges (edges) connecting them. This can be represented as a graph such as the one on the right. Nodes A, C and D have a degree of three, as they have three edges connected to them while Node B has a degree of five with five edges connected to it. As more than two nodes have an odd degree, there is no Eulerian path in this graph.

The travelling salesman problem

The travelling salesman problem (TSP) is another key problem in computer science and mathematics. It involves finding the shortest possible route that visits a set of cities and returns to the starting city, such that each city is visited exactly once. In other words, imagine a salesman who wants to visit a number of cities to sell his products, and he wants to find the shortest route that takes him through each city once before returning to his starting point (Figure 5.6).

The problem is commonly represented as a graph, where each city is a node and the distances between the cities are edges. The TSP asks for a path that visits each node exactly once and has the shortest total length.

KNOWLEDGE BUILDER

TSP simulation

This activity may be completed physically using a corkboard, six pins, string, a tape measure and a printed map of your school location. Otherwise, it may be completed digitally using Google Maps.

Simulation option 1

1. Fix a printed map of the area around your school (~2–20 km wide depending on how populated your area is) to a display board.
2. Place the first pin at your school and tie a piece of string to the pin. This will be pin A.
3. Select five other significant locations around your school and place pins in these locations. Label these B–F.
4. Use the string to 'visit' each of the pins before returning to the school pin.
5. Measure and note down the length of the string and the order of the pins.
6. Repeat this process a few more times, trying to reduce the length of string required to visit each pin once and return to school.

Shutterstock.com/LightField Studios

Figure 5.6 A physical simulation of the TSP provides a simple demonstration but there are issues with inaccuracy as roads rarely travel in straight lines like the string.

Simulation option 2

1. Go to Google Maps and search for your school (Figure 5.7).
2. Click the 'Directions' button and drag your school up to the starting position.
3. Add a destination in your surrounding area (e.g. a park or shopping centre). Remove the 'Your location' destination.
4. Add four more destinations around your school.
5. Note down the time and distance the journey in the current order would take.
6. Drag to rearrange the five destinations to see if you can reduce the time and/or distance.

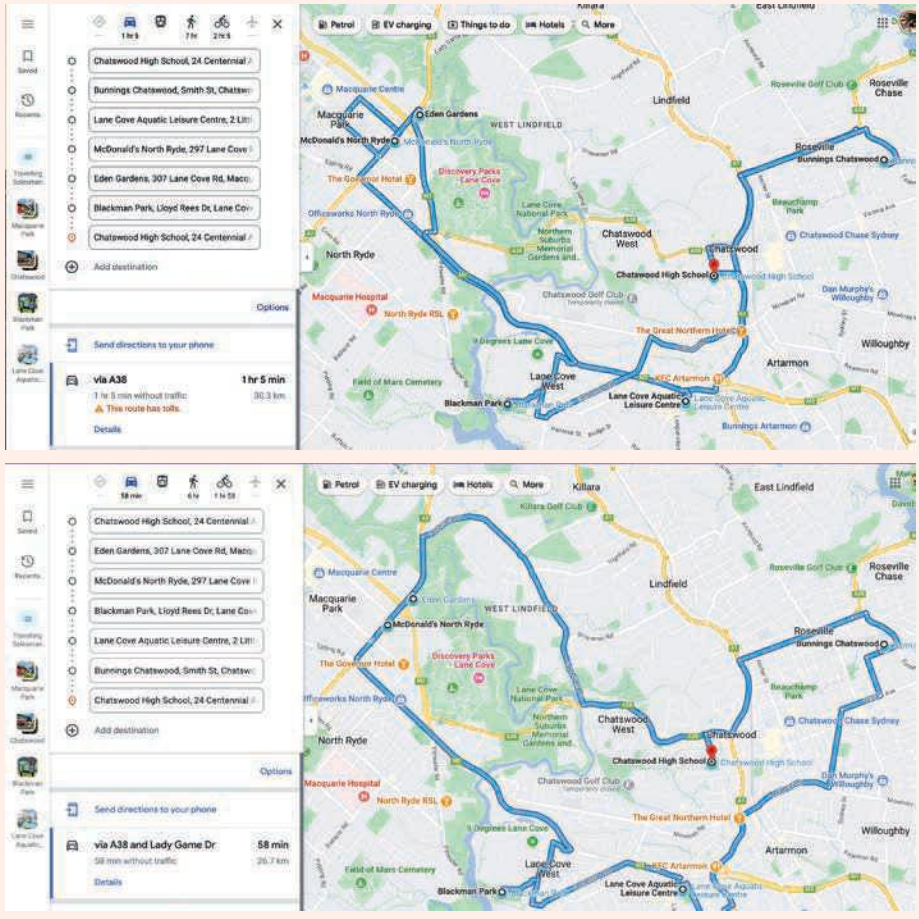


Figure 5.7 The shortest route to visit all destinations and return to school will take 58 mins and is 26.7 km long.

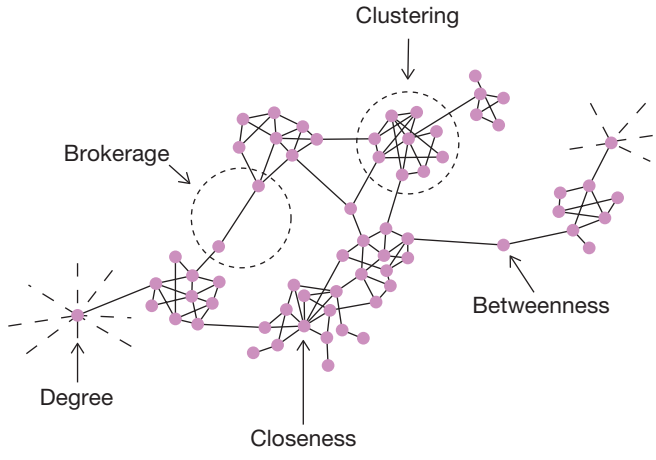
Graph theory and social networks

Graph theory is widely used to understand social networks. This includes network structure, community detection, influence and spread, network dynamics and network visualisation.

Network structure

Graph theory is used to analyse the structure of social networks, including the number of nodes, edges and the connectivity between them.

Common metrics used to analyse social network structure include degree **centrality** (the number of connections a node has), betweenness centrality (the degree to which a node lies on the shortest path between other nodes), closeness centrality (the average of the inverse of distance to all other nodes) and clustering coefficient (the degree to which nodes in a network tend to cluster together) (Figure 5.8).



Source: KnowledgeBrief

Figure 5.8 Network analysis metrics.

Community detection

Graph theory is used to identify communities or clusters of nodes within social networks. Community detection algorithms aim to identify groups of nodes that are densely connected within the group but sparsely connected outside of the group.

Influence and spread

Graph theory is used to analyse the influence of individuals or groups in social networks, and how information or behaviours spread within the network. Centrality measures, such as betweenness centrality, are used to identify influential individuals, while diffusion models are used to simulate how information or behaviours spread through the network.

Network dynamics

Graph theory is used to study the evolution and dynamics of social networks over time. Dynamic network models are used to model the growth and change of social networks, while longitudinal studies are used to analyse how social networks change over time.

Network visualisation

Graph theory is used to visualise social networks in a way that makes them easy to understand. Network visualisation tools, such as Gephi and Cytoscape, allow researchers to visualise and interact with social network data in a meaningful way. Sociograms are used to visualise non-computer social networks (Figure 5.9).

Overall, graph theory provides a powerful framework for analysing social networks, helping researchers to understand the structure, dynamics and behaviour of these complex systems.

Sociograms

A sociogram is a graphical representation of the social relationships among a group of individuals. Sociograms represent social networks by using symbols to represent individuals and lines or arrows to represent the relationships between them (Figure 5.9).

Sociograms can be used to study a variety of social phenomena, such as social influence, group dynamics and communication patterns. For example, a sociogram of a classroom might show who interacts with whom, who is most popular or well-liked, who is isolated or excluded from the group or other social dynamics that can affect the learning environment.

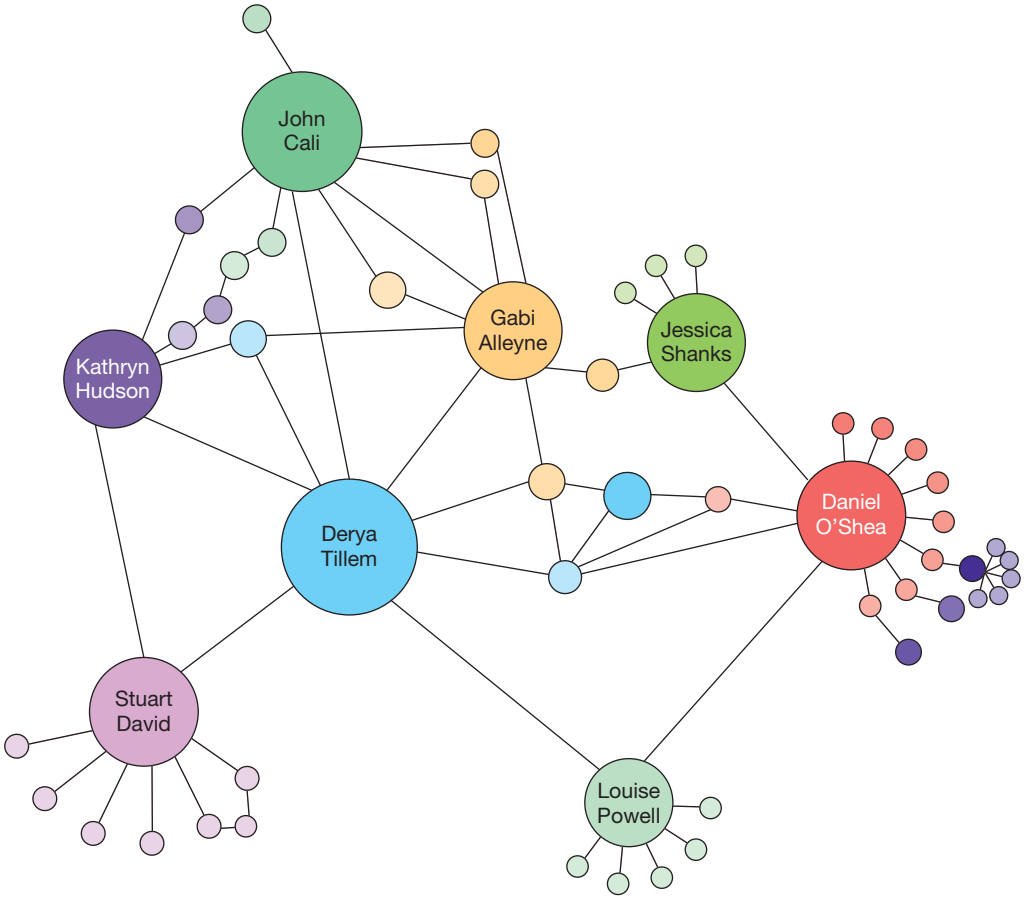


Figure 5.9 In this social network visualisation, circles represent the entities or individuals and lines show relationships between them. Circle size is also used to show entities with lots of relationships or a high degree.

When analysing non-digital social networks, sociograms can be created through a variety of methods such as surveys, interviews and observation. Researchers typically ask individuals to identify their social connections or relationships with other group members, and then use this information to construct a graphical representation of the social network.

Sociograms can provide a useful visual representation of social relationships, allowing researchers to identify patterns and trends that might not be immediately obvious through other methods. They can also be used to communicate social dynamics to a wider audience, such as teachers, managers or policymakers, who might use the information to make decisions about group management or intervention strategies.

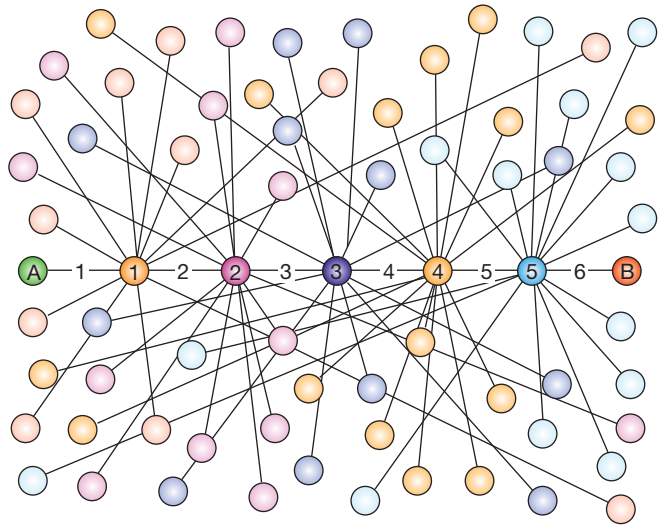
Six degrees of separation

The six degrees of separation theory, also known as the small world phenomenon, suggests that any two individuals in the world are connected by a chain of no more than six acquaintances (Figure 5.10). Graph theory provides a mathematical framework to explain this phenomenon.

The degree of a node in a graph is the number of edges that connect it to other nodes. So, in the context of social networks, the degree of an individual is the number of acquaintances (or significant connections to other individuals). In a typical social network graph, most nodes have a relatively low degree, representing individuals who have only a few connections to others. However, a few nodes have a very high degree, representing individuals who have many connections to others.

The small world phenomenon arises when there are a few nodes in the graph that have a high degree, enabling them to act as hubs that connect many other nodes. These hubs form shortcuts that significantly reduce the distance between any two nodes in the network. The distance between two nodes refers to the minimum number of edges that must be traversed to go from one node to another in a graph. This is also known as the shortest path between two nodes.

Graph theory looks at the average distance between any two nodes in a graph and shows the larger the graph, the smaller the average distance between nodes. This means that as social networks grow larger, the average distance between any two individuals decreases, making it easier to connect with others and reducing the number of degrees of separation. In other words, the structure of social networks naturally leads to the formation of hubs that connect many individuals, reducing the average distance between any two people in the network. As you will see in the next chapter, efficient computer network design and routing aims to use switches or routers to minimise the distance and number of 'hops' data needs to travel between two computer nodes.



Source: Wikimedia Commons/Laurens van Lieshout/CC

Figure 5.10 The six degrees of separation theory shows that any two people can be connected via six relationships or five entities.

KNOWLEDGE BUILDER

The shortest path between unrelated famous people

In this Knowledge builder you will show the *diameter* (shortest path) between two apparently random famous people in the 'worldwide' social network. The first activity gets you to do this manually and the second uses a computer network to demonstrate just how many paths there are between nodes in social networks, particularly ones with a high *degree* like famous people.

Activity 1

1. Each student writes down the name of two famous people (alive today) on Post-it notes (Figure 5.11).
2. All Post-it notes are folded, placed in a hat and mixed up.
3. Students pull two Post-it notes out of the hat.
4. Show the degrees of separation between these two people, using your own knowledge and online sources to find evidence of the 'significant' relationship between each node.

Optional: Redo the activity, this time showing the degrees of separation between you and one of the famous people.

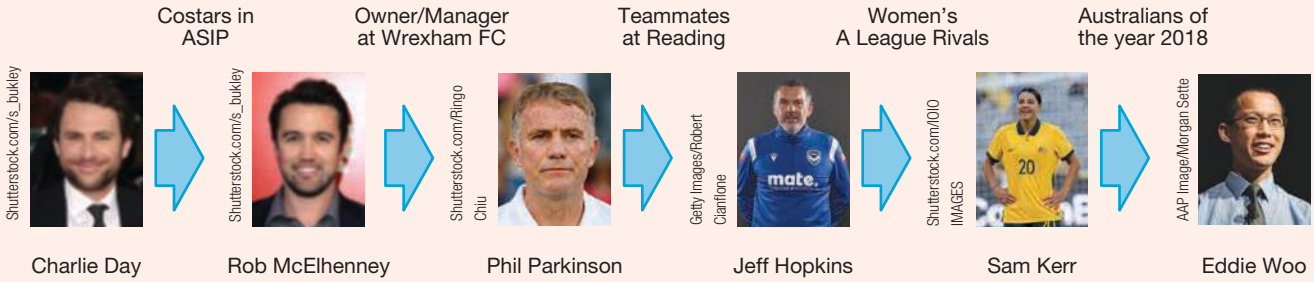


Figure 5.11 The American comedian Charlie Day and the Australian maths teacher and YouTuber Eddie Woo have five degrees of separation as demonstrated by the intermediate nodes and relationships.

Activity 2

1 Now use the Six Degrees of Wikipedia website to see if the distance between these people is actually smaller and how many different paths can be taken.

2 Evaluate the use of this tool in visualising a social network. Does a link in Wikipedia equate to a 'significant acquaintance'?

Weblink
Six Degrees of Wikipedia

SKILL BUILDER

Character sociogram

In this Skill builder, you will create a character sociogram of your favourite television show or book. It will be demonstrated using a TV series you may have watched as a small child, *Paw Patrol*. Aim to have at least a dozen characters included. Label the nodes, relationships and any significant event that is key to the show or book's story. You may use pen and paper or an online diagram tool like diagrams.net to create your sociogram. Using a digital tool gives you the flexibility to rearrange as your sociogram builds, as they are not always easy to visualise in the beginning.

Systems thinking

Alamy Stock Photo/Everett Collection Inc

Figure 5.12 The characters in *Paw Patrol* are connected by a variety of relationships, which can be represented in a sociogram.

1 Place the main character(s) in the middle of the page in a large circle. For example, in the television series *Paw Patrol*, the main character is Ryder (Figure 5.12).

2 Arrange the other characters around the main character with consideration of their relationship with the main character. Use different sized circles to represent their importance within the story. You could also position them in a way that conveys their level of closeness and the quality of their relationship (i.e. closer placement indicates a stronger relationship).

In *Paw Patrol*, the first group of characters we will consider are the *Paw Patrol* pups (Figure 5.13). The leader of the pups is Chase and the other five pups (Marshall, Skye, Rubble, Rocky and Zuma) are friends/teammates with each other, forming a full network of links. Chase will be in a slightly bigger circle than the other pups.

Other side characters to include in slightly smaller circles are:

- Mayor Goodwin and her pet chicken, Chickaletta
- Mayor Humdinger and his pet cats that form the opposing Kitten Catastrophe crew
- Farmer Yumi and her husband Farmer Al
- Mr Porter and his grandson Alex
- Katie, the pups' vet and her pet cat, Cali.

3 Use lines to show the relationship between characters. You may like to style these lines to visually represent the type of relationship. For example, different colours could be used to show family, friends or enemies, or an arrow might show the direction of power in the relationship. In the *Paw Patrol* sociogram, the person-to-pet relationship can be shown as a one-directional arrow. Friendly or positive relationships can be represented in green and conflict relationships in red.

4 In pairs, share your sociograms, including a brief explanation of how you developed them. Evaluate your peers' process and final products. Consider:

- **a** the quality of the data used to build it
- **b** whether it was hand drawn or or created using a software tool
- **c** how the nodes were arranged on the page to help visualise the relationships
- **d** any styling that was used to better show/emphasise relationships and other sociogram measures.

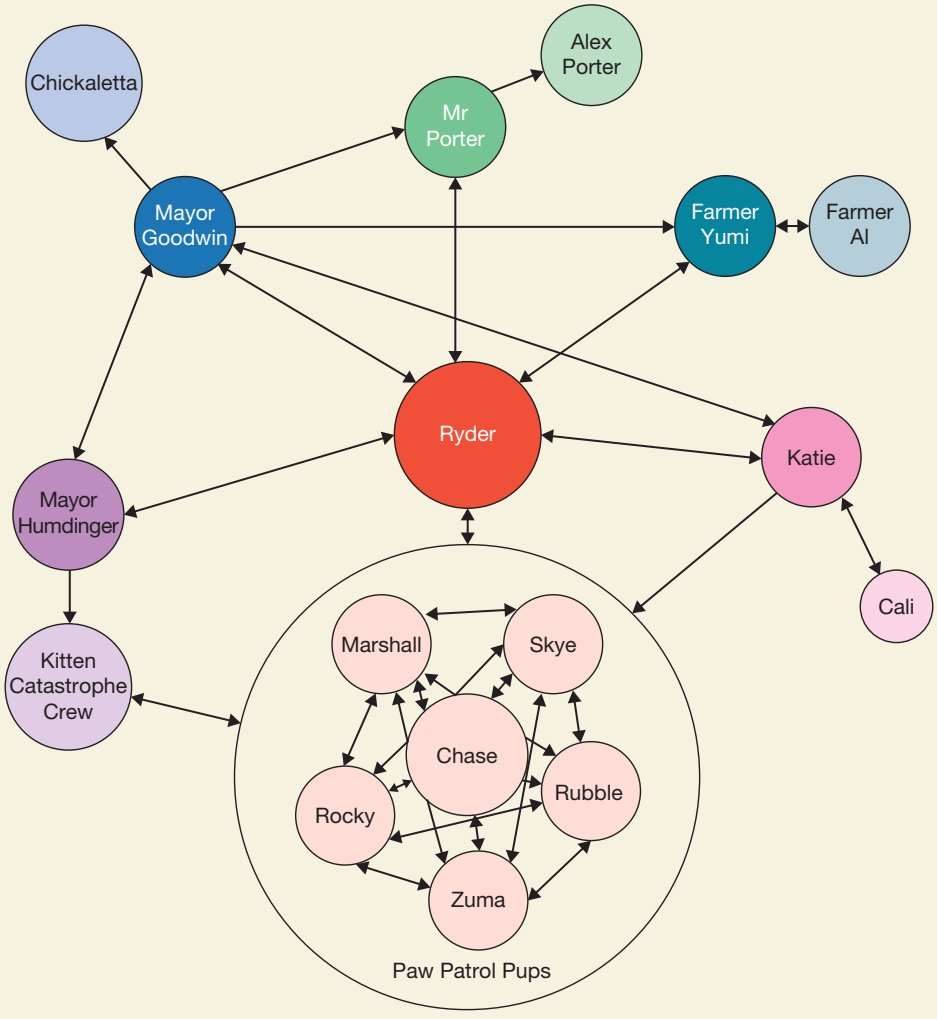


Figure 5.13 Sociogram of the main characters in the kids' television show *Paw Patrol.*

ACTIVITIES

Identify

1 What tool is used to visualise a social network?

2 How many nodes are there in the *Paw Patrol* sociogram?

3 How many relationships are there in the *Paw Patrol* pups sub-sociogram? Is this a triangular number? Draw a sociogram to illustrate this.

Analyse

4 Would you say Mayor Goodwin or Mayor Humdinger has more influence in the community based on the network connections? Why?

5 Look at the graph in Figure 5.14. Can it be solved? How do you know this?

6 Analyse the graph and find the solution.

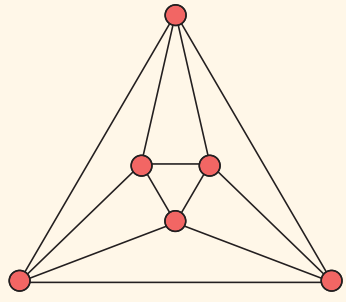


Figure 5.14 Graph that has a Eulerian cycle, meaning there is a path that visits every edge exactly once and starts and ends at the same node.

Extend

7 Design your own graph that has a Eulerian cycle and give it to a friend to solve.

8 Add additional characters to the *Paw Patrol* sociogram.

5.3 Social network examples

Non-computer social network – Aboriginal kinship connections

Aboriginal **kinship** is a system of relationships that govern social interactions among Australian First Nations communities. It encompasses complex networks of interdependent relationships between individuals, families and communities, and defines the roles and responsibilities of each member within these networks. Kinship determines how people relate to one another, how they should behave towards one another, and how resources and responsibilities should be shared. It includes the recognition of a broad range of relationships, including biological and non-biological ties, such as through marriage or adoption. The system of Aboriginal kinship is deeply rooted in the culture, traditions and history of Aboriginal communities and plays a fundamental role in shaping their social structures, values and ways of life.

Kinship connections

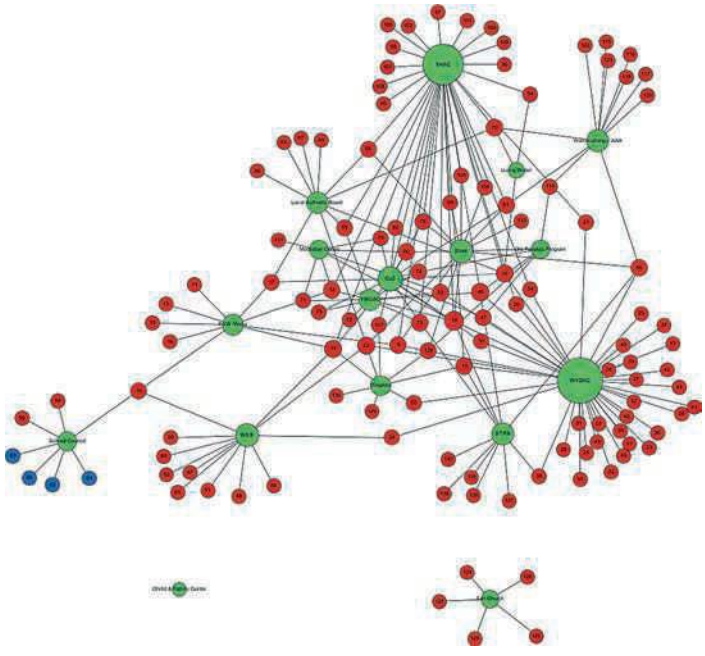
Kinship connections in Aboriginal communities is an example of a non-computer social network. These connections are a key feature of many Aboriginal cultures, and they are generally very complex, with intricate systems of relationships that are tied to social roles and responsibilities. The analysis of these networks using graph theory has allowed people to better understand Aboriginal culture.

Kinship networks

Graph theory has been used to analyse the structure of kinship networks in Aboriginal communities. In one study, researchers used graph theory to analyse kinship networks among the Yanyuwa people of Northern Australia, mapping out the complex relationships between individuals and groups. The study found that the Yanyuwa kinship system was characterised by a high degree of interconnectedness and a strong emphasis on reciprocity as well as mutual obligation.

Social networks

Graph theory has also been used to study the social networks of Aboriginal communities. In one study, researchers used social network analysis to examine the patterns of social interaction of community board members among the Warlpiri people of Central Australia. The study found that the Warlpiri social network was highly clustered, with a small number of individuals occupying central positions in the network and exerting a high degree of influence over the community (Figure 5.15).



Source: Whole of Community Engagement (WCE) initiative

Figure 5.15 Sociogram showing the community boards (green) and the individuals who were members of the boards. Can you identify the individuals who sat on more than one board, potentially having more influence than the ones on a single board? Individual 18 sat on six boards.

Land management

Graph theory has been used to understand the complex relationships between Aboriginal people and the land. In one study, researchers used graph theory to analyse the social networks of Aboriginal rangers who were responsible for managing natural resources on their traditional lands. The study found that the rangers' social networks were strongly linked to their cultural identity and values, and that these networks were critical to the success of their land management activities.

Overall, graph theory provides a powerful framework for understanding the complex social relationships and cultural practices of Aboriginal communities. By using graph theory to map out these relationships, researchers can gain a deeper understanding of the intricate systems of kinship, social interaction and land management that are so important to many Aboriginal cultures.

KNOWLEDGE BUILDER

Non-computer social network – LEGO Star Wars characters

Joshua C Fjelstul, a data scientist with an interest in data visualisations and LEGO, conducted an analysis of the character minifigures included in 362 Star Wars LEGO sets in order to see the portrayal of the character relationships, including the depiction of conflict. Clusters of nodes (characters that are closely linked in the network) show the closer relationship between characters in the same Star Wars series or episode. Clusters of the same colour also show there are many sets that only contain protagonists or antagonists, therefore not promoting conflict play by children using the LEGO (Figure 5.16).

Read through the article on this analysis found on the linked website. What other findings did the scientist observe through his network visualisations? How have the relationships of minifigs changed over time?

Weblink
How does LEGO depict conflict in Star Wars sets?

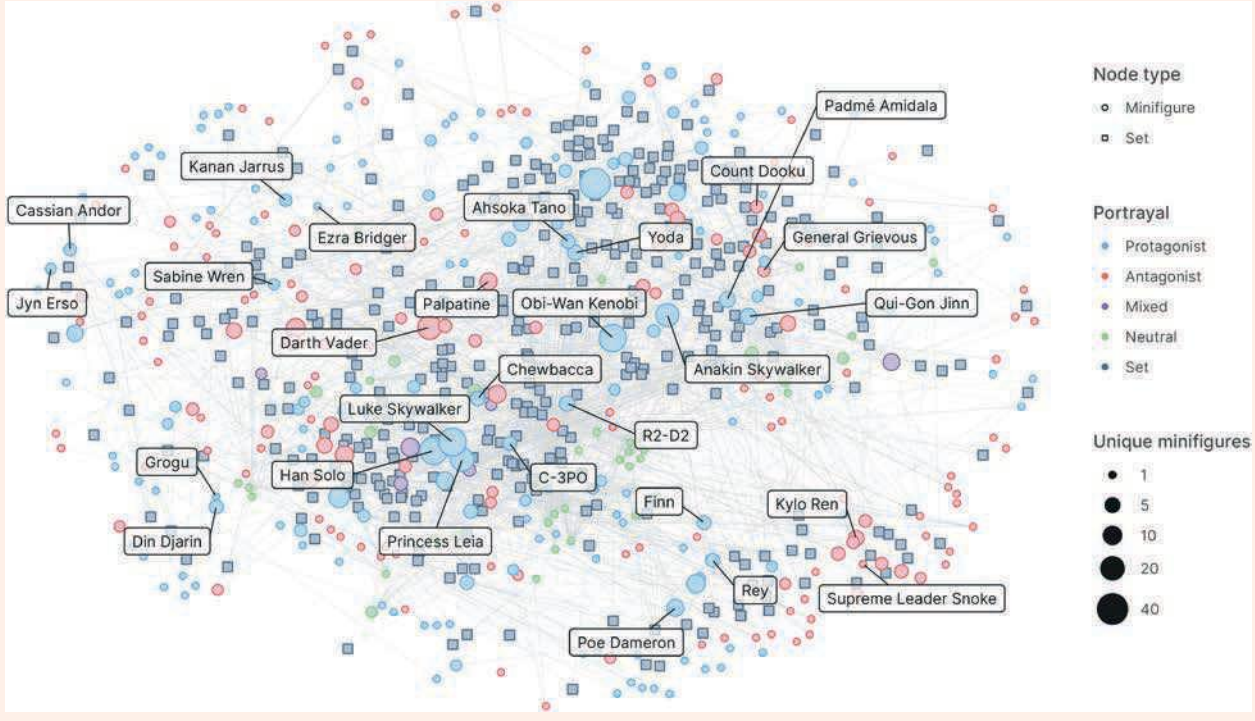


Figure 5.16 Network visualisation showing the relationship between minifig characters that appear in the Star Wars LEGO sets.

Weblinks
Excel Tool for Producing Sociograms

Gephi

Skill builder resource
Dummy data for sociograms

SKILL BUILDER

Non-computer social network – School friendship groups

In this skill builder you will use a customised spreadsheet system to help determine the best arrangements of students in cabins for a school camp based on existing social friendship networks.

1. Download the spreadsheet from the linked website.
2. Inspect the spreadsheet. The system allows each student to nominate up to three other students they would like to be in a cabin with and, optionally, nominate up to three students they do not want to be in a cabin with.
3. Use the dummy data provided on Nelson MindTap.
4. View the generated sociogram, identifying the clusters of students and therefore groups of individuals who would like to be together. Use this information to group students into cabins of up to six students.
5. Identify other potential relationships that can be initiated when previously unrelated friends are put in a cabin together based on determined friends-of-friends relationships.

Optional: You may like to do this activity with your peers in your class, but carefully consider what data is collected, who will have access to the data and how the final results will be used. Why is it important to consider the use of this type of personal data?

KNOWLEDGE BUILDER

Network analysis software – Gephi

In the previous Knowledge builder, you saw how a data scientist programmed their own tool to create network visualisations. Then in the Skill builder you used a simple application made in Excel to create a sociogram. Dedicated software is available for building network visualisations including Gephi, an open-source application (Figure 5.17).

Wikimedia Commons/Gephi/GPLv3

Figure 5.17 Gephi is a free network analysis and visualisation application.

1. Download and install Gephi, then open the sample file that shows the relationships between the characters in the classic musical *Les Miserables* (Figure 5.18).
2. Click on the Data Laboratory tab to view the two files used to create the network: a table of nodes that simply lists the node ID number and the name of the node; and a table of edges that shows the link between two nodes (the source and the target) via their ID plus other optional features of the link including a possible weight.
3. Go back to the Overview tab to inspect the visualisation. How has size and colour been used to help visualise the data? What do the clusters of characters show?
4. Run your mouse over a character to see how the interactive features of the software allow you to see the sub network for that particular character.
5. Explore some of the other tools within the software, such as finding the average degree and path length of the network in the Statistics menu.

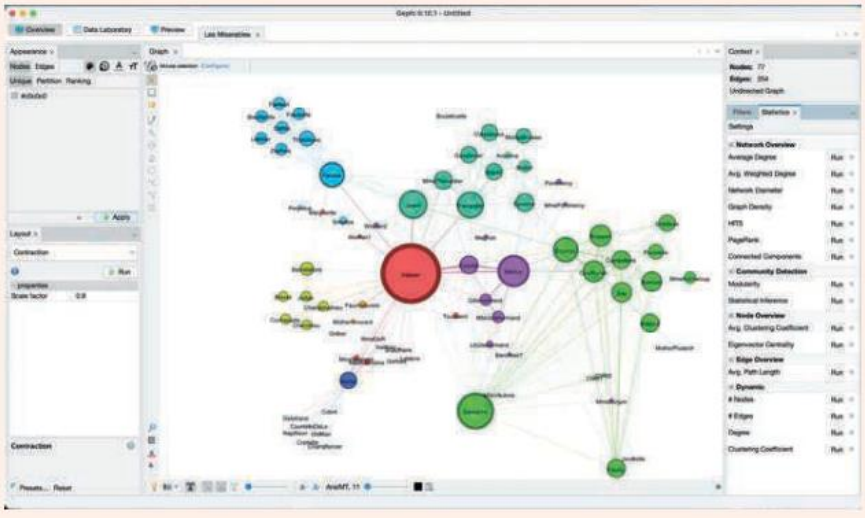


Figure 5.18 The Gephi interface includes windows for defining the nodes and relationships (data laboratory), then viewing the calculated network and adjusting the appearance. Network metrics can also be calculated in the statistics window. This sample data set shows the relationships of the characters in the musical *Les Miserables*.

Digital social networks

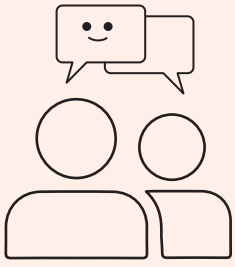
Not long after Tim Berners-Lee made the Internet accessible to everyone through the World Wide Web, entrepreneurs saw how the often physical boundaries of social networks could be overcome when translated onto an online platform.

The following timeline shows the relatively short but rapidly evolving history and development of online social networks including the key players and major milestones.

- 1997: Six Degrees is considered the first social networking site, and allowed users to create a profile, list their friends and send messages.
- 2000s: In the early 2000s, Friendster, MySpace and LinkedIn launched. Friendster was popular in Asia, while MySpace was the dominant social networking site in the US. LinkedIn focused on professional networking and is now used mostly by job seekers and recruiters.
- 2004: Facebook was launched and introduced several new features, such as news feeds and the ability to share photos and status updates.
- 2006: Twitter was launched as a microblogging platform, allowing users to share messages of no more than 140 characters. Hashtags (#) were used to categorise messages and make topics or themes easily searchable, a feature that has since been adopted by other platforms.
- 2010s: Several new social networks emerged, including Snapchat, which allowed users to share short-lived photos and videos; Twitch, a live-streaming platform mostly for video game broadcasting; and WeChat, which has become China's 'app for everything', including instant messaging, social media and mobile payment.
- 2018: TikTok was launched as a platform for short-form videos featuring music, comedy and other user-generated creative content.
- 2020s: Elon Musk acquired Twitter and later rebranded it as 'X'. Meta (who owns Facebook, Instagram and WhatsApp) launched Threads, a text-based 'social sharing' app linked to Instagram to compete with Twitter/X.

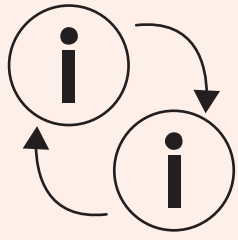
KNOWLEDGE BUILDER

Top 10 reasons people use social networks



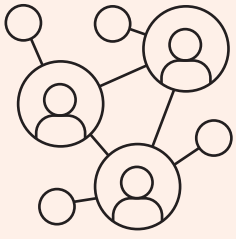
Socialising and staying connected: People use social media to stay in touch with friends, family and colleagues, to share updates and to maintain relationships.

Figure 5.19 Socialising.



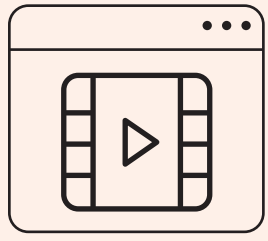
Information sharing: People use social media to share news, information and opinions. Social media platforms have become important sources of information and news for many people.

Figure 5.21 Information sharing.



Networking: Social media is used for professional networking, job hunting and business promotion.

Figure 5.20 Networking.



Entertainment: Social media provides a platform to share and consume entertaining content, such as funny videos, memes and other media.

Figure 5.22 Entertainment.

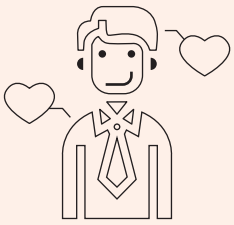


Figure 5.23 Personal expression.

Personal expression: Social media allows people to express their personalities, creativity and individuality through posts, comments and profiles.

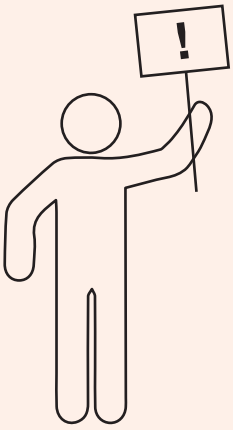


Figure 5.24 Social activism.

Social activism: Social media is used to raise awareness about social and political issues, to mobilise people for causes, and to organise protests and campaigns.

Figure 5.25 Shopping.

Shopping: Social media is increasingly used for shopping, as users can browse and buy products directly on social media platforms.

Figure 5.26 Education.

Education and learning: Social media can also be used for education and learning, as there are many resources and communities dedicated to various topics and interests.

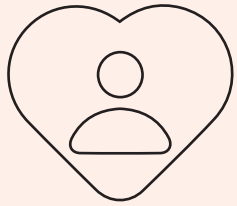


Figure 5.27 Inspiration.

Inspiration and motivation: Social media can inspire and motivate people with uplifting messages, inspirational stories and motivational content.

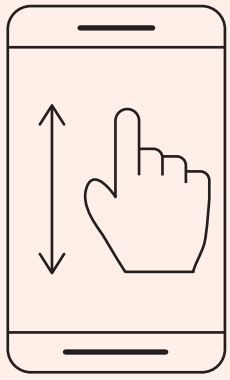


Figure 5.28 Time pass.

Time pass: Many people use social media as a way to pass the time, relax and unwind, especially during leisure hours or while commuting.

1 Consider the social network benefits illustrated in Figures 5.19 to 5.28. Pick the three reasons that most apply to you. Share with a peer. Are their reasons the same or different to yours?

2 Ask one or two adults to choose the three reasons that most apply to them, recording the data. As a class, collect this data. What is the most common reason among students? Is there a difference between ages or genders?

ACTIVITIES

Identify

1 List three ways graph theory has allowed for a better understanding of Aboriginal kinship.

2 Identity three other individuals who have connections with three or more boards in the sociogram in Figure 5.15.

3 What new functionality did Facebook provide that fuelled its popularity over other social media platforms?

Analyse

4 How do you think the development in network hardware and infrastructure influenced the development of social network platforms?

5 What styling has been used in the sociogram in Figure 5.18 to better represent the relationships?

6 What are the benefits of using tools like spreadsheets, Gephi or specialised code (like for the Star Wars analysis) to create sociograms?

Extend

7 Use Gephi to visualise your television network from the previous chapter.

8 Create a survey for the use of social network activity, collecting data from at least 20 people. Visualise the data collected using charts.

5.4 Social network data

Nearly all social media platforms are free to join or use. You may therefore wonder how are they some of the wealthiest companies in the world? Data. You are paying for their service with *your* data. But why is your data so valuable and are you aware of all the data they are using and how it is being used?

Profiles

When we are using the online world to engage socially with others, a digital profile needs to be created that represents us. The data in the profile commonly includes names, photos, interests and other personal information provided by the user (Figure 5.29). Although a user has a lot of control over the data included in and the customisation of their profile, there are significant volumes of data behind the profile that is used by the system to influence your involvement within the network.

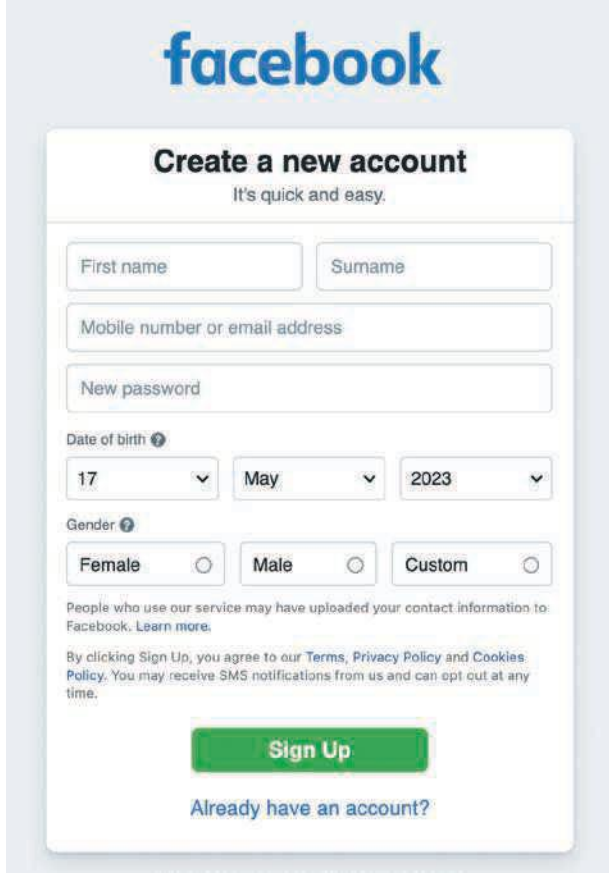
All of this related data is organised in a way that means what a system can determine about you (and your connections) is more than your closest acquaintances (or even yourself) would know.

Shutterstock.com/Alexey Boldin

Figure 5.29 People provide a range of personal data to platforms like Facebook when creating their online profile.

Data collection

Data collection in a social network system can be active or passive. When you first sign up to a service you actively supply data such as your name, a profile photo and other personal details (Figure 5.30).



Source: Facebook

Figure 5.30 When you sign up to a social media platform like Facebook, it is important that you read the linked data and privacy policy to be aware of how they are using and managing your data as well as other data they may be passively collecting.

A form is used to collect data with user-friendly control structures and validation to improve the efficiency and accuracy of the data collection process. Many services also offer the option to use an existing account with another service (e.g. Google) to create an account, so personal data can be shared from there.

Passive data collection refers to the practice of collecting information about individuals' behaviours, preferences and activities without their active participation or knowledge. This type of data collection typically occurs through the use of tracking technologies such as cookies or beacons which are embedded in websites, mobile apps and other digital platforms. These tracking technologies gather information such as the user's browsing history, search queries, location data and device information.

Passive data collection is often used by advertisers, market researchers and data brokers to create targeted advertising and marketing campaigns, as well as to develop insights into consumer behaviour and trends. However, it is also a controversial practice because it can raise privacy concerns and lead to the unauthorised sharing of personal data.

For this reason, laws have been put in place in many countries requiring companies to obtain user consent before collecting personal information and to also provide users with the ability to opt-out of data collection.

KNOWLEDGE BUILDER

Baxta pet social network – Part 1

Baxta is an Australian social networking platform for pets or, more accurately, human pet enthusiasts. The platform can only be accessed from an app. After you download the app, you will be asked to create an account or log in using an existing platform including Apple, Google or Facebook. On this initial page, there is also a link to the terms of service and privacy policy with the statement 'By continuing you agree to Baxta's terms of service'.

The minimum personal data they actively collect is your email address. Your name and other information is optional, such as your address if you would like to use location-based services such as distance to parks. The other data that is actively collected in a simple form are the details and media related to your pets (Figure 5.31).

In the privacy policy they outline how data may be used including 'From time to time we may also use personal information about Registered Users and potential Registered Users to inform you about Baxta products, services, promotions or for lost pets.' They also state 'Baxta does not sell, rent or trade personal information about you to or with third parties'.

Weblink
Baxta Pets Privacy Policy

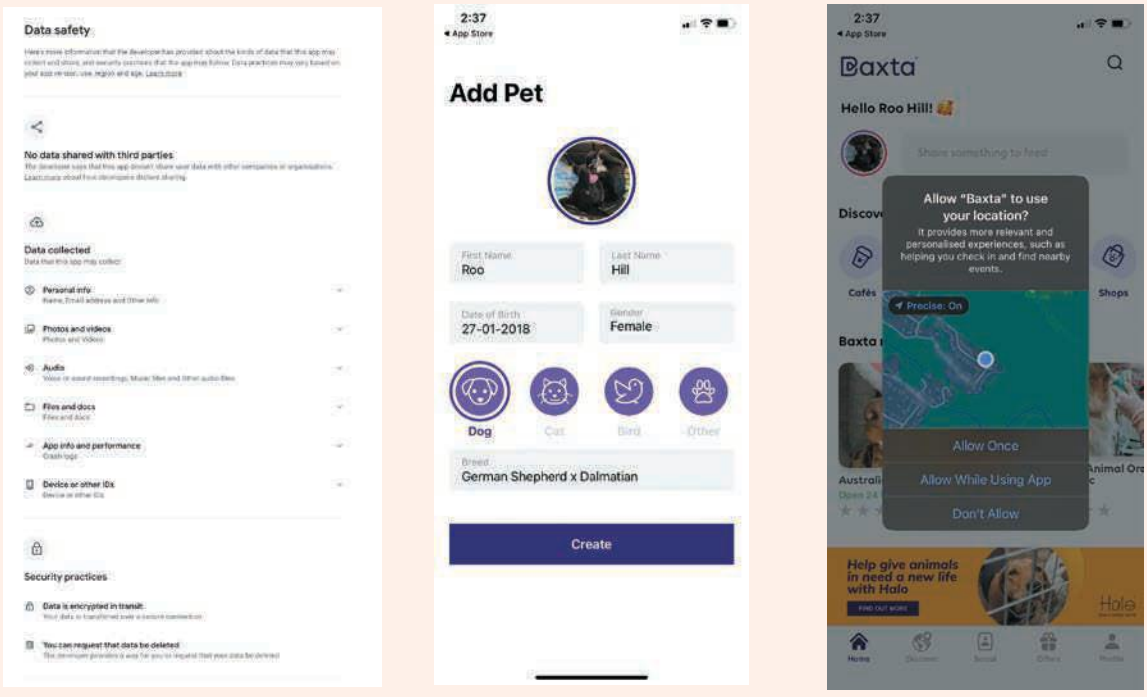


Figure 5.31 The data safety information from the Google Play store; the Baxta app form that collects the key data about your pet; and the prompt for the user to optionally allow location data to be collected and used by the Baxta app.

1 Either explore the Baxta app by creating your own account, or look through the features outlined on the Baxta website.
2 List the different types of data collected by Baxta and give an example of each.
3 How has the 'Add Pet' form been designed to make data collection efficient and to minimise data inaccuracy? How could it be improved?
4 Go to the Google Play store and compare the Data Safety page of Baxta to that of Facebook. How much more data does Facebook collect? Do they follow any additional security practices?

Database structure

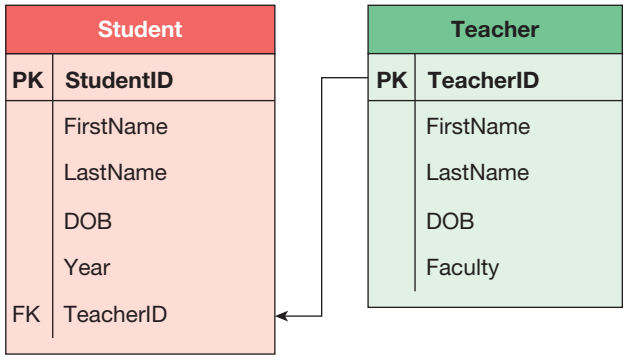
In Chapter 10 you will learn how related data can be stored in tables with relationships. For example. in a simple primary school, there would be a table for student records and a table for teacher records, with a field in the student record linking that student to their teacher's record. This is called a one-to-many relationship, as one teacher is linked to many students. In reality the relationships between teachers and students can be a lot more complex. Consider your school; you have many teachers and the teachers have many students (a many-to-many relationship). You also possibly have relationships to teachers who do not explicitly teach you; they may coach you in a sports team or act as a mentor.

In social networks relationships are commonly complex, so rather than organising data in a traditional relational database using a relational database management system (RDBMS), a graph database system is used, drawing on the natural web structure of a network. In a graph database, nodes represent entities such as people, products or locations, while edges represent the relationships between these entities.

For example, a graph database could represent a social network by using nodes to represent individual users and edges to represent the connections between them. Graph databases are optimised for querying and analysing complex, interconnected data as the focus is on the relationships. When we are trying to organise data in a database that has many-to-many relationships and is more heterogeneous in nature (not always the same fields), a graph database is therefore commonly a better tool.

Nodes in a graph database represent an individual entity (e.g. Joe Bloggs) but can be tagged with label that shows the type of entity (e.g. Student) and hold the attributes for that entity (e.g. name, date of birth). Relationships are directed and named (e.g. Mr Hill –*mentors*→ Joe Bloggs). They can also have properties (e.g. time).

Simple Primary School Database Schema



High School Database Schema

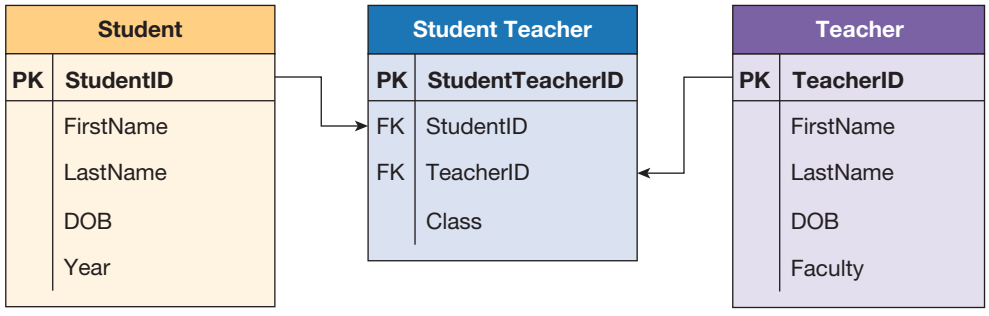


Figure 5.32 A simple primary school database would store the related data between teachers and students in two tables with a one teacher to many students relationship.

In a high school database, the entities include students and teachers with attributes such as FirstName, LastName, DOB, etc. (Figure 5.32). The relationship between a student and teacher is many-to-many, so an intermediate table is created that stores the various 'events' of their relationship, such as a student being in a particular class with a teacher (Figure 5.33).

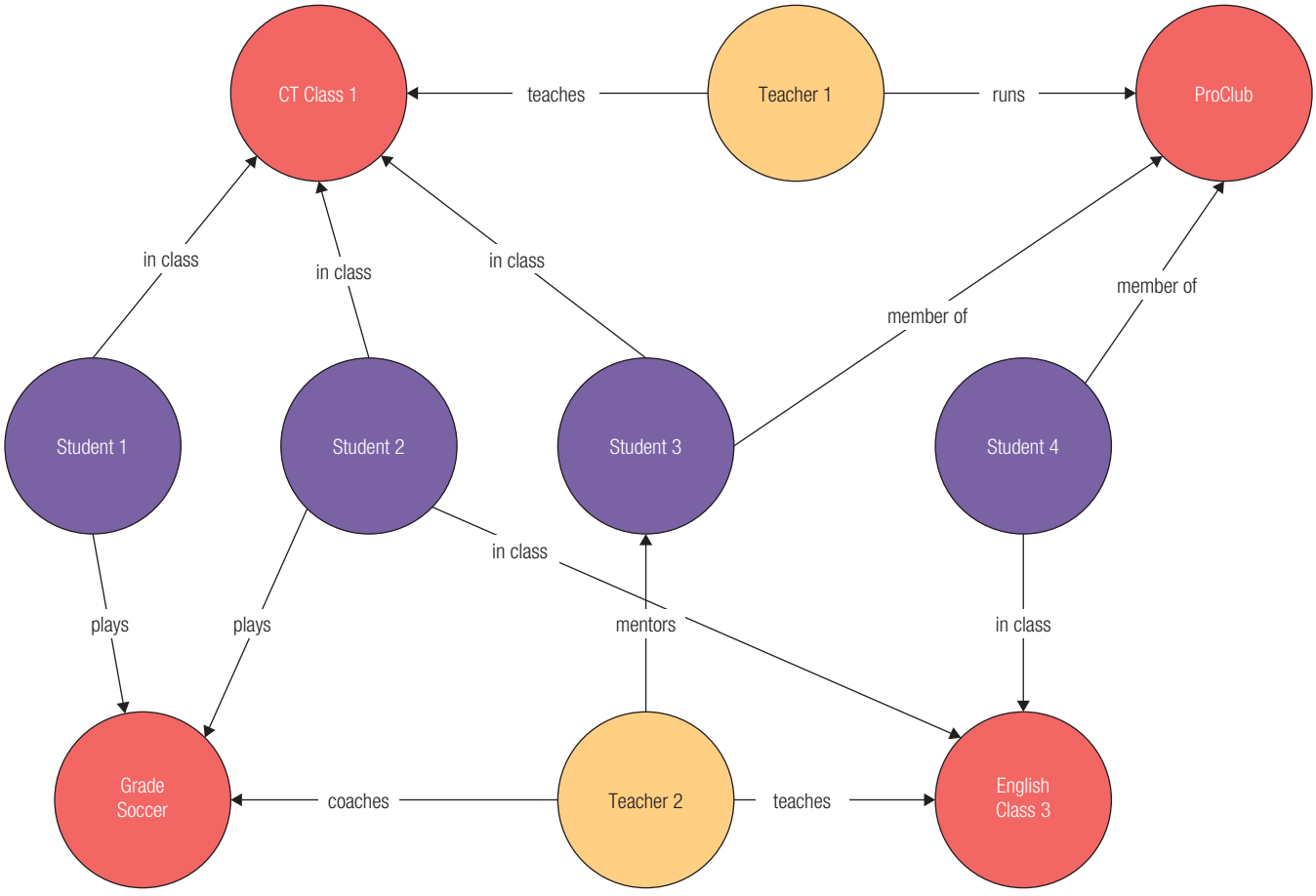


Figure 5.33 A graph database provides the flexibility to show the more complex nature of the relationships in a social network such as a school. This one shows a small subset to represent how the various relationships could connect teachers, students and other school entities.

KNOWLEDGE BUILDER

Baxta pet social network – Part 2

Baxta is a social network that stores data about pets, their owners, social media posts and venues the pets may use, such as dog parks and grooming services. To understand how the related data may be organised, we will consider how these entities, attributes and relationships may be defined in a relational database structure and a graph database arrangement (Figure 5.34).

Relational database structure

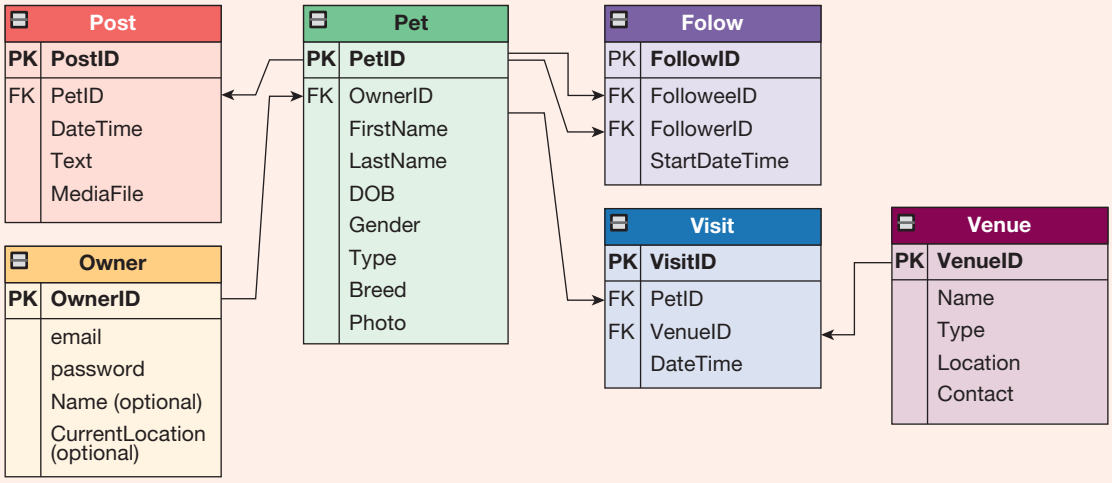


Figure 5.34 Simplified relational database structure for the data collected in the Baxta pet social network system including the entities/events of Pet, Owner, Post, Follow, Venue and Visit. An owner can have one or many pets. A pet can post one or many times. The event of a pet visiting a venue captures the otherwise many-to-many relationship of Pets and Venues. The event of a 'follow' captures the many-to-many relationship of pets following pets.

1 Create a data dictionary for the Pet table.

2 How would a review be stored in this database: as an attribute in an existing entity or as its own entity linked to one or two tables?

Graph database structure

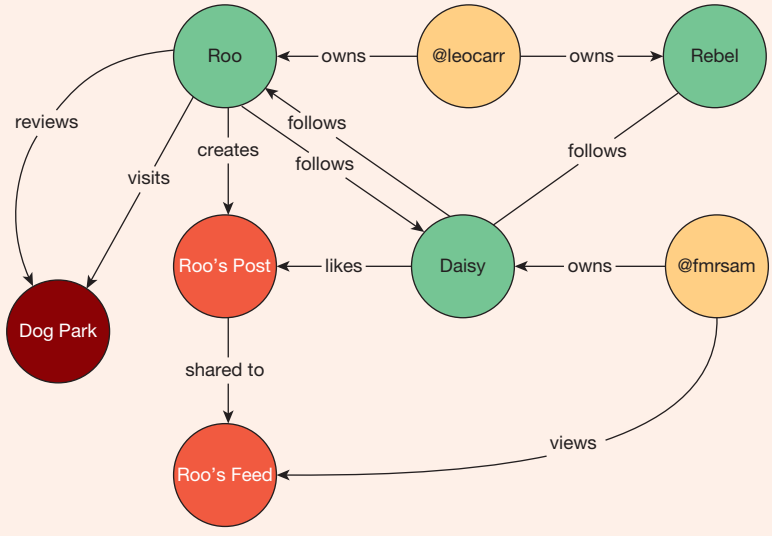


Figure 5.35 The entities are very similar to the relational database; however, nodes represent individual entities such as a particular pet (e.g. Roo), a particular owner (e.g. @fmrsam) or a venue (e.g. Dog Park). Relationship arrows show the direction of the relationship and are labelled with the type of relationship (e.g. @leocarr – *owns* → Roo).

3 Recreate Figure 5.35 using a diagram creation tool such as diagrams.net. Style or tag the diagram to show the type of entity and add labels to the entities (and relationships) to identify their potential properties.

4 Add some additional entities and relationships to the diagram, using the Baxta app or website for inspiration.

Access and analysis

Every time we load a new page or scroll down in a social network app or website numerous query requests are occurring. Sometimes this involves a simple search to retrieve existing data stored in the database, such as when you view your account profile. Other times complex analyses are occurring in order to display targeted information based on constantly updating entities and relationships.

Structured Query Language (SQL)

Computational thinking

In a traditional database, these queries are done using Structured Query Language (SQL). For example, to find all the students in a particular class (e.g. Maths01) the SQL would read:

```
SELECT Student.FirstName, Student.LastName
FROM Student, StudentTeacher
WHERE Class = "Maths01", Student.StudentID =
   StudentTeacher.StudentID
ORDERBY LastName
```

Graph database query languages (GQL)

There are a number of different graph database query languages (GQL). A common query in these languages is the MATCH query. A match query is used to find and retrieve specific patterns of nodes and edges in a graph. The user specifies a pattern of nodes and edges using a graph pattern syntax and the database returns all instances of that pattern in the graph.

The syntax for a match query typically involves specifying a set of nodes and edges along with any relevant properties or conditions. For example, a simple match query might look like this:

```
MATCH (student:Student)-[:IN_CLASS]->(class:Class)
WHERE class.name = "Maths01"
RETURN student.FirstName, student.LastName
```

In this example, the query is looking for pairs of nodes representing students who are connected by an "IN_CLASS" edge to Classes with the name property equal to "Maths01". The query returns the first and last names of each student.

The following outlines how a match query works:

- The query engine begins by identifying the starting node or nodes specified in the query pattern.
- It then looks for edges that match the specified relationship type and direction, connecting the starting node to other nodes in the graph.
- The query engine continues to traverse the graph, following edges that match the pattern until it reaches the end of the specified pattern.
- If the entire pattern is matched, the query engine returns the result set containing the nodes and edges that match the specified pattern.
- The result set can then be processed further, filtered, sorted or aggregated as needed using additional query clauses.

Match queries can be used to find specific patterns in a graph, explore relationships between nodes and perform graph analysis and modelling. They are a powerful tool for working with graph data and are a key feature of most graph database query languages.

Weblink
Keeping Culture

INFOBIT

The Keeping Culture system, introduced in Chapter 3, organises the media, including photos, documents, movies and sounds, into archives records, and the knowledge into profiles records such as people and places. These are all linked to create a network graph of knowledge for storage, access and analysis.

To find out more, go to the Keeping Culture website.

KNOWLEDGE BUILDER

Baxta pet social network – Part 3

Let's consider the following common processes performed on the Baxta app (Figure 5.36) and compose likely SQL and GQL commands that may be used when a user:

- views their pet's profile
- views their social feed
- views the social feed of the pets they are following.

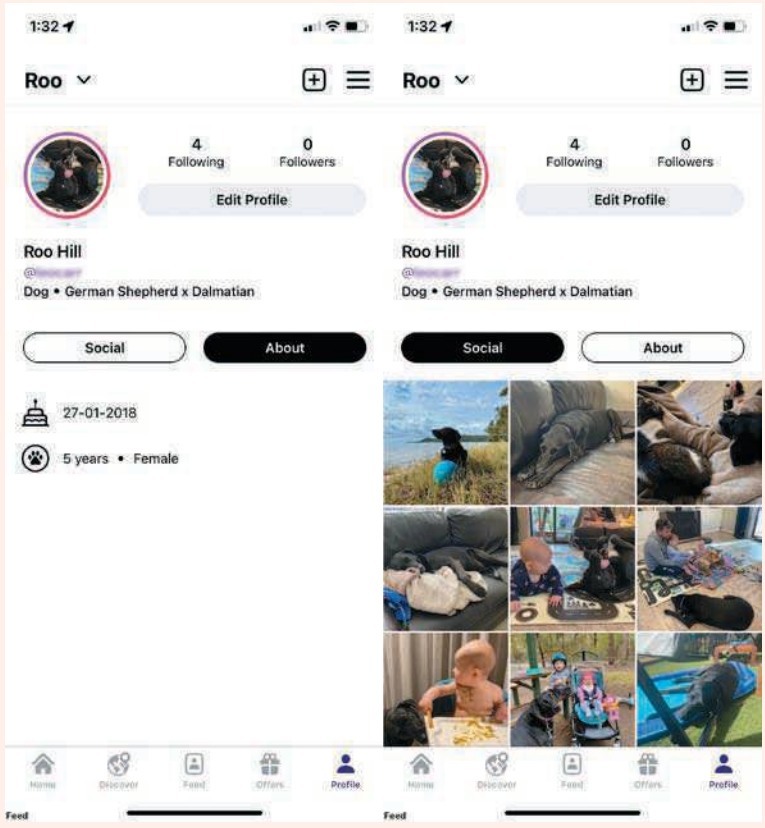


Figure 5.36 The profile page of the pet Roo, including the 'About' view and the 'Social' view.

A user viewing their pet's 'About' profile

SQL	GQL
SELECT Pet.Photo, Pet.FirstName, Pet.LastName, Owner.OwnerID, Pet.Type, Pet.Breed, Pet.DOB, Pet.Gender FROM Pet, Owner WHERE Pet.OwnerID = Owner.OwnerID AND Pet.PetID = "RH027"	MATCH (pet:PET)-[:OWNED_BY]->(owner:Owner) WHERE owner.OwnerID = @leocarr AND pet.PetID = "RH027" RETURN Pet.Photo, Pet.FirstName, Pet.LastName, Owner.OwnerID, Pet.Type, Pet.Breed, Pet.DOB, Pet.Gender

A user viewing their social feed summary

SQL	GQL
SELECT Post.Media WHERE Post.PetID = Pet.PetID ORDER BY Post.DateTime DESC LIMIT 9	MATCH (pet:PET)-[:CREATES]->(post:POST) WHERE pet.PetID = "RH027" RETURN post.Media ORDER BY post.DateTime DESC LIMIT 9

A user viewing the social feed of the pets they are following

SQL	GQL
SELECT Post.Media, Pet.FirstName, Pet.LastName, Post.DateTime, Post.text WHERE Follow.FollowerID = "RH027" AND Follow.FolloweeID = Pet.PetID ORDER BY Post.DateTime DESC LIMIT 20	MATCH (pet1:PET)-[:FOLLOWS]->(pet2:PET) MATCH (pet2)-[:CREATES]->)(post:POST) WHERE pet1.PetID = "RH027" RETURN post.Media ORDER BY post.DateTime DESC LIMIT 20

ACTIVITIES

Identify

1. What data is typically collected for a profile on social media? What data is typically collected as users create content for their social media account?
2. How can users be aware of how their data is being used, how it is being kept secure and what control they have over their data?
3. If organising data about students and teachers in a high school context in a relational database, what is the relationship between students and teachers? How is this managed in an RDBMS?
4. Why are graph databases used to organise and manage data in social media systems?

Analyse

5. Select another social media application and analyse its data safety.
6. Compare and contrast the syntax of the SQL and GQL in the Baxta examples.

Extend

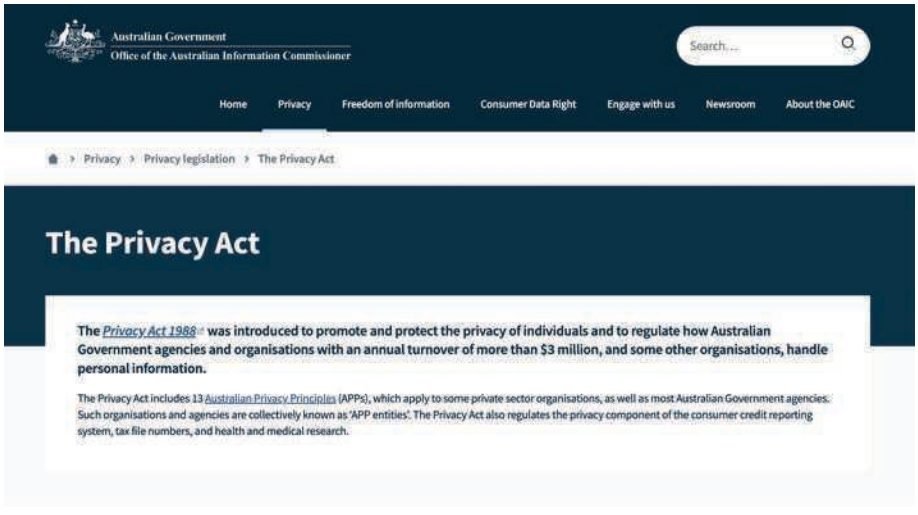
7. Write an SQL and/or GQL statement that may be used in the Baxta system to return the three closest dog parks for a pet.
8. How would the event of a 'like' on a post be organised in the relational database or the graph database structure?

5.5 Social, ethical and legal considerations

While social networks have many benefits, there are also several issues that are associated with their use. These issues may have social, ethical and/or legal implications and need to be thoroughly considered when using the platform. Developers also have a responsibility to implement features and practices to remove or at least minimise the risks associated with these issues.

Privacy

One of the most significant issues associated with social networks is privacy, which has social, ethical and legal implications. Users often share **personally identifiable information (PII)** on these sites. As this information identifies a specific individual, it is particularly vulnerable to hacking or misuse. Social networks can also collect large amounts of data on users, raising concerns about how this data is being used and who has access to it. Legally, social media platforms need to comply with data protection and privacy laws, such as the Australian *Privacy Act* (Figure 5.37).

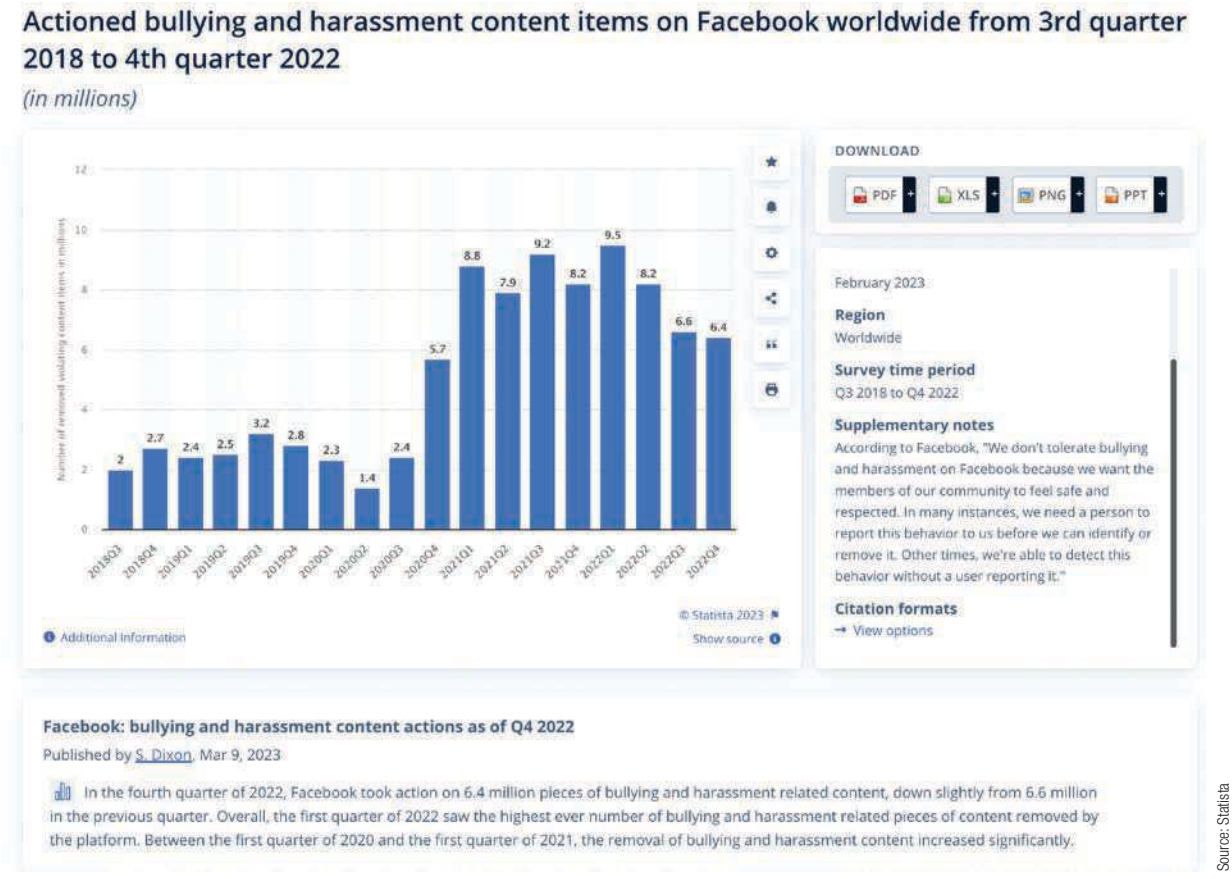


Source: Office of the Australian Information Commissioner website — www.oaic.gov.au

Figure 5.37 The Australian Privacy Act was originally introduced in 1988, before the World Wide Web, but has been continuously updated in line with technological advances to address all contexts where individuals should have their privacy protected.

Cyberbullying

Social networks can also be a platform for cyberbullying, with users harassing or attacking each other online. This can have serious consequences, including emotional distress and even suicide. Ethical considerations require social media companies to implement measures to prevent cyberbullying and provide support to victims. The bullying and harassment of individuals can also lead to legal consequences. Companies need to take measures to prevent cyberbullying and harassment on their platforms, such as providing reporting mechanisms and investigating complaints (Figure 5.38).



Source: Statista

Figure 5.38 Statista's report on content removed due to bullying and harassment. In the first quarter of 2022, Facebook removed 9.5 million content items that were identified as bullying and/or harassment. Analysis of the graph shows a sharp increase between the third and fourth quarter in 2020, where it has remained high, but starting to show a trend down since early 2022.

Addiction and social isolation

Some people become addicted to social networks, spending excessive amounts of time on these sites and neglecting other aspects of their lives (Figure 5.39). This can have negative impacts on mental health, social relationships, and work or school performance. Companies need to be responsible for designing their platforms in a way that promotes healthy social media usage and addresses addiction.

Ironically, social networks can also contribute to social isolation, as people spend more time communicating online and less time engaging in face-to-face interactions. This can have negative effects on mental health and wellbeing.

Shutterstock.com/Fabio Principe

Figure 5.39 Reflect on how much time you spend on social media. Do you make the most of physical relationships when with your peers or do you spend significant time on your phones while in the company of your peers?

Fake news and misinformation

Social networks can also be a breeding ground for fake news and misinformation, which can have serious consequences for society. This is especially concerning in the context of political campaigns and other important events, where false information can sway public opinion and lead to harmful decisions. Social media companies need to be responsible for identifying and removing misleading or false information from their platforms. This can be achieved through a range of validation and verification processes, such as user reporting, collaborating with independent fact-checking organisations and the use of AI and machine learning to detect patterns and characteristics associated with fake news (Figure 5.40). When social media is used to spread false or harmful information about individuals it can lead to defamation lawsuits. Social media companies may be held liable if they fail to take reasonable steps to prevent defamatory content from being published on their platforms.

Bias and discrimination

Social media can not only perpetuate but also amplify systemic biases and discrimination, such as racism, sexism and homophobia. This is especially problematic when the platform's algorithms create echo chambers or bubbles that only reinforce a user's existing beliefs and perspectives.

Censorship and freedom of speech

Social media platforms need to balance the need to allow freedom of speech with the responsibility to prevent the spread of misinformation and hate speech. This is a delicate balance, and it requires ethical considerations from social media companies to ensure that they are not biased against certain groups or particular viewpoints.

Intellectual property

Social media platforms commonly host content that is protected by intellectual property laws, such as copyright or trademark. Companies need to take steps to ensure that user-generated content does not infringe on the intellectual property rights of others (Figure 5.41).

Advertising and marketing

Social media platforms host advertising and marketing content, which may be subject to advertising and marketing laws. Companies need to ensure that their advertising and marketing content complies with these laws.

Source: Facebook

Figure 5.40 Facebook allows users to report posts and ads that they feel breach the platform's rules including false news. The report option is easily accessed by clicking the three dots in the top right corner of any post.

Terms of service and user agreements

Social media platforms have their own terms of service and **user agreements**, which are legal documents that users agree to when using the platform. Companies need to ensure that their terms of service and user agreements comply with relevant laws and regulations.

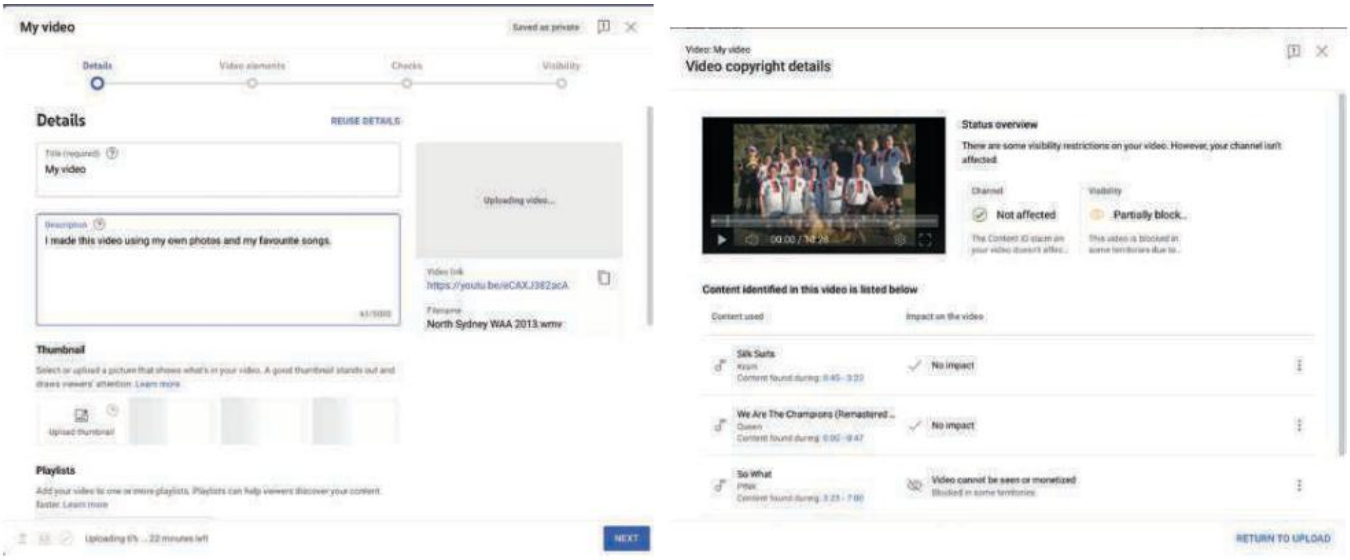


Figure 5.41 YouTube performs checks on content being uploaded to the platform including checking to see if the content contains copyright material, such as this video, which uses copyrighted music.

These issues are complex and multifaceted, and addressing them will require a combination of individual responsibility, regulation and social change. As social media continues to evolve, new issues will likely emerge and need to be addressed and evaluated from social, ethical and legal perspectives.

Accessibility

Like all information systems, social media platforms should be designed in ways that make them accessible to people with disability. This includes implementing various features and practices such as being compatible with screen readers and keyboard navigation, including alt text and descriptions for images as well as captions or transcripts for other media, ensuring the visual design is easily readable for individuals with colour vision deficiencies as well as text resizing and zooming options for people with other vision impairment disability. Social media companies commonly encourage users to provide feedback on accessibility issues they encounter. This feedback helps platforms identify and address accessibility barriers, ensuring continuous improvement. Platforms also follow established **accessibility** guidelines, such as the Web Content Accessibility Guidelines (WCAG), to ensure they meet industry standards.

Chapter review quiz

SKILL BUILDER

Social media use issues infographic

Design thinking

In this Skill builder you will create an infographic to visually communicate metrics associated with some of the key issues of social media use.

1 Use the web to find out key statistics about issues associated with social media use.
   - **a** You want to make sure the data behind the statistics is of quality. This includes assessing the source of the data (*Is it reputable?*), how up-to-date the data is (*When was the data captured?*) and how relevant it is to our context (*Is it capturing the whole world, just our country or another country not relevant to us?*).
   - **b** Websites like Statista have a good reputation for data quality and clearly show the context of the data including dates and locations.
   - **c** Aim to collect 6–12 statistical points. You may like to pick a 'theme' for your statistics as there are a lot to choose from. For example, you could find statistics related to social media addiction.

Weblink Statista

2 Create an infographic to display the statistics.

a For each statistic decide how it might best be visualised. For example, a statistic that is a simple fraction could be represented by coloured-in people icons. A simple number statistic could be accompanied by a relevant icon such as a clock to represent time (Figure 5.42).

b Use an application like Canva or Adobe Illustrator to create your infographic. Make sure you follow design principles to ensure your infographic is attractive and readable. For example, select a colour palette to keep the look consistent and use colour to emphasise key metrics.

Alternative: Create a poster that promotes the safe and ethical use of social media.

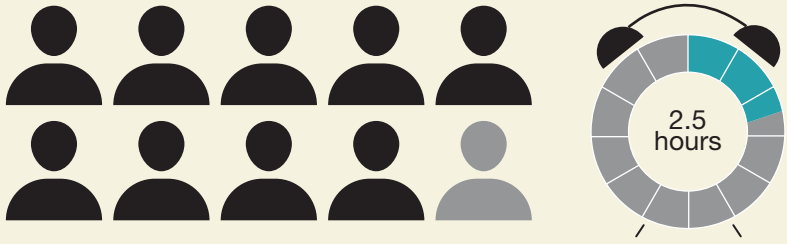


Figure 5.42 Example visualisations for an infographic showing the statistics that 9 out of 10 internet users use social media every month and a user spends an average of 2.5 hours a day on social media.

ACTIVITIES

Identify

1 What is PII? Give examples that you might share online.

2 Advertising and marketing is a legal issue. How could it also be considered a social and/or ethical issue?

3 Give an example of user content that could potentially infringe on the IP rights of others.

Analyse

4 One suggested strategy to limit the amount of personal data social media platforms use (as your form of 'payment') is to instead add a cost to the service and allow users to opt out of certain data sharing. Alternatively, users could be paid for their data. Consider the benefits and potential issues with these strategies.

5 Go to the Statista website and search for Facebook or another significant social networking company. Look at the statistics for 'content removal' including key metrics or trends in the data. Discuss with a peer.

Extend

6 In a group discuss what social rules you think should be followed when posting content on social media that includes information about other people (e.g. where someone has been, people's faces in photos).

Weblinks:

- Six Degrees of Wikipedia
- How does LEGO depict conflict in Star Wars sets?
- Excel Tool for Producing Sociograms
- Gephi
- Baxta Pets Privacy Policy
- Keeping Culture
- Statista
- Kickstarter

Skill builder resource:

- Dummy data for sociograms

Chapter review quiz

Nelson MindTap

To access resources above, visit
cengage.com.au/nelsonmindtap

5.6 Project: New social network pitch

In this chapter you read about some of the big, or should we say gigantic, social networks in use today. Facebook, for example, has almost 3 billion monthly active users, and of those, 2 billion log in every day. It has a very general purpose of connecting everybody around the world with smaller groups within the platform allowing for communication about particular interests to be fostered. You also looked at a more specialised social network for pet enthusiasts, Baxta. There are many more 'boutique' social networks, designed to connect individuals with niche interests or needs. For example, 7 Cups is an online platform that provides counselling and therapy by connecting people with professional 'active listeners', and Ravelry is a social network for people who like to knit or crochet.

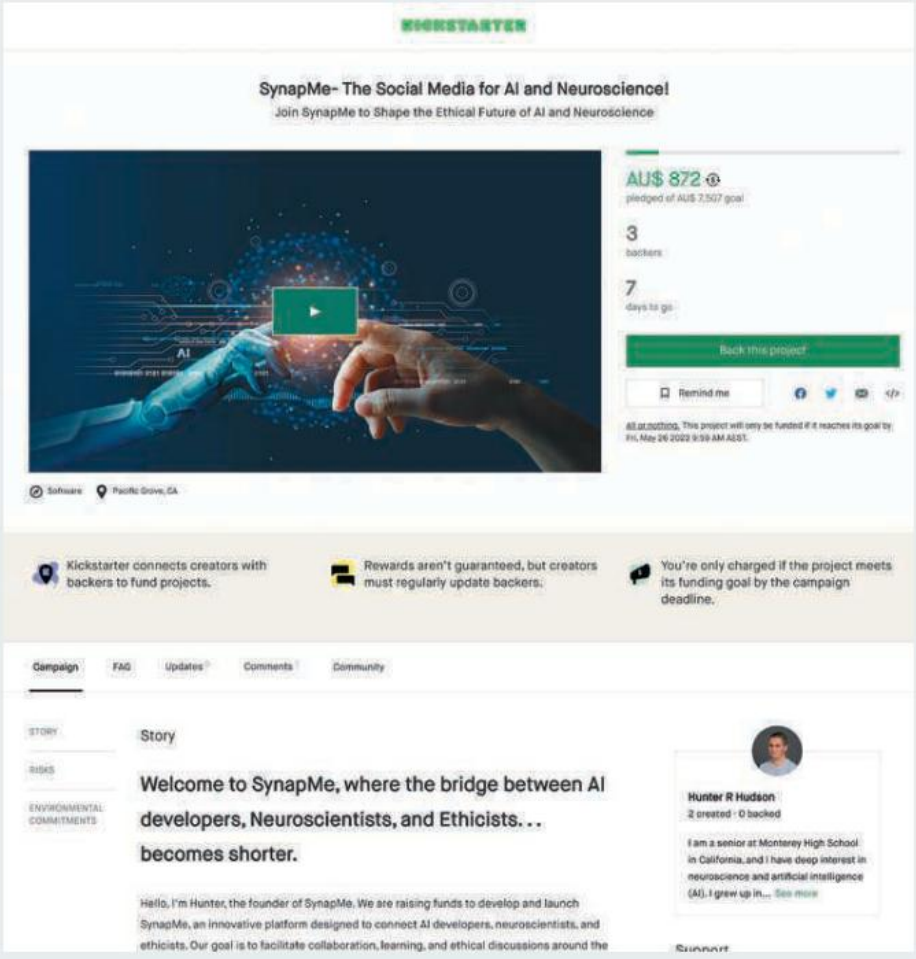
Your task

For your project, you are required to design a social network related to a particular interest you have. The functionality of the network would allow individuals to communicate about this interest using the most appropriate media forms and structure.

You will then need to create a pitch to get the network funded on a platform like Kickstarter (Figure 5.43). This should include a proposal report that addresses functional and non-functional requirements such as security, and a prototype of the network using a design and prototyping tool such as Figma or Adobe XD so you can demonstrate how it works. How the data will be organised, either as a relational database or graph database, may also be demonstrated along with some example queries showing key functionality.

Design thinking

Weblink
Kickstarter



Source: Kickstarter, SynapMe

Figure 5.43 SynapMe is a social media platform seeking funding via Kickstarter. The developer has provided a story to explain why they are taking on this project, the goal of the project and what the platform will do. The risks associated with the development including data security are also addressed along with their consideration and commitment to minimising their environmental impact.

Tracking your project development

Set up a document to record the key processes during project development. This document will likely show an iterative process, as you continually plan, produce and evaluate different parts of the whole project over the project's timeline. A logbook is an informal way to easily note down progress. If a more formal documentation report is required at the end of the project, your logbook can be used as a reference when creating a report.

Idea generation for your social network theme

1 Use a mind map to brainstorm your interests (e.g. sporting, reading, social justice or other hobby). For each interest, brainstorm ways you interact with other people when engaging with the interest. For example, for soccer you may go to training, play in a game, get feedback from your coach, discuss tactics with your teammates, keep track of your results, talk about your favourite famous players, etc. If producing your mind map digitally you may like to add imagery to help visualise your ideas (Figure 5.44).

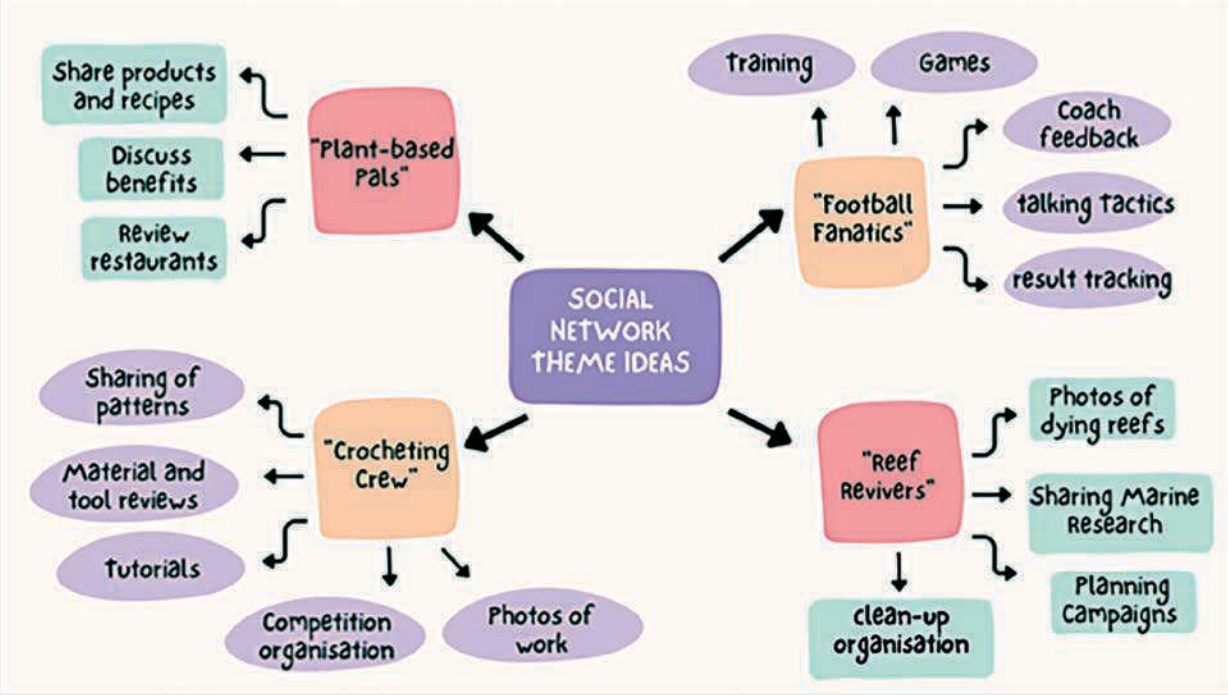


Figure 5.44 Online tools like Canva can help you to create documents like mind maps quickly by providing templates.

2 Reflecting on your mind map, think about which of your interests you would most likely benefit from engaging with your peers using a social network. Maybe for a sport, a physical social network is enough as you see each other so frequently. Perhaps one of your interests where you do not get to physically engage with your peers regularly or where you would like to engage with a wider range of people would be more in need of a social network. Select an interest for your social network based on these reflections.

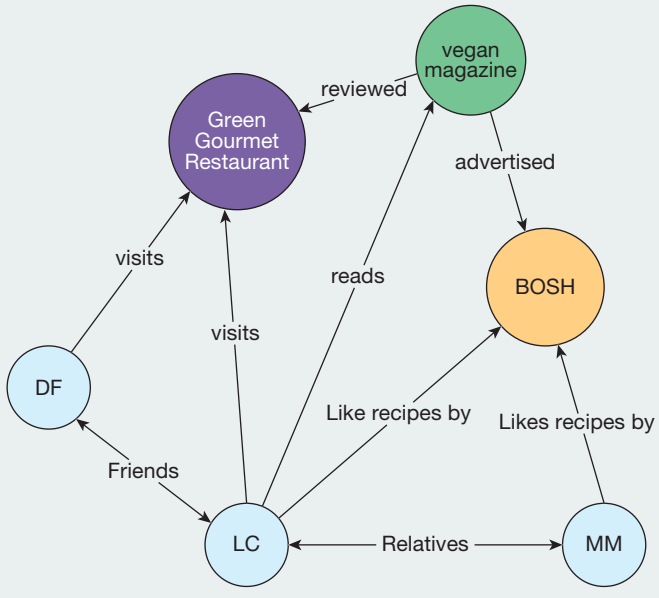


Figure 5.45 Part of a sociogram for a plant-based diet social network.

9780170442787

Modelling the network and defining the requirements and considerations

3 Draw a sociogram to represent your physical social network related to this interest such as the one shown in Figure 5.45.

4 Define some of the functional requirements for your social network based on your experience with the Skill builder and Knowledge builder activities, discussions with your teacher and the details in the description of the project (e.g. what will be the nodes/entities and the relationships between them?).

5 Define some of the non-functional requirements such as data security, including what personal details need to be collected, how they will be kept secure and how this will be communicated to users.

6 Discuss factors/issues that could impact the development of your solution (e.g. time, resources and money).

Researching and planning

Examining similar products

7 Explore the Google Play Store or Apple App Store to see if other similar products exist. If so, look at their website or description in the app stores to learn more about their features and functionality. Record summaries of the following in your project documentation:

- **a** Identify what data is collected and evaluate how it is kept secure.
- **b** Describe how users engage with each other on the network.
- **c** Read some reviews to get an insight into features or functionality that users particularly like and don't like. List what you interpret to be the top three of each.

8 Explore Kickstarter to see what other social media projects are being developed. Select two that interest you and inspect the project pages. Outline the titles and headings under which the project is described. Record a summary of the two sites that interest you and reflect on how you could also document your project idea in a similar way.

Refining the project requirements

9 Conduct interviews and/or a survey with your target users (i.e. your peers with the same interest who you would likely engage with in the social network) (Figure 5.46). The purpose of the interviews/survey is to better understand what the users' needs are.

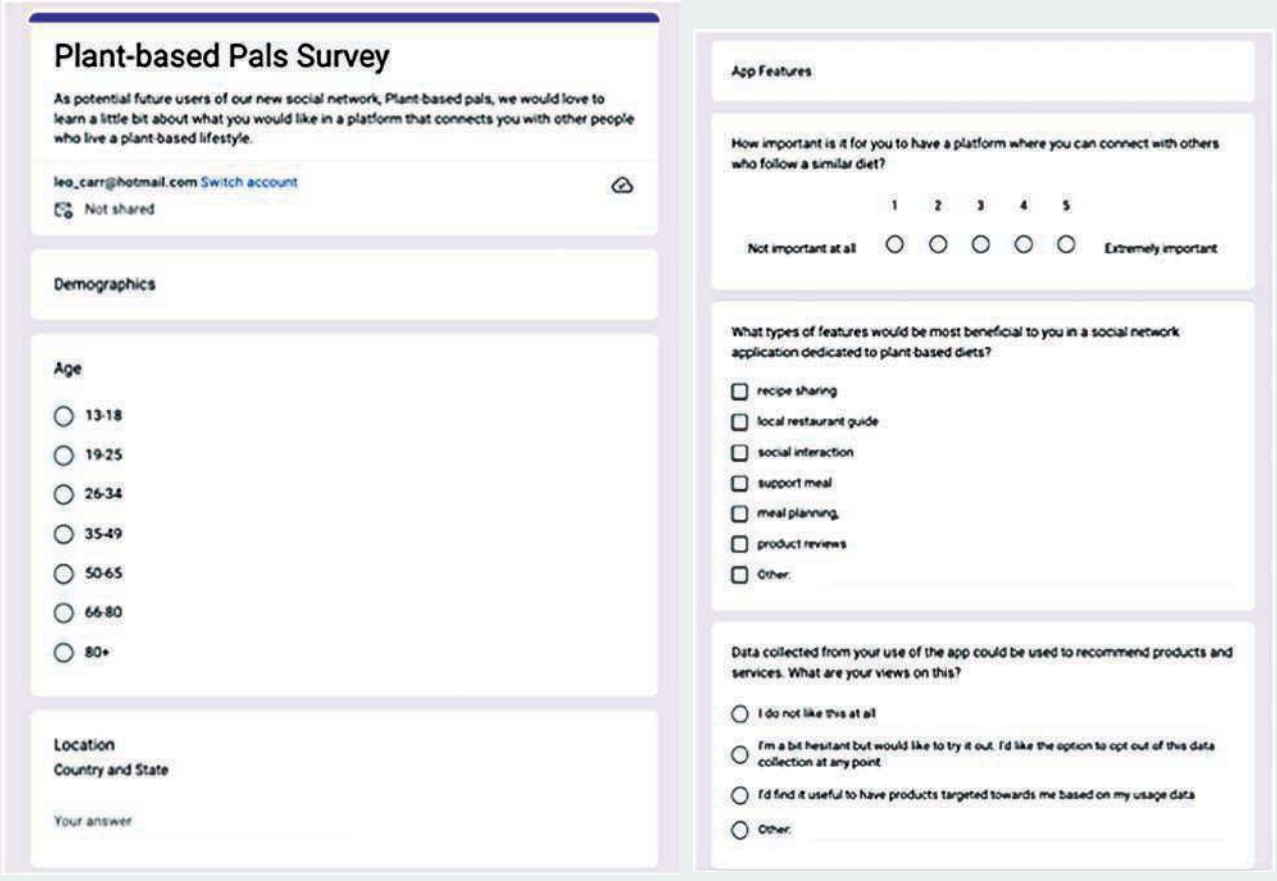


Figure 5.46 Google forms allows you to quickly create and share surveys using a range of question types to make them user friendly and to minimise invalid data entry. It also provides summaries of the collected data for fast analysis or you can export the data into a spreadsheet for further analysis.

10 Update your functional and non-functional requirements based on your research into other products and the results from your interviews/survey.

Sketching and modelling your social network idea

11 Draw a network graph to show the entities, events and relationships in your social network. Label the entities with the key attributes. For a person, for example, this would include email, name and ID. Remember to consider data security and privacy, only including data that is needed for the network's functionality; for example, if the age of the person is not relevant, do you need to collect their date of birth?

12 Sketch the interface for the main pages of the application, such as the sign-up page, profile page and the main page where the users interact. Label how a user will navigate between the pages and any other interactive features. Remember to apply your knowledge of design principles and user experience when planning the colour, fonts, page layout, components and navigation.

13 Sketch or collect any other assets you will need for your application, such as a logo, icons and photos.

Producing and implementing

Asset creation

14 Create any assets you need using an appropriate application and ensure you export in a format compatible with Adobe XD. Table 5.1 provides you with some suggestions.

Table 5.1 Application assets.

Asset	Hardware/Software	Export format
Logo	Illustrator	svg or png
'Like' icon	Illustrator	svg or png
Demo profile picture	Camera, enhanced and cropped in a lightroom	jpg
Image that might be shared (e.g. a photo of a prepared meal)	Camera, or generate using AI software such as Adobe Firefly	jpg
Map	Screenshot from Google Maps, modified in Photoshop with pins, labels, etc	jpg
Short video clip	Camera, edited in Premiere Pro	mp4 (< 25 mb)

Adobe XD prototype creation

15 Use Adobe XD or another tool like Figma to create a prototype of your app (Figure 5.47). If you have not used this software before, look at the final project in Chapter 7 for instructions or find an alternative tutorial online to follow.

- **a** Create a screen for each of the main windows, also adding the assets and including dummy data that represents typical entities, events and relationships.
- **b** Add any interactive features that can be implemented in XD, such as a like button that changes colour to indicate it is active or a scrolling window for the main social media feed. Label any other interactivity you are unable to implement.
- **c** Add the navigation between the main pages, such as clicking on the profile button to view the user's own profile and clicking on the ID of another user to view their profile.
- **d** Regularly click the Play button (top right) to view and test your working prototype.

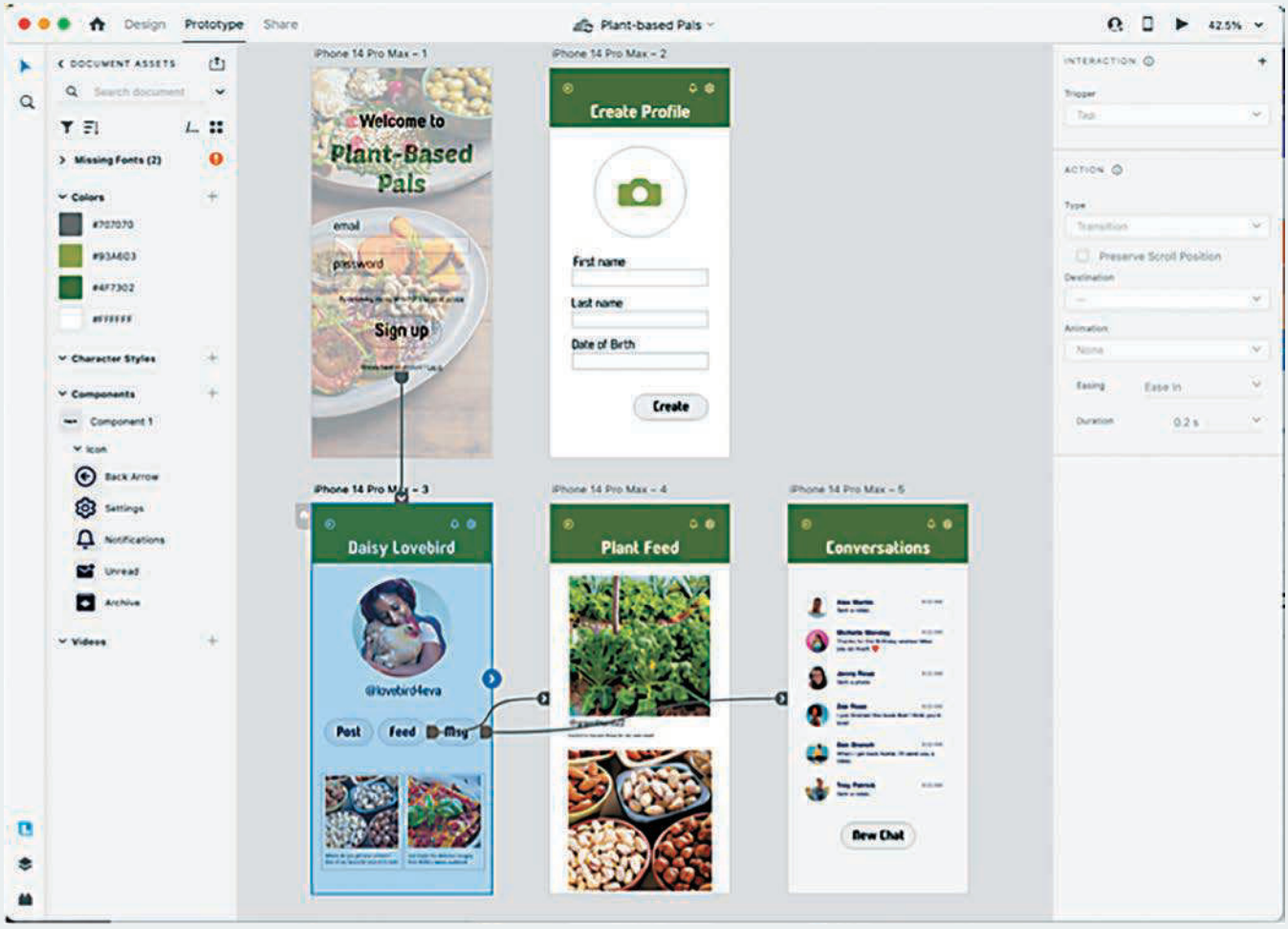


Figure 5.47 Adobe XD is a prototyping tool that allows you to design the content, layout and navigation for social media platform.

Testing and evaluating

Pitch documentation

16 Create a presentation to pitch your social network, highlighting the need, users, key functionality and benefits over existing products.

17 Investors and potential users will also want to know how personal data will be kept secure. Prepare documentation on the theoretical data security including what data is being collected, how it will be used and how it will be kept secure. Add this to the presentation.

System and user testing

18 For a final functionality test, simulate your app on your own mobile device so you and others can test it. You will need to download the Adobe XD app on your phone to do this.

19 Use the 'Share' button at the top right to create a link and share your app with your target users and other peers for feedback. Ask them to evaluate the features (e.g. can they communicate with people in their network effectively?), the attractiveness of the windows (e.g. colour and layout) and other user experience features (e.g. was it intuitive to navigate?).

20 If time allows, make any adjustments based on this final testing and feedback.

Finalising the pitch

21 Record the use of the app in a video using the screen record function on your phone.

22 Finalise your pitch, including screenshots or video recordings to best demonstrate the product and also include any positive comments you received from users during testing that emphasise the potential of your product.

Taking it further

If this project is being completed in combination with database systems, or you have completed the database systems unit prior, you should include the design of the database organisation and analysis including a schema and example SQL/GQL. You can also build a prototype of the database using Microsoft Access or another suitable database management system, adding your own test data and experimenting with some of your suggested queries.

6 Modelling computer networks

FOCUS AREA: Modelling networks and social connections

Chapter outcomes

In this chapter, you will learn:

- about the purpose and components of computer networks, including their role in modern communication
- how different network hardware devices function and contribute to the seamless flow of data
- about various transmission media and their significance in enabling data transfer within networks
- about network topologies and navigation maps, and how they impact the efficiency of data transmission
- the principles of data processing and transmission, including the concept of packet switching and relevant protocols
- how to build and program simple networks using microcontrollers, exploring their applications and potential
- about network security measures and their importance in safeguarding data from potential threats
- about the Internet of Things (IoT) and its integration of microcontrollers in creating interconnected devices.

Key terms

Bluetooth
client-server
handshaking
Internet of Things (IoT)
LAN
malware
mesh
microcontroller
navigation map
node
optical fibre
packet switching
protocol
router
server
social engineering
switch
topology
UTP
VPN
WAN

IN ACTION

Driverless car networks

Computer networks play a crucial role in the operation of driverless cars, enabling communication and coordination between various components and systems (Figure 6.1). These networks facilitate the exchange of data and commands, allowing the car to perceive its environment, make decisions and control its movements. There are networks within an individual car as well as networks between a car and other systems.

Driverless cars have complex in-vehicle networks that interconnect multiple electronic control units (ECUs) responsible for various functions like perception, decision-making and control. These networks are generally composed of a bus system and use protocols such as Controller Area Network (CAN), Ethernet and FlexRay to enable real-time communication between the ECUs. Driverless cars also rely on a range of sensors, including cameras, light detection and ranging (LiDAR), radar and ultrasonic sensors, to perceive their surroundings. These sensors generate large amounts of data that need to be processed and shared within the vehicle. Sensor networks enable the collection, organisation and distribution of sensor data to different systems for perception and decision-making.

Vehicle-to-Everything (V2X) communication is a critical aspect of driverless cars. It enables communication between the car and its external environment, including other vehicles (V2V), infrastructure (V2I), pedestrians (V2P) and the cloud (V2C). V2X networks use wireless technologies like Cellular Vehicle-to-Everything (C-V2X) and Dedicated Short-Range Communication (DSRC) to exchange safety-critical information, traffic data and other relevant updates. The cloud-based services are used for high-level decision-making, map updates, traffic information and software updates. Cloud connectivity allows the car to access real-time data and leverage cloud computing resources for advanced computations and machine learning algorithms.

Computational thinking

Table 6.1 The IPO table for a driverless car stopping at a red light.

Input	Input source	Process	Output
Traffic light status/colour	Camera, traffic light signal	If red or orange, prepare to stop	Start to turn on brakes
Distance and speed of cars in front	Distance sensor, V2V if other car is driverless	Calculate rate at which to slow down and when to stop	Brake level
Weather data	Cloud	If raining or has rained recently, brake sooner	Brake level adjusted for wet road.

Due to the critical nature of driverless car systems, network security is essential. Robust security measures are implemented to protect against cyber threats, such as unauthorised access, data manipulation and denial-of-service (DoS) attacks. Secure communication protocols, authentication mechanisms and encryption techniques are employed to ensure the integrity and privacy of data exchanged within the networks.

Systems thinking

The computer networks used in driverless cars facilitate seamless communication between the car's internal components, sensors, external environment and cloud services. These networks enable the car to perceive its surroundings, make informed decisions and execute precise control actions, all while ensuring data security and safety (Table 6.1).

Figure 6.1 V2X communication relies on creating a complex network between cars and all other systems that can influence their function, including other cars, traffic data, maps, traffic lights, weather, etc.

Questions

1. What sensors can a driverless car use to perceive their surroundings? Give an example of how each may be used.
2. What is meant by a bus system? Hint: It has nothing to do with the large vehicle that might take you to school!
3. What is a protocol? Compare the protocols CAN, Ethernet and FlexRay.
4. Apart from accessing key data for decision making, how else may driverless cars use the Cloud?
5. Create an IPO table similar to Table 6.1 but for a driverless car approaching a pedestrian zebra crossing.

6.1 Introduction to computer networks

In the simplest form, a computer network is two devices connected together. For example, a computer and a printer or a remote control and a television. These two **nodes** are sharing data over a wired or wireless transmission medium. They also follow rules, or **protocols**, that define how the data is transmitted to ensure communication is successful and the data arrives efficiently, accurately and securely. In these two examples, a computer sends the printer instructions of what dots to print on the paper via a USB cable, whereas a remote control sends the button commands for the television via infrared light signals. In reality, computer networks are generally composed of a complex arrangement of devices, commonly spread across the world.

As computer networks are part of so much of our day-to-day processes, we may not appreciate how amazing it is that data can be shared almost instantly to a precise location anywhere in the world.

In this chapter you will start to learn about computer network technology and systems with the goal of being able to create your own network solution to effectively solve a problem.

A little bit of history

In 1969, one of the first demonstrations of a computer network, ARPANET, occurred after eight years of development work.

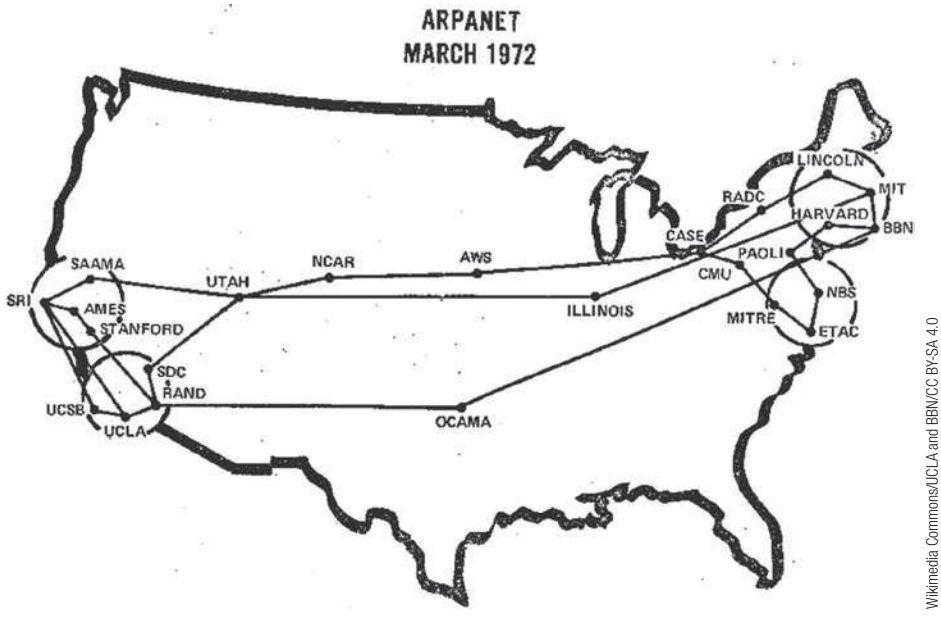
ARPANET's technology was based on the existing circuit-switching process used in the telephone network, but applied it to computer data in a process called **packet-switching**. ARPANET allowed data to be transmitted between the four main computers (or nodes) in the offices of the Advanced Research Projects Agency (ARPA) in Los Angeles, Stanford, Santa Barbara and Utah (Figure 6.2). Over the following years, many other institutions developed their own networks until these were all eventually connected to form what we now call the Internet. Today the Internet is used to transmit data to almost every corner of the world and into space. Although our main use of computer networks (the Internet) is to communicate and access content via the world wide web, there are many more uses underlying nearly all of the systems we rely on, from day-to-day processes to unexpected, critical events.

Structure and components

First, we introduce the general structure and components of a computer network (Figure 6.3), which we will look at in more detail in subsequent sections.

Nodes

Each node in a computer network is a hardware device that creates (source), transmits, forwards, redirects and/or receives (sink) the data communication. Typical hardware devices that are the source or sink of data



Wikimedia Commons/UCLA and BBN/CC BY-SA 4.0

Figure 6.2 The Advanced Research Projects Agency (ARPA) established one of the first packet-switching networks, which would eventually expand and connect with other networks to become the Internet.

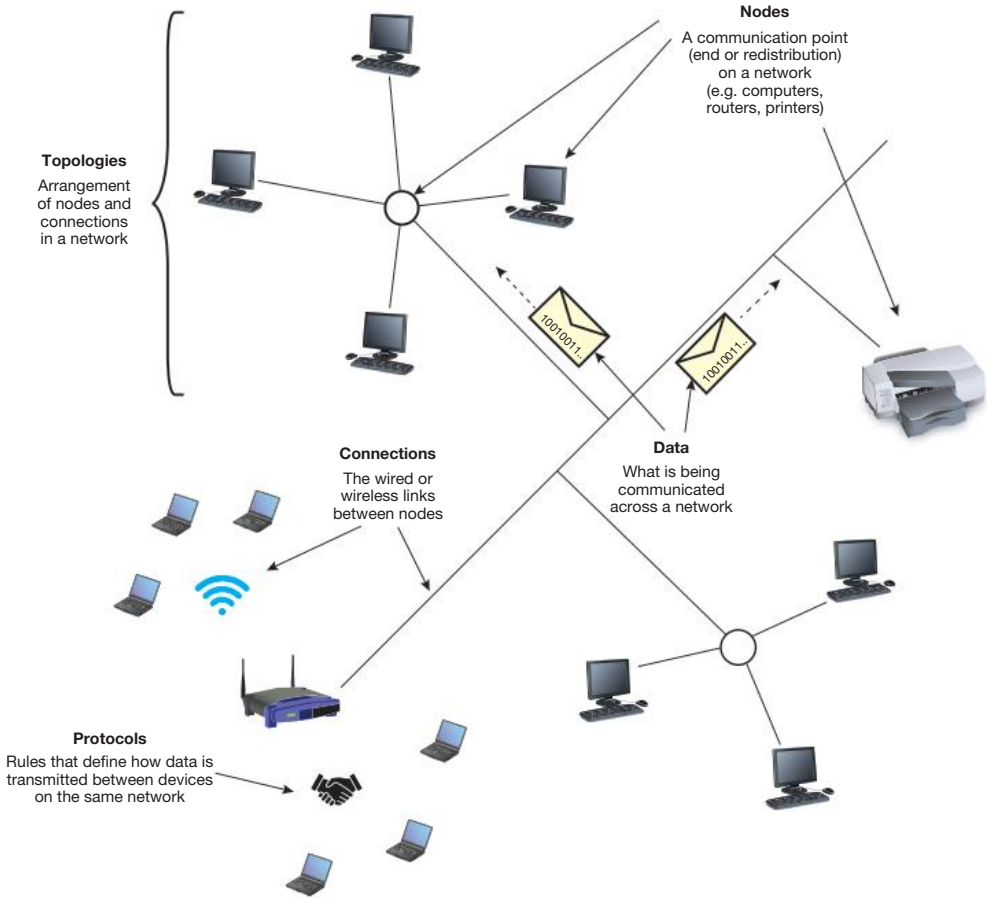


Figure 6.3 The general structure and main components of a computer network.

include computers, smartphones, **microcontrollers** and peripheral devices for input (e.g. sensors) or output (e.g. printers). Devices that allow the transmission of data through a network between the source and sinks include **routers**, **switches** and hubs.

Connection

Each node can be connected to one or more nodes in a network. Not all nodes need to be connected to every other node as this would become exponentially difficult to set up with each new device added to the network.

Instead, as long as there is a path between the source and the sink via other nodes, data can be transmitted. These connections can be wired through cables such as ethernet or **optical fibre**, or wireless such as radio waves. As data can travel in both directions along a connection, collisions can occur when computers try to communicate at the same time.

Topologies

The arrangement of the connections between nodes is called the **topology**. Different topologies are used in different contexts that consider the amount of cabling or use of wireless transmission, the level of redundancy in the possible paths data can take and the security required for the data. Bus and star topologies are common arrangements.

Protocols

Like any form of communication, rules need to be followed so all involved understand what is being communicated. Humans follow the linguistic rules for their particular language, such as using words from that language and organising them to form correct sentences. Computers follow protocols. Two devices need to agree to the protocols before communication can occur in a process called **handshaking**. Common protocols include TCP/IP, Ethernet and HTTP, however, there are many more. Some are used for quite specific communication processes.

Data

The 0s and 1s that make up the data being sent across the network need to be translated into a form for the specific medium. For example, a network cable sending data as electricity uses high or low voltage to represent a 1 or 0 bit, respectively. The data being sent also needs to be packaged in a way that allows it to efficiently and safely reach its destination and combine again with the other packets to reform the original message. Data packets, therefore, include other information such as the destination address and encryption details.

Client-server model

One of the most common models used to define a computer network arrangement is the **client-server** model. It describes how a **server** (e.g. a web server) provides resources and services to one or more clients (e.g. desktop computer, laptop, tablet or smartphone).

To connect to a server, a client sends a request that can either be accepted or rejected. If the connection is accepted, the server establishes and maintains a connection with the client over a specific protocol. This model is used in various networked applications and services, including web browsing, email, file sharing, online gaming and database management systems. It provides a flexible and efficient way to distribute and access resources in a networked environment.

LANs and WANs

LAN (local area network) and **WAN (wide area network)** are two types of computer networks that differ in terms of their geographical coverage, size and functionality (Table 6.2). Your school network is an example of a LAN (Figure 6.4).

There are other types of computer networks including WLAN (wireless LANs) and **VPNs (virtual private networks)**, which you will learn more about later.

Table 6.2 A comparison between LANs and WANs.

	LAN	WAN
Coverage	Typically limited to a small geographical area, such as a home, office building or campus.	Span large geographical areas, connecting devices across different cities, states or even countries.
Size	Usually smaller in scale, connecting devices within a close proximity, such as computers, printers and servers within a single building or group of buildings.	Can be extensive, connecting multiple LANs or other networks together.
Ownership	Commonly privately owned and controlled by a single organisation or individual.	Typically operated by multiple organisations or service providers, enabling communication between geographically dispersed locations.
Speed	Generally offer higher data transfer speeds and lower latency since they operate within a localised area.	Generally have lower data transfer speeds compared to LANs due to the longer distances and potential for network congestion.
Examples	Home networks, office networks, school networks.	The internet, global enterprise networks, telecommunications networks.

Figure 6.4 A school would use a local area network (LAN). Each desktop in a computer lab would be a client that could access server resources such as files and printers.

CASE STUDY

Smart grid

A smart grid is an advanced electrical power distribution network that incorporates microcontrollers and other intelligent devices to enhance the efficiency, reliability and sustainability of electricity delivery.

The key nodes in the network are microcontrollers. A microcontroller is a small computer on a single integrated circuit that includes a CPU, memory, clock, input/output and analogue/digital converter (ADC) (Figure 6.5).

Systems thinking

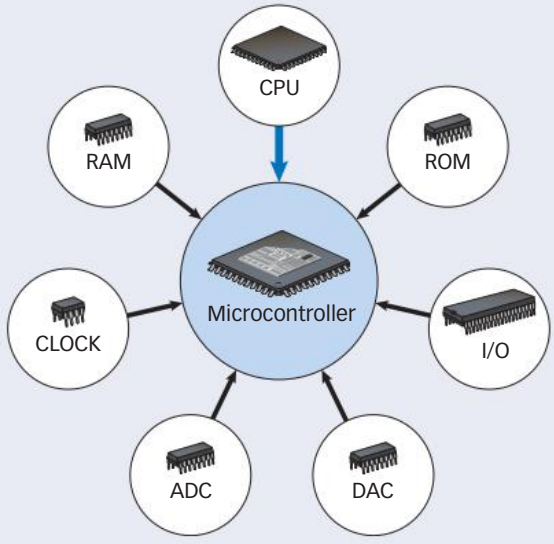


Figure 6.5 The components within a microcontroller.

Microcontrollers are embedded in various components of a smart grid, including smart meters, distribution automation devices and grid sensors. These microcontrollers enable real-time monitoring, control and communication capabilities within the grid infrastructure.

Smart meters

Smart meters are digital devices installed in households and businesses that measure and monitor electricity consumption (Figure 6.6). They are equipped with microcontrollers and communication modules to provide real-time data on energy usage. These meters enable two-way communication between the customer and the utility company, allowing for more accurate billing, load management and demand response.

Distribution automation devices

Microcontrollers play a crucial role in the automation of distribution systems within a smart grid (Figure 6.7). They are integrated into devices that monitor and control the flow of electricity. These devices can detect faults, isolate sections of the grid and restore power automatically, improving system reliability and reducing outage durations.

Wikimedia Commons/Mirko Tobias Schäfer/CC BY 2.0

Figure 6.6 Smart electricity meter.

Grid sensors

Microcontrollers are used in grid sensors and placed at various points within the electrical network. The microcontrollers process and transmit this data to control centres, allowing utilities to monitor the grid's health, identify abnormalities and take proactive measures to prevent disruptions.

Microcontrollers in the smart grid rely on communication networks to exchange data and commands. These networks can include wired systems, such as power line communication (PLC), or wireless technologies, such as Wi-Fi, cellular networks or dedicated radio frequencies. The microcontrollers use these networks to communicate with each other and with central control systems, including energy management systems that collect, organise and analyse data from various sources in the smart grid. These systems use algorithms and control strategies to balance electricity supply and demand, optimise energy distribution and facilitate integration of renewable energy sources.

There are many cyber security risks associated with the use of a smart grid system, such as unauthorised access, malware and DoS attacks. The mitigation of these needs to be prioritised to safeguard the smart grid infrastructure from power outages that could cause huge disruptions, cost millions of dollars and potentially lead to catastrophic events. The privacy and security of the personal data of the electricity customers also needs to be maintained. Strong access controls, encryption, network segmentation, intrusion detection systems, regular security assessments and employee training are some cyber security measures that should be employed.

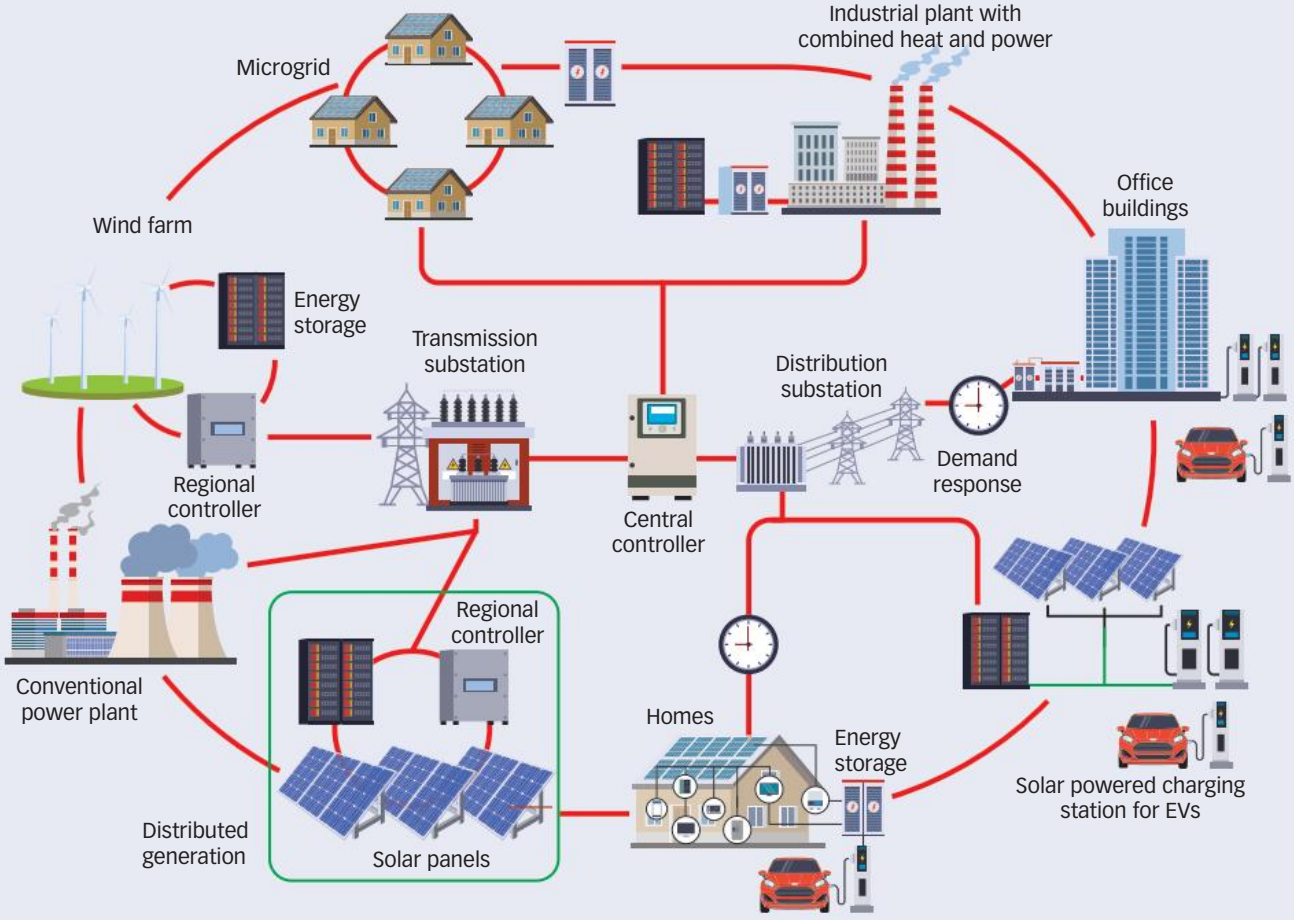


Figure 6.7 The smart grid is a complex network of microcontrollers and other network technology. By utilising real-time data, automation and communication capabilities, the smart grid improves grid management, reduces energy losses, supports renewable energy integration and enables consumers to make informed decisions about their energy usage.

SKILL BUILDER

Explore a microcontroller – micro:bit

The micro:bit is a microcontroller development board created by BBC to provide students with a hands-on tool to learn about physical computing and programming (Figure 6.8). The micro:bit includes a range of inbuilt inputs and outputs as well as the option to extend the functionality through I/O pins and edge connectors. The pins along with the USB connector and radio antenna also allow us to create wired and wireless networks with other micro:bit nodes, serving as a practical learning tool for understanding computer networks.

In this Skill builder, you will explore some of the features of the micro:bit to become familiar with the components, coding environment and set up. There are more micro:bit Skill builders throughout this chapter. As this chapter is not focused on programming, you will mostly be provided with the code for each of the Skill builders. However, it is expected you have or are developing some understanding of computational thinking, so should be trying to understand how the code works, adding comments to demonstrate this. In chapters 12 and 13 you will have the opportunity to build and code more complex devices using the micro:bit.

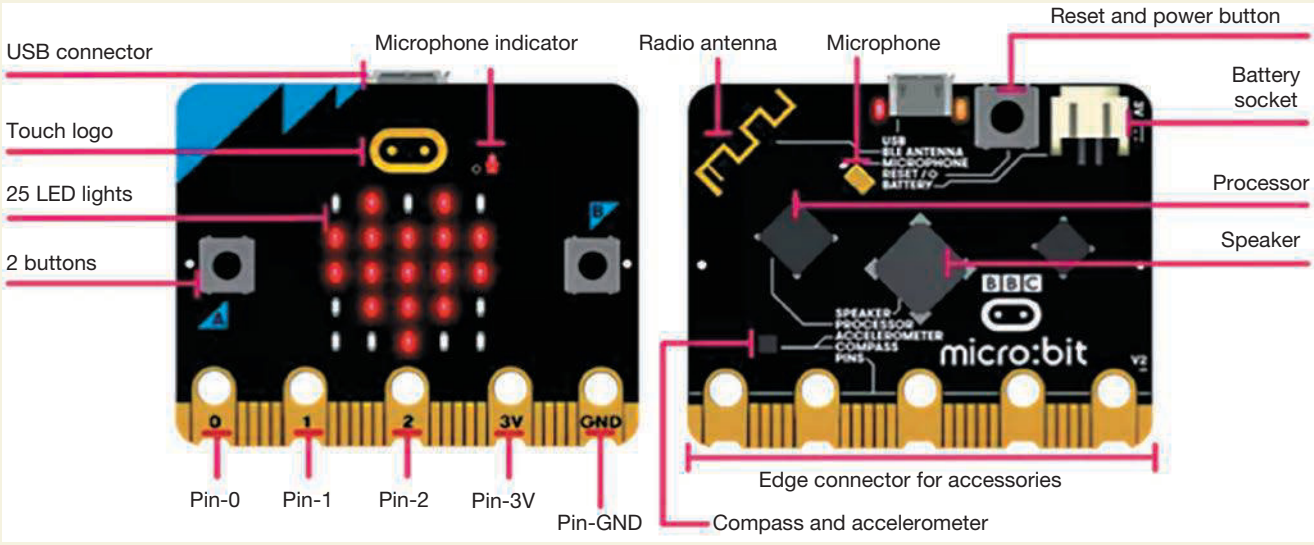


Figure 6.8 The micro:bit is an educational microcontroller that includes many built-in components, as well as pins to allow expansion for more functionality. Look at your micro:bit and identify all the components in the diagram.

There are many coding environments available for the micro:bit, including visual programming (e.g. Microsoft MakeCode) and coding in general-purpose languages, such as Python and JavaScript. In this activity we will use the micro:bit Python editor (Figure 6.9).

1. Go to the micro:bit Python editor.
2. Run the default code in the simulator on the right. What does it do?
3. Using the code reference library on the left, attempt to create your own simple program that takes an input and produces an output. For example, when you press a button, something is displayed on the LED matrix. The reference library has example code that you can copy and paste into your file.
4. First run the program in the simulator to test, then connect your micro:bit via USB to your computer and press the 'Send to micro:bit' button. This will flash your program to the micro:bit so you can test your program on the physical device.
5. Explore more of the reference library and apply some of the general programming principles covered in Chapter 4 to create a more complex system.

Weblink
micro:bit Python Editor

Computational thinking

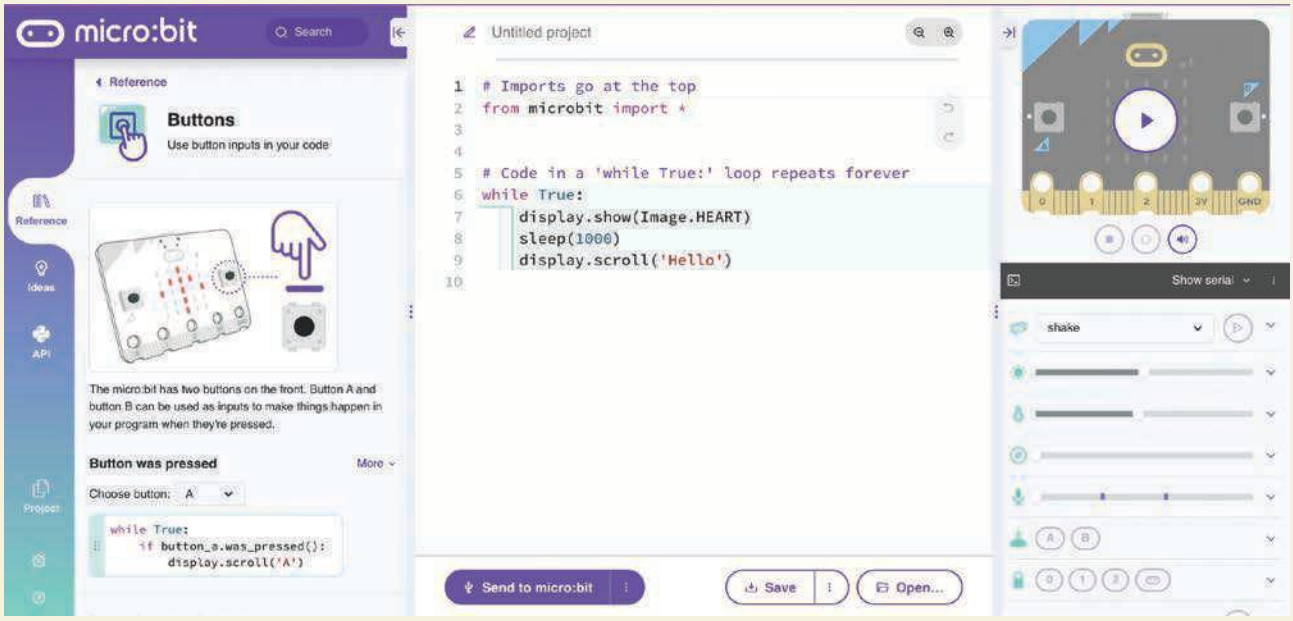


Figure 6.9 The micro:bit Python editor provides a user-friendly interface for coding your micro:bit. It includes a reference library for easy code creation (left panel) and simulator for testing prior to installing on your device (right panel).

ACTIVITIES

Identify

1 What is the purpose of a computer network?
2 Define the main components of a computer network.
3 What is the difference between a LAN and a WAN?
4 List the inputs and outputs on a micro:bit.

Analyse

5 How would a smart grid electricity network benefit individuals and the environment? What are the risks of implementing a smart grid network and how should these be reduced?
6 Consider two networks, a LAN and a WAN, used for a similar purpose (e.g. gaming). What are the advantages and disadvantages of each?

Extend

7 Your school network will apply the client-server model. Explain this arrangement in your school context. What devices are the clients? What servers does the school have and what resources and services do they provide? How do you connect to the server?

6.2 Network hardware

In this section we will look at the key hardware used in most home networks and LANs.

Hubs

An early networking solution was to connect all the computers to a central device called a hub (Figure 6.10). When a hub receives a message, it transmits it to all linked computers. Hubs also act as repeaters. A repeater strengthens signals travelling over large distances. The problem with a hub is that it is likely to cause collisions between data because all the computers can transmit in unison (Figure 6.11).

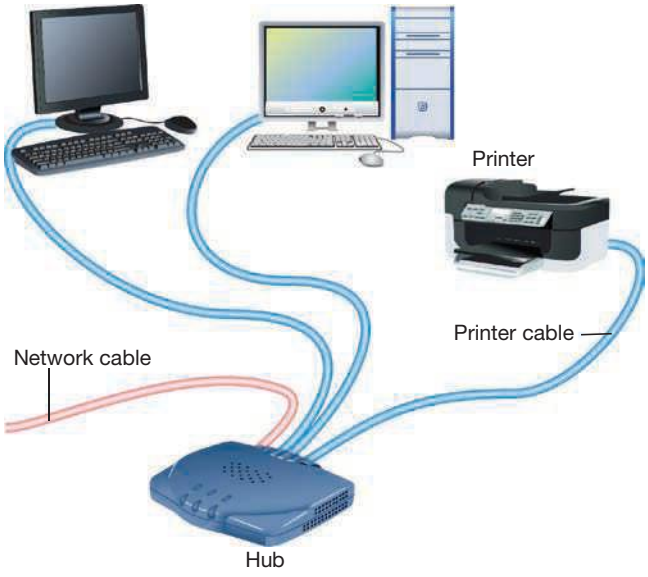


Figure 6.10 A hub is a device that connects a small number of network devices.

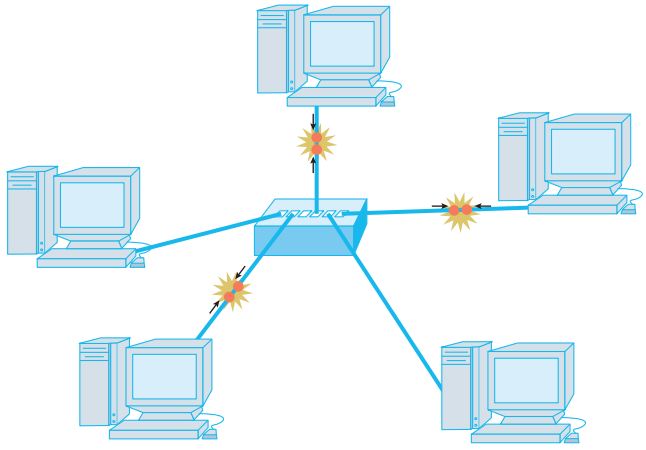


Figure 6.11 Computers can transmit to the hub at the same time, which means data collisions are likely to occur.

Switches

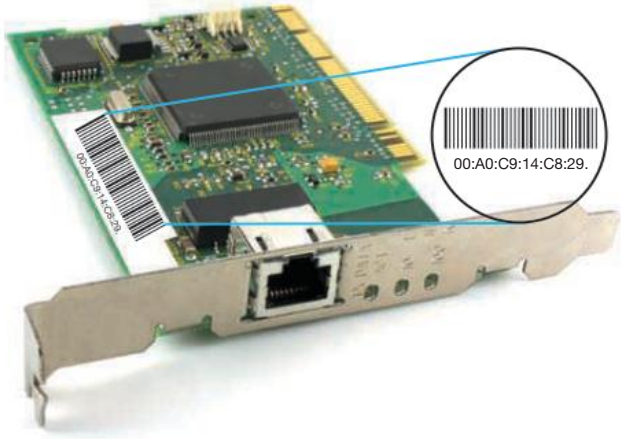
A network switch (Figure 6.12) possesses an important advantage over a hub. A switch can look at an address being sent to it and identify the correct destination port or computer rather than transmitting it to all connected computers as a hub does.

A switch dramatically reduces the problem of data collisions as they now only occur when a computer and its switch try to talk to each other at the same time. If this happens, a collision is detected and retransmitted after a short delay. Switches have now largely replaced hubs in networks. Switches work by keeping a table of addresses for each connected device.

Shutterstock.com/Anucha Cheechang

Figure 6.12 A network switch.

Each device has its own network interface card (NIC) that is numbered with a media access control (MAC) address (Figure 6.13). MAC addresses are also known as ethernet addresses. Connecting computers with switches solves the problem of a room or two of computers needing to communicate. But how do we communicate with a computer in another building, city or country? How does a switch know which of billions of connected devices it needs to connect to?



Shutterstock.com/DeSerg

Figure 6.13 A network interface card.

Routers

The problem of how vast amounts of data are communicated is solved by routers (Figure 6.14). Routers keep track of which device a data packet has come from as well as the device it should send it to next, based on stored MAC addresses.

Shutterstock.com/lucadp

Figure 6.14 A typical wireless router.

The most familiar type of router connects the devices in your home to the internet through an internet service provider (ISP) (Figure 6.15). Routers may be wired or wireless. You probably have a wireless router in your home. There are also routers connecting cities and countries at high speed along the fibre optic cables of the internet. The Internet is made up of nearly 50 billion interconnected devices.

How do routers perform the seemingly impossible task of sending a message quickly and accurately and decide the best device to send it to for its next hop on the journey? The answer is packet-switching, which we will look at in section 6.5.

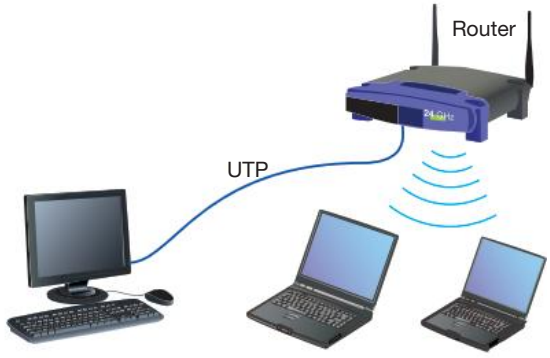


Figure 6.15 A home wireless router can provide access to the Internet for several computers.

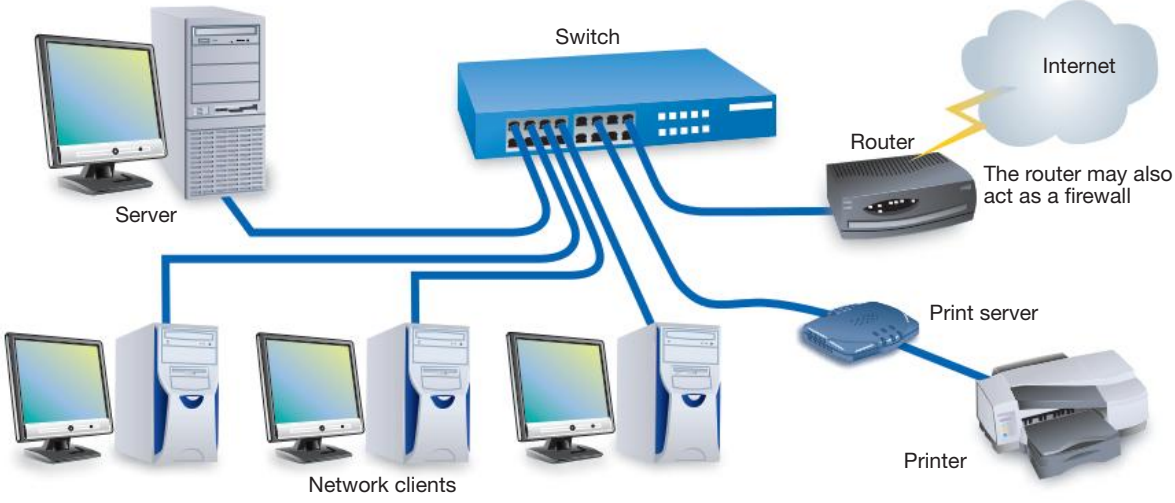


Figure 6.16 A typical network for a small business.

Wireless access points (WAPs)

A wireless access point (WAP) enables wireless devices to connect to a wired network (Figure 6.17). It acts as a central point of communication for wireless devices, such as laptops, smartphones and tablets, allowing them to access a LAN or connect to the internet wirelessly. The WAP receives data packets from wireless devices and transmits them to the wired network and vice versa.

Figure 6.17 An Aruba WAP in a high school classroom has the capacity to connect up to 50 devices with high-speed access to the school network.

Servers

A server is a computer or a system that provides resources, services or functionality to other devices or programs (clients) within a network (Figure 6.18). Servers are designed to handle and respond to requests from clients, facilitating data storage, communication, computation or other specific tasks. Some examples of servers are file servers, print servers, web servers, email servers and database servers.

File server

A file server is dedicated to storing and sharing files and data within a network. It allows users to access and share files centrally, facilitating collaboration and efficient data management.

Print server

A print server manages network printers and facilitates print job processing and distribution. It allows multiple users to send print jobs to shared printers, handles print queues and manages printer resources.

Web server

A web server hosts websites and serves web pages to client devices over the internet or intranet. It handles requests from web browsers, processes web applications and delivers web content to users.

Email server

An email server manages email communication within a network. It handles the storage, sending, receiving and retrieval of email messages.

Database server

A database server stores and manages databases within a network. It allows users and applications to access, retrieve and manipulate data stored in the databases. Database servers handle database queries, transactions and data integrity.

Shutterstock.com/Hadrian

Shutterstock.com/a4ndreas

Wikimedia Commons/Florian Hirzinger/CC BY-SA 3.0

Figure 6.18 Depending on the storage, processing and data traffic demands servers can come in all shapes and sizes, from just a single small computer like the Mac mini to arrays of computers stacked together to form huge server farms. Servers can also be set up on part of an existing computer in a system called a virtual server.

Smart hubs

Although they are named after the older networking technology superseded by switches, smart hubs are used in many modern networks, particularly in our homes. More than just hubs, they are specialised computers on the network that connect and communicate with smart devices (Figure 6.19).

Smart hubs act as a central control point for managing and coordinating the various smart devices within a home. These hubs typically have processing power, memory and connectivity capabilities to facilitate communication between different devices. They interact with the smart devices using various communication protocols, such as Wi-Fi, **Bluetooth**, Zigbee or Z-Wave, depending on the devices' compatibility. The hub serves as a bridge or gateway, enabling these devices to work together and be controlled through a single interface or mobile app.

iStock.com/palliki

Figure 6.19 The Genius Hub is at the centre of a smart heating system that includes a mesh network of devices that communicate using Z-wave protocol.

SKILL BUILDER

Design a network for a cafe – Part 1

'Bryce's Brew' is a new cafe opening up in your area. You have been asked to plan and implement the computer network they and their customers will use for a variety of day-to-day processes (Figure 6.20).

1 Brainstorm the variety of computing technology used by a cafe. This could include technology used by:
   a. the cafe workers: to serve customers, keep track of orders and receive payment
   b. the customers: to make orders, use their own private devices or for entertainment (television, music)
   c. the cafe owner: to manage the cafe including keeping track of stock, security and ensuring the cafe environment is inviting for customers.

Shutterstock.com/Ground Picture

iStock.com/MStudioImages

Getty Images/manusapon kasosod

iStock.com/WhiteLacePhotography

Figure 6.20 A modern cafe relies on a variety of computing technology including networking hardware for day-to-day processes.

2 Considering all of the above, what network hardware would the cafe need? Complete Table 6.3.

Table 6.3

Hardware	Needed?	Purpose in cafe	Type / Key specifications
Hub	☐		
Switch	☐		
Router	☐		
Server	☐		
WAP	☐		
Smart hub	☐		

3 Perform your own research online to find current network products that may be used in the cafe network. Using spreadsheet software, create a budget for the network hardware.

4 Add to the table and the budget any additional hardware you will need to purchase that will be other nodes in the network, such as the transaction processing terminals, iPads for taking orders, television for customers to watch music videos, the office printer, Wi-Fi controlled air conditioning, security cameras, etc.

ACTIVITIES

Identify

1 How do routers function in a network?

2 What is the difference between a hub and a switch?

3 Why do devices on a network need a NIC?

Analyse

4 How do switches avoid data collisions?

5 What type of WAPs are in use in your school? Look up their specifications including what frequencies they operate on, their advertised speed and how many connections they support.

6 Why do you think smart hubs use the term 'hub' in their name?

Extend

7 Find out about the different servers used in your school network. Where are they located? What is the purpose of each?

6.3 Transmission media

We have learnt about the nodes of a network and now we look at how those nodes are connected, either physically through wires or through wireless technology. We will also look at how each connection method achieves fast, error-free communication through the use of protocols.

Wired transmission

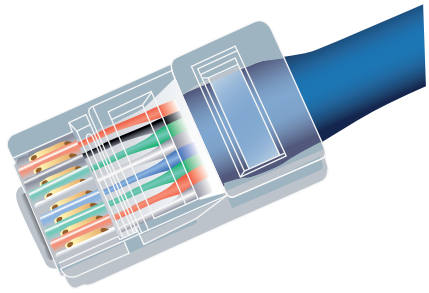
The main forms of wired transmission media used in computer networks send data using electrical energy or light energy.

Unshielded twisted pair (UTP) cable

Arguably the most widely used LAN connection is the clearly recognisable blue Ethernet cable.

This cable has four twisted pairs of copper wires surrounded by an outer jacket (Figure 6.21). Electrical energy is conducted through the wire to send data. By twisting the wire pairs and further weaving the pairs together, the interference from the data signal is reduced. These cables are used to connect each computer, WAP or other network device to a switch or router on the network. Standard **UTP** cabling can support 100 Mbps and Gigabit Ethernet can support 1000 Mbps.

An RJ45 connector for UTP cable. This is a standard type of connector for network cards.



UTP cables contain four twisted pairs of wires, each pair twisted around the others like a rope, to reduce interference with the signal on the wire. Each wire is colour-coded to make it easy to assemble the RJ45 connectors.

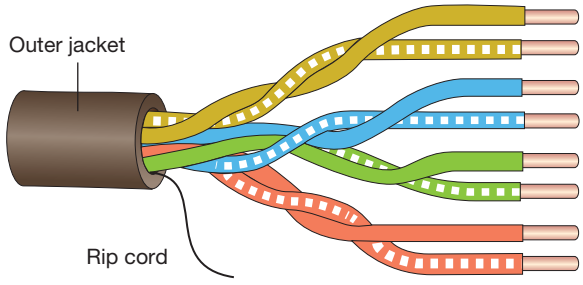


Figure 6.21 Unshielded twisted pair cables connect easily to other network devices with the RJ45 connector.

Coaxial cable

Coaxial cables are another electrical energy transmission medium. However, these cables are commonly used in cable television and broadband networks.

They consist of a central conductor wire surrounded by insulation, a metallic shield and an outer protective layer. Coaxial cables can transmit data over longer distances compared to twisted pair cables and provide better resistance against external interference.

Optical fibre

In recent years coaxial cables have largely been replaced by optical fibre technology. Optical fibre cables use strands of glass fibres as thin as a human hair to transmit data as pulses of light (Figure 6.22).

Optical fibre offers extremely high bandwidth (bps) and is known for its ability to transmit data over long distances without significant loss or degradation. Optical fibre cables are widely used in high-speed networks, including long-distance telecommunications and high-capacity data centres. In Australia, nearly all homes and businesses are connected to the Internet via the National Broadband Network (NBN), a high-speed optical fibre network, which gradually replaced the copper-wire network between 2011 and 2020.

iStock.com/STRINGERimage

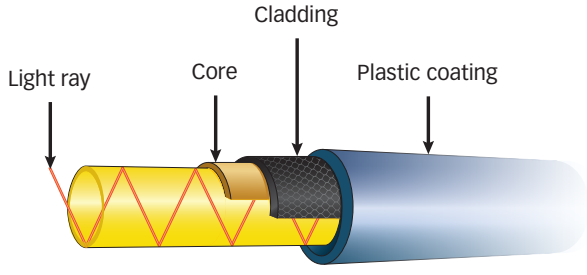


Figure 6.22 Optical fibre cable.

Skill builder resource
Build a wired micro:bit network

SKILL BUILDER

Build a wired micro:bit network

Resources

2 micro:bits, 2 power packs, 2 alligator cables, personal computer, micro:bit code environment (e.g. micro:bit Python Editor)

Introduction

In this Skill builder you will build a simple wired network between two micro:bits. The micro:bits will be connected via electrical jumper cables using the expansion ports. The cables are made of copper wire, just like Ethernet. The 1s and 0s are pushed out as high or low voltages via the ports.

For example, a number might be transmitted as a single byte of eight bits (e.g. decimal number 72 = binary 01001000) (Figure 6.23).

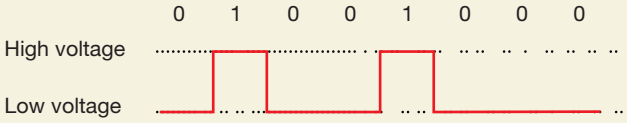


Figure 6.23 The pattern of high and low voltages used to send the decimal number 72 (in binary).

Letters or other character might be encoded using a system like Unicode or ASCII (the ASCII encoding system is actually a subset of Unicode). For example, to send the word 'Hi' you would transmit two bytes of data, one for each ASCII letter code (Figure 6.24).

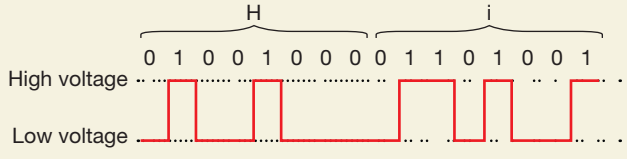


Figure 6.24 The pattern of high and low voltages used to send the word 'Hi' (in binary).

Wired network 1: 2 nodes, 1 direction (transmitter to receiver)

1 Set up your micro:bits as in Figure 6.25. Connect pin 1 of the transmitter micro:bit (green) to pin 0 of the receiver micro:bit (red). Complete the circuit by connecting the ground pins from both micro:bits to each other.

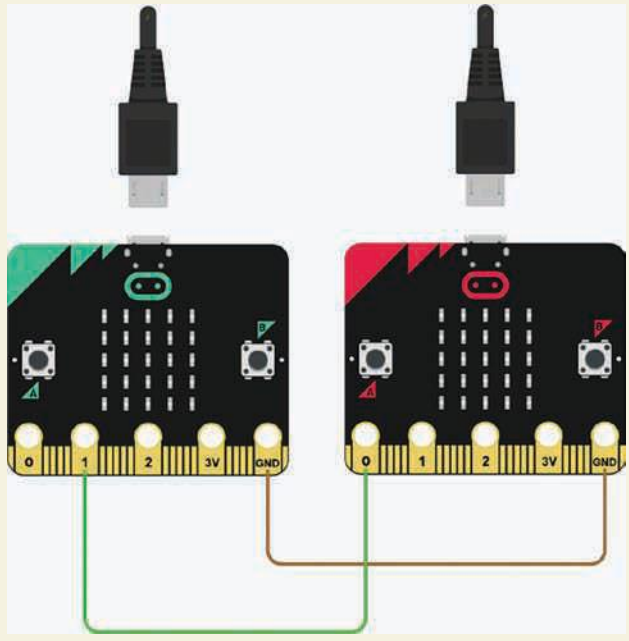


Figure 6.25 Wired network (2 nodes, 1 direction) of two micro:bits, a transmitter and a receiver.

2 Create the code for the transmitter micro:bit, which sends a '1' if button 'b' is pressed, otherwise it sends a '0'. When it sends a '1' it also displays an 'up' arrow on its LED array to indicate it is transmitting data (Figure 6.26).

3 Test the code using the micro:bit simulator in the coding environment.

Figure 6.26 Coding the transmitter micro:bit in the micro:bit Python editor.

4 Open a new tab and create code (Figure 6.27) for the receiver micro:bit, which is constantly checking its pin input for a '1'. If received, it will indicate this by displaying a 'down' arrow on its LED array indicating it is receiving data.

```
from microbit import *

while True:
    if pin0.read_digital() == 1:
        display.show(Image.ARROW_S)
    else:
        display.clear()

```

Figure 6.27 Python code for the receiver micro:bit.

5 Download the code to the corresponding micro:bits and test using the physical micro:bits (Figure 6.28).

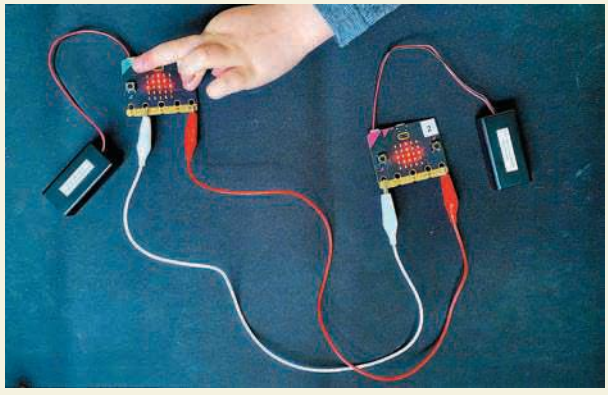


Figure 6.28 Micro:bit 1 transmitting to micro:bit 2.

Wired network 2: 2 nodes, 2 directions

We will now allow both micro:bits to transmit and receive data.

6 Add an additional cable to the two-node network, this time connecting pin 1 of the second micro:bit to pin 0 of the first micro:bit (Figure 6.29).

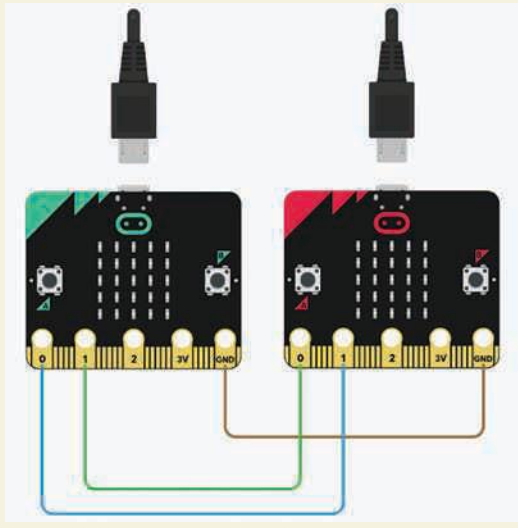


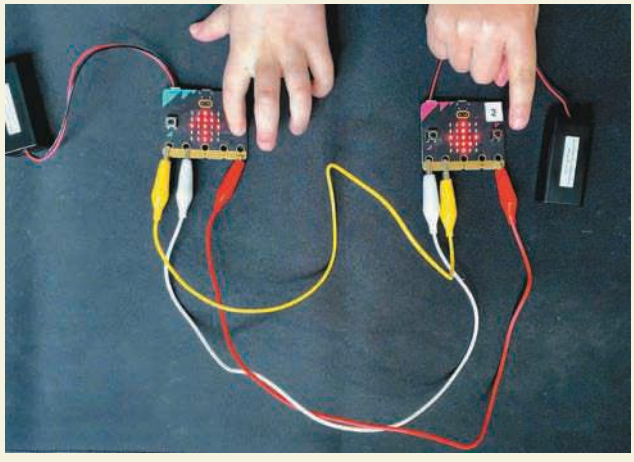
Figure 6.29 Wired network (2 nodes, 2 directions) of two micro:bits that both act as transmitters and receivers.

7 Combine the transmit and receive code and download a copy on each micro:bit for testing (Figure 6.30).

```
from microbit import *

while True:
    if button_b.is_pressed():
        pin1.write_digital(1)
        display.show(Image.ARROW_N)
    elif pin0.read_digital() == 1:
        display.show(Image.ARROW_S)
    else:
        pin1.write_digital(0)
        display.clear()

```

Figure 6.30 Python code for the micro:bit two-direction wired network.

8 Experiment with the network (Figure 6.31). What happens when you try to send data from each micro:bit at the same time?



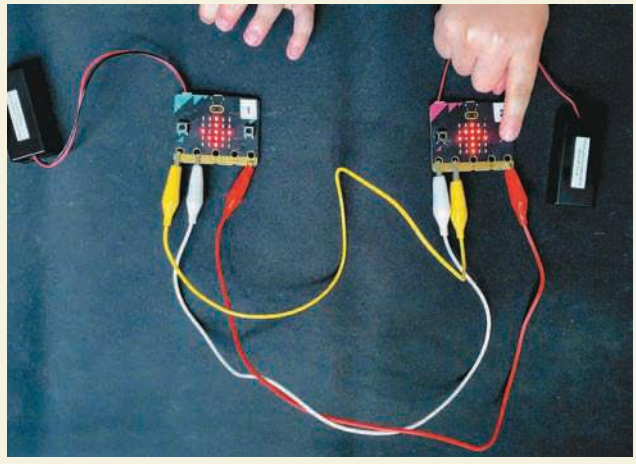


Figure 6.31 Micro:bit 1 and 2 can send and receive the digital signal as indicated by the display arrows.

Wireless transmission

Today, most individuals and businesses rely on the flexibility and convenience of wireless network connection. For this, data needs to be transmitted through air using electromagnetic energy.

Most technologies use radio-wave frequencies within the electromagnetic spectrum (Figure 6.32). Different frequencies are used for different purposes depending on the required speed, distance and reliability of the transmission.

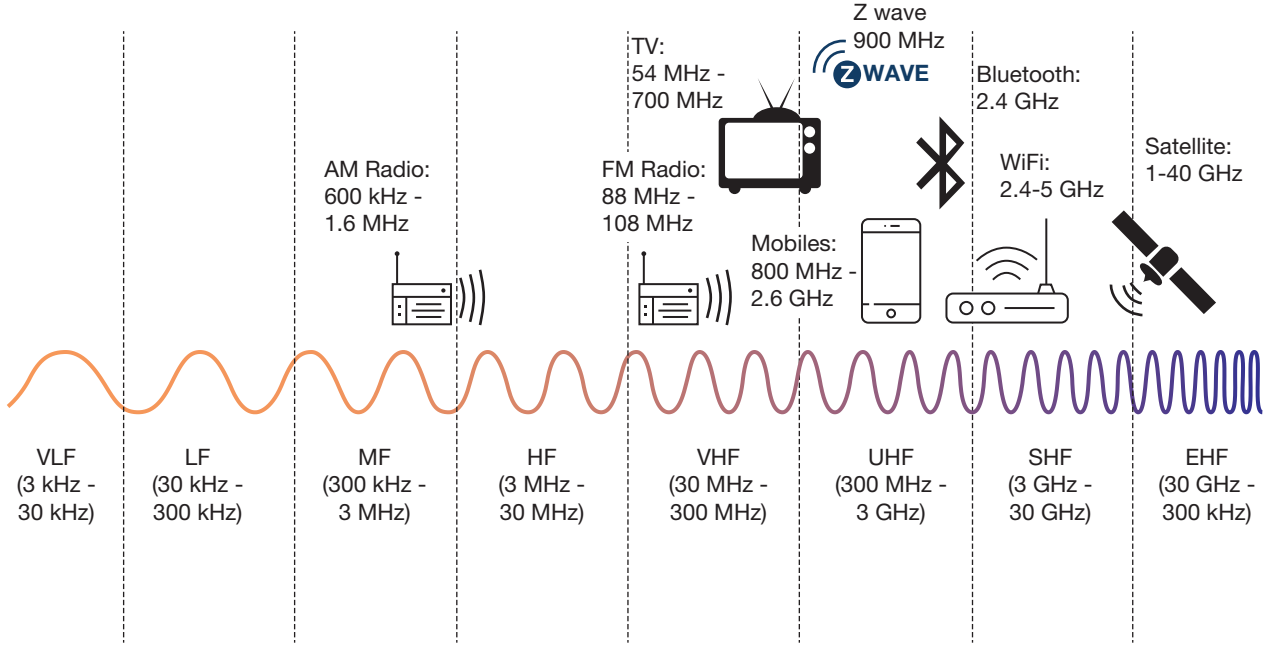


Figure 6.32 The different radio frequencies used for wireless data transmission.

Wi-Fi

Wi-Fi is used to provide high-speed internet access to many devices without the need for a wired network. The technology has become ubiquitous, providing wireless connectivity in homes, schools, offices, public spaces and various industries. It has revolutionised the way we access the Internet, connect devices and enable wireless communication, offering convenience, flexibility and mobility. Wi-Fi technology uses radio waves to transmit data, operating mainly on the 2.4GHz and 5GHz frequency band.

Cellular networks

Mobile phone networks also use radio waves to send data. As the wireless signals from individual phones only reach a limited distance to connect to another phone, the signal needs to be received by a mobile phone tower, which is then connected to the vast cellular network composed of wired and wireless connection (Figure 6.33).

The various generations of cellular networks have gradually improved the data transfer rates, latency (reduced delay in data transmission), network capacity (ability to handle more connected devices simultaneously) and reliability. For example, 5G introduced new radio (NR), a new air interface technology for improved data transmission efficiency and millimeter wave (mmWave), a technology that allows it to operate in high-frequency bands, which offer large bandwidths and the potential for extremely high data transfer rates (Figure 6.34).

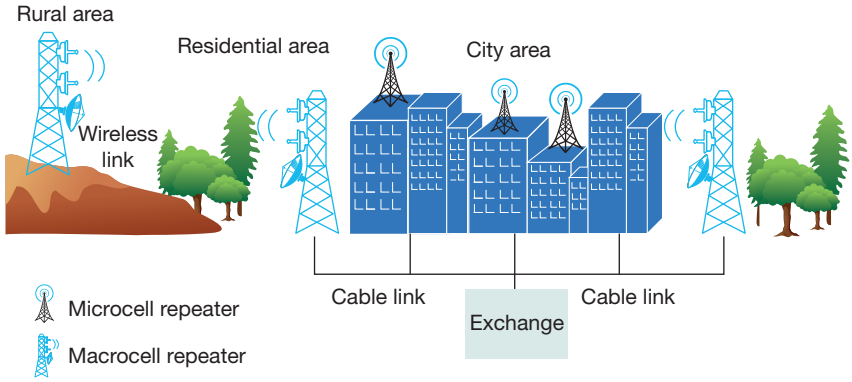
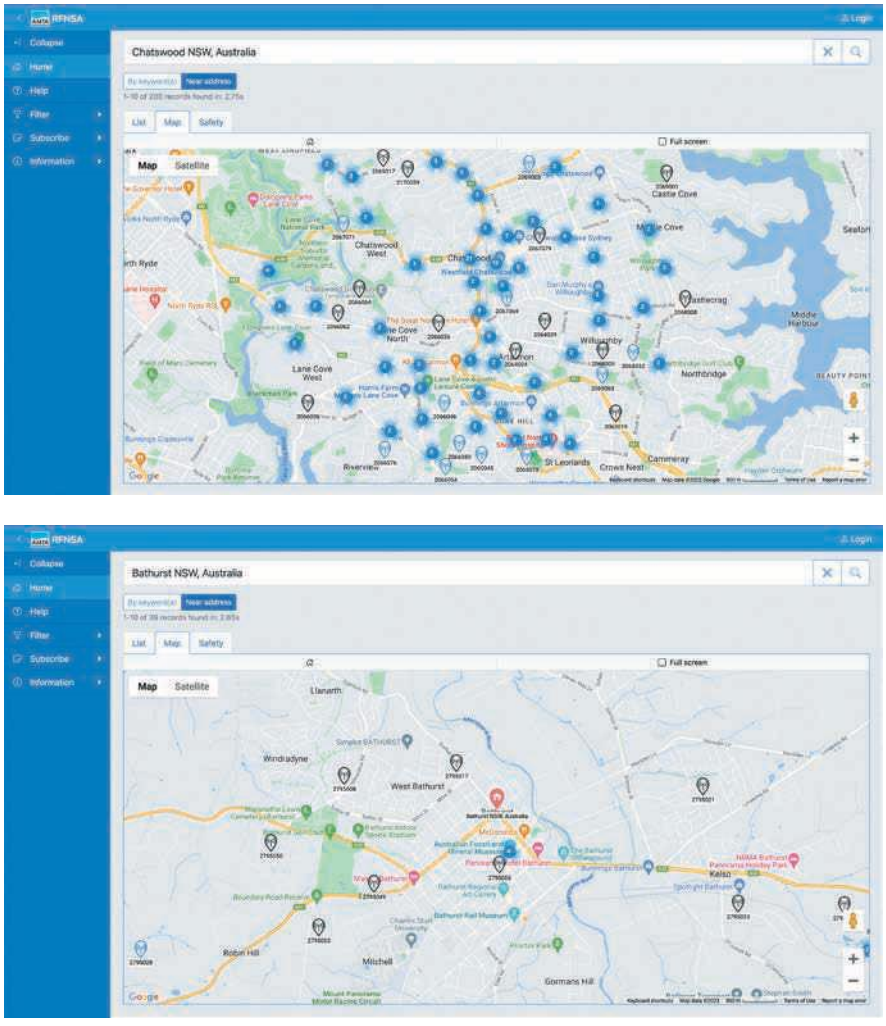


Figure 6.33 The mobile phone network.



Source: https://www.rfnsa.com.au/

Figure 6.34 A typical mobile base station can handle about 160 voice channels. Large buildings can limit coverage. Hence, city areas with high-rise buildings and large population density need significantly more stations than country areas. The 20 km radius around Chatswood has 200 base stations while Bathurst only has 39 for the same physical area.

Satellite

It is not always economical to run cabling to a node, particularly in rural or remote areas. In these situations, satellite communication is used. Satellite communication relies on a network of satellites in orbit around the Earth, which receive and transmit data signals on radio waves.

To access the Internet via satellite, you need a satellite dish installed at your location with a clear view of the sky. The dish is equipped with a transmitter and receiver, which communicates with satellites in orbit. The disadvantages of satellite communication are that it has a noticeable delay in signal transmission, known as latency, small bandwidth due to the limited allocated frequency spectrum and the performance can be impacted by weather conditions.

Starlink is a satellite Internet constellation launched by SpaceX in 2019. The constellation consists of thousands of small satellites in low Earth orbit (LEO), working together to create a network that can deliver high-speed and low-latency Internet access to regions where traditional Internet infrastructure is lacking or impractical to deploy.

Bluetooth

Bluetooth enables short-range data exchange between devices such as your phone to headphones, speakers, remote-controlled devices or your computer for quick file sharing.

Bluetooth is designed to be power efficient to reduce the battery requirements for wireless communication in portable, battery-powered devices. It typically allows connection between devices around 10 metres apart.

Zigbee, Z-Wave and Matter

Zigbee and Z-Wave are short-range wireless communication technology that connect devices used in home automation and **Internet of Things (IoT)** devices (Figure 6.35). They also focus on low power consumption, enabling devices to operate on battery power for extended periods.

Matter is a new, open-source protocol based on IP that aims to unify the various IoT devices, allowing compatibility between different products.

All these technologies support what is called '**mesh** networking', which is a type of network architecture where multiple devices or nodes work together to create a decentralised network. You will learn more about mesh networks later.

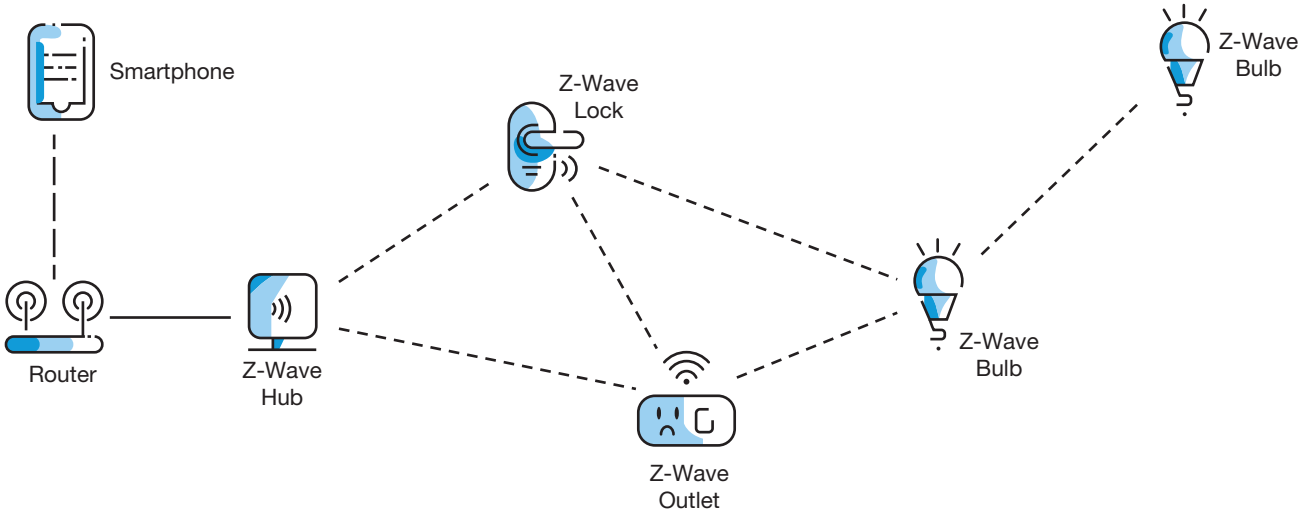


Figure 6.35 Devices on a smart home network generally use low power wireless transmission technology like Z-wave, which creates a mesh network. Individual smart devices in a mesh network act as repeaters of the signals, extending the range of the hub. The smart bulbs in this network are not within range of the hub but they are still able to transmit and receive data signals to it via other devices in the mesh network.

Skill builder resource
Build a wireless micro:bit network

SKILL BUILDER

Build a wireless micro:bit network

In this activity you will create a simple wireless network between 2 micro:bits. Micro:bits transmit via radio waves between the frequencies of 2.402 and 2.480 GHz. You will use wireless communication first to repeat the previous Skill builder wirelessly and then to create a connected game of scissors, paper, rock.

Wireless micro:bit communication

1 Remove all crocodile clip wires (if still connected from the previous Skill builder). All you need is a computer and two micro:bits with battery power (Figure 6.36).

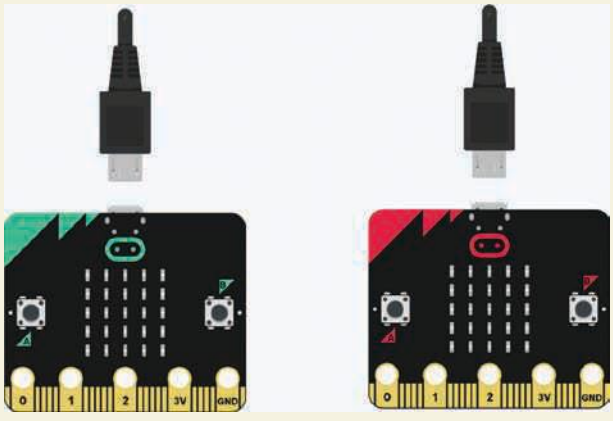


Figure 6.36 Wireless network (2 nodes, 2 directions) of two micro:bits that both act as transmitters and receivers.

2 Modify your previous transmit and receive code for wireless communication (Figure 6.37):

- a add code to import the micro:bit's radio module (line 2)
- b set the same radio group on both micro:bits (line 3)
- c check if messages have been received at the start of our loop (line 6)
- d replace the previous write instruction with a radio send instruction (line 8)

```
1  from microbit import *
2  import radio
3  radio.config(group=23)
4
5  while True:
6      msg = radio.receive()
7      if button_b.is_pressed():
8          radio.send('1')
9          display.show(Image.ARROW_N)
10     elif msg:
11         display.show(Image.ARROW_S)
12     else:
13         display.clear()
```

Figure 6.37 Python code for the micro:bit two-direction wireless network.

3 Download this code to each micro:bit for testing.

4 Experiment with the network. What happens when you try to send data from each micro:bit at the same time?

A wireless scissors, paper, rock game

How it will work:

- Two players in the game, each will have a micro:bit.
- When a player shakes their micro:bit it will trigger a random selection of scissors, paper or rock on the micro:bit.
- Each micro:bit will display the selection on the LED matrix and transmit it via radio to the other micro:bit.
- The micro:bits will determine whether they have won based on the data they receive in the transmission and the rules of scissors, paper, rock.
- After the selection has been displayed for at least two seconds, the LED matrix will change to show whether the player has won, lost or drawn.
- The game can be reset by pressing the 'a' button on the micro:bit.

5 Code the micro:bit as in Figure 6.38.

Computational thinking

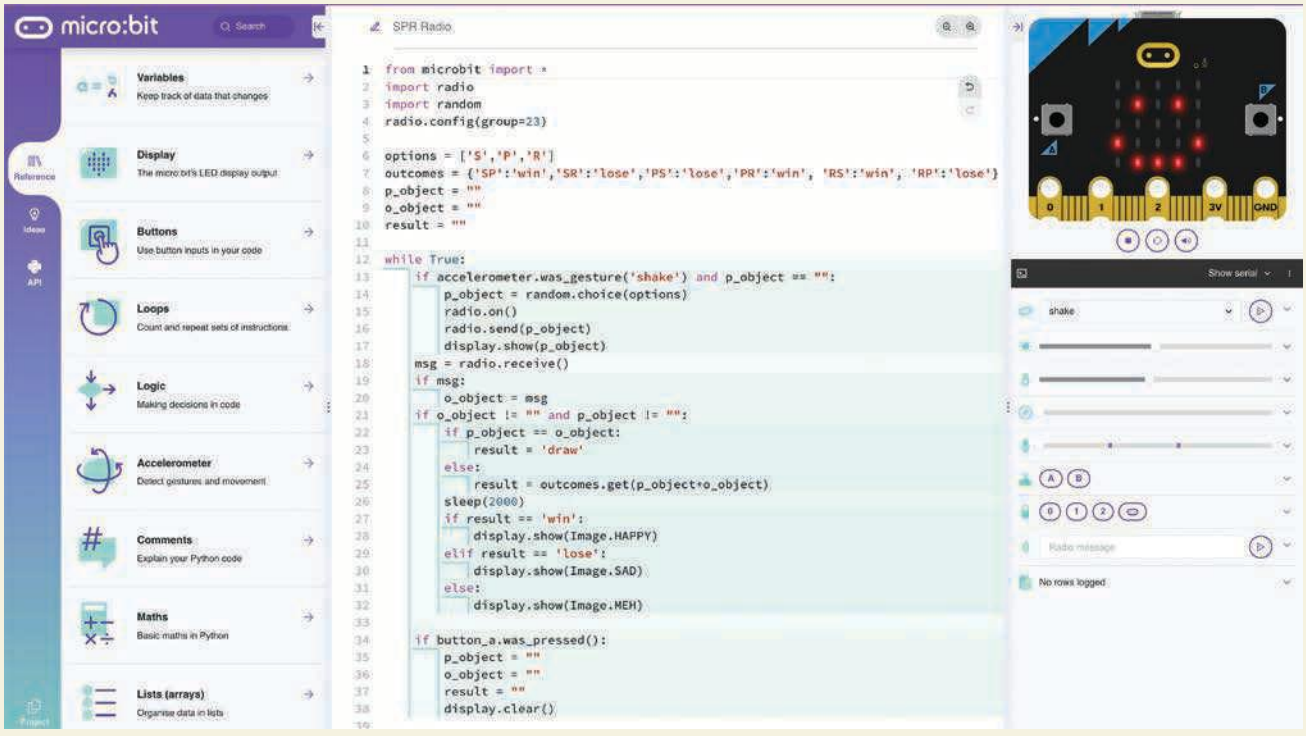


Figure 6.38 Python code for the wireless micro:bit network game scissors, paper, rock.

6 Add comments to the code to demonstrate your understanding.

Optional: Run the code in the simulator to test it. You will need to provide the radio message from the second micro:bit. Test all possible combinations (win, lose and draw).

7 Download the code to two micro:bits and test by having two players shake their micro:bits.

8 Reset the game and test a few more times.

9 Experiment:

a How far away from each other can you be before the game no longer works?

b How does changing the power of the radio (e.g. `radio.config(power = 1)`) impact the distance you can be?

c Change the channel of the radio so the different micro:bits are operating at different frequencies (e.g. micro:bit 1: `radio.config(channel = 3)` and micro:bit 2: `radio.config(channel = 4)`). Does the game work? Do you notice any interference when the channels are close in frequency?

Transmission level protocols

All the technologies you have just read about including Ethernet, Wi-Fi, NR, mmWave, Bluetooth, Zigbee and Z-wave are named for the protocols behind them.

A network protocol is a set of rules and conventions that governs how devices communicate and exchange data over a computer network. It defines the format, sequence and meaning of the messages exchanged between devices, as well as the actions to be taken at each step of the communication process.

Network protocols provide a standardised way for devices to establish connections, transmit data and ensure reliable communication across different network architectures and technologies. They enable devices from different manufacturers and with varying operating systems to communicate effectively with each other.

The key features of a network protocol include message format, addressing, handshaking and connection establishment, error detection and correction, flow control, routing and security (Figure 6.39).

Message format

Protocols define the structure and format of messages exchanged between devices. This includes specifying how data should be organised, the order of data elements and any required control information.

Addressing

Protocols define how devices are identified and addressed on the network. This includes assigning unique addresses to devices and specifying how these addresses are used in communication.

Handshaking and connection establishment

Protocols define mechanisms for devices to establish connections with each other. This involves a process of negotiation and verification, known as handshaking, to ensure both devices are ready to communicate and agree on parameters for the communication session.

Error detection and correction

Network protocols include algorithms for detecting and correcting errors that may occur during data transmission. This can involve adding error-checking codes, checksums or parity bits to the data to detect errors, as well as processes for requesting retransmission of lost or corrupted data.

Flow control

Protocols may include flow control mechanisms to manage the rate of data transmission between devices, ensuring that a receiving device can handle the incoming data without being overwhelmed.

Routing

Protocols define how data is routed through a network. This involves determining the most efficient path for data transmission.

Security

Many protocols incorporate security measures to protect data confidentiality, integrity and authenticity. This can include encryption of data, authentication mechanisms and access control to ensure only authorised devices can access the network.

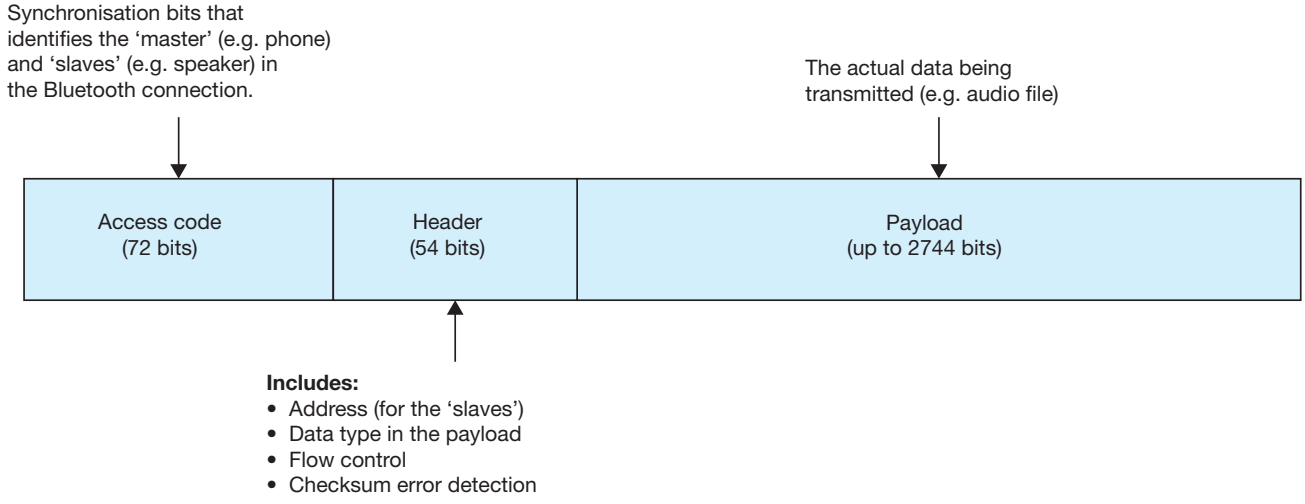


Figure 6.39 The structure and contents of a Bluetooth data packet.

SKILL BUILDER

A data transmission protocol simulation

Your task is to be the first team in the class to accurately transmit a message to your partner.

1. Find a partner. Go online to access an ASCII chart, which shows hex and binary code for the letters of the alphabet.
2. Agree together on a body signal to represent 1 and another to represent 0. No other signals are permitted apart from the length of pauses.
3. For accurate transmission of data, certain protocols need to be agreed upon between partners. Together, agree on your protocols. For example, how will the transmitter know if the receiver has received the last bit? How will the transmitter know if they have to start again? Decide what protocols are needed from those listed in Table 6.4 and agree on what it will be. Assume both partners can signal each other (i.e. duplex transmission), but only using signals for 0 and 1.

   Note that waiting times are necessary to allow the receiver to receive the signal and look up the chart before the next bit is transmitted. This equates to a buffer as used in printers and screen image temporary storage.
4. One partner decides on a message of at least 15 characters, including spaces and writes it on a piece of paper. This must be kept a secret from the other partner and it must not be a well-known phrase!

Table 6.4 The components of a protocol.

Protocol	What it does
handshaking	agreement on protocols
synchronous or asynchronous	will data be sent at timed or non-timed intervals?
start bit and stop bit	indicate where a transmitted byte starts and where it stops
parity check	for even parity, an eighth bit (either 0 or 1) is added to the seven-bit ASCII code to make the total of the 1s an even number
error correction	agreed protocol for retransmission if data is lost

   They then translate this into binary code using the chart, write it on paper and hide it.
5. Partners stand on opposite sides of the room.

   The game is conducted in complete silence! Teams are disqualified by the teacher if sounds are heard.
6. Transmission begins on the signal from the teacher. The receiver records bits as they are received. They then use the ASCII chart to translate the message back into English.
7. When the message is decoded the receiver alerts the teacher, who then checks the partner's message and decides if transmission was exact and correct.

SKILL BUILDER

Design a network for a cafe – Part 2

With the network hardware and other computing technology for Bryce's Brew Cafe selected, we will now compare and evaluate possible transmission media we can use to connect the nodes together.

First, we will consider the media we will use to connect to the Internet via an ISP. In Australia, NBN is the most common choice of connection, however, not all homes and businesses have fibre to the premises (FTTP). Instead, the optical fibre cable ends at a main network node (or the curb) and the house or business is connected to this node via copper wire. Some homes and businesses also opt to still use coaxial cable internet access. In remote areas, internet is provided via satellite.

1. Compare these options considering the availability, speed, bandwidth, latency and reliability. Use Table 6.5 to summarise your findings.
2. Using these findings, select an appropriate medium for your area.

   Now we will consider the internal network connections.

Table 6.5

	Optical fibre	Ethernet	Coaxial cable	Satellite
Availability				
Speed				
Bandwidth				
Latency				
Reliability				

3. Identify an appropriate transmission media and protocol for the following node connections. Hint: First list all the transmission media and protocols covered in this section and reflect on which is best for each scenario.

Table 6.6

Node connections	Transmission media	Protocols
Router to desktop computer in office		
EFTPOS terminal to sales computer		
Customers' laptops to WAPs		
Manager's smartphone to cafe speakers		

ACTIVITIES

Identify

1 What is the metal used in the wire of Ethernet and coaxial cables?
2 How is digital data represented in electrical signals?
3 What is a network protocol?

Analyse

4 Explain when and why you would use optical fibre instead of Ethernet cabling.
5 Explain when and why you would use Bluetooth instead of Wi-Fi.
6 Use a speed test website or app to measure the download and upload speed in different network–device scenarios, including using a desktop with ethernet connection, a laptop with Wi-Fi connection, a phone with Wi-Fi connection and a phone using cellular (4G/5G) Internet. Take at least three measurements of each and calculate the average before comparing. Can you suggest a reason for the different performances?

Extend

7 Complete modules nine and ten of Grok Learning's micro:bit crash course to learn more about radio communication including channels and protocols.
8 Go to the Australian Radiation Protection and Nuclear Safety Agency (ARPANSA) website to view the official data relating to possible health risks associated with mobile base stations, including watching the video "Electromagnetic Energy and You". Summarise the key points.

Weblink
Grok Academy

6.4 Topologies and navigation maps

Now that we have an understanding of the typical nodes that make up a computer network and what technology is used to connect them, we will look at how these connections are arranged in topologies and how these arrangements can be documented in **navigation maps**.

Network topologies

Network topology refers to the arrangement of devices and connections in a computer network. It defines how devices are interconnected and the paths through which data flows in a network. The most common network topologies are bus, star and mesh.

Bus topology

In a bus topology, all devices are connected to a single communication line called a bus (Figure 6.40). Data transmitted by one device is received by all other devices on the bus. This topology is simple and inexpensive to implement but can suffer from performance degradation if multiple devices transmit simultaneously. Bus topologies are commonly used in small LANs or small/home office setups.

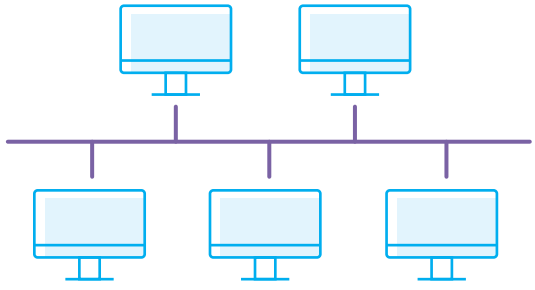


Figure 6.40 Bus topology.

Star topology

In a star topology, each device is directly connected to a central device such as a switch or hub (Figure 6.41). All communication between devices must pass through the central device. This topology provides a centralised control point, easier troubleshooting and better performance compared to a bus topology. There is also the ability to add or remove devices without affecting the rest of the network. Ethernet networks and other LANs commonly use star topologies.

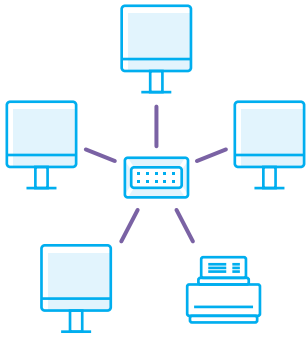


Figure 6.41 Star topology.

Mesh topology

A mesh topology provides a fully interconnected network where each device is connected to every other device (Figure 6.42). This results in redundant connections and multiple paths for data transmission. Mesh topologies offer high reliability and fault tolerance but require a large number of connections, making them costly and complex to implement with wired media. You therefore typically see mesh networks use wireless transmission.

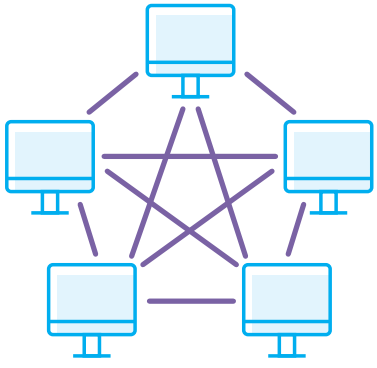


Figure 6.42 Mesh topology.

Ring topology

In a ring topology, devices are connected in a closed loop, forming a ring-like structure (Figure 6.43). Data travels in one direction around the ring, passing through each device in the network until it reaches its destination. Although ring topologies are rarely used in LANs these days, some industrial networks, such as those used in manufacturing plants or utilities, may employ them for their high fault tolerance.

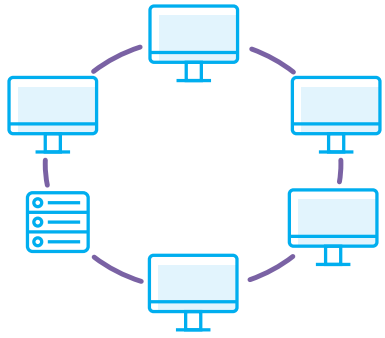


Figure 6.43 Ring topology.

Hybrid topology

The reality of most networks is that they use a combination of topologies in order to use the advantages of each for the different parts of the network (Figure 6.44). For example, your school network may include a main bus network with a number of star networks branching off it. This hybrid topology is referred to as a tree topology. Hybrid topologies are designed with graph

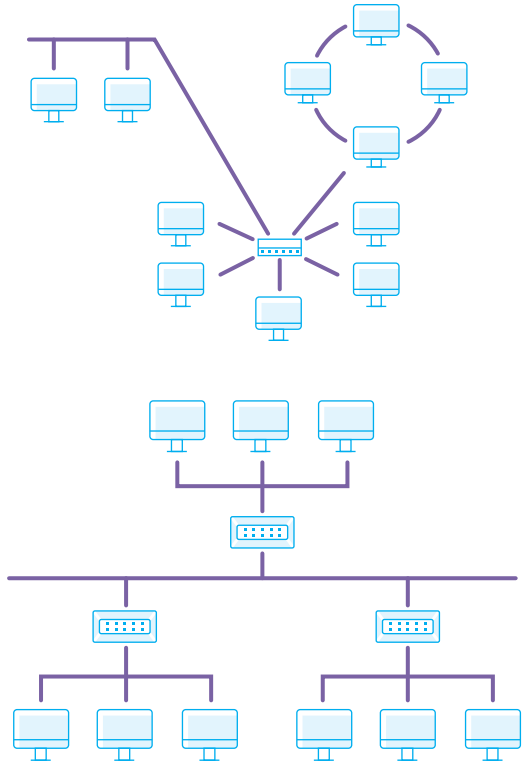


Figure 6.44 Two examples of hybrid topologies, including a tree topology.

theory principles in mind (covered in Chapter 5). Using thoughtfully positioned switches or routers, these high-degree nodes create the most efficient data transmission paths between any other nodes on a network.

Navigation maps

A network navigation map, also known as a network topology map or network diagram, is a graphical representation of the structure and connections of a computer network. It provides a visual overview of the network, including its devices, connections and the flow of data between them. Navigation maps are used when designing a new network and are essential for network administrators managing a network, helping them identify how everything is connected. By visualising the network's layout and connections, administrators can identify potential bottlenecks, analyse traffic flows, plan for network expansion and ensure efficient network operations.

There are various software tools available for creating network navigation maps, ranging from simple diagramming applications to more advanced network management systems that can automatically generate and update maps based on network discovery or monitoring. Microsoft Visio or diagrams.net are applications you may like to use when creating your own navigation maps.

Icons or symbols

The main purpose of a navigation map is to communicate the arrangement of the different nodes and connection types. We want a simple, clear and easy to read diagram and it is important the types of nodes and connection types are clear.

This can be achieved through the design of the nodes and through labelling. Standardised icons, including generic designs and branded ones (e.g. Microsoft Visio), have been developed in order to allow network designers to clearly communicate a network layout.

If drawing by hand, you can try and copy some of these symbols. When using diagram creation software, a range of icons, symbols or pictures may be available to use through the menu (Figure 6.45). Whatever you choose, it is important to be consistent and include labels and keys.

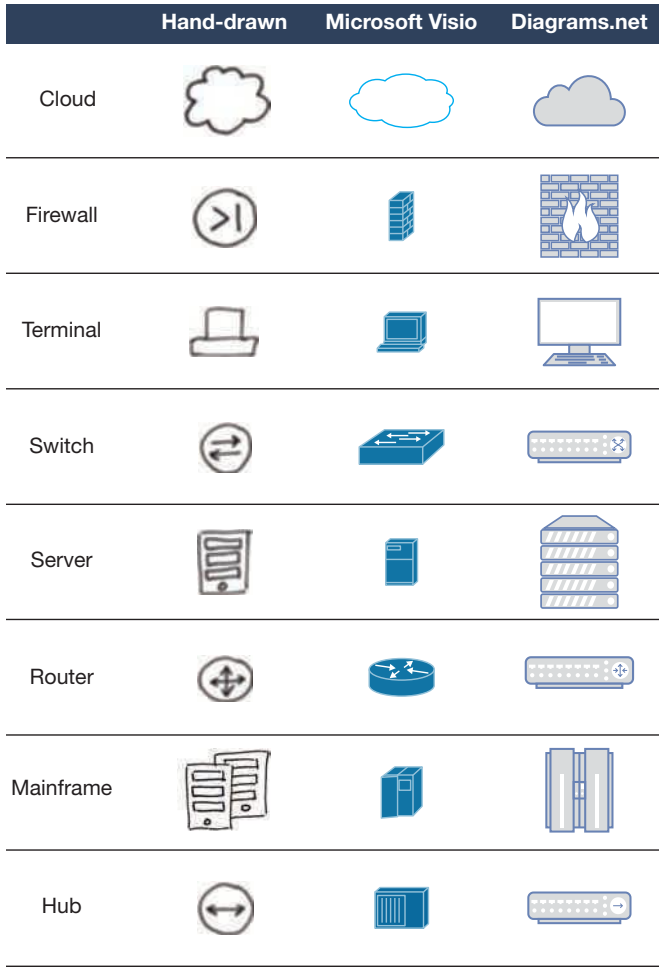


Figure 6.45 There is not one way to identify the nodes in a network diagram.

SKILL BUILDER

Create a micro:bit LAN

In this Skill builder, we will create a simple LAN with four micro:bits (Figure 6.46). Data will only be sent in one direction, with one device acting as the transmitter and the other three as receivers.

1. Starting at the set up from the Skill builder from section 6.3, connect the second receiver to the first receiver, by using another cable to connect port 0 on receiver 1 to port 0 on receiver 2 and a cable to connect the grounds together. Repeat for the third micro:bit receiver.
2. Download the original transmit code from the micro:bit Skill builder to the first micro:bit and the original receive code to the other micro:bits.
3. Test the system.
4. Draw a network diagram to represent your micro:bit LAN.
5. How could you modify this system to allow all micro:bits to be transmitters and receivers? Experiment, reflecting on why the network may not have worked as you expected.

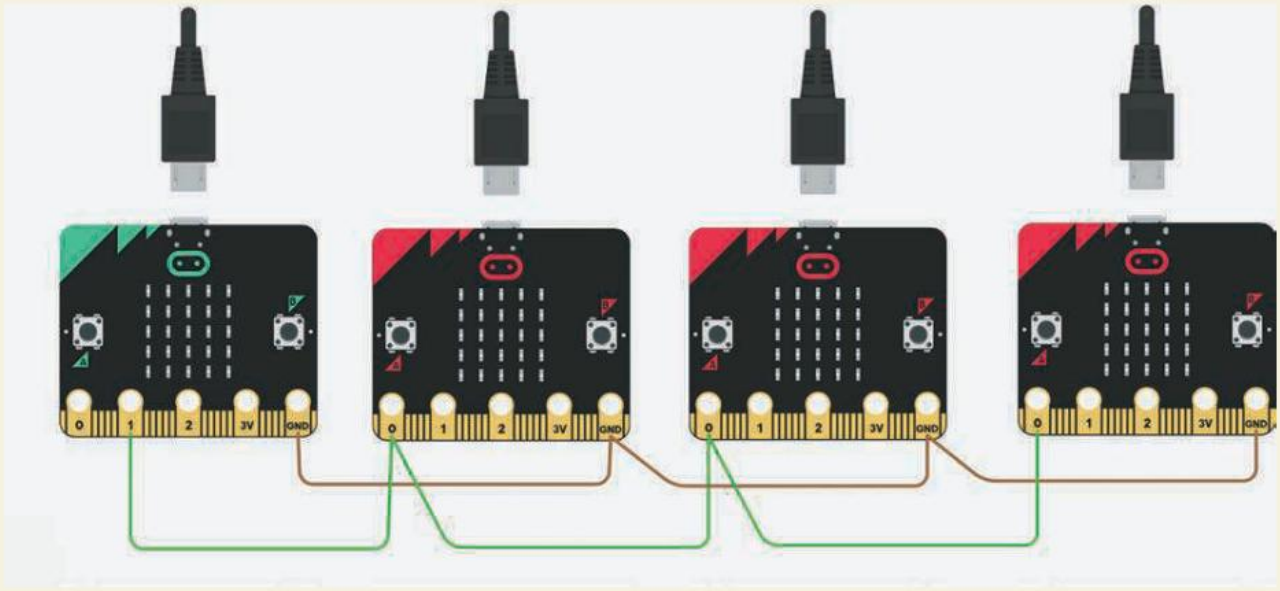


Figure 6.46 Wired network of four micro:bits in a bus topology including one transmitter (green) and three receivers (red).

SKILL BUILDER

Design a network for a cafe – Part 3

With our nodes and transmission medium identified, we can design the arrangement of the devices and connections in a navigation map. This can be done on paper, but it will be demonstrated using the free online tool diagrams.net.

1 To help visualise the arrangement of the nodes and connections, it is helpful to have a map of the physical area, on top of which you can layer the diagram. Source or sketch a floor layout of a typical cafe (Figure 6.47).

Design thinking

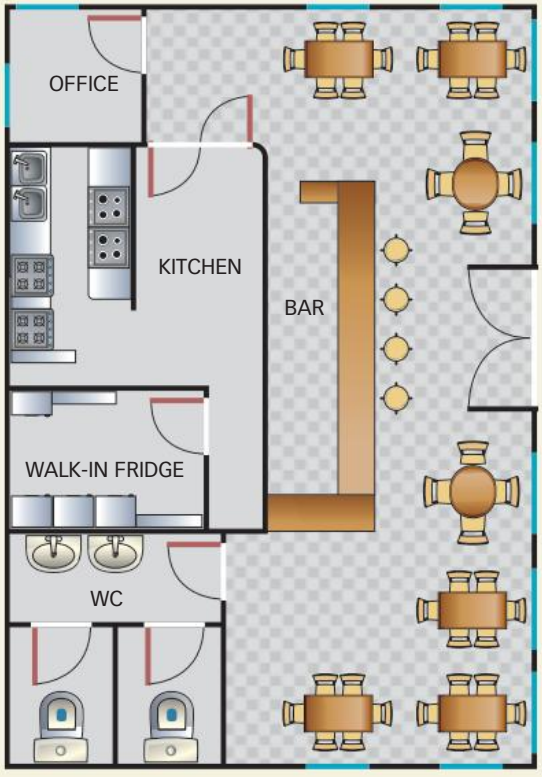


Figure 6.47 A floorplan of the cafe can be used to help arrange the placement of the nodes and connections.

Skill builder resource
Cafe floorplan

2 Create a new diagrams.net blank diagram and insert the floorplan image (Figure 6.48). Drop the opacity of the image down to 50%.

3 Click the 'Add shapes' button at the bottom and select 'Network' check box.

4 Drag out all the network shapes you think you will use including the nodes identified (Figure 6.49).

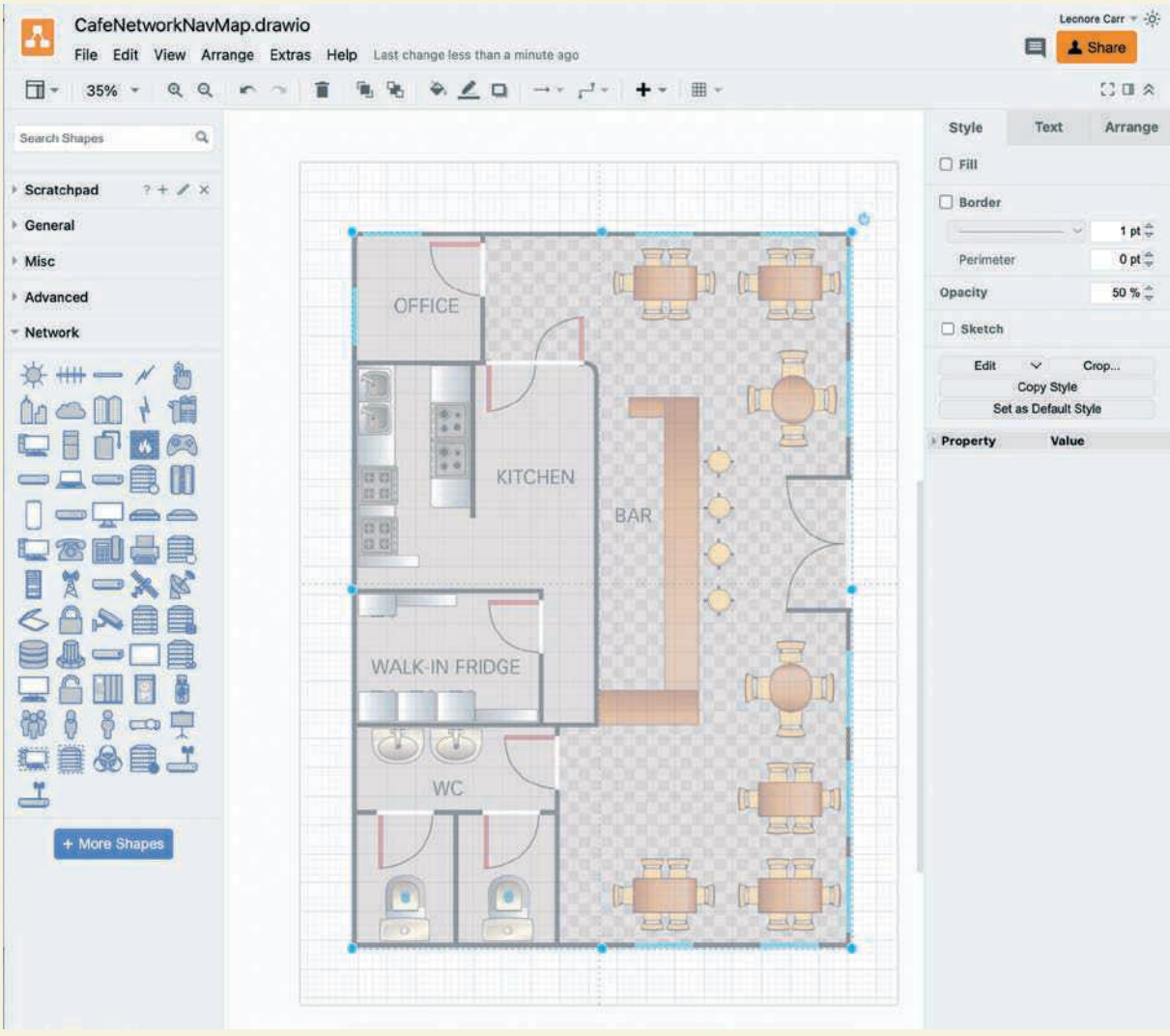
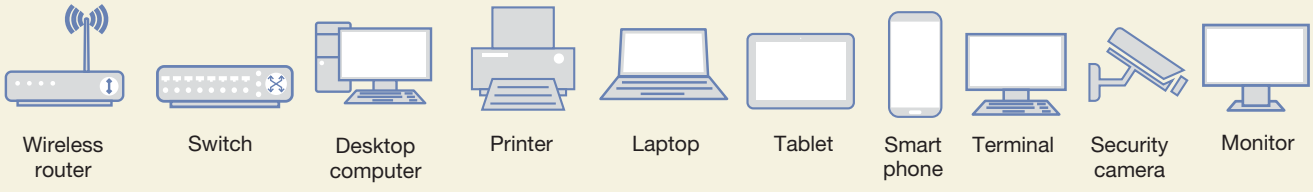


Figure 6.48 It can be helpful to include a floorplan in your network diagram to help visualise the arrangement of the nodes and connections.



Source: Diagrams.net

Figure 6.49 Standard network icons available in diagrams.net

5 Arrange the nodes on top of the floorplan.

6 Use lines to show wired connections (Figure 6.50). You may also like to show some key wireless connections using a dashed line.

7 Add labels and/or a key to make it clear what each node is, including its purpose in the cafe network and connection type such as Ethernet, optical fibre, Wi-Fi, Bluetooth, etc.

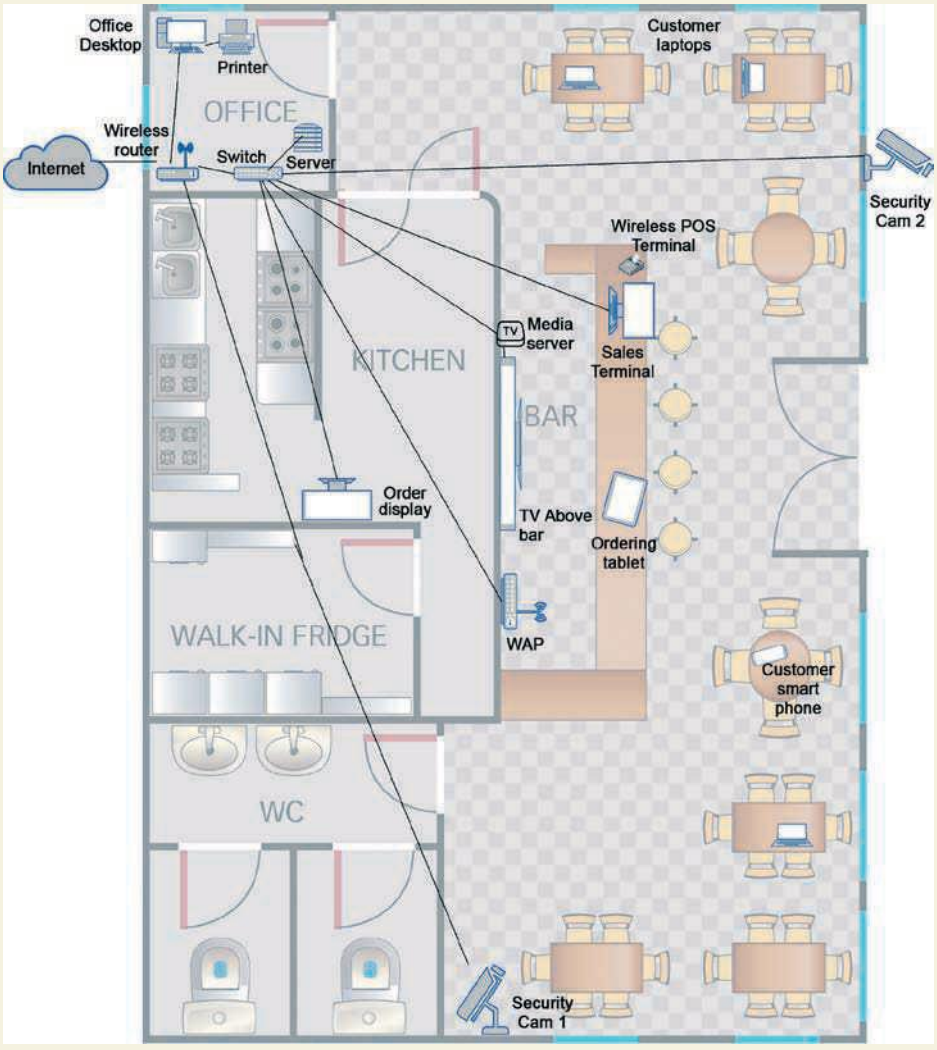


Figure 6.50 A network diagram showing the network design for Bryce's Brew Cafe.

SKILL BUILDER

Placement of WAPs in a school

In this activity you will design and budget the placement of WAPs at a school in order to ensure all staff and students can access high-speed, uninterrupted wireless Internet at the various locations they may be using their devices at school.

There is a huge range of WAPs available on the market, so to simplify this activity you will have the choice of two fictional WAPs, with the goal of efficiently placing them to minimise the cost while maximising the coverage and performance (Table 6.7).

Design thinking

Table 6.7 Comparison of budget and mid-range WAPs.

WAP	Practical coverage	Practical number of clients	Connection speed	Cost (including installation)
Budget WAP	8 m indoor, 20 m outdoor	50	2 Gbps	$500
Mid-range WAP	15 m indoor, 30 m outdoor	100	1.3 Gbps	$700

Requirements

- The school has 300 students and 20 staff.
- Each classroom can have up to 30 students and a teacher who may need to connect to the WAP with their device.
- Most classrooms would not need super-fast network speed, however, the multimedia computer lab (you can nominate which room this is) requires fast speeds.
- The hall is used for exams, which include online testing. This can involve up to 100 students sitting in the gym for the exam on their devices, with each needing access to the network.
- Most of the students move to the outdoor areas for recess and lunch.
- WAPs can be placed on the walls or ceilings of rooms.
- You have a total budget of $5000.

Process

1. Find an image of a school layout and place it into a graphic software application (e.g. Adobe Illustrator).
2. Determine the distance ratio by drawing a line along the key distance and observing how many cm/pts long the line is (Figure 6.51). Adjust the size of the school image to make this an easy conversion (e.g. 1 m = 50 mm).
3. Create two circles to represent the indoor coverage areas for the different routers (i.e. one with a radius of 8 m = 4 cm and one with a radius of 15 m = 7.5 cm). Colour these differently to distinguish them.
4. Repeat for the outdoor coverage, using the same colours but with dotted lines.
5. Copy and paste your circles to show your placement of the WAPs across the school to meet the requirements.
6. Calculate the total cost of the WAPs.

Skill builder resource
School layout

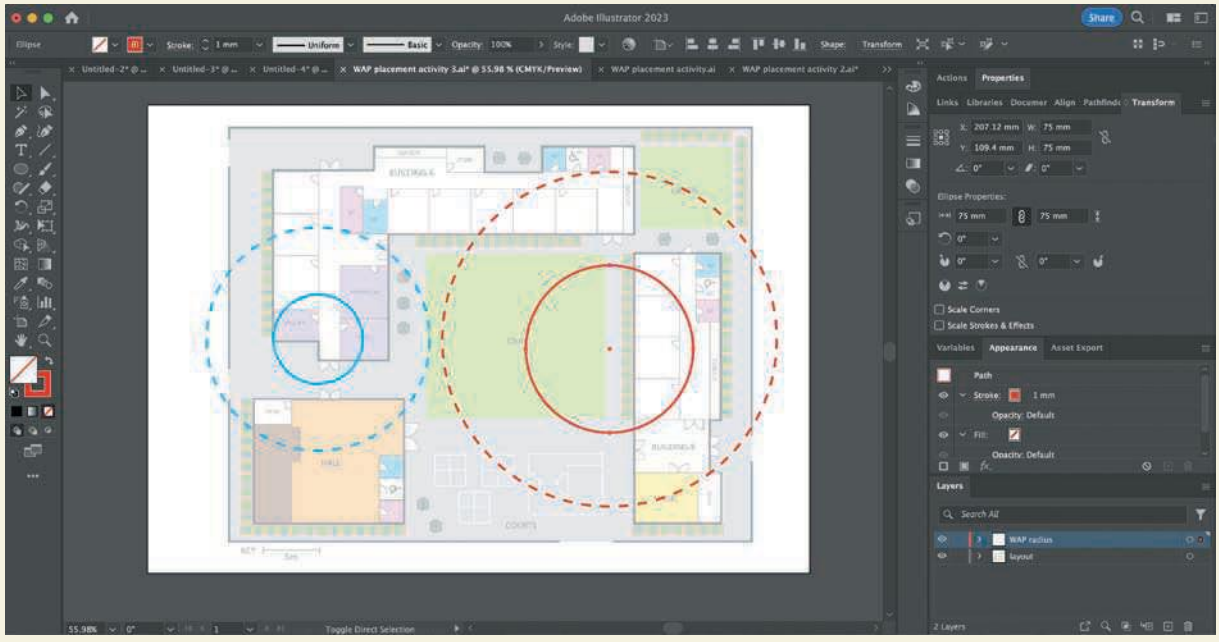


Figure 6.51 Circles represent the practical coverage of the WAP, which would be positioned at the centre point.

Questions

1. What was a greater restriction on the placements: the number of clients or the coverage?
2. Most WAP networks will involve significant redundancy in coverage (overlapping circles). Explain why is it important to have redundancy?
3. Compare your network design with a peer. Evaluate each other's designs based on cost and whether the design will meet the needs of the school.
4. How could the location of the network switches impact the placement of the WAPs?
5. Why is coverage significantly lower in indoor spaces? How would this impact the number of devices needed in multi-story buildings?
6. Discuss the need to access the network in outdoor spaces, primarily intended for screen-free play.
7. There are many more specifications to consider when selecting WAPs. Research the specifications of the WAPs used in your school.

ACTIVITIES

Identify

1. Where are bus topologies typically used?
2. What is an advantage of a star topology as opposed to a bus topology?
3. Do mesh topologies usually use wireless or wired connections? Why?
4. Draw the network symbols used to represent the Internet, a computer terminal and a router.

Analyse

5. Think about the devices and connections in your home network. Create a simple navigation map to show the arrangement. Have you noticed connection issues for devices far away from the router? How could this be resolved?
6. Compare the symbols for a switch and a router. Reflecting on the purpose of these, explain why you think they are designed in this way.

Extend

7. See if you can source a copy of your school's network layout diagram. Identify the different topologies. If one does not exist, ask if you can go on a tour of the school network and create your own. How has graph theory been applied to design an efficient network? Demonstrate this by counting the distance or the minimum number of 'hops' between any two nodes. See Chapter 5 for more graph theory metrics.

6.5 Data processing and transmission

So far you have heard about the various components that make up a computer network and the different ways these can be arranged for specific contexts. How data actually gets sent through the transmission media between the nodes has been hinted at, including the mention of the use of energy, packet switching and the role of protocols. Now we will look at the journey of data through a network in more detail. For example, how does clicking on a video link on YouTube result in the video being streamed back to you on your computer or phone?

Compression

One of the first steps to make data transmission as efficient as possible is compression. The data needs to be reduced in size while still maintaining all or most of the quality. Depending on the type of data, lossless and/or lossy compression may be used.

Where it is essential that the original data can be perfectly reconstructed from the compressed data, lossless compression algorithms need to be used. Lossless compression is commonly used for text-based data, configuration files, executable files and any data that must remain exact. Examples of lossless compression algorithms include ZIP, LZW (used in GIF images) and DEFLATE (used in PNG images).

Lossy compression algorithms achieve higher compression ratios by selectively discarding some data that is less perceptually significant. These algorithms remove non-essential details or irrelevant information from the data to reduce its size. Lossy compression is commonly used for multimedia data such as images, audio and video, where minor loss of quality is viable.

Examples of lossy compression algorithms include JPEG (used for image compression), MP3 (used for audio compression) and MPEG, for example H.264 (used for video compression).

Data packets

Once the data has been compressed it needs to be divided up into smaller parts called data packets. These packets are labelled with the destination IP address, then sent along the best available route to get to the destination via routers. At the destination they are reassembled in the correct order to form the original whole data file. You can think of packet switching like sending a long sentence using postcards, but with only one word on each postcard (Figure 6.52). Each postcard is numbered and contains the destination address so although they may take different postal routes and arrive at different times, once they all arrive the recipient can order them correctly to read the sentence.

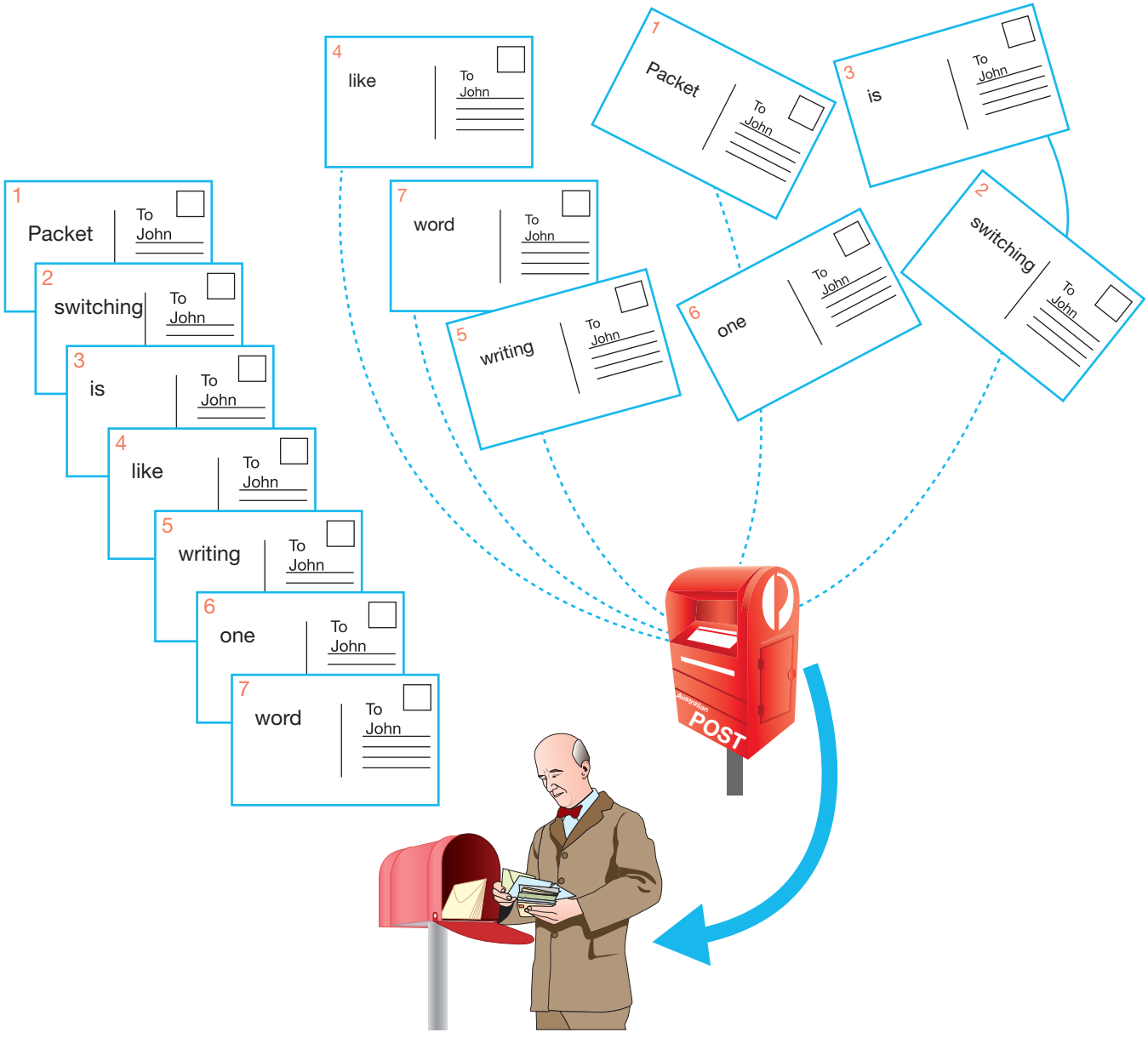


Figure 6.52 Packet switching is like sending numbered postcards with one word on each.

Communicating using packets has been one of the biggest advances in networking as it has many benefits including efficient network utilisation, increased robustness and scalability.

Efficient network utilisation

As packets can take different paths and be transmitted independently across the network, it enables better utilisation of available network bandwidth as multiple packets from different sources can be interleaved and transmitted simultaneously.

Increased robustness

If a particular network link or node fails, packets can be rerouted dynamically through alternative paths to reach their destination. Also, when network problems result in data loss or errors, only the affected data packets need to be resent, not the entire message.

Scalability

As each packet can take its own route through the network, bottlenecks and congestion can be avoided. This scalability is particularly important in large networks with numerous connected devices and high data traffic.

Digital signals and modulation/ demodulation

Once the data has been broken down into packets, the zeros and ones that make up the data packet need to be transmitted in a form carried by the transmission media (Figure 6.53).

Data sent over copper wire (e.g. Ethernet) will be converted to a digital signal. Digital signals are electrical voltage levels that represent the binary values.

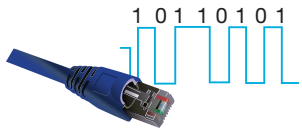
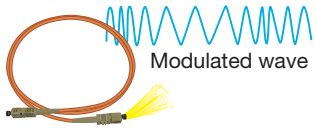
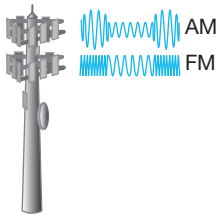
Data transmission types	Bit represented as	
Network cable	Voltage high or low	
Fibre-optic cable	Change in phase or intensity of light	
Radio waves	AM – varying the amplitude (height) of the wave FM – varying the frequency of the wave	

Figure 6.53 Media and methods for data transmission

For example, a high voltage level (e.g. +5V) could represent a binary 1, while a low voltage level (e.g. 0V) could represent a binary 0.

Data sent via optical fibre or through the air using wireless technology use waves. Modulation involves modifying some property of the carrier wave, such as its amplitude, frequency or phase, to represent the digital signal. A common modulation technique is amplitude modulation, where the varying amplitude of the carrier wave corresponds to the digital signal.

At the receiving end, the electrical or wave signals are detected and converted back into digital data.

Navigating the Internet

For a data packet to get from one computer to another, possibly on the other side of the world, it needs to navigate through countless routers. How does this happen? To communicate using the Internet a combination of protocols is used. The main ones are transmission control protocol (TCP), Internet protocol (IP), domain name server (DNS) and whatever media protocol may be used, such as Ethernet. The first of these form the well-known combination known as TCP/IP.

Let's consider the scenario where you want to watch a YouTube video to learn how to make pancakes. You go to YouTube, type in 'How to make the fluffiest pancakes' and hit return. The data packets begin their journey.

DNS

The first thing to understand is that your computer and all servers on the Internet are assigned unique addresses known as internet protocol (IP) addresses, which look like this: 208.65.153.238. When you type 'www.youtube.com' into your browser, a DNS has the job of converting this written URL into its matching numerical IP address for that computer. DNS was created because, unlike computers, humans are not very good at remembering long strings of numbers. When you hit return on your search in YouTube, your router sends a request to DNS to translate the URL to its corresponding IP address.

TCP/IP

In addition to the DNS request, your browser passes your search 'message' to the TCP/IP software in your computer's operating system. TCP breaks down this message into smaller packets (Figure 6.54).

The TCP packet is then wrapped in an IP packet, which includes the IP address of where the data is being sent (YouTube) and your own IP address to allow a response. Based on the destination IP address of the IP packet, each router along the journey knows the best hop to select. This is because it maintains a routing table that contains information about various network destinations and the corresponding paths or next hops to reach them. Routing algorithms use graph theory metrics to create the routing tables. The network is represented as a graph with each edge attributed with the 'cost' (physical distance, bandwidth and congestion) of data being transmitted along that link. Algorithms such as shortest-path algorithms use the graph theory metrics to determine the fastest route from this node to the next hop or destination (Figure 6.55).

The data packet is then wrapped in another layer containing your unique device MAC (Ethernet) address and the MAC address for the next router on the journey. When the packet completes a hop, the current router strips off the last device's MAC address and adds its own and that of the router for the next hop. In this way, the IP addressing information in the inside packet remains the same but the destination device MAC address changes each hop until it reaches its final destination IP address, in this case YouTube's server.

At the receiving computer, the wrappers are removed in reverse order by the protocols and TCP reassembles the separate packets in their correct order (Figure 6.56).

The message has been delivered.

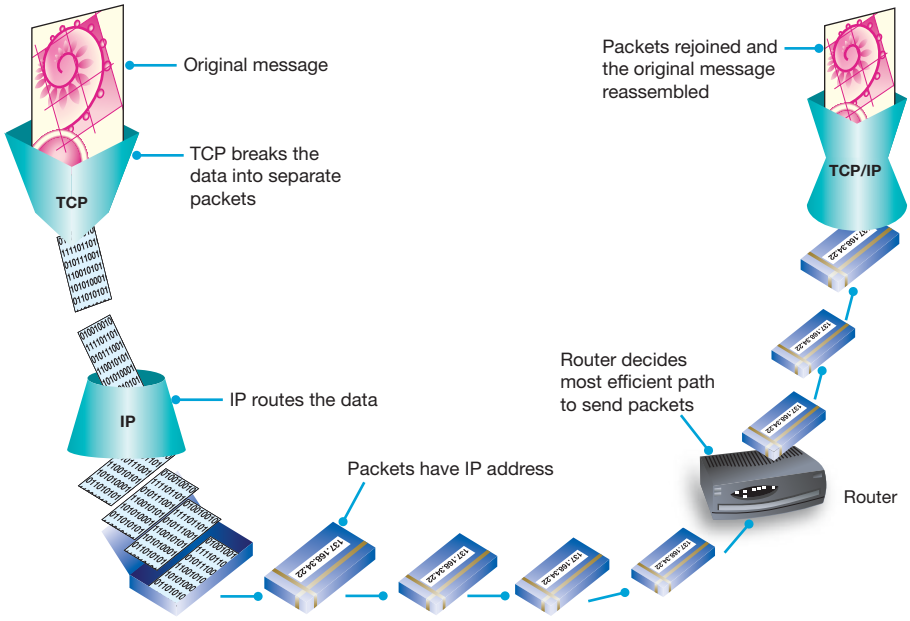


Figure 6.54 A message is divided into separate packets to send over the Internet and these are re-joined in the correct sequence at the receiving end.

Computational thinking

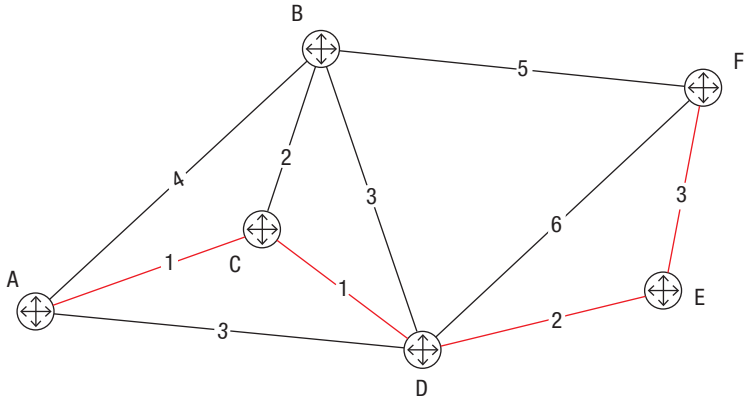


Figure 6.55 Routing algorithms are applied to the graph of the computer network to find the shortest/fastest path to send data. In this case from node A to C to D to E to F with a total 'cost' of 7.

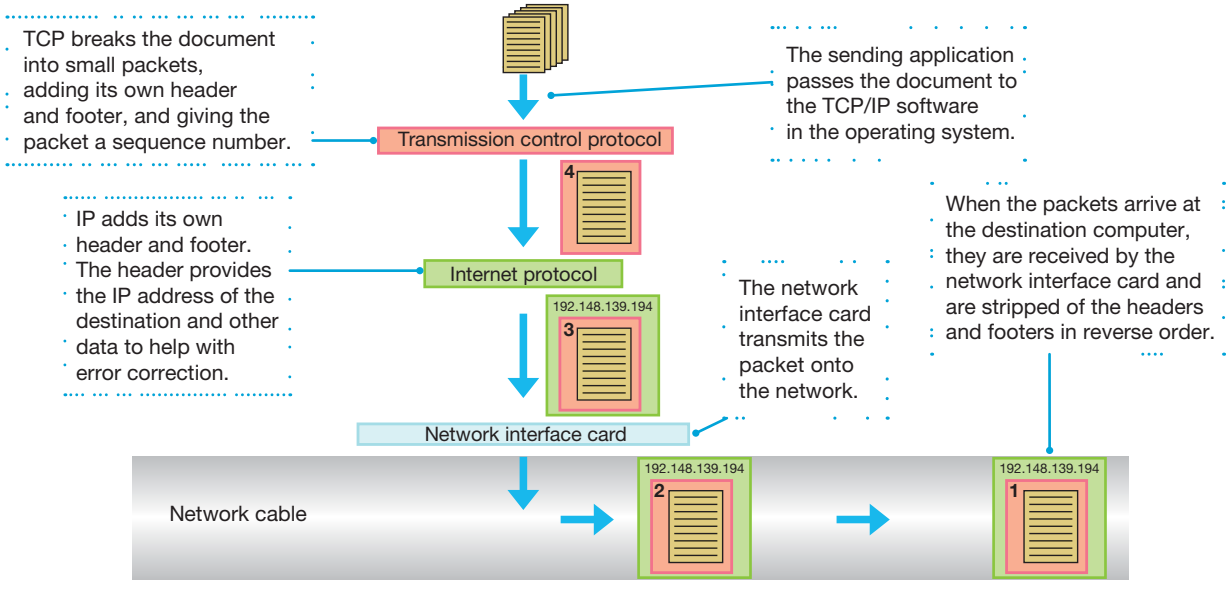


Figure 6.56 TCP/IP protocol is now built into most operating systems. Data packets 1–4 follow each other through the protocol layers.

KNOWLEDGE BUILDER

Internet role play

In this activity you will be role playing how data travels along the internet using people as the nodes, string to represent the internet connections and toilet rolls as the message data packets.

Resources

- five students (one playing a blue computer, one playing a red computer, three playing separate routers)
- six toilet rolls (three coloured blue, three coloured red, each set numbered 1–3)
- a ball of string
- scissors to cut the string

Aim

To be first to deliver a three-word message, which the receiving computer must reassemble and read aloud.

Both red and blue computers must flip three rolls along strings to routers, which forward them to the other computer where they are reassembled to decipher a three-word message.

Systems thinking

Rules

One student is elected to be the blue computer and is given three numbered red toilet rolls. Another is the red computer and given three numbered blue toilet rolls. The three other students are elected as routers.

- All students stand apart. Both computer students hold three lengths of string, each reaching to one of the three router students. The router students each hold four strings, which reach to the four other students (including both computers) (Figure 6.57).
- Blue computer sends red roll number one along a string to any router. At the same time, red computer sends blue roll number one along a string to any router.
- Routers forward messages to computers of identical colour by flipping each toilet roll along the string as quickly as possible.
- Only one roll on each string at any time! If a string has a message already on it, then it must not be used and another route must be found.
- If there is a collision, then messages must go back to the senders. How is this like TCP/IP?

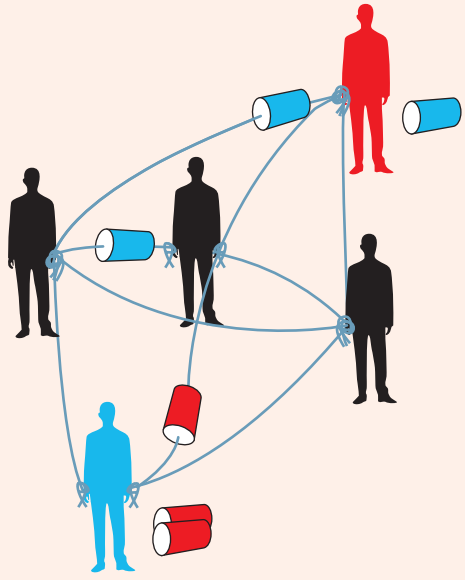


Figure 6.57 A human network.

SKILL BUILDER

Using an Internet simulator

Code.org has designed an excellent free online Internet simulation tool (widget) to visualise, experiment with and solve different kinds of problems associated with networked computers. This tool was created by computer engineers from Microsoft, Google, Facebook and Twitter.

Teachers: Follow the weblink for the Internet Simulator.

Click the 'Try now' button on this page for full instructions on setting up a class. Note that students' names can be entered in one step by entering names on separate lines.

Students: You cannot use this tool by yourself. It simulates a network, so it needs more than one person linked up for it to work.

Once names are entered, your teacher will provide the class with a single URL and passwords. Your name represents your domain name and the tool has a built-in DNS server to translate names to IP addresses on this simulated network.

1. Log in using the URL provided by your teacher. You will use the advanced version of the tool.
2. Click on the radio button next to the words 'Internet Simulator: DNS'.
3. Click to join one of the routers.
4. Examine the network map at the top left of the simulator window. You will only be able to see the router to which you are connected, along with other users of that router. You will see your DNS but not those of other users. You will not be able to see any other users in your class, even though they are connected to your network. In what ways is this like the real Internet?

Computational thinking

Weblink
Code.org: Internet Simulator

5 Use the DNS server to find the IP address of another user of your router. The DNS server will respond if you ask in the correct way. You can request the IP address of anyone in your class, even if they're connected to a different router!

6 Type the IP address of the DNS into the 'To' field, then use the following protocol to retrieve the IP address of another user (Figure 6.58).

```
GET <hostnameOfPerson>
```

For example: `GET cv9`

7 Once you've sent your request, DNS responds with the hostname and IP address (Figure 6.58).

Figure 6.58 Using GET instruction to ask DNS for IP address of user cv9 and the IP address 1.13 is received for user cv9.

8 After DNS has returned the user's IP you can contact them directly by typing the IP address into the 'To' field, entering a message and then pressing 'send' (Figure 6.59).

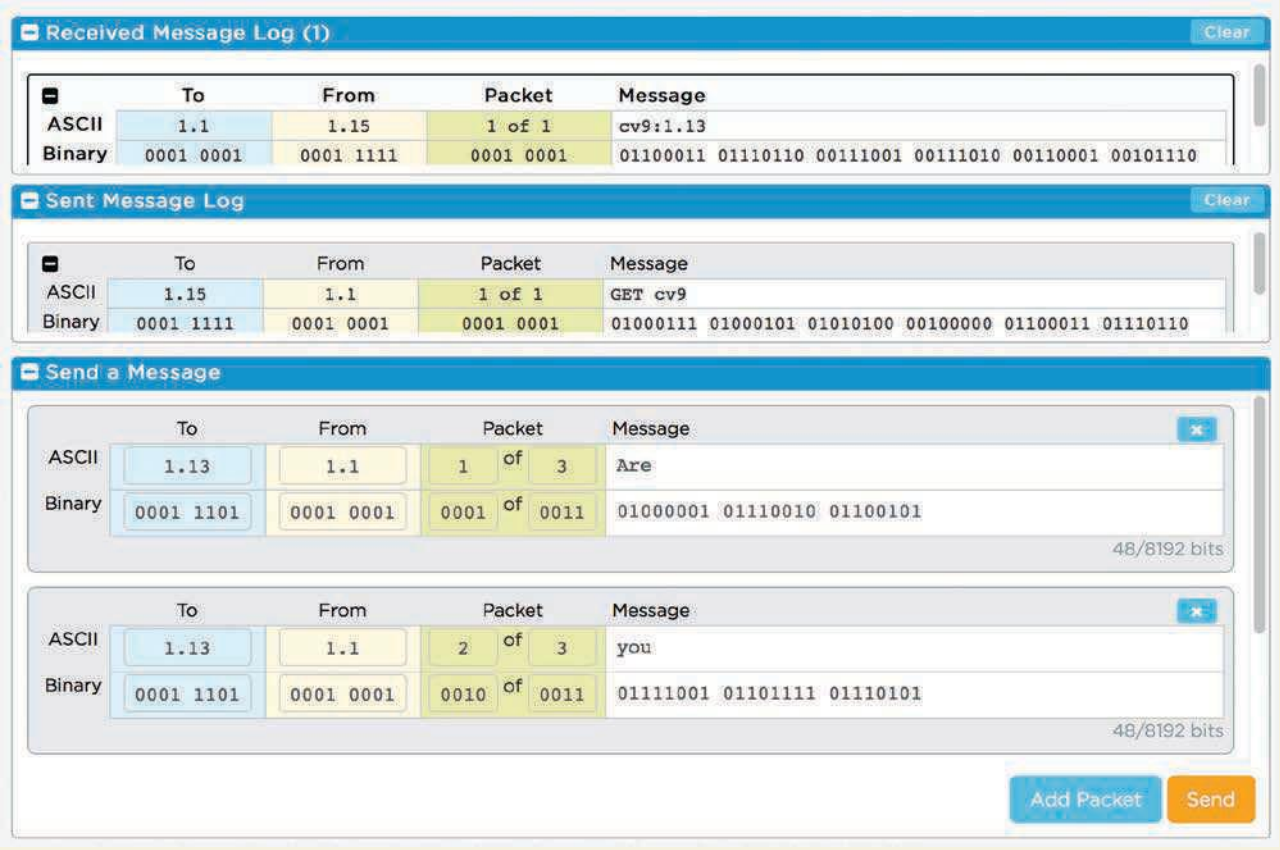


Figure 6.59 Now their IP is known, messages can be sent to user cv9.

9 Complete Table 6.8 by finding the IP addresses for five other users.

Table 6.8 Users' IP addresses.

Name	IP Address
Your name	
1	
2	
3	
4	
5	

10 You will be assigned a partner. (Your teacher will randomly assign a partner to each class member by drawing names out of a hat.)

11 You have already learned that in the actual Internet, long messages are split into packets. The simulator allows you to do this. Each packet is labelled with a sequence number and the total number of packets.

12 First find the address of your partner using DNS.

13 Decide on a question to ask, remembering that each word must be sent as a separate packet. Any user being asked a question must wait until all the packets making up the question have arrived and only then reply to the correct sender, again using one word per packet.

14 Record the conversation in Table 6.9.

Table 6.9 Internet simulator conversation.

Partner's name	Partner's address
Your question	Response
Partner's question	Response

15 How is this like the Internet?

ACTIVITIES

Identify

1 What makes up a data packet?

2 What is the role of DNS?

3 How does a router know which connection to send a data packet along? How does it decide which is the 'best' option for the next hop on the Internet?

Analyse

4 How might an email be compressed for transmission compared to a video file?

5 What is the difference between an IP address and a MAC address?

6 Trace a data packet around the world using a traceroute tool. How many hops did it take? Do the longer distance hops take a longer time? Repeat the trace – were the hops and times the same or similar?

Extend

7 Research how TCP/IP takes care of 'lost' data packets.

8 Find five locations where international data cables arrive on land in Australia.

9 Learn how Dijkstra's algorithm is used to find the shortest path from a node to all other nodes in a network. Teach the method to a peer or the rest of the class. Is this method used in Internet routing? Why or why not?

Weblink
G Suite.
Tools: Visual Traceroute

6.6 Network security

In 2022, the Australian Cyber Security Centre was receiving reports on cyber security crimes on average once every seven minutes. For all the benefits and positive impacts networks have brought about, there has unfortunately been a new wave of crimes that can have disastrous impacts on individuals and businesses. In order to prevent or minimise the risk of cyber crimes, there are many tools and processes that need to be implemented when creating and managing a network.

In this section we will look at some of the common threats and vulnerabilities and the ways these can be reduced or mitigated.

Threats and vulnerabilities

Malware and viruses

Networks are vulnerable to **malware** and viruses that can be introduced through malicious emails, infected websites or removable media (Figure 6.60). Once inside the network, malware can spread, disrupt operations, steal data or perform other malicious activities.

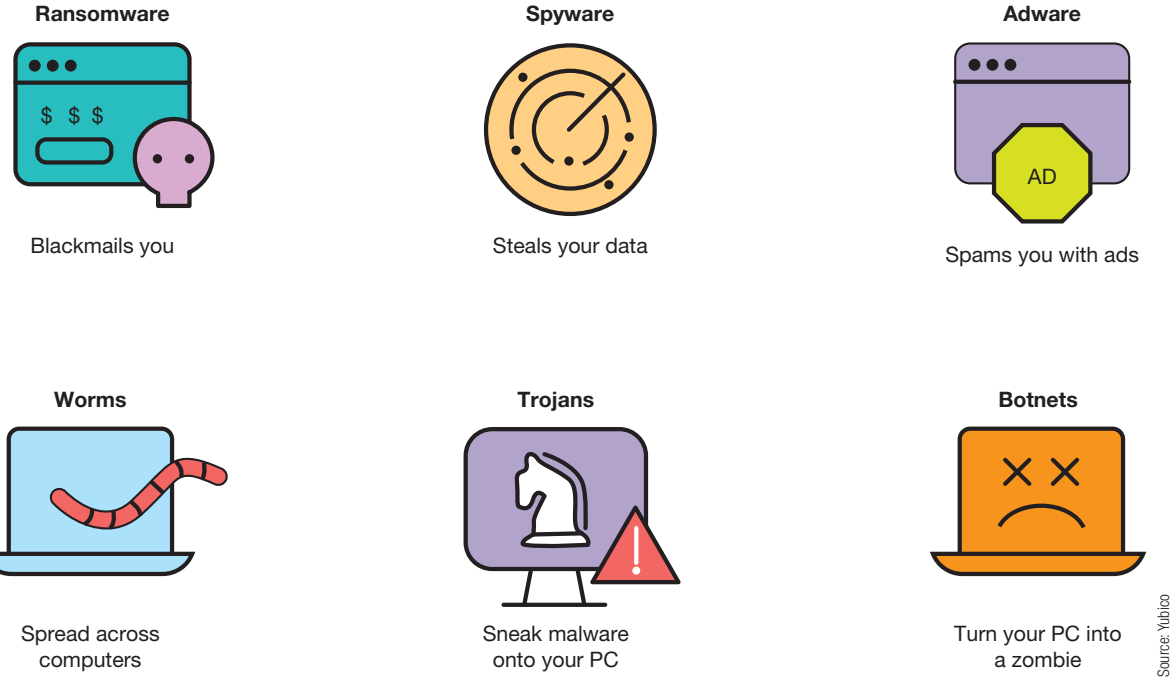
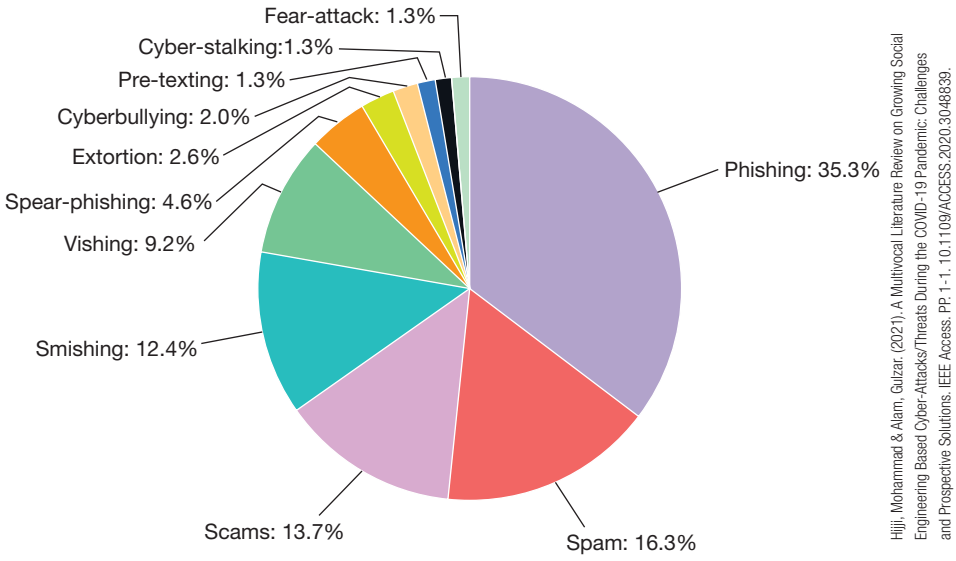


Figure 6.60 The different types of malware ('mal' means 'bad' in French) that devices on a network could be exposed to.

Social engineering

Attackers may use **social engineering** techniques to manipulate individuals within the network to gain unauthorised access. This can include phishing emails, impersonation or exploiting human weaknesses to trick employees into revealing sensitive information or performing harmful actions (Figure 6.61).



Hijji, Mohammad & Alam, Gulzar. (2021). A Multivocal Literature Review on Growing Social Engineering Based Cyber-Attacks/Threats During the COVID-19 Pandemic: Challenges and Prospective Solutions. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.3046839.

Figure 6.61 COVID-19 saw a significant rise in people using devices for work and personal communication. With this came an increased risk of social engineering attacks. The most common form of social engineering attack was phishing, where attackers attempt to trick individuals into providing sensitive information, such as passwords, credit card details or personal information, by impersonating a trustworthy entity or organisation.

Lack of encryption

Without proper encryption, data transmitted over the network is susceptible to interception and eavesdropping. This is particularly critical for sensitive information such as login credentials, financial data or personally identifiable information (PII). Encryption algorithms are continuously being improved to keep up to date with current threats and meet the generally growing concerns most individuals have about data privacy and security.

Weak network security

Poorly configured network devices, such as routers, switches and firewalls, can create vulnerabilities. Default or weak passwords, unnecessary open ports or outdated firmware can be exploited by attackers to gain unauthorised access or control over the network. Data in wireless networks can be easily intercepted if not properly secured. Weak encryption and default or easily guessable passwords are two common vulnerabilities.

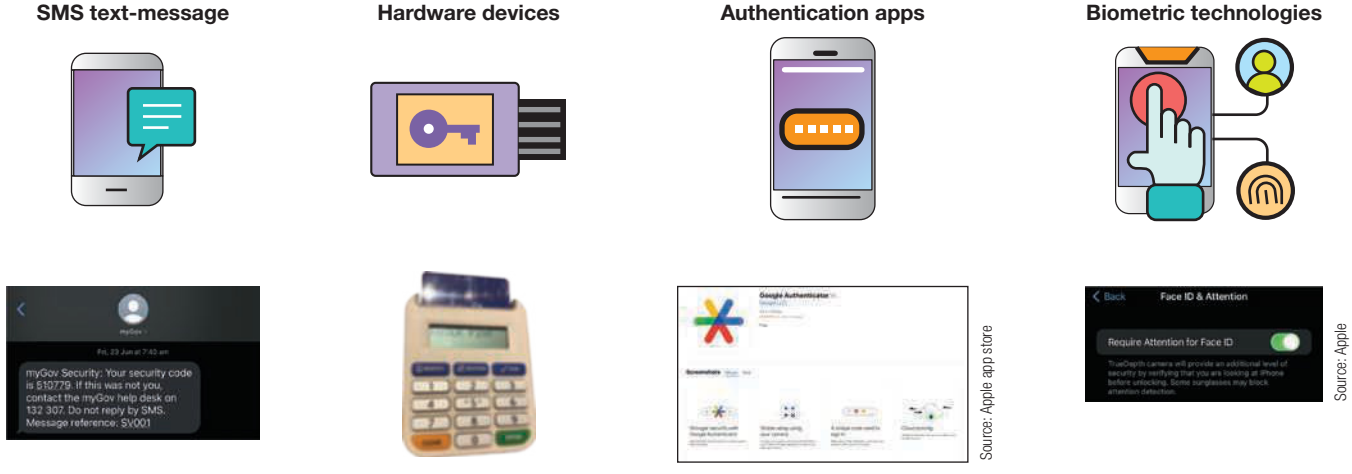
Mitigation strategies

Access controls

One of the most important steps to mitigate cyber security risks is to implement strong access controls. This includes the use of unique, complex passwords for all network devices and the enforcement of regular password changes. Using two-factor authentication (2FA) where possible also reduces the risk (Figure 6.62). This process has become more prevalent in recent years in response to the need and demand for stronger security measures. Finally, administrative access should be restricted to authorised personnel only.

Firewalls

A firewall is a network security device or software application that acts as a barrier between an internal network (e.g. a LAN) and external networks (e.g. the Internet) (Figure 6.63). Its primary purpose is to monitor and control incoming and outgoing network



Source: Apple app store

Source: Apple

Figure 6.62 Common types of two-factor authentication used to mitigate the risk of unauthorised access.

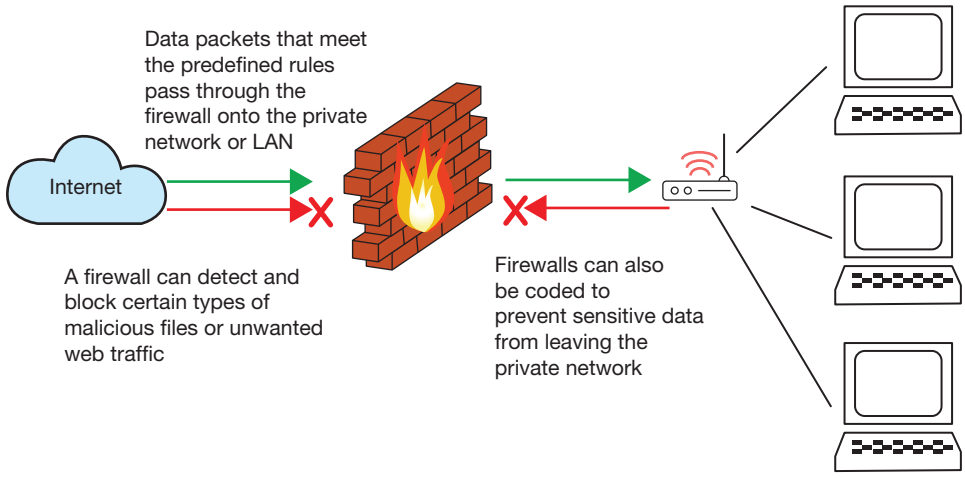


Figure 6.63 The function of a firewall.

traffic based on security rules. Firewalls use various techniques to examine packets of data and determine whether to allow or block them based on specified criteria. They function to prevent malicious data from entering a LAN, such as a virus and prevent sensitive data from leaving a LAN, such as personal information.

Encryption

When unauthorised access does occur, it's unlikely the unauthorised person will be able to use it if data is strongly encrypted. Encryption should occur during network transmission through the use of encryption protocols like HTTPS and SSL/TLS and when being stored somewhere on a network like a server or other connected storage devices.

Virtual private networks (VPNs)

A virtual private network (VPN) establishes a secure and encrypted connection over a public network such as the Internet (Figure 6.64). It allows users to create a private network connection remotely, as if they were directly connected to a private network, even if they are accessing it from a different physical location. When you connect to a VPN, your device establishes a secure and encrypted connection, commonly referred to as a tunnel, to the VPN server. By connecting to a VPN server, your internet traffic appears to originate from the VPN server's location rather than your actual location. This masks your IP address and helps protect your online privacy

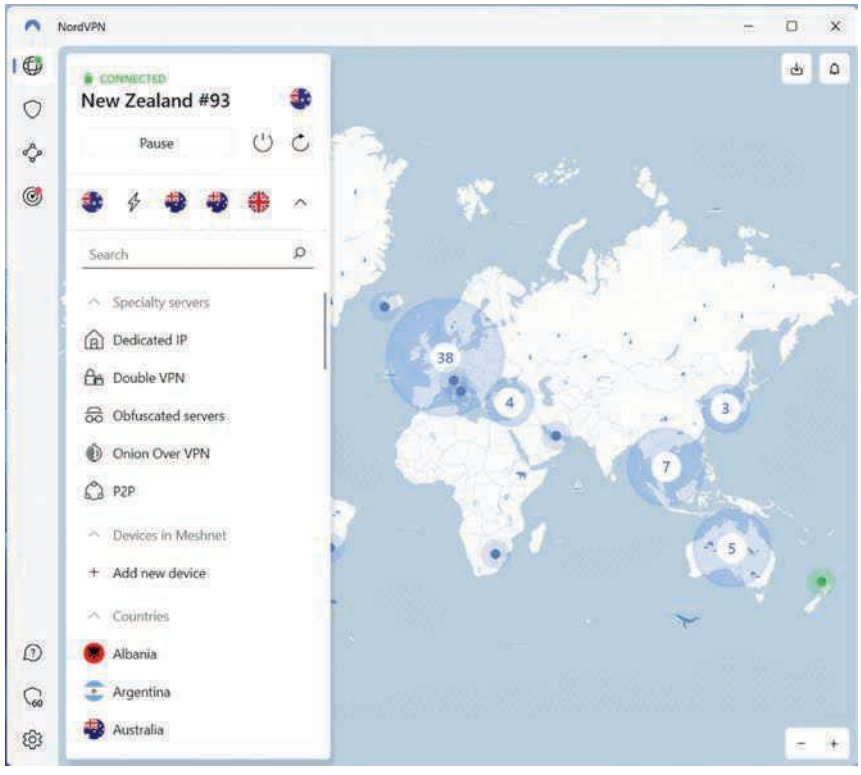


Figure 6.64 VPNs are easy to download on your device and allow you to select a server in another country where your data will 'tunnel' through, disguising its real origin.

and anonymity. When connecting to public Wi-Fi networks, which are often insecure, using a VPN adds an extra layer of security.

Segment the network

Divide the network into separate segments or virtual LANs (VLANs) based on the sensitivity of data and systems. Implement firewall rules to control traffic flow between segments, allowing only necessary communication.

Update and back up regularly

Keep all network devices, operating systems and software up to date with the latest security patches. Regularly check for firmware updates for network equipment and apply them promptly. Implementing a robust backup strategy for critical data means that if data is damaged or lost due to a security breach, it can be recovered.

Monitoring and reporting

Network monitoring plays a critical role in detecting and preventing security threats. It involves monitoring network traffic for suspicious activity and analysing security logs to identify potential breaches, malware infections, unauthorised access attempts or data leaks or thefts. Security monitoring helps in early threat detection and incident response, as well as providing valuable insights into potential security incidents and ways security could be improved.

Education

Given a significant percentage of crimes result due to weak network security and social engineering, it is essential that individuals are educated about the risks. Security awareness training should be conducted regularly and address issues such as avoiding suspicious email attachments, practising safe browsing habits and promptly reporting potential security incidents.

User expectations

People's changing expectations and concerns for security and privacy online have had a significant impact on the development and evolution of computer technology. All of the mitigation strategies above are continuously being improved based on advancements to the technology behind the threats and user experience and expectation.

The COVID-19 pandemic was a significant event that accelerated the need and therefore development of such technology, as most people began working or learning from home. Security measures in place for the safety of the LAN needed to be bolstered to ensure the security of data as it travelled across the Internet.

In general, more of our lives have moved online in recent years. With many people's improved awareness of potential threats and desire for increased anonymity, computing technology has greatly improved. This includes improved encryption algorithms, 2FA, better user control over data and privacy settings, the use of privacy-focused web browsers and VPNs, and other cyber security system developments.

ACTIVITIES

Identify

1 What are the main network vulnerabilities? What threats target these vulnerabilities?
2 What is a VPN and how does it mitigate cyber security threats?
3 Why is education an important mitigation strategy?

Analyse

4 Go to the 'How secure is my password' website to see how secure your school network login password is.
5 Students often go to public libraries to study, using the free Wi-Fi to access the Internet. Explain how the effective use of access controls, VPNs and backups could help them to stay safe online.
6 As privacy concerns have grown, the concept of 'privacy by design' has emerged. What is this and how can it help to mitigate privacy related cyber security issues?

Extend

7 Create a poster to help educate students and teachers at your school about the importance of robust cyber security practices, including the risks if these are not implemented.
8 Complete one of Grok Learning's cybersecurity challenges.

Weblinks
How secure is my password?

Grok Academy

PEOPLE

Network administrator

A network administrator is a person responsible for managing, maintaining and securing the computer networks within an organisation (Figure 6.65). They ensure that the network operates smoothly, efficiently and securely to meet the organisation's needs. Their role and responsibilities include:

Network design and implementation

Network administrators plan and design the organisation's computer network infrastructure. They determine the hardware and software requirements, network topology and network protocols necessary to meet the organisation's needs.

Network configuration and maintenance

Administrators set up and configure network devices, including routers, switches and wireless access points. They establish network connections and IP addressing and implement security measures such as firewalls and intrusion detection systems. Network configuration software helps administrators centrally manage and configure network devices. Regular network maintenance tasks such as firmware upgrades, patch management and performance monitoring are also performed by network administrators.

User management and support

Network administrators handle user accounts and permissions, managing user access to network resources. They provide user support and troubleshooting for network-related issues, such as connectivity problems, slow network speeds and network outages. They also assist users in setting up and configuring network services, such as email, VPN and file sharing.

Network security

Network administrators play a critical role in maintaining network security. They use network security software such as firewalls and VPNs, and implement security measures to protect the network from unauthorised access, data breaches and malicious activities. This includes managing user authentication and access controls, implementing network encryption, monitoring network traffic for security threats and conducting security audits and vulnerability assessments.

Network performance optimisation

Administrators use network monitoring software to monitor the health and performance of network devices, servers and applications (Figure 6.66). It provides real-time information on network traffic, bandwidth utilisation, device status and other performance metrics such as latency and packet loss. Administrators can use this information to optimise network configurations and make adjustments to enhance network speed, reliability and efficiency.

Backup and disaster recovery

Network administrators are responsible for implementing backup and disaster recovery plans for network infrastructure. They ensure that network data and configurations are regularly backed up and can be restored in the event of data loss or network failures. They may also test and simulate disaster scenarios to assess the effectiveness of recovery procedures.

Figure 6.65 The primary role of a network administrator is to ensure the smooth and secure operation of network infrastructure. This includes maintaining network hardware including the often highly complex arrangement of switches with ethernet and optical fibre connections in LANs.

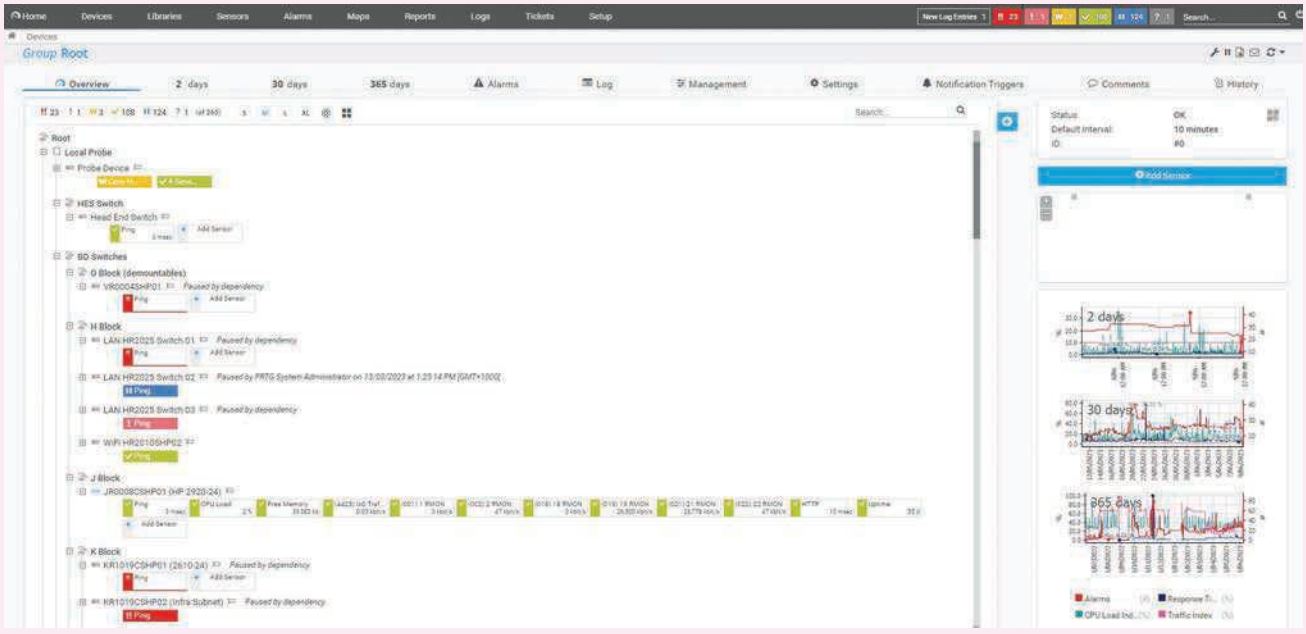


Figure 6.66 Network administrators use management software to keep track of network devices to ensure they are running as expected and to identify where an issue may be located when a problem with the network arises.

Questions

1. What software do network administrators use to fulfil their role?
2. Why is it important that network administrators have good communication skills?
3. Go to Seek or another job site and find an advertisement for a network administrator. What knowledge, skills and experience are they expected to have?
4. Conduct an interview with the network administrator at your school to find out about their role and how they manage the school network.

Possible questions:

- **a** Explain or demonstrate how you use the software to test and monitor the network.
- **b** How have network monitoring tools improved over time and how has this has impacted your role and work efficiency?
- **c** Have you been involved in the (re)design of a LAN? If so, how did you design the network to be efficient, work within the physical constraints of the environment and be within budget?

6.7 Internet of Things (IoT)

If you were to ask a random person on the street what they use the Internet for, the chances are they would say it was for communication (e.g. email), social media, entertainment, information, eCommerce, banking, work or education. These products nearly always involve a website or app that serves as an interface for the user accessing content from and providing content to a central server.

The end nodes in the network are generally the terminal computers with people providing or receiving the content and the servers processing the requests. It's the *Internet of People*!

What is IoT?

In 1982, before the web had been developed, the Internet (or, more accurately, ARPANET), was mainly being used by researchers, academics and government institutions for communication via email and file sharing protocols. It was an *Internet of Professionals*. One such professional was David Nichols, a graduate

student at Carnegie Mellon University. Nichols was frustrated by the number of times he had walked to the university vending machine only to find it out of stock. So, he, John Zsarnay and Mike Kazar engineered a solution that would allow them to communicate with this thing remotely. They made the thing a node on ARPANET by adding microswitches to the vending machine slots that could monitor the indicator lights. An interface was developed that connected the microswitches to the main department computer via a serial port. To monitor the machine's status, a server machine was programmed to record the time of the last slot's transition. Additionally, a simple network protocol was created, enabling any machine on the local Ethernet (and eventually on the Internet) to inquire about the current status of the vending machine. They had created the precursor to the *Internet of Things (IoT)*, an evolution of the internet, where objects and devices are connected and able to communicate and interact with each other.

Modern examples

Today, the IoT encompasses a wide range of interconnected devices, including smart home appliances, wearables, industrial sensors, autonomous vehicles and more. These devices collect and share data, enabling new applications, automation and improved efficiency in various domains (Figures 6.67 and 6.68).

Shutterstock.com/panuwat phimpha

Figure 6.67 Many modern vending machines are connected to the Internet to not only allow real-time stock metrics, but also cashless payment, data analytics for optimisation and advertising, and other remote management processes.

Smart homes

IoT enables the automation and control of various home devices and systems, such as lighting, thermostats, security systems, appliances and entertainment systems. Smart home devices can be interconnected and remotely managed through a central hub and/or mobile app.

Industrial automation

IoT is extensively used in industrial settings for monitoring, control and optimisation of processes and equipment. It enables real-time data collection, predictive maintenance, remote monitoring and efficient resource utilisation, leading to increased productivity and cost savings.

Health care

IoT has numerous applications in health care, including remote patient monitoring, smart medical equipment and wearable devices. IoT devices can collect and transmit vital signs, track medication adherence, enable remote consultations and improve patient outcomes. Wearable devices such as fitness trackers and smartwatches collect data on physical activity, heart rate, sleep patterns and more, providing insights for personal health and fitness management.

Smart cities

IoT is utilised to make cities more efficient, sustainable and liveable. Applications include smart energy management, intelligent transportation systems, waste management, environmental monitoring, public safety and infrastructure optimisation.

Agriculture

IoT technologies are used in agriculture to monitor and control factors like soil moisture, temperature and humidity. IoT sensors and actuators help optimise irrigation, manage crop health, monitor livestock and automate farming operations, leading to increased productivity and reduced resource usage.

Retail

IoT is employed in retail for inventory management, supply chain optimisation and enhancing the customer experience. IoT devices such as beacons, RFID tags and smart shelves enable real-time inventory tracking, personalised marketing and seamless checkout processes.

Energy management

As you learnt in the case study in Section 6.1, IoT plays a crucial role in smart grid systems and energy management. Primarily, it enables monitoring and control of energy consumption, demand response programs, renewable energy integration and efficient grid operations.

Environmental monitoring

IoT devices are used to monitor and collect data on air quality, water quality, noise levels and other environmental parameters. This information helps in early detection of pollution, natural disaster monitoring and effective environmental management.

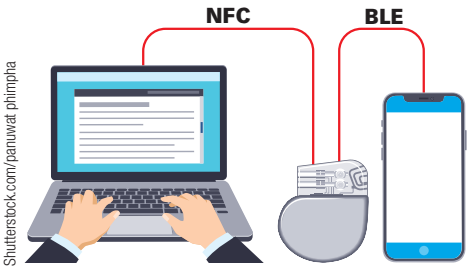
Transportation and logistics

IoT is employed in fleet management, asset tracking, logistics optimisation and vehicle telematics. Real-time data from sensors and GPS devices enable efficient route planning, vehicle maintenance and improved supply chain visibility.

Shutterstock.com/Barry Paterson

Smart bulbs used in home automation

Industrial processes — energy monitoring



Shutterstock.com/panuwat phimpha

IoT pacemaker (health care)

Shutterstock.com/Eric Buermeyer

Monitoring in agriculture — tracking of livestock

iStock.com/Kinwun

Environmental monitoring

iStock.com/gorodenkoff

Product tracking and logistics

Figure 6.68 Some examples of IoT devices used in different industries.

Microcontrollers

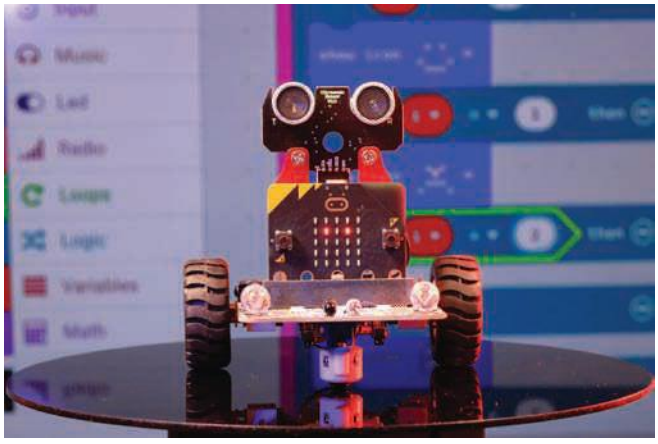
The hardware requirements of IoT devices are different to those of personal computers. One clear difference is devices do not need a graphical user interface to communicate with one another. Given the volume of data being collected is generally less and fewer processes are being undertaken, the CPU and memory requirements are also lower. For this reason, many IoT devices use microcontrollers for the computational processing within the object. As you learnt earlier in the chapter, a microcontroller is a small computer system contained within a single integrated circuit (IC) or chip. They are used in IoT as they offer a combination of low power consumption, cost effectiveness and sufficient processing capabilities for many IoT applications. Microcontrollers have a CPU, memory, timers/counters and input/output ports that allow them to communicate with external devices or sensors. There are a range of microcontroller products available for IoT including Arduino (popular among hobbyists, students and professionals) and PIC (popular in industrial automation, automotive systems and consumer electronics). We will continue to explore the micro:bit (Figure 6.69).

Sensors

One of the key features of any IoT device are the sensors; how it collects data from the object's environment. There are a range of sensors that can be connected to microcontrollers, such as temperature, humidity, pressure, proximity, light, motion and sound. The micro:bit V2 sensors include a temperature sensor, light sensor, touch sensor, buttons, microphone, accelerometer and compass, plus several general-purpose input/output (GPIO) pins that can be used for connecting external sensors to expand the sensing capabilities of the micro:bit, such as an ultrasonic or soil moisture sensor.

Output

Like any system, once the input has been processed, the calculated output needs to be displayed or actuated. This can be as simple as an LED turning on or sound playing through a speaker, or more complex movement through the use of motors. The micro:bit has a 5x5 array of LEDs, which can be used to display simple graphic elements of text, and a basic speaker. For more complex visual and audio display, you can connect screens and speakers to the edge connectors. Actuators such as motors and fans can also be connected in this way.



PublicDomainPictures/Vasiliu Gabriel

Figure 6.69 A micro:bit can form the brain of a robot, with inputs collected by built-in sensors such as the compass and from connected sensors such as US-distance sensors. Output can use built-in displays, such as the LED matrix and the speaker, and connected actuators, such as motors to turn the wheels.

Connections

Like all computer networks, the microcontrollers in an IoT system need to be able to communicate with other nodes in the system network (e.g. computers, other microcontrollers, routers and peripheral devices). These connections can be wired (e.g. USB or Ethernet) or wireless (usually using radio waves such as Wi-Fi, Bluetooth or Zigbee protocols). The micro:bit has a USB connection and can communicate with other micro:bits by radio and with other devices using Bluetooth.

In addition to Starlink, Space X has recently launched Swarm, a network of small, low-orbit satellites about the size of a hand that provide low-bandwith connectivity to every point on Earth. This allows IoT devices to function even in the most remote locations.

Chapter review quiz

Skill builder resource
Build a micro:bit wireless data logger

SKILL BUILDER

Build a micro:bit wireless data logger

In this task we will build a data logger network to record a simple science observation: the acceleration of a car as it goes down a ramp (Figure 6.70). The variables that you change could include the height of the ramp or the surface of the ramp. One micro:bit will be attached to the back of a toy car and will use its built in accelerometer to log the combined force acting on the car (in x, y and z directions). This micro:bit will transmit this reading to another micro:bit, where it will be logged with a timestamp. This second micro:bit will only receive readings if it has been told to start recording by someone pressing the 'a' button. Pressing the 'b' button stops the recording. This second micro:bit will be attached via USB to a computer so once the observation is complete, the recording can be stopped and the logged data transferred as a CSV file to be graphed in a spreadsheet application.

Resources

2 micro:bits (V2), 1 power pack, 1 USB cable, personal computer, micro:bit code environment, toy car, cable ties/elastic bands to attach micro:bit to car, ramp. Note: If you have V1 micro:bits, you will need to devise a way to create your own log by printing lines to serial that can then be copied and pasted into a spreadsheet. You could alternatively use the Python Mu editor, which has a built-in plotter for serial data.

Figure 6.70 By attaching the micro:bit to a car, you can use the accelerometer to measure the forces acting on the car as it goes down a ramp. These readings can be transmitted to another micro:bit attached to a computer that keeps a log of the readings.

How it works

1. Write the Python code in Figure 6.71 for the car micro:bit (transmitter) in your choice of micro:bit Python editor.
2. Add comments to the code to demonstrate your understanding of how it works.
3. Test the code in the editor, using a simulator if possible.
4. Download the code to the micro:bit and test. You could add a display to the LED matrix to help you determine if data is being accurately recorded and transmitted.
5. Write the Python code in Figure 6.72 for the logging micro:bit (receiver).

Figure 6.71 Python code for the wireless data logger.

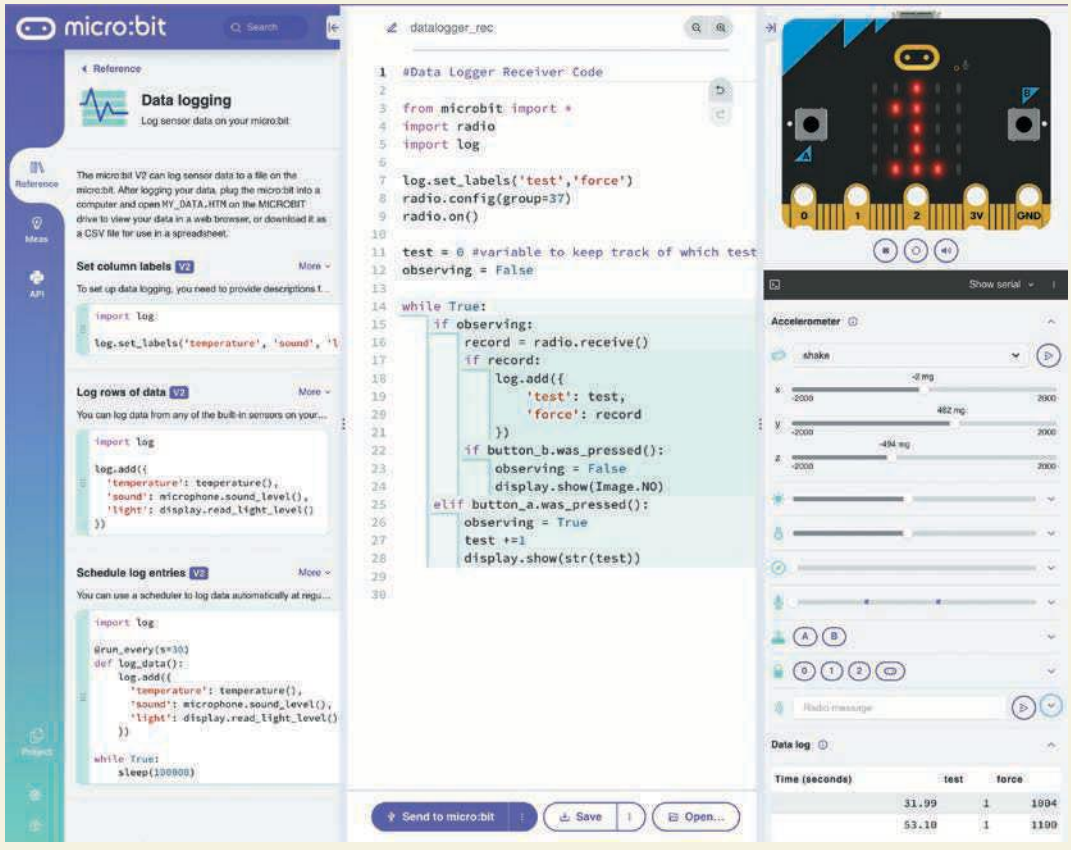


Figure 6.72 Python code for the receiver micro:bit in the data logger network.

Computational thinking

6 Add comments to the code to demonstrate your understanding of how it works.

7 Download the code to the second micro:bit and test by moving the sensor micro:bit (remember to press 'a' on the receiver so it can begin recording).

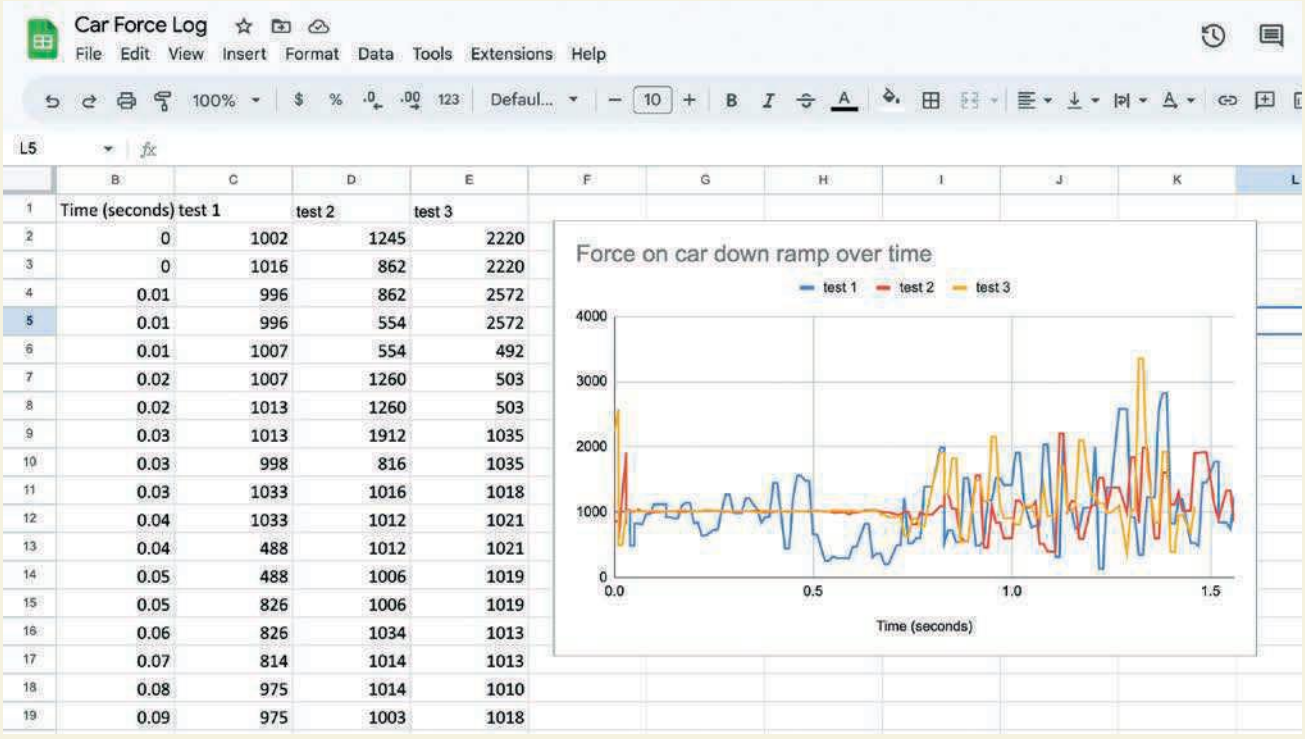
8 Pause the receiver by pressing 'b', then open the MY_DATA.HTM file to check data has been logged. If so, move on, otherwise you may need to do some troubleshooting/debugging to find out what is not working as expected (e.g. are your radio groups configured to different channels?).

9 Redownload/flash the program on to your micro:bit and run the tests, changing the variables (e.g. height of ramp). Press the 'a' button to start the recording of each and the 'b' to finish.

10 Open the MY_DATA.HTM file and download the table as a CSV file.

11 Open the CSV file in spreadsheet software and create a line graph of the force measured over time (Figure 6.73). You should 'clean up' the data (e.g. remove data points captured in error) and set each test up as a series before graphing.

12 To create an interactive online visualisation, refer to the dashboard Skill builder in Chapter 9 in section 9.8.



	B	C	D	E
1	Time (seconds)	test 1	test 2	test 3
2	0	1002	1245	2220
3	0	1016	862	2220
4	0.01	996	862	2572
5	0.01	996	554	2572
6	0.01	1007	554	492
7	0.02	1007	1260	503
8	0.02	1013	1260	503
9	0.03	1013	1912	1035
10	0.03	998	816	1035
11	0.03	1033	1016	1018
12	0.04	1033	1012	1021
13	0.04	488	1012	1021
14	0.05	488	1006	1019
15	0.05	826	1006	1019
16	0.06	826	1034	1013
17	0.07	814	1014	1013
18	0.08	975	1014	1010
19	0.09	975	1003	1018

Figure 6.73 The data logged on the micro:bit can be downloaded as a CSV file for analysis in spreadsheet software.

SKILL BUILDER

Skill builder resource
Build a silent tip game with micro:bits

Build a silent tip game with micro:bits

We will now build a two-directional network with multiple micro:bits. In order to do this effectively we need to define the communication protocol so that the rules of data transmission are established.

We will create a network for a game of silent tip (Figure 6.74).

Resources

At least 3 micro:bits, each numbered with a label, 1 power pack per microbit, 1 USB cable, personal computer, micro:bit coding environment.

Note: If you have V1 micro:bits, simply replace the touch logo gesture with the shake gesture.

How it works

1 Form groups of at least three numbered players (1, 2, 3…), each player with a micro:bit. Players must stay in their seats within the classroom.

2 An image display on the micro:bit LED matrix indicates if the player is in or not (e.g. GHOST for in and PACMAN for not in).

3 Player '1' starts 'in'. Using their micro:bit, they need to 'tip' another student in their game by sending that person's number to their micro:bit. As the micro:bit only has two buttons, you will need to encode the number in binary with button 'a' representing a 0 and button 'b' representing a 1. For example, to tip person 3, they would

need to send the binary number '11'. To indicate the number has been completed, the gold logo is touched and the person's number is transmitted to all micro:bits. The player cannot tip themselves.

4 The person who has just been tipped will see their display change (e.g. to a GHOST), so then needs to tip another student using the same process.

5 The game is played for a set period of time. Once the time is up, the person who is in loses and is out of the game. The remaining people play until only one remains – the final winner!

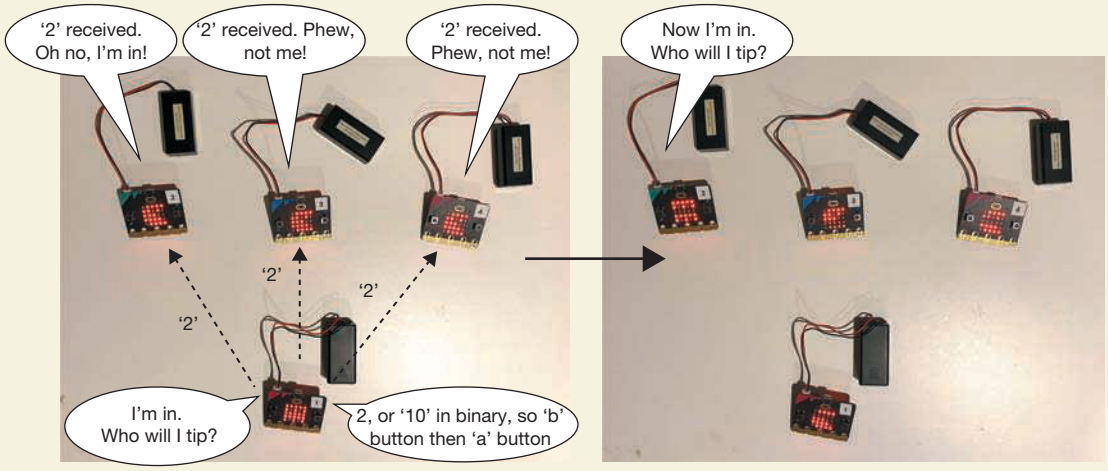


Figure 6.74 How a silent tip game with four micro:bits might look.

Computational thinking

Creating the game

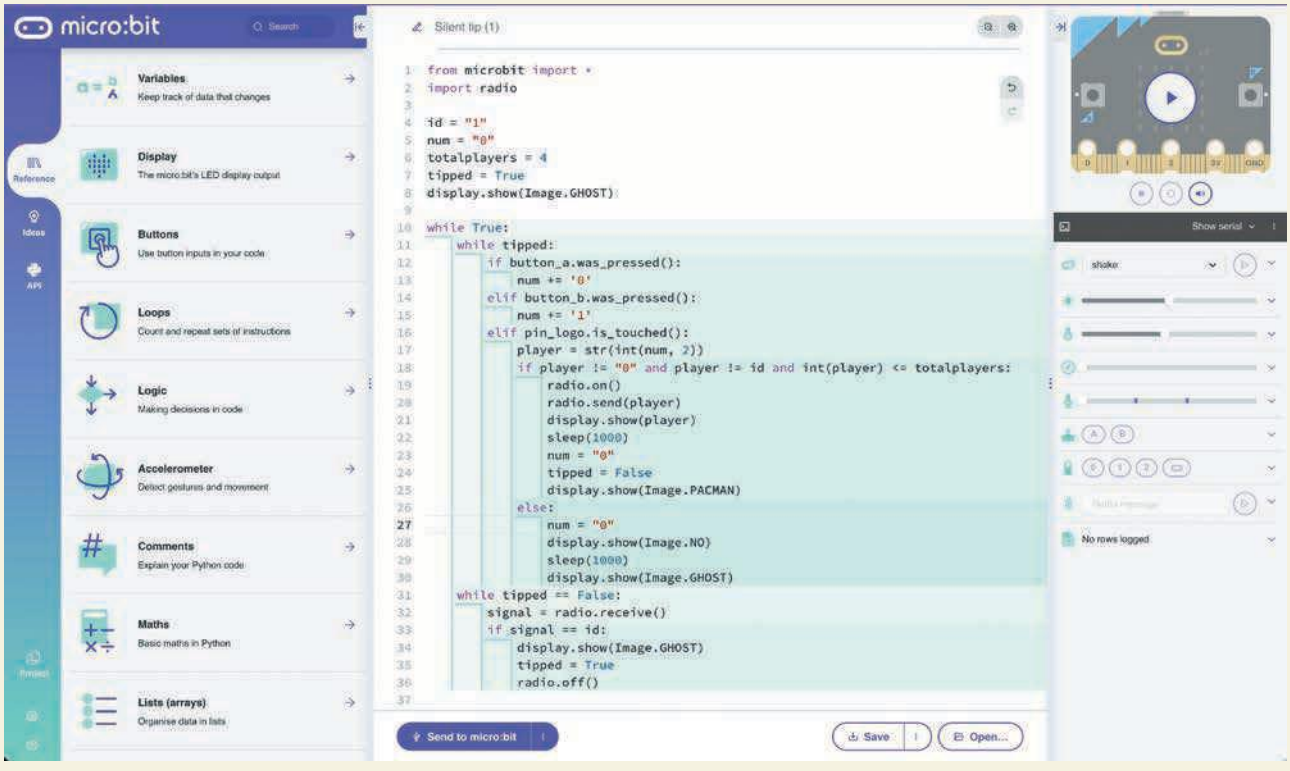


Figure 6.75 Python code for the wireless silent tip game.

6 Code the micro:bit using the code in Figure 6.75, adding your own comments to demonstrate your understanding of how it works.

7 Test in the simulator, observing the radio signal which is being sent.

8 Download to micro:bit '1'.

9 Modify the code, changing the ID number to '2', tipped to False and display to Image.PACMAN. Download to micro:bit '2'.

10 Repeat for all remaining micro:bits, changing the ID number to the one on the label.

11 Attach battery packs to each micro:bit. Micro:bit '1' should have the GHOST image and the rest should have the PACMAN image.

12 Test the game, starting with micro:bit '1'.

Questions

1 What protocol(s) allows this game to work?

2 What are some of the problems with the function of this game system? How could the code be modified to address this?

3 One improvement to the game would be to allow players to 'dodge' a tip. Suggest how you could code this including the protocol for communication between micro:bits.

CASE STUDY

Consumers tracking the food supply chain

Consumer tracking of the supply chain typically involves the use of complex computer networks. These include IoT devices and various information systems to provide visibility into the journey of a product from its origin to the end consumer (Figure 6.76).

Each food product is assigned a unique identifier such as a barcode, QR code or RFID tag. This identifier contains information about the product, such as its manufacturing details, batch number and other relevant data.

Shutterstock.com/STEKLO

Figure 6.76 Systems like IBM Food Trust allow not only businesses, but also consumers to track the source of all ingredients in a product.

Throughout the supply chain, different entities such as manufacturers, distributors and retailers collect and record data at various stages using sensors, scanners or other tracking devices. This data may include information about the product's origin, production process, transportation details and handling procedures (Figure 6.77).

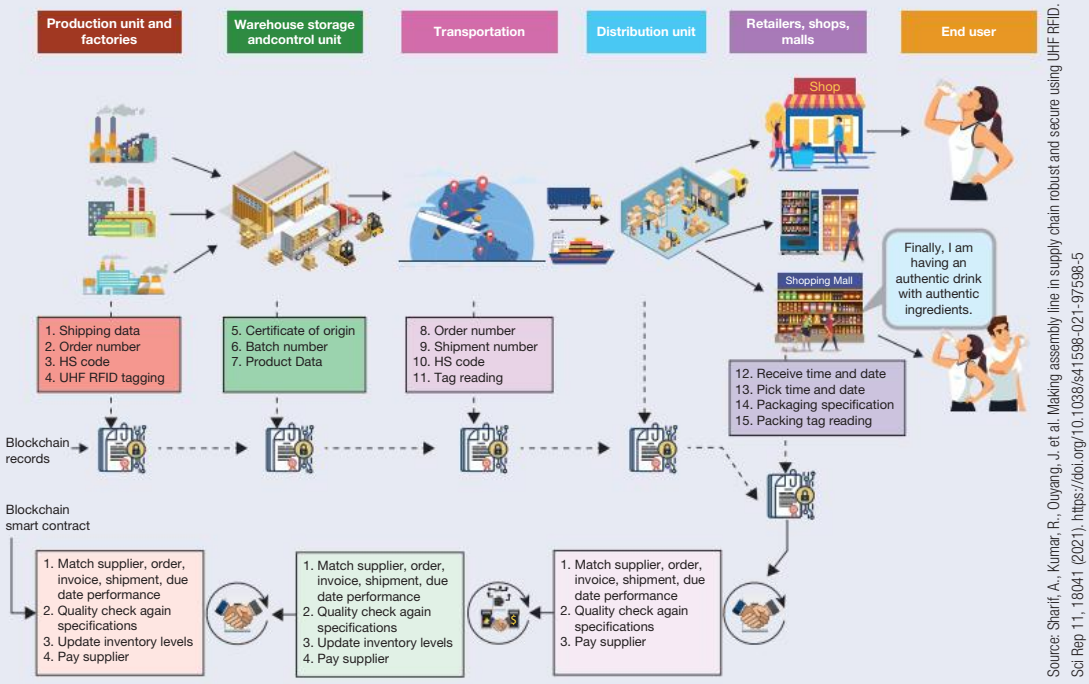
The collected data is shared among supply chain partners, including manufacturers, suppliers, logistics providers and retailers. Integration between their information systems enables the seamless flow of data across different stages of the supply chain. This integration ensures that information is up to date and accessible to all authorised parties.

Blockchain technology is used in supply chain tracking to enhance transparency, traceability and trust among the various participants. It provides a decentralised and un-editable record of transactions, ensuring the integrity of supply chain data.

To empower consumers with supply chain visibility, companies provide user-friendly interfaces such as mobile apps or websites. Consumers can scan a product's barcode or QR code, or input its unique identifier to retrieve relevant information about its origin, manufacturing process, quality certifications, environmental impact and more.

By tracking the supply chain, consumers can gain insights into the authenticity, quality and sustainability of products they purchase. This promotes accountability and encourages companies to maintain high standards throughout their supply chains.

IBM Food Trust is one system available to producers to facilitate consumer supply chain tracking.



Source: Sharif, A., Kumar, R., Ouyang, J. et al. Making assembly line in supply chain robust and secure using UHF RFID. Sci Rep 11, 18041 (2021). https://doi.org/10.1038/s41598-021-97598-5

Figure 6.77 The food product tracking process using IoT devices and blockchain records from the sourcing of the individual ingredients to the final product sitting on supermarket shelves ready for consumers.

Systems thinking

ACTIVITIES

Identify

1 What is the IoT?

2 How are IoT devices being used to improve the health of individuals?

3 What are the built-in sensors and outputs on the micro:bit microcontroller?

4 What is the purpose of the consumer product tracking system?

Analyse

5 How does a microcontroller differ from a personal computer, including the features and use?

6 Find a food product that allows supply chain tracking and perform a track. Reflect on the process.

7 Why is blockchain an important feature of the supply chain tracking system?

Extend

8 Find out about the new and emerging term the Internet of Everything (IoE). Describe what it is, in comparison to IoT and with examples.

9 Build another network game using the micro:bits. An example is heads or tails. The host has a series of true or false questions. Each player must select true (button 'a') or false (button 'b') on their micro:bit to indicate their guess, which is also displayed on the LEDs. The host sends the answer to all micro:bits. Those that correctly guessed have their display show a smiley face while the others have a frowning face. Added challenge: keep score!

Design thinking

Weblinks:
- micro:bit Python Editor
- Grok Academy
- Code.org: Internet Simulator
- G Suite. Tools: Visual Traceroute
- How secure is my password?

Skill builder resources:
- Build a wired micro:bit network
- Build a wireless micro:bit network
- Cafe floorplan
- School layout
- Build a micro:bit wireless data logger
- Build a silent tip game with micro:bits

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

6.8 Project: Create a model of a smart home or smart garden

The IoT evolution has led to a vast array of technology that allows us to automate processes in our daily lives. Our home is one environment that has been especially revolutionised by smart devices. Lighting can automatically be turned on and off as we move through rooms and our vegetable patch can be automatically watered when the soil moisture gets too low.

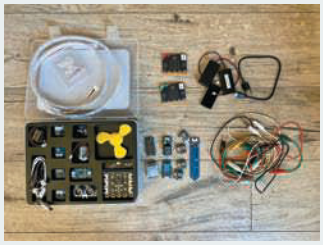
Your task

In this project you will create a model of a smart home and/or smart garden that demonstrates two or three automated functions and utilises a network of at least three nodes (e.g. main micro:bit with relevant sensors and actuator, receiver micro:bit and a personal computer). Ideally, you will build a physical model using materials such as LEGO, corflute, cardboard or other recyclable materials/items to construct the house/garden along with the micro:bits and relevant additional sensors and actuators (Figure 6.78). However, the model can be demonstrated using diagrams and micro:bit simulations. Alternative microcontrollers and accessories, such as the BitGadget Kit for micro:bit or Arduino boards with MAAS ThinkerShields, may be used.

Design thinking

It is recommended you opt for a theme such as smart home security or a smart study space for yourself, or focus on a system that has environmental benefits.

Building or household materials (eg lego, empty packaging)



Micro:bits (or other microcontroller) x2 with battery pack and USB cable. Sensors and actuators with cables and a shield to connect to the micro:bit

Figure 6.78 The resources you may need to create a model of your smart home or garden.

Identifying and defining

Tracking your project development

Set up a document to record the key processes during the project development. This document will likely show an iterative process, as you continually plan, produce and evaluate different parts of the whole project over the project's timeline. A logbook is an informal way to easily note down progress. If a more formal documentation report is required at the end of the project, your logbook can be used as a reference when creating a report.

Initial research and idea generation

1. There are many smart home award events that occur each year. Look through recent awards to see what categories of awards were given and what the winning products were.
2. In one of the categories that interests you, select one of the winning products and learn more about how it works. Summarise your findings, including the system name, purpose, users, inputs, processes, storage and outputs. Draw a network diagram that shows how this system may be integrated into a standard home.
3. Considering the categories you have observed and your interests, select a theme for your smart network model. Brainstorm related features and functions in a home (reflect on your home if this helps) that fit your theme (Figure 6.79). For example, if your theme is lowering the environmental impact of the home, features could include electricity consumption from lights and appliances, water usage in the shower or garden, wasted food in the fridge or cats escaping at night and harming wildlife. For each of these features, identify what manual processes are put in place to minimise the impact. Reflect on whether these could be automated in some way by identifying the manual data input (observation), process (decision making) and output (action) you do. You may like to discuss this with members of your family to find out their experiences.

9780170442787

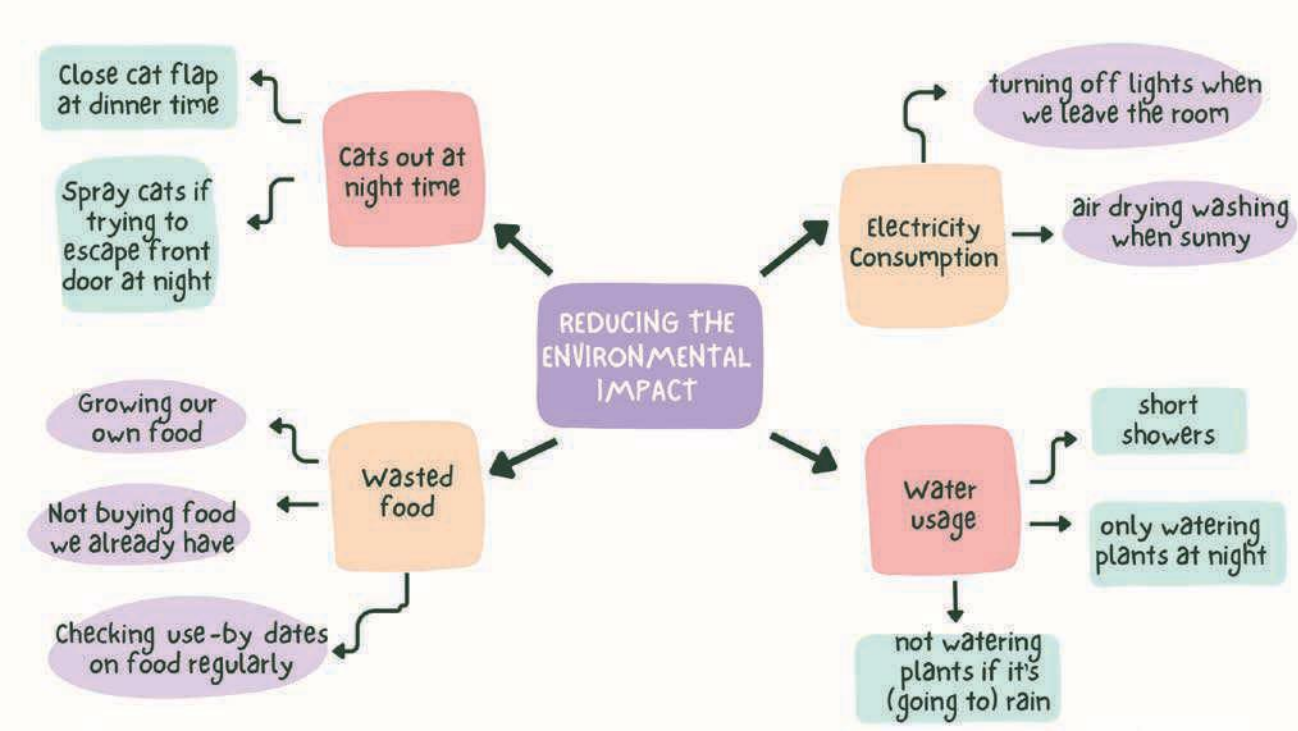


Figure 6.79 Mind map showing a brainstorm of ideas related to the theme of reducing the environmental impact through a smart house design.

Project requirements and considerations

4 Define some of the functional requirements for your model. Do this by selecting three automated functions you think you could include in a model of a smart home network. For example, to reduce electricity, your fan heater could be automatically turned off once the room reached a particular temperature or when there was no one in the room. Your cat flap could automatically lock or unlock based on the light level outside, preventing your cat from going outside at night time. You should define what the nodes in your computer network are, including their function and what data will be communicated through the network.

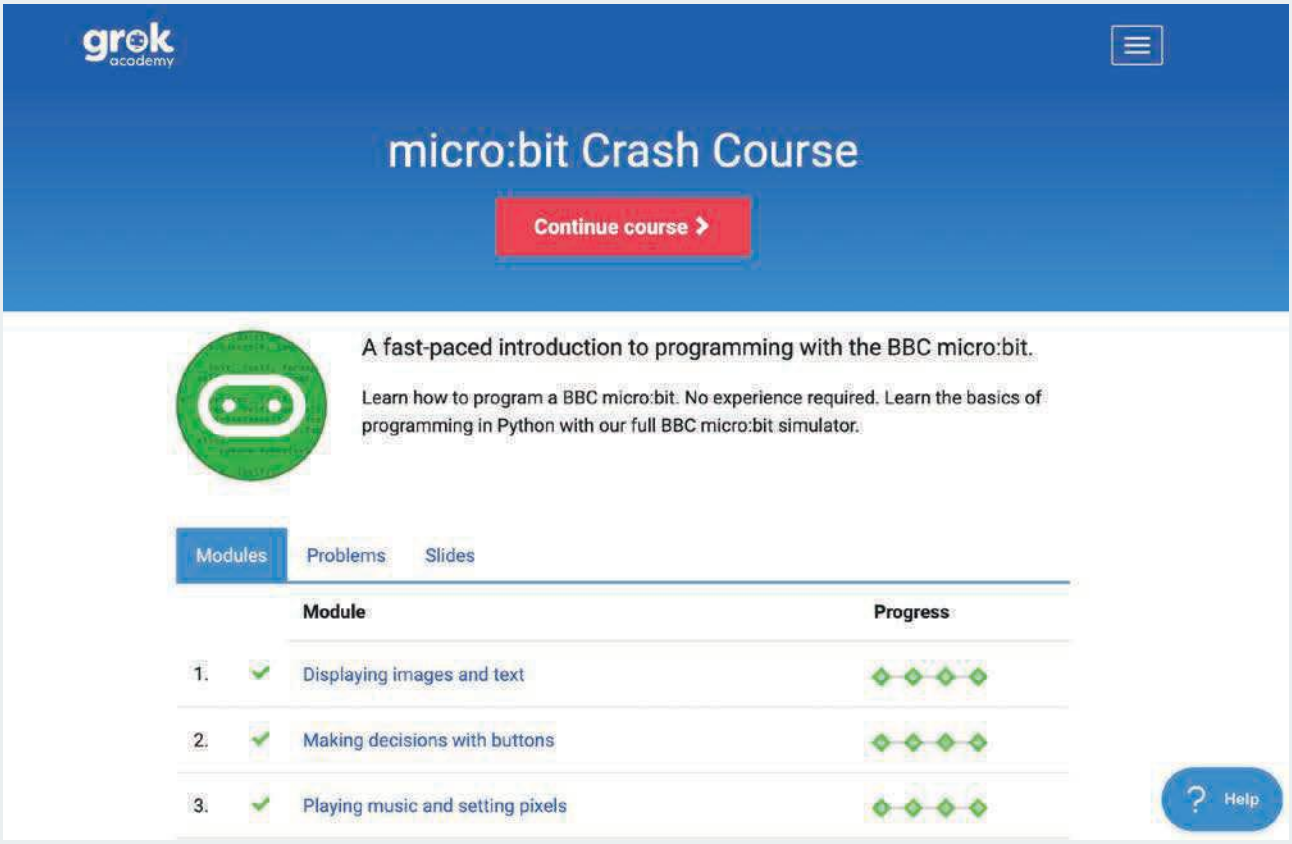
5 Define some of the non-functional requirements, including security of the data as it is transmitted across the network and stored on the cloud.

6 Discuss the factors or issues that could impact the development of your solution (e.g. time, resources, money).

Researching and planning

Further skill development

7 Complete any relevant tutorials to help you learn how to code your micro:bit(s), for example the Grok Learning courses (Figure 6.80).



Source: Grok Academy

Figure 6.80 You may like to complete a tutorial series, like the Grok Learning micro:bit crash course to build your knowledge of what is possible using the components and coding.

Sketching and modelling your smart home idea

Systems thinking

8 Sketch and label a design for the physical model of your house and/or garden.

9 Draw a network diagram to show how the technology in your smart home and/or garden is arranged and connected. Ideally, overlay this on your house sketch.

10 Construct an IPO chart to summarise how the automated functionality will work. Table 6.10 provides an example to get you started.

Table 6.10 IPO table for cat flap functionality.

Input	Process	Output
Light reading from light sensor	If below threshold indicating night time	Turn on motor to move arm to lock cat flap

Further research and designing

11 Research the technology available that could allow you to collect the necessary inputs and perform the required outputs. Ensure you consider what is compatible with micro:bit and what you already have available at school. Compose a list of equipment and parts, and put together a budget if required.

12 Identify what other resources you need in your model and where these can be sourced (e.g. LEGO stash at home, a hardware store) or how they can be created (e.g. 3D printer, repurposing materials like boxes and bottles).

13 Create labelled sketches of each device that shows the sensors, actuators and other linked components and how they are all connected.

14 Design the code for your micro:bit using pseudocode and/or flow charts.

Creating a prototype

15 Create a prototype of the main functionality using the micro:bit simulation functionality in your chosen coding environment (Figure 6.81).

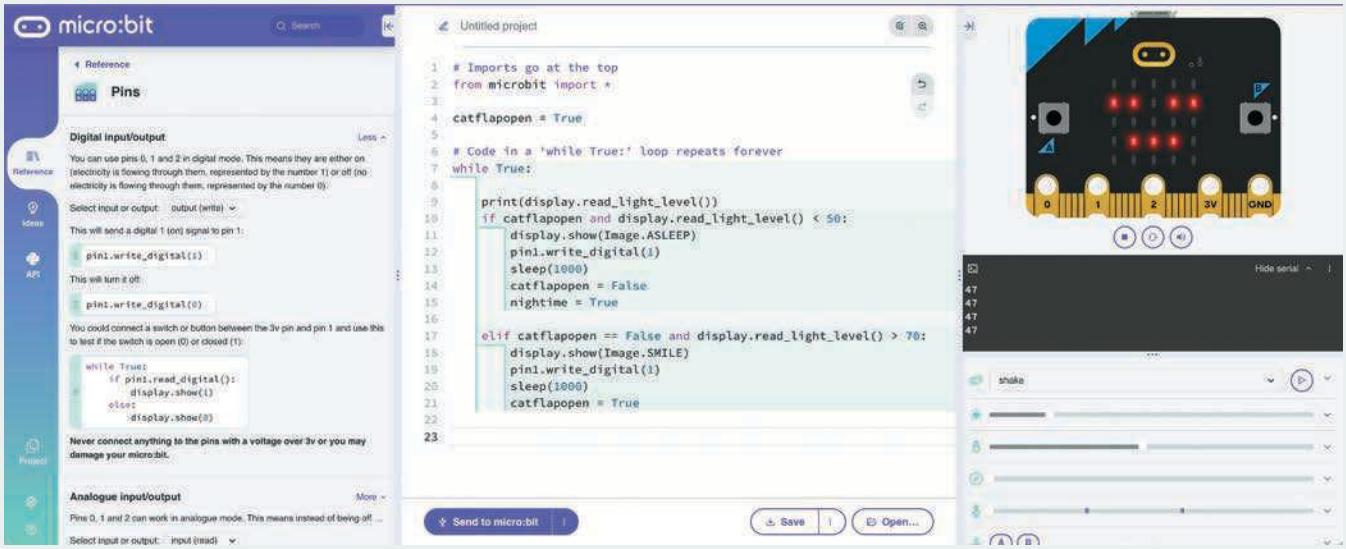


Figure 6.81 Prototyping the functionality for a cat flap that automatically locks at night time indicated by the amount of ambient lighting.

Producing and implementing

Constructing your smart home

16 Build your room, house or garden using your identified materials (e.g. LEGO).

17 Source and create any physical resources or components you need for your system, including the micro:bits, cables, sensors, actuators and other component materials.

18 Connect your micro:bit to any additional sensors or actuators using wires or expansion boards.

Coding your smart home

Computational thinking

19 For each function, create the code and test in the online simulator. Once functioning as expected in the simulation, upload to the micro:bit then test in the 'real' environment. If output is not as you expected, use programming debugging strategies to try and resolve the issue. One useful strategy is to include output 'statements' to help you identify what parts of the code is being executed and whether the collected data is as expected. The LED matrix can be used for this or serial output if the micro:bit is still connected to the computer via USB.

20 Create the code to transmit any relevant data to the receiver micro:bit. Create the code on the receiver micro:bit to receive the code. Define a protocol to ensure that the data is transmitted accurately and securely.

Testing and evaluating

System testing and evaluation

21 With all components individually tested and set up, perform a whole system test to ensure it works as a whole. You may need to simulate events in the environment to do this, such as adjusting the ambient light.

22 Reflect on whether the functional and non-functional requirements have been met. Identify ways the product could be improved to meet the requirements. List all the requirements and annotate the ones that were not met with justifications and/or plans for future improvements.

23 Demonstrate your smart home to your peers and ask for their feedback. Summarise the feedback in your documentation.

7 Designing the user interface

FOCUS AREA: Designing for user experience

Chapter outcomes

In this chapter, you will learn:

- the importance of the user experience (UX)
- the principles of user interface (UI) and UX design
- to evaluate the effectiveness of a range of UIs
- about the evolution of UI in hardware
- the importance of accessibility issues in UI and UX design
- the history of the graphical user interface (GUI)
- to evaluate user requirements
- to establish functional and non-functional requirements
- to design and build an effective UI for an application
- to use heat maps and analytics to evaluate a UI
- to avoid common errors in UI design.

Key terms

accessibility

command line interface (CLI)

functional requirement

graphical user interface (GUI)

natural language interface (NLI)

non-functional requirement

user interface (UI)

user experience (UX)

wireframe

9780170442787

IN ACTION

An immersive user experience

iCinema Centre/Neil Brown, Dennis Del Favero, Jeffrey Shaw, Peter Weibel

Figure 7.1 Once inside the T_Visionarium, viewers are surrounded by hundreds of 3D video clips.

iCinema Centre/Neil Brown, Dennis Del Favero, Jeffrey Shaw, Peter Weibel

Figure 7.2 A remote control enables the user to select and arrange video clips by 'moving' them in 3D space.

A *T_Visionarium* may be the ultimate user experience (UX) for searching a database! When the doors to the T_Visionarium open, viewers experience what it might be like to step inside a giant visual search engine. The aim of the project is to provide a better user experience for managing visual files in large databases.

The University of New South Wales Centre for Interactive Cinema Research's T_Visionarium was the world's first 360-degree 3D (stereoscopic) cinema. Inside, the viewers are surrounded by hundreds of 3D video clips swirling around them on a huge cylindrical screen that has an area of 120 square metres.

The experience is like a 3D IMAX theatre-in-the-round. However, wearing 3D glasses, the viewers are not only able to move about, but what makes this cinema unique is the ability of users to change the flow of images that unfold around them.

Over a period of one week, 28 hours of digital free-to-air Australian television was captured for the T_Visionarium. This footage was then converted into a large database that contains more than 20 000 video clips.

The viewer wears 3D glasses and stands inside a 360-degree cylindrical cinema screen with an interactive visual database and selects from the range of digital video clips. The software then merges clips chosen by the viewer into a universe of images encircling the screen (Figures 7.1 and 7.2).

All the images are projected in stereoscopic 3D, through a bank of data projectors mounted on the ceiling. Looping video clips are displayed and distributed around the huge circular screen and 'hang in space', surrounding the viewer and filling the circular cinema. It is possible to 'walk through' these clips as the viewer travels from one side of the theatre to the other. Using a remote control and interface, the viewer can select, rearrange and link these video clips in whatever order they wish by 'picking up' a clip in the air and dragging and joining it to another clip.

Each clip is tagged with descriptors (metadata) that define its properties. This information includes the gender of the actors, the main emotions they are expressing, the pace of the scene and actions such as standing up, lying down, arguing and kissing. A viewer can choose an emotion and a colour (say 'fear' and 'red') on the hand-held controller and each matching clip will sweep hover in front of the viewer. Those that do not have these qualities hang in space behind the viewer.

To see the T_Visionarium in action, visit the UNSW iCinema Centre website.

Weblink
T_Visionarium Project

Questions

1. In one paragraph, describe the user experience (UX) of the T_Visionarium project and describe how it assists users managing video database data.
2. How does 3D technology contribute to this project?
3. Comment on the following statement: 'Interactive cinema is blurring the boundary between audience and screen that has stood firm since cinema's dawning more than a century ago'.

7.1 The user experience

In this chapter you will learn what makes an effective **user experience (UX)** and discover the principles of effective **user interface (UI)** design. You will then use these principles to build interfaces that engage users and learn about the evolution of the modern UI for both software and hardware. You will learn how to meet the needs of a wide variety of users in your UI designs.

The first project at the end of this chapter is a guided project in which you design and build an intuitive user interface for a working mobile app that displays data from an online linked spreadsheet.

In the second project you use Adobe XD (a UI prototyping tool) to design and build a web-based app for a mobile phone.

In the next chapter, you will extend your knowledge of UX design by building a 2D game, exploring exciting 3D experiences using augmented and virtual reality, learning about careers in digital interface design and designing your own virtual 3D world using Unity 3D.

The very best user interfaces are the ones you do not notice. They are so intuitive and natural you don't think about them. This topic will help you learn the tools and principles to create such interfaces yourself.

What are UI and UX?

The UI describes all the features of a product a user interacts with, especially its screen design elements: graphics, icons, buttons, text entry fields and mouse pointer. Its design creates the look and feel of the object. If it involves hardware, it may also include light emitting diodes (LEDs) and haptic (touch) and audio feedback as well.

UX includes all the aspects of a UI but with an emphasis on the overall experience of a person using a product. It describes how easy and how pleasing it is to use from the viewpoint of the user. It includes the useability, engagement and emotional response provided in the interaction between a user and product. Not only computers, but also mobile apps, websites, video games, handheld controllers, household devices and car dashboards all depend on good UI design to produce a positive overall UX.

SKILL BUILDER

A UI of the future?

Alamy Stock Photo/Collection Christophel

Figure 7.3 Chief John Anderton (played by Tom Cruise) reflects the 2022 film *Minority Report's* view of the user interface of the future.

Weblink
Pointing to the future of UI

Watch the TED Talk, 'Pointing to the future of UI' by John Underkoffler, the designer of the UI in *Minority Report* (see weblink). Here he talks about what a UI of the future might look like. As we live in real 3D space, a big challenge for a modern UI is to be able to provide users with a useable and intuitive 3D interface when all we have is flat screens.

It was *Minority Report* that first popularised gesture user interfaces (see Figure 7.3). Tom Cruise (as Chief John Anderton) strapped on his infogloves accompanied by classical music and stood in front of a large semicircular computer screen. It was director Steven Spielberg who requested an interface like 'conducting an orchestra'. Cruise used his hands to fast forward and rewind video images on his screen. He zoomed in and out by moving one hand in front of the other. When he shook someone's hand his files were thrown to the floor. Strangely, all other workers seemed to be using keyboards for data access!

The man acting as consultant for these sequences was virtual reality engineer Jaron Lanier, who later developed the Kinect gesture-recognition system for Microsoft's Xbox 360®.

Tom Cruise's interface was science fiction back in 2002. Many of his gestures seem bizarre to us now, but others became reality and now form part of our lives: pinch to zoom, rotation and swipe-to-dismiss.

Three factors of UX

Three factors create an overall UX when using a digital hardware or software product:

1. the product
2. the context in which it is experienced
3. the user.

A product's design and features, especially the design of its UI are critical. The context in which the product is experienced is also important – is it being used as a unique tool or was one of many products that might have been used? The user's own background, their emotional and intellectual reactions will also play a role (Figure 7.4).

A UI designer's job is to create an interface which is logical, intuitive, functional and aesthetically pleasing but a UX designer's job is to create a product that provides the best possible UX. For a product to be successful, a positive UX will depend on a well-designed UI.

A tradesperson would not tolerate being forced to use a tool that was difficult to use or poorly designed, yet these issues often plague computer users. Companies often put great energy and expense into making their products perform their required functions and very little time into user-friendly design. Many companies' successes have depended on designing superior UX to those of their competitors despite having the same basic functionalities.

Designers need always to keep the users of their products as the central focus of their work. It is very easy for a developer or designer to become so absorbed in a project that they begin to experience it through their own eyes rather than that of the end users.

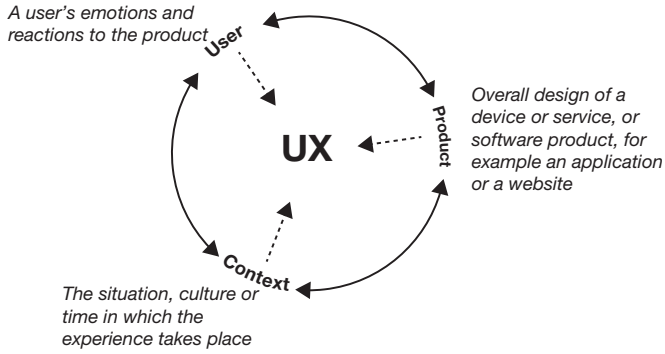


Figure 7.4 These three aspects contribute to the overall UX: the product itself, its context and the user.

INFOBIT

UI and UX designers have existed from the earliest times under various names and as part of other professions. It is only recently that the term UX has been widely used and strongly associated with digital technologies.

It can be very confronting for a software developer to stand back while a user tests their product. What they thought was a well-designed and intuitive interface can be a very different experience in the hands of a newcomer.

A few years ago, a boy filmed his father, an experienced user of Windows XP, trying the Mac OS X operating system for the first time.

His video shows the importance and impact of interface design on a user's experience!

Weblink
Using Mac OS X for the First Time

7.2 The purpose of the user interface

The purpose of a UI is to allow interactivity between the user and the system, its hardware and software. In software, this often comprises fields for text and number entry, mouse pointers, buttons and other graphical elements. In hardware, it may involve switches, keyboards, touch screens and LEDs, which provide information about the interactions between a user and a machine.

To use a device, you need to know what functionality the system has – what it can do for you. A well-designed UI should allow you to interact naturally.

Today, user interfaces are most often screens, however, there may be other forms of interfaces, such as the spoken word and physical devices.

The interaction between a digital device and the user is commonly via a GUI. This involves some mechanical action by a user – you move the mouse, click mouse buttons, tap the keyboard or touch the screen. A graphics drawing tablet or interactive whiteboard responds to touch and the movement of a stylus or pen. Similarly, a touchscreen provides an intuitive interface for young children and adults.

In effect, users are having a 'conversation' with a computer system through an exchange of signals. The software responds by providing visual and/or audio 'cues' and by performing other actions unseen by the user.

KNOWLEDGE BUILDER

Issues affecting UX

In this Knowledge builder we look at three UI examples involving both software and hardware that commonly affect UX.

UI design

1 Consider these before and after screenshots of the home page (also known as a 'landing page') of a website (Figures 7.5 and 7.6).

Figure 7.5 Original website.

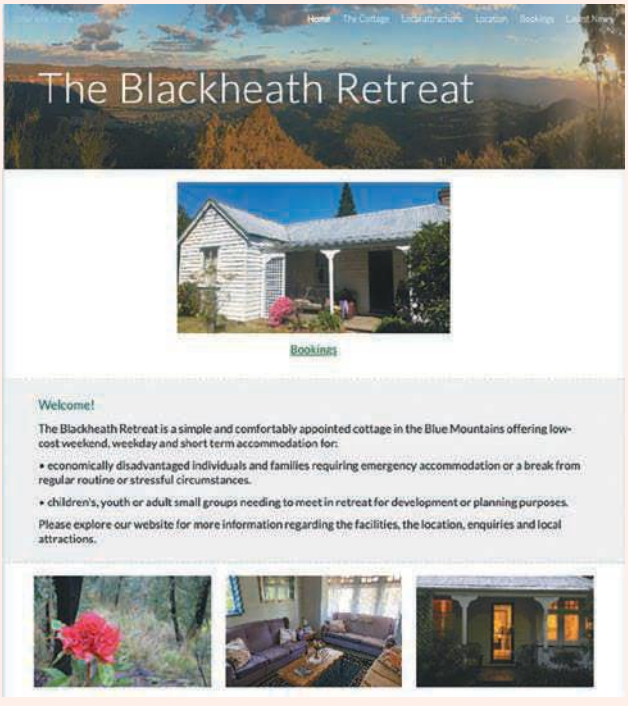


Figure 7.6 Result of website's home page makeover.

2 Copy and complete the table template (Table 7.1) below to help identify the improvements in UI design. Identify two other features and comment on them.

The new UI for this site was created in Google Sites, which achieves design consistency, as do many web builder tools, by offering website builders limited choices for object placement and fonts.

Table 7.1 Comparison of UI features template

UI features	Old version of site	New version of site
layout		
white space		
colour palette		
eye flow		
navigation		
typography		

Software UI functionality

3 Figure 7.7 shows an interface from a booking app for a fitness class. Before reading further, can you identify a UI issue?

The screenshot (showing Friday, 22) has only four classes scheduled, giving users no data on whether these are morning or evening times.

Identify at least two other UI issues.

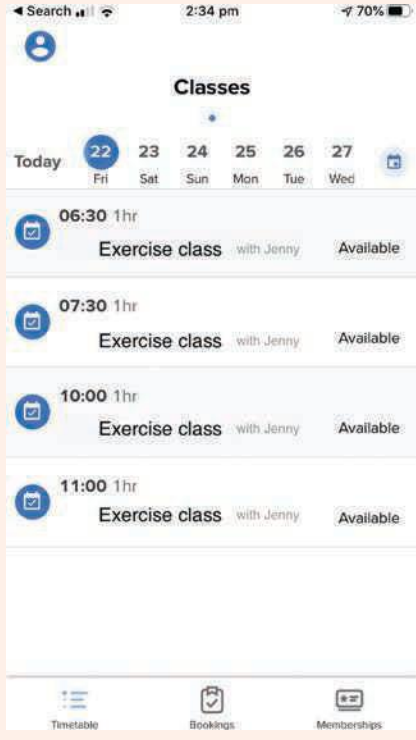


Figure 7.7 A booking screen from a fitness class mobile app. Can you identify three significant UI issues?

9780170442787

Hardware UI

4 Examine the image of a traditional television/DVD remote in Figure 7.8. Buttons are included for every function. With the advent of onscreen buttons and interactivity, it was possible to simplify this UI. Sketch your own design concept of both shape and buttons for a new remote.

iStock.com/mbbirdy

Figure 7.8 A typical television remote.

5 Figure 7.9 shows two generations of the Apple television remote. The first two generations used a circular rocker wheel for four switch functions. An infra-red capability enabled the device to be used without direct pointing. The fourth generation moved a microphone button to the right side for spoken instructions and added a mute button, a back button and replaced the large touch pad with a circular one with four directional rocker buttons.

Compare the interfaces of these new remotes with the traditional television remote in Figure 7.8 and comment on the design changes. Is there any loss of functionality? Suggest further improvements.

6 How has the UX been affected by using these new remote devices?

Alamy Stock Photo/Adam Bronkhorst

Shutterstock.com/Vladimka production

Figure 7.9 The UI evolution of a hardware device.

CASE STUDY

UI and product success: The mobile phone

A UI design is rarely a finished product; instead, it is gradually improved over many iterations.

Sometimes these improvements are due to changes in fashion. Fonts and colours go in and out of style and UX designers respond to these changes by modernising the UI. At other times, new functionality becomes available and forces UI changes.

At times, radical new approaches to a UI can make a dramatic impact on the use and popularity of a product. Consider the example of the mobile phone (see Figure 7.10). For many years, phones depended on keyboards and used 'nested' menus. Users typed messages or selected menu items using physical keys which occupied a lot of space. The Blackberry mobile phone had a full QWERTY keyboard with tiny keys. Each key of the biggest seller at the time, the Nokia 3310, represented four letters accessed by repeated presses. To select the phone's functions (address book, display settings etc.), users 'drilled down' through multiple hierarchical menus.

In 2005 at Apple, CEO Steve Jobs instructed his design team to research touch screens for a planned tablet product (which eventually became the iPad). While doing this he realised the advantages a touch interface would have for a mobile phone, overcoming the problems of physical keys and the awkward menus described above. Although it was not the very first touchscreen mobile phone, its design and UI were superior to others and its overall approach revolutionary. The iPhone could take photos, play music, send and receive email, browse the web, send and receive text messages, record notes, perform mathematical calculations and receive voicemail.

Its phenomenal success demonstrated the impact of good UI design on the total UX and showed the effect of good design on a product's success.

Shutterstock.com/Razulation Shutterstock.com/H_Ko Alamy Stock Photo/Chris Willson Shutterstock.com/marleyPug

Figure 7.10 The evolution of the mobile phone UX. L to R: The first 'handheld' mobile phone, the *Motorola DynaTAC 8000X* weighed over 1kg and took more than 10 hours to charge for 30 minutes' use; *Nokia 3310*, an iconic and durable mobile phone which sold 125 million and has recently been reissued; *Blackberry* offered a complete QWERTY keyboard and was very popular with business people; a *1st generation Apple iPhone*, which revolutionised the phone, replacing physical keys with a touchscreen.

ACTIVITIES

Identify

1 Describe the difference between UI and UX.

2 Identify and define three components of the UX.

3 What major UI feature was the trigger for the development of the smartphone?

Analyse

4 Justify the claim that hardware, as well as software, must be considered as part of UX.

5 Explain how the need to deliver websites and applications on many different devices affects UI design.

6 Compare the experience for a user of an early mobile flip phone with that of a touchscreen smartphone.

Extend

7 Analyse the UX of using a smart watch such as the Apple watch and identify the UI challenges it presents to UX designers, accessibility advisers and software engineers.

7.3 Types of user interfaces

Application software communicates with the operating system to drive the hardware to process, save or transmit your data. You can also talk directly to the operating system; for example, changing your screen settings, printing or installing new software.

A well-designed graphical UI enables a user to do all this in an intuitive and consistent environment whether using an application or the computer's operating system.

The way you interact with a computer system depends on the type of interface you are using. There are three main types of interfaces. These are the:

1. **command line interface (CLI)**
2. **graphical user interface (GUI)**
3. **natural language interface (NLI)**.

Command line interface (CLI)

Early UIs on personal computers were CLIs. These are still used today by terminals running the UNIX® operating system as well as by some network administrators to control busy servers. They do not require as much processing power as a GUI, and so they have an advantage.

In Figure 7.11, you can see several lines of information displayed on a CLI after the user has run the 'dir' command, which displays information about folders and files on the computer.

A CLI works a little bit like a chat line, except that you 'chat' with a computer operating system. However, to use a CLI, you need to know which commands to use, and you need to type them using a keyboard.

The graphical user interface (GUI)

The main idea of a GUI is to create a visual metaphor, a representation of another more familiar idea. A GUI uses the screen to represent a business desk and small pictures called icons to represent working documents. 'Window' is a metaphor for a 'window into a document' through which you see and work on the document (Figure 7.12).

The click of a mouse or drag-and-drop functionality is a natural action and easy for a user. The mouse pointer becomes an electronic extension of the user's hand. The introduction of the mouse is still considered by some to have been the turning point for the rapid adoption of computing around the world. (We look at the interesting story of its origins briefly in the next section.)

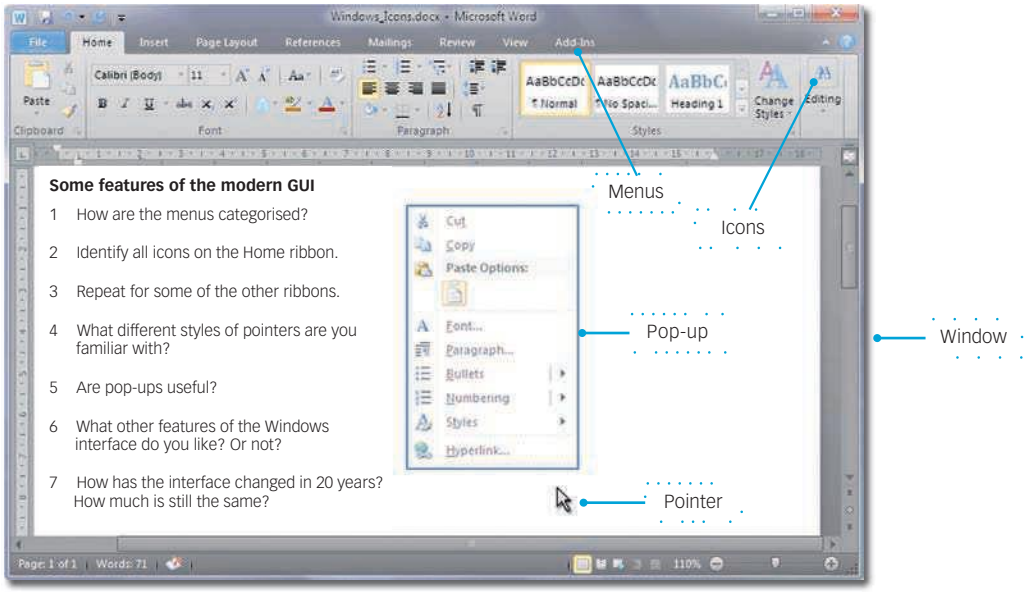


Figure 7.11 A typical graphical user interface on a personal computer provides windows, icons, menus and pointer (WIMP) for user and computer to interact. **Answer the questions** listed in the document displayed in this figure.

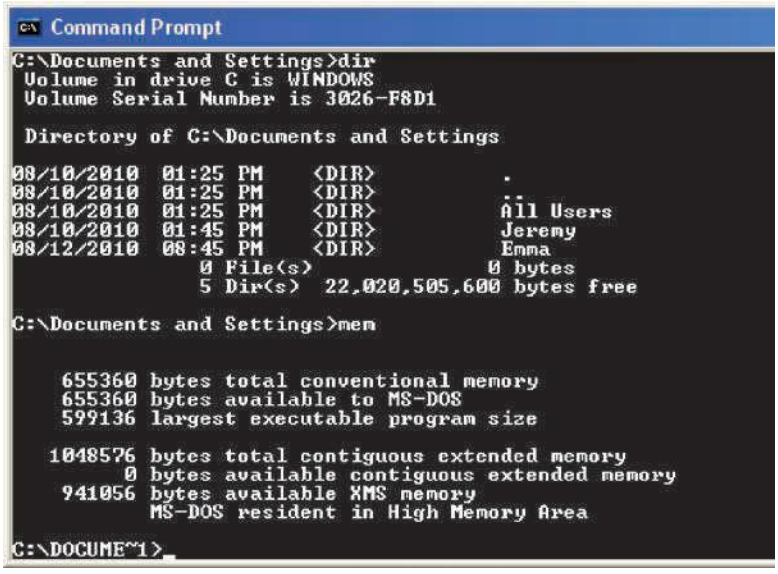


Figure 7.12 A command line interface. The 'dir' and 'mem' commands are used to display details of the directories and memory useage, respectively.

Natural language interface (NLI)

It is only recently in the evolution of UIs that speech recognition has allowed users to control their devices using their voices. *Siri* and *Alexa* are used extensively on mobile phones which are then often linked to smart home applications, allowing a user to perform many daily functions by voice alone, such as dimming lights or playing a particular song.

Natural language tools use speech recognition technologies allowing users to issue commands by speaking to the computer or request a word processor to take dictation (Figure 7.13).

Many such interfaces have been developed and are included in operating systems or as a part of special packages. We have already referred to the Apple television remote which uses speech commands to select television programs and performs many other functions. For example, a command such as 'What did he/she say?' rewinds 15 seconds and turns on closed captioning for about 30 seconds. Using AI, it responds to questions such as 'Show me action films from the 1980s' and 'Who played [character] in [movie]?'

An NLI is extremely helpful for those who find it difficult to use a keyboard or mouse, particularly people with disabilities.

Apple's Siri uses speech recognition and the phone's contact list to match spoken names, allowing users to dictate, 'Message Peter that I will be there in 20 minutes via WhatsApp'. There have been dramatic improvements in the successful interpretation of spoken instructions. Companies such as Apple and Google have been employing human operators to sample unsuccessful attempts and fine tune their algorithms. Sampling of users' conversations can also be used to customise advertising by revealing user interests.

Getty Images/Bloomberg

Shutterstock.com/DenPhotos

iStock.com/RyanJLane

Figure 7.13 Apple's Siri uses speech recognition and the user's contact list to match spoken names.

KNOWLEDGE BUILDER

Gesture recognition

Gesture recognition can be used as a way for computers to understand human body language, either with the hand or facial expression. In this way, we may be able to interact with gaming consoles, robots or other machines to input commands. Various gesture input devices have been created, many depending on sensors in gloves.

Examine Figure 7.14 showing Leap Motion's gesture input device. The device can be used with hundreds of games and to zoom over the Earth as if a helicopter pilot using Google Earth's application. A disadvantage is its absence of haptic (the sense of touch) feedback.

Current areas of research and development in gesture recognition involve recognition of human facial emotion using advanced AI techniques.

Such technological possibilities raise serious ethical concerns and one research academic at a Sydney university was so concerned he abandoned his research as a result of such issues.

Questions

1. Find a video of the Leap Motion's gesture input device in action and comment on its advantages and disadvantages.
2. Describe a game which where this device would be a suitable input for the game's UI.

Alamy Stock Photo/Aleksey Popov

Figure 7.14 The Leap Motion Controller captures movements of your hands and individual fingers so you can interact with your computer by swiping, pinching or grabbing in the air.

ACTIVITIES

Identify

1. Identify three types of UIs.
2. What is the purpose of a UI?
3. Briefly describe the ways in which a UI allows us to interact with a computer.

Analyse

4. Describe how gesture recognition can be used in a UI and identify two limitations.
5. Outline ethical concerns that might arise from advanced gesture recognition techniques.

Extend

6. Research three alternatives to the standard keyboard for text entry and discuss advantages and disadvantages for each.

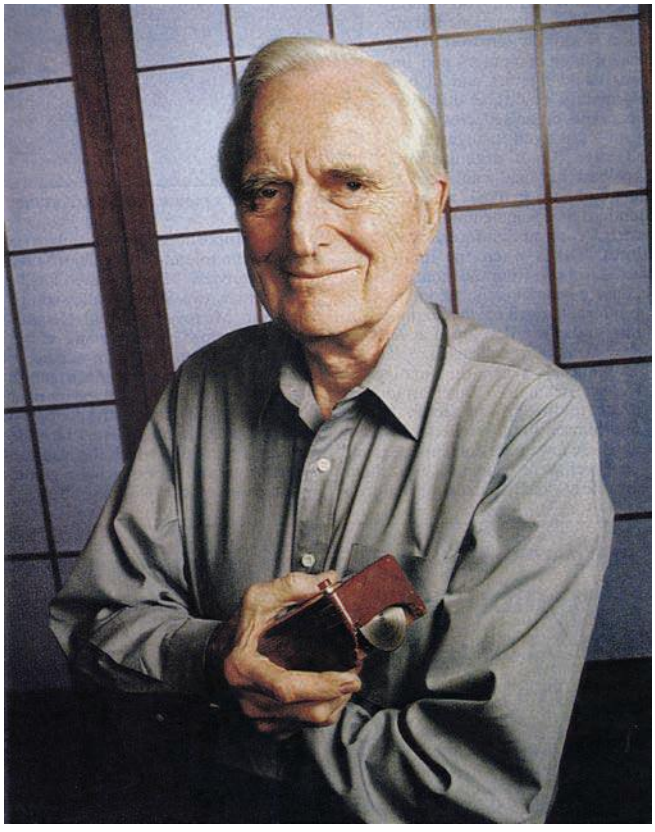
7.4 Evolution of user interfaces

A brief history of the GUI

In Chapter 3 you learned that the first computer interfaces involved plugs and cables, followed by paper tape then punched cards. In the 1970s, even the display of a letter of the alphabet on a television monitor connected to a keyboard was ground-breaking to an ordinary person! For a personal computer to exist, new UI methods were needed.

The first personal computer interfaces required instructions and data to be typed and were known as CLIs. MS-DOS is an example of a CLI that can still be seen on some computers today.

In 1968 Douglas Englebart developed a human interface for a computer which used an early version of the mouse (Figure 7.15). He used two wheels, aligned with x and y axes, in which movement could be measured as the wheels rotated using potentiometers, unlike later mice which sensed light falling through narrow slits in the rotating wheels.



Getty Images/Apic

Figure 7.15 Douglas Englebart with a prototype of his 1968 invention: the computer mouse. Its two opposing wheels can be seen.

Attendees called his 90-minute presentation at the Association for Computing Machinery, 'The Mother of All Demos'. This was the first time a computer system used the elements of the modern personal computer: windows, the computer mouse, hypertext, graphics, smooth navigation, typed commands, video conferencing, word processing and the collaborative editing of documents. It was his demonstration that led to Xerox's GUI project (Figure 7.16) which influenced the Apple Macintosh and later Microsoft Windows graphical user interface operating systems.

Flickr.com/cmnit

Figure 7.16 The Xerox Alto was the first computer to use the desktop metaphor and the mouse-driven GUI. The separate hard drive seen below the computer could hold 2.5 megabytes (MB) of data. A screen pointer could be moved by a 'mouse' (the small brown box) allowing icons to be selected on their 'Alto' computer (see Figure 7.15).

When Steve Jobs from Apple visited Xerox in 1979 with his engineers, he recognised the potential of the device and its GUI to change the user experience of the personal computer. Xerox had little interest in their invention and allowed Apple to use it in Apple's new Lisa computer. However, it was the Macintosh computer launched in 1984 that first popularised the mouse as an input device. Later, when Apple launched a court case against Microsoft's Windows 1.0 for copying its Macintosh technology (Figure 7.17), Bill Gates of Microsoft defended his company by saying, 'Hey, we both got it from Xerox'.

Old-computers.net

Figure 7.17 The Macintosh computer, introduced in January 1984, was the first to use a WIMP (windows, icons, menus and pointer) GUI.

Another important user interface innovation on the Macintosh was the use of proportional fonts. Until then, screen fonts looked like typewriter fonts where all letters (even the letter i) were the same width! It was Steve Job's only year at college (university) that inspired this innovation.

INFOBIT

In 1984, the launch of the Macintosh computer with its GUI and proportional fonts, the arrival of the first affordable laser printers and Adobe's new PostScript file format drove the sales of the Macintosh and dramatically changed the printing industry.

The emergence of the GUI on personal computers was a significant turning point in software development. No longer did the computer belong only to technical experts. Many people were now able to use one without learning special commands. Companies soon began to develop graphically based software applications for it and the technical specifications of computers expanded. More sophisticated programs demanded more powerful central processing units (CPUs) and monitors, greater storage space and ultimately, the need to communicate over the Internet.

GUIs had come a long way in 40 years.

KNOWLEDGE BUILDER

Trying UIs from the past

1 Early operating systems

Perhaps the only way to really appreciate how far operating systems have come is to experience the earliest operating systems through a number of sites which offer web-based emulations of the complete user experience.

Experience a fully emulated operating system by trying the following three weblinks and playing some of the hundreds of games and applications saved by Internet Archive.

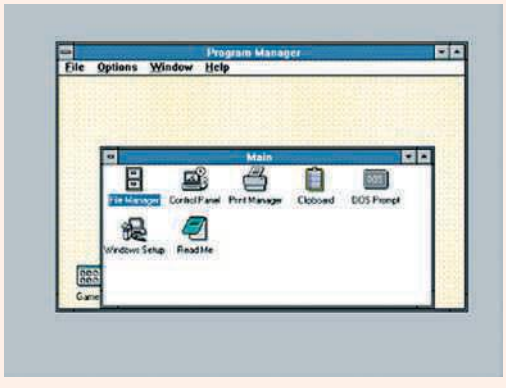
Experience an early computer with the Mac Plus emulator running OS System 7 weblink.

Play early Macintosh games with the Software Library: Macintosh weblink.

Play early Windows games with the Software Library: Windows 3.x Games weblink.

Questions

1. Identify three characteristics that distinguish early versions of Windows and Mac operating systems (as in Figure 7.18) from their current versions.
2. Identify three features distinguishing the current versions of Windows from that of the current Mac operating system.
3. Imagine and describe the UI changes you think might occur in operating systems 10 years from now.



Source: How to Geek

Source: PCMag

Figure 7.18 Two early computer emulations: 286 IBM PC compatible running Windows 3.0 and a Mac SE running System 7.

Weblinks
Mac Plus emulator running Mac OS System 7

Software Library: Windows 3.x Games

Software Library: Macintosh

Weblink
Old Web Today

INFOBIT

Free information advocate, Brewster Kahle, founded the Internet Archive in 1996 by 'collecting' the Internet itself! Like newspapers, the content published on the web is temporary – but unlike newspapers, no one was saving it. Today, over 20 years of web history and millions of artefacts are all accessible through the Internet Archive's 'Wayback Machine'.

2 Early browsers

Visit *oldweb.today* (Figure 7.19) and experience what it was like to browse the web 20 years ago (on the computers of the time). You may have to be patient as this site also emulates typical download speeds of the time. It not only emulates the browsers from the 1990s, it also recreates them using a choice of platforms (Figure 7.20).

Questions

1. Visit the oldweb.today site to view an early browser. After opening a browser and entering the URL for the *Sydney Morning Herald* (http://www.smh.com.au), select a date. View today's page of the paper by visiting today's paper using a modern browser.
2. Comment on the web design of the paper 20 years ago.
3. Identify UI differences now and comment on your overall UX in both cases.
4. Identify and comment on three UI features of a modern browser experience which were lacking in early browsers.

Source: oldweb.today

Figure 7.19 The site *oldweb.today* allows you to experience what it was like to browse the web 20 years ago using the download speed of the computers of the time.

Wikimedia Commons/Netfreak

Figure 7.20 Netscape's browser was the first experience of the Internet for many in the 1990s.

7.5 Planning a user interface

By completing the previous Knowledge builders, you will have encountered many UI and UX principles. We now collect and summarise these and you will begin to put them into practice in UI designs of your own.

Users rule!

The UI design process follows the stages of design thinking: empathise, define, ideate, prototype, test and modify (see Chapter 2).

Design thinking

At the very start, users are interviewed and their requirements decided.

User stories

A 'user story' is a technique for making sure the design of a digital solution meets the needs of the stakeholders, as they are the end-users of the product.

Usually, a variety of users are interviewed and asked to fill in the blanks in a sentence such as 'As a <type of user>, I want <some goal> so that <some reason>' (Figure 7.21). For example, 'As a user of the Fitness booking app, I want to be able to see what dates and times are available so I can easily make a booking online'.

User requirements

Before commencing any UI design work, the profile and attributes of end users of the product must be understood:

- determine your target audience (e.g. shoppers, tourists or school students)
- obtain user stories
- consider the attributes of the users (e.g. consider cultural sensitivities, age appropriateness, gender and interests)
- examine **accessibility** needs (e.g. hard of hearing, vision or physical limitations)
- security needs – the users' vulnerability to scams (e.g. the elderly and children).

Legal issues such as legislative compliance should be also considered throughout the project.

Finally, decide what criteria you will use to evaluate the success of the final product (such as the performance of included functions, the level of user engagement or sales).

Functional and non-functional requirements

Before commencing the design of a product, a designer needs to clearly define its requirements. These requirements fall into two categories:

1. **Functional requirements** describe what a product *must* be able to do. For example, a word processor must be able to create, edit, save and print documents.
2. **Non-functional requirements** describe *desirable* features of the product. For example, a word processor should be easy to use, be able to word count and save in various formats.

Figure 7.21 A UI is for the end-users of a product. Examining user stories will give UI designers a set of requirements for their design.

SKILL BUILDER

Weblink
Adobe Color

Create a UI colour palette from an image

Once a concept for your app has begun to form, it is a good time to think about your choice of colour palette. You will need a dominant thematic colour and some accent colours that go well alongside it, as well as a neutral colour for backgrounds, etc.

1. Adobe Color is free and allows you to create a palette (Figure 7.22) using an image. Find an image online representative of the Blue Mountains in New South Wales.
2. Visit the weblink Adobe Color.
3. Play around with the colour wheel under *Create*.
4. Create your own colour palettes by selecting *Extract Theme* and provide an image. Drag the small circles to sample colours and use the radio buttons provided to try out various Color Moods.
5. Select *Explore* and take a look at others' projects. See Figure 7.23.

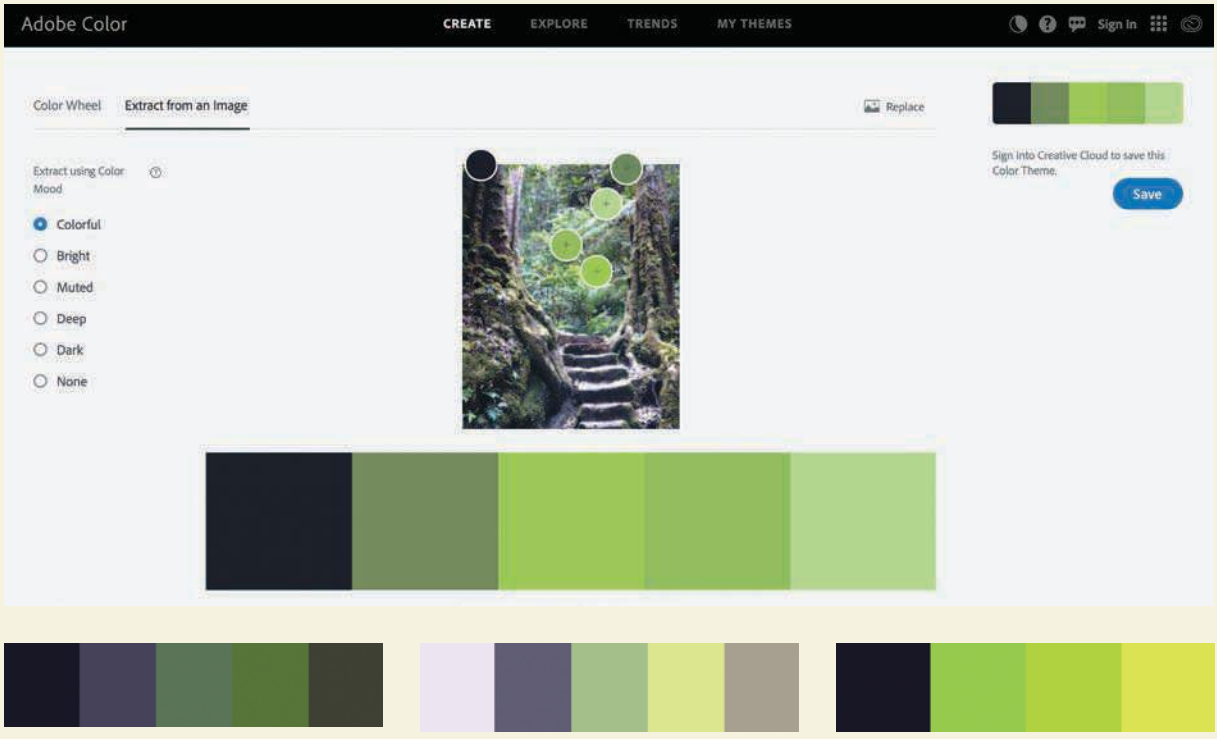
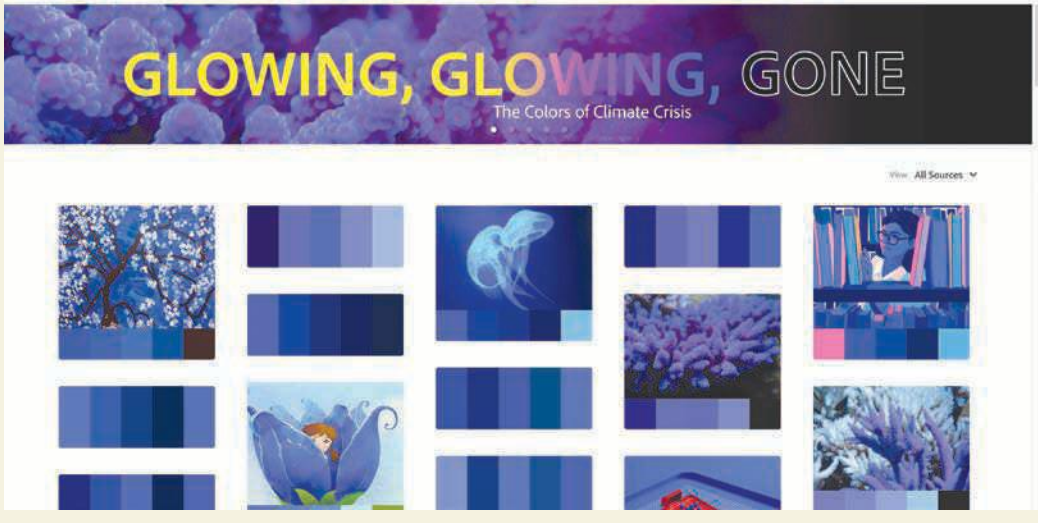


Figure 7.22 Adobe's free online application *Adobe Color* automatically creates custom palettes from any image provided. Above are four different 'colour moods' generated for one image.



Source: Adobe Color

Figure 7.23 *Adobe Color* stores palettes developed by users for their projects. (Choose *Explore*.)

KNOWLEDGE BUILDER

Functional and non-functional requirements

For each of the following common applications, list three functional and three non-functional requirements. Reproduce this table and complete it.

Application	Typical user story	Three functional requirements			Three non-functional requirements		
Photo editing software such as Photoshop	*e.g. 'As a professional photographer I want to be able to crop my images so that my subjects are framed well.'*						
A web browser							
Contacts app							

INFOBIT

Human interface guidelines

Operating system (OS) software controls everything that happens on a personal computer and is the main interface between user and device. Companies such as Microsoft and Apple put a great deal of time into the overall UX of their OS but they encourage developers who produce third-party applications for their systems to treat UI design just as seriously.

Both companies have published extensive Human Interface guidelines, outlining these principles. Take a look at each of these online resources via the weblinks provided.

Apple Human Interface guidelines

Microsoft User Interface Principles

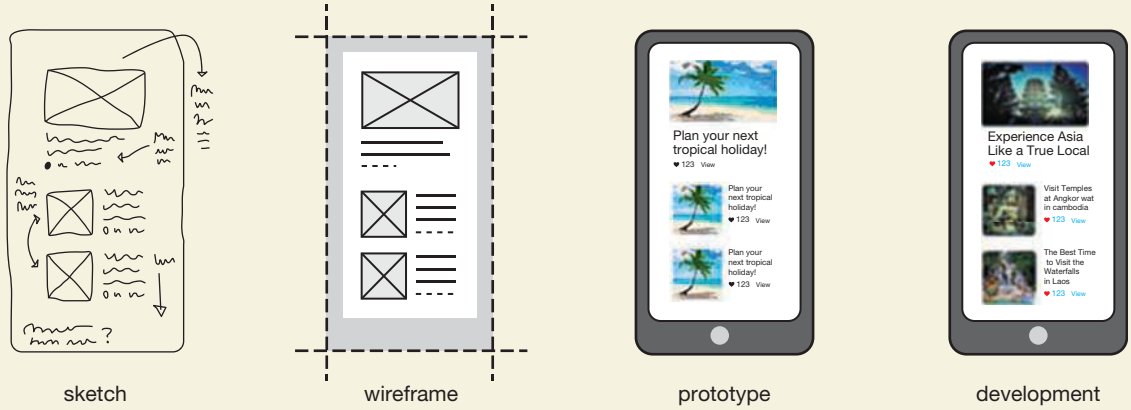
Questions

1. Compare the UX of accessing the design guidelines for each of these two companies. How easy are they each to access? How easy is each to interpret?
2. Identify briefly the main themes in these two documents.

Weblinks
Human Interface Guidelines

Getting Started Developing User Interfaces for Windows Applications

SKILL BUILDER



sketch wireframe prototype development

Source: https://artandlogic.com/solutions/design-services

Figure 7.24 Stages in the development of a mobile app: sketch, wireframe, prototype and final product.

Designing a UI

In the project at the end of this chapter you will build an interactive prototype for a mobile app for tourists about the (New South Wales) Blue Mountains. You will design the application on a desktop computer then view it either on your mobile device or via a simulated phone screen on your computer.

You have an idea for a mobile app for tourists that will allow them to select a town in the Blue Mountains, see thumbnails of tourist attractions, select one, read about it and see a link to its location.

In this Skill builder, you will start the design for the project to progress through the stages UI designers use when developing a product.

User stories

1 Gather user stories from classmates who will act as tourists and decide what your app should do.

Sketch the screens

Figure 7.24 shows stages in the development of a mobile app: sketch, wireframe, prototype and final product.

2 Decide what screens are needed for your app.

3 Designers decide on a basic grid to use for their design. If you look through the images in the project at the end of this chapter, you will see how a grid is used. A phone app with a small screen will need to use a different grid to a desktop app. Design a basic grid for your app and sketch its lines lightly in pencil.

4 Sketch some screen designs using your grid.

Colour palette, typography and design documentation

Note that as well as making sketches and mock-ups which form part of the design process, UI/UX designers keep a record of the evolution of the product using folios, versioning and other digital records.

5 Colour palettes and typography achieve consistency in design for both a single product and the image of a company. Select a colour palette for your site (see previous Skill builder). Select a main font. Try to avoid using more than two fonts on a single screen.

Navigation and layout sitemap

6 Decide on a navigational sitemap for your website. Start by drawing a very rough sketch, then create a navigational map using a free online tool such as *Figma* (free for three maps) or *MindMup* (Figure 7.25).

7 Create a rough sketch of the layout of the pages showing navigational arrows. A sketch is the only way to effectively show your idea to others and a quick and efficient way is to simply draw it on paper. No detail is required at this stage. Figure 7.26 shows our sketch for a tourist app. Many other layouts are possible.

Wireframe

This stage indicates the main features, functions and content of the UI without starting visual design. This is known as a **wireframe**.

A *wireframe* can be constructed using online tools (many of which are free) or by using standard drawing apps or even sticky notes arranged on a whiteboard, with marker lines drawn between elements.

8 Use sticky notes on a desk, wall or whiteboard to construct a wireframe for your tourist app (Figure 7.27).

Mock-up

For a mock-up, the UI designer provides actual design examples of typography, colour palettes, logos and styles. Usually, this is mocked up for a few pages or screens only.

Users and management will be involved at this stage to evaluate the design. UX and UI designers work together at this stage.

9 Optional: Use Adobe Illustrator, InDesign or Photoshop to mock up a design for two different pages of your tourist app. Consistency in your design is critical. Specify your typographic style definitions and use Adobe Color to specify the colour palette. Record these in a separate designer's brief.

Working prototype

Finally, a working prototype is produced which is close to the final product. In the project at the end of this chapter, you will use Adobe XD to move from design to wireframe to working prototype.

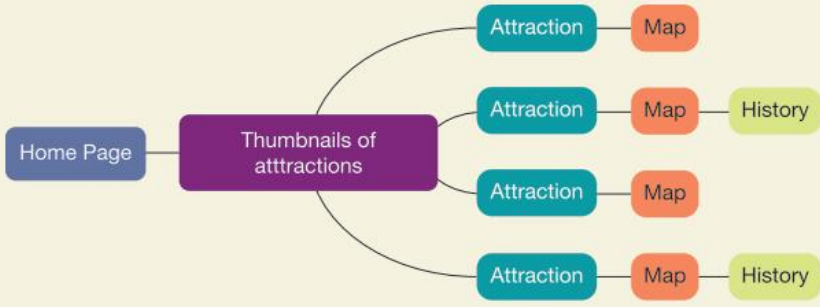


Figure 7.25 A navigational sitemap.

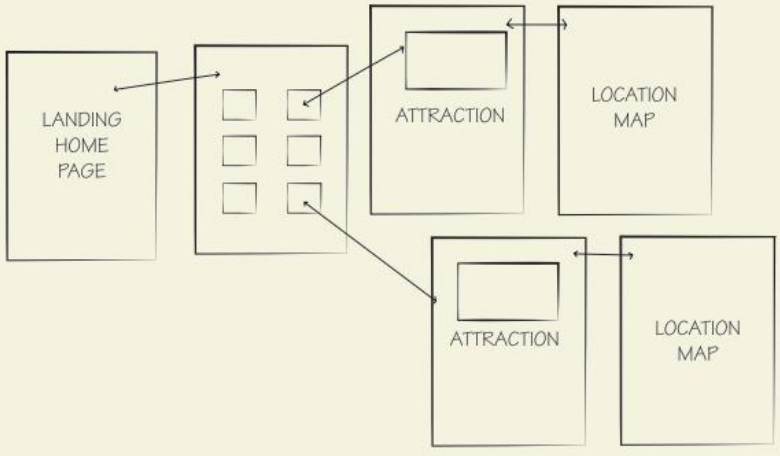


Figure 7.26 UI design begins with an idea then progresses to a rough sketch – usually on paper.

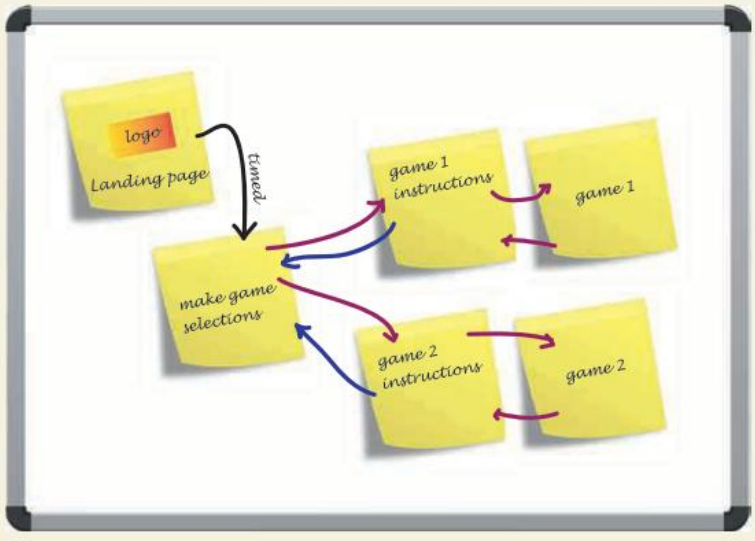


Figure 7.27 Example of wireframing using sticky notes.

ACTIVITIES

Identify

1 What was the first personal computer to popularise the mouse?
2 Who invented the mouse?
3 What is a WIMP?

Analyse

4 Outline three key differences between early web browsers and those of today.
5 Describe the difference between functional and non-functional requirements and provide an example, other than the examples provided in this chapter.

Extend

6 Use Google's *Ngram* viewer and identify when the term 'user experience' was first used, as well the year when it started to become highly popular.

Weblink
Google Books Ngram Viewer

7.6 UI design principles

Consistency

The design of software applications requires graphical user interface elements to be applied consistently. In a well-designed and consistent user interface, once you have learnt how something works, you can use the same knowledge in a new application.

A good GUI should be consistent in the:

- colours it uses for its elements
- layout of controls on forms and dialogue boxes
- behaviour of controls – drop-down lists, radio buttons, checkboxes and other types of controls should behave consistently across applications.

Another aspect of consistency is to use commonly recognised icons where appropriate (such as a garbage can for deletion or a pen for editing). This helps users feel at home with your app and to be able to use it intuitively.

There are widely used standards to help software designers maintain consistency – this helps make the software more intuitive.

Functionality

Menus and toolbars are used to provide access to a range of commands. Simply by pulling down menus and exploring dialogue boxes, users should be able to discover the basic functionality of the software.

Help menus usually provide helpful links to online tutorials and indexed searches (Figure 7.28).

Navigation

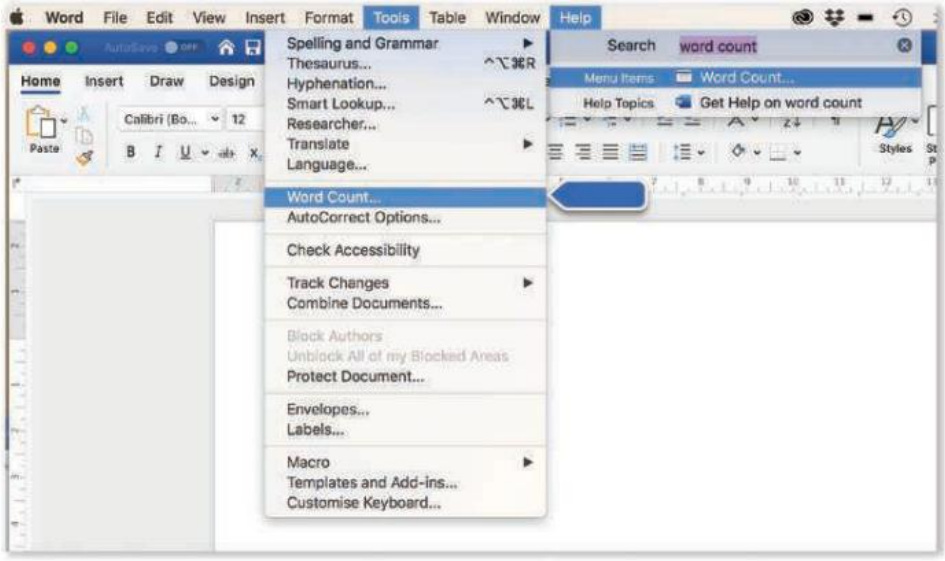
Navigation is the process by which you find your way in a software application. Software may have a hierarchical menu and command structure. For example, to insert a picture into a Word document, you might do any of the following:

- click the *Insert* menu on the ribbon
- choose the *Pictures* command from Insert on the menu bar
- choose from a submenu under the picture command
- make choices from a dialogue box.

You may choose to use either the mouse or keyboard shortcuts to work your way to the same part of the software. ***Important:*** GUI designers should always try to minimise the number of steps required to perform an action.

Similarly, if you are designing a layout for a website, you would minimise the number of mouse clicks the user will need to make to find any page on the site.

The unofficial 'three-click rule' states a user of a website should be able to find any information with no more than three mouse clicks.



Source: Microsoft Word

Figure 7.28 Can't locate word count? Typing 'word count' in Help in this Mac version of Microsoft's Word results in the item being revealed contextually by a gently rocking blue pointer.

Balance

Balance is achieved by placing items creatively and using white (blank) space effectively.

Balance is not only achieved by placement (see Figure 7.29). Items can also be balanced using size, colour, contrast and density. Items do not need to be placed symmetrically, however. Asymmetry can be dynamic and effective but requires greater design skill. Here is an example of the arrangement where balance and asymmetry are working together effectively.

Figure 7.29 An example of how balance can not only be achieved by symmetric placement, but also by using size, colour, white space, contrast and density.

Flow

A good design pays attention to the way a user's eye tracks a page. This is known as 'flow'. There are three main ways in which user's eyes track text.

For languages that read from left to right, on pages with little text and plenty of images or objects, user's eyes naturally move from the top left corner of the screen (the POA or Primary Optical Area) to the bottom right (TA or terminal anchor), as if under gravity (see Figure 7.30). The other two corners tend to be ignored by the eye and these 'dead spots' are known as fallow areas which should not be used for important information.

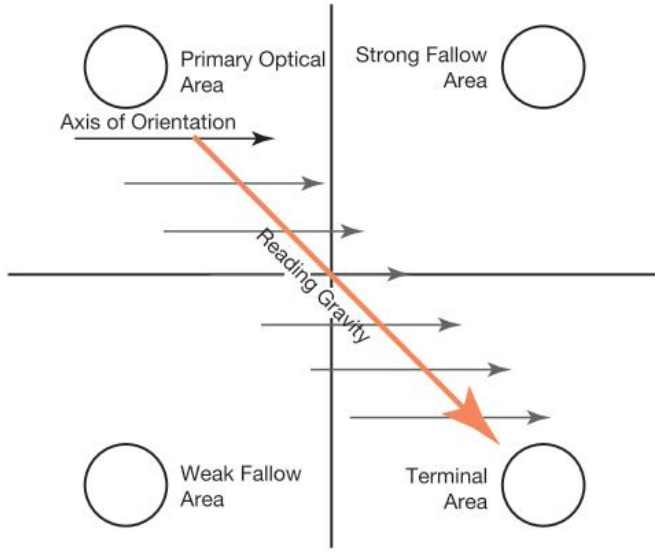


Figure 7.30 For screens or pages with little text and plenty of images or objects, a user's eyes move naturally from the top left corner of the screen.

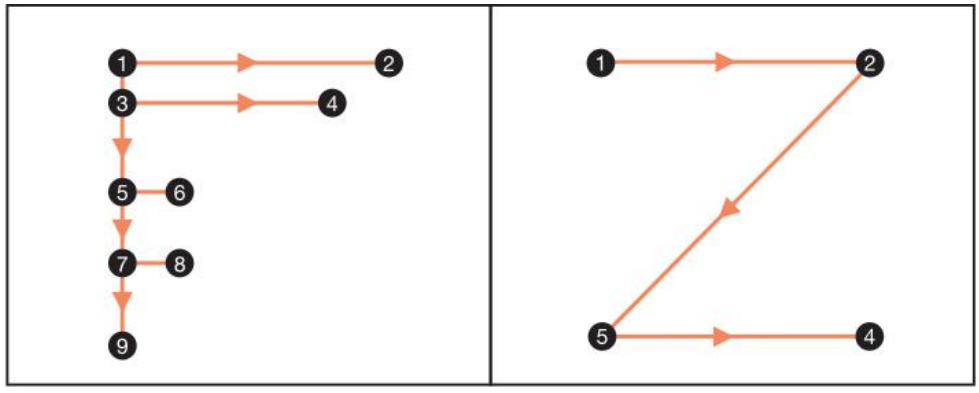


Figure 7.31 For screens or pages with lots of text and not many images or objects, the eyes often follow an F or Z path.

For pages with lots of text and not many images or objects, the eyes often follow an F or Z path (Figure 7.31). You can confirm this by looking at pages in this text, particularly those split by an illustration.

A viewer's eye will move around a screen in some direction anyway so the designer might as well take control!

UX engineer has an excellent illustrated summary of UX design principles (see the weblink).

Weblink
Unity: Principles of Design

Borders and white space

Most people think of white space as wasted space, but margins and white space need to be used carefully when designing an interface. White space is not necessarily white. It is empty space without text, graphics or objects.

Include enough white space so that your design does not look crowded. The amount of white space to include is subjective, so the designer needs to consult with users to find out how they respond. Figure 7.29 demonstrates how the use of 'white' space can be an effective design tool.

Borders can be used to group things together, to focus on some specific controls. The use of too many borders, or borders that are too thick, can be distracting. Again, feedback from test users can guide designers in the appropriate use of borders.

Clear instructions to users

Good software design includes well-thought-out instructions to help the user achieve their goal. Some actions require little explanation as they are quite intuitive.

If an online multimedia presentation contains audio or video material, the user may need to be reminded to turn on the speakers or click an icon to play the clip. Any instructions given should always be quite brief and very clear in their meaning.

When a specific date needs to be entered in an online form or application, a drop-down calendar, as shown in Figure 7.32, is a good solution to avoiding user errors.

In some software, the user is asked for confirmation of their intentions, particularly if the action is potentially damaging. Warnings such as 'This will delete all files – proceed?' give the user a last chance to change their mind.

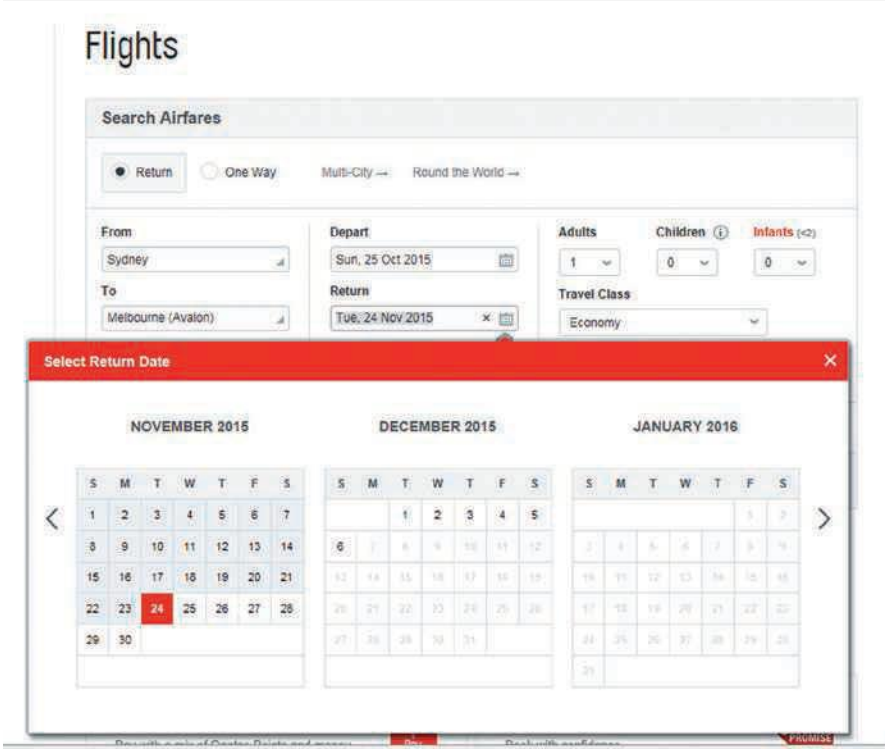
Control elements

Effective use of control elements is an essential UI feature for interactive products. In Figure 7.33 you can see some of the control elements used in software applications. The most commonly used include buttons, drop-down menus, palettes and text boxes.

Some regular controls that you may have used are detailed in Table 7.2. Other important control element examples are hyperlinks and tabs.

Table 7.2 Control elements are essential UI features for interactive products.

Control element	Purpose
sliders	provide a graphic representation of a sliding switch; e.g. a volume control
tab controls	used to group things together in a logical way on a dialogue box or form, and to save a lot of screen space
spin boxes	allow us to cycle through a list of choices
radio buttons	allow only one choice to be made for which the designer should set a default
dropdown menus	save space and declutter a screen when choices have to be made
check boxes	allow multiple choices to be made from a list of items(unlike radio buttons)



Source: Virgin Australia

Figure 7.32 Using a drop-down calendar to enter dates can avoid confusion about formatting.

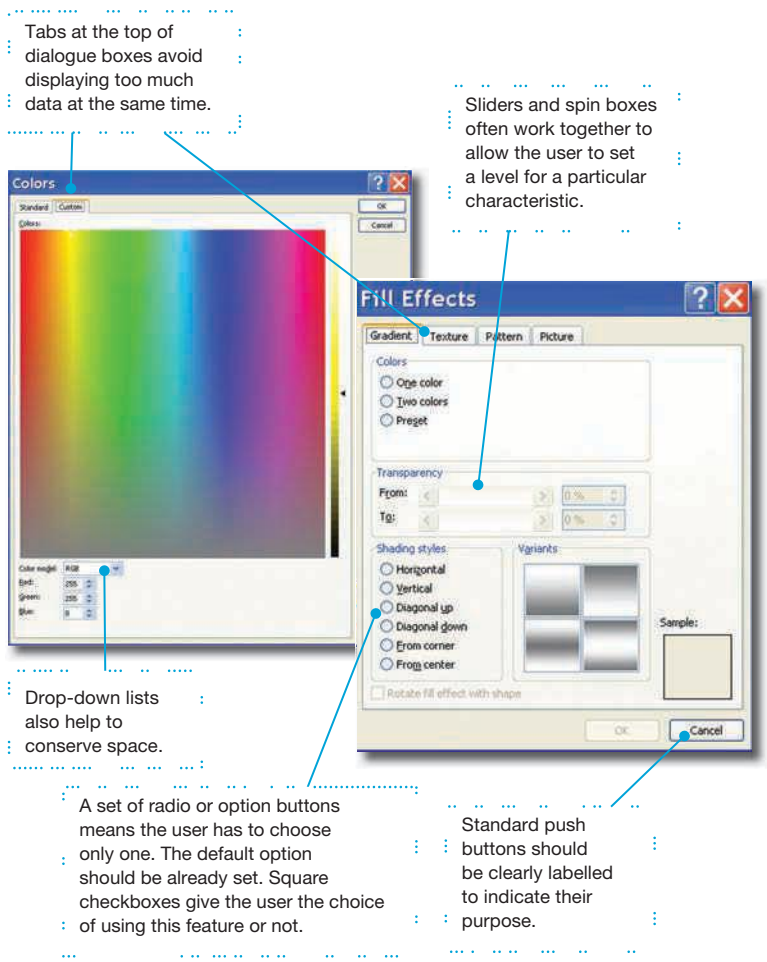


Figure 7.33 Examples of typical control elements in a GUI.

SKILL BUILDER

Designing the UI for a presentation

Shutterstock.com/Giovani Dressler

Figure 7.34 A popular online Australian graphic design app.

Canva (Figure 7.34) is a popular online Australian graphic design app, which has achieved international success. It can be used to create posters, presentations, simple websites and more.

Here, you will use it to create two linear presentations for two different audiences but using the same content dealing with the basic design principles outlined in this chapter:

- one for a friend in another school
- one for a graphic designer job application.

While you work, remember to apply the principles of UI design covered previously in this chapter.

1. Plan your content by creating a simple storyboard. Choose five design principles for five slides. Create a linear sequence showing the basic text and images but do not yet plan your design.
2. Open Canva and create a new Presentation. *From the Design Templates*, select a style that you believe would appeal to your friend from those provided at the left of the Canva window. We suggest type in 'graphics design' to start.
3. Replace the template's text and images in each slide as required. Import an audio track or sound effects using the *Uploads* button.
4. Your second design is to be suitable for a job interview, showing the interviewer your knowledge of design principles. Select a suitable base template from Canva. Replace the template's text and images in each slide as required. Import an audio track or sound effects using the *Uploads* button.
5. Compare the two design approaches you took and compare your approaches with others in the class.

Weblink
Canva

SKILL BUILDER

A website makeover

Earlier in this chapter, we examined a makeover of the home page of a website. Now you will perform your own makeover of a site you decide needs it on the Internet. You will use *Google Sites* to create a responsive UI – one that adjusts depending on the size of the screen on which it will be viewed.

Adaptive and responsive webpage design

For many years, following the launch of HTML in 1991, websites used fixed layouts. Web designers used tables to lay out their text and graphics. When small-screen devices arrived, websites looked terrible and web designers had to create three different versions of their sites: for desktop, tablet and phone.

Adaptive design tools meant web designers had to create three different UI versions of their websites, and browsers would display the appropriate version.

Responsive web design is an improved feature which automatically recognises the device on which the website is being viewed and uses algorithms to control the position and size of elements on the page to suit different screen sizes. Designers need only create one layout.

In Figure 7.35, the responsive design built into *Google Sites* adjusts elements in the layout tablet or phone. Compare the navigation here to the desktop version of the same website shown earlier in this chapter in Figure 7.6, which used a navigation bar at the top. For a phone screen, the title text changes position, the background image is cropped and the navigation bar changes to a 'hamburger' style menu which slides out from the left of the screen.

1. Identify a website with a UI you believe you can improve.
2. Use *Google Sites* to create your own version of the home page for the site you selected in step 1. Create an empty folder and download the images from the website's home page or take screen shots of the images. Copy and paste the text content into a text editor such as Notepad (Windows) or TextEdit (Mac). Create additional blank pages as placeholders for linked pages so they appear in your navigation menu.
3. Publish and view your site using a mobile phone, a tablet and a desktop computer. What features of responsive design do you observe?
4. Suggest two further improvements.

Weblink
Google Sites

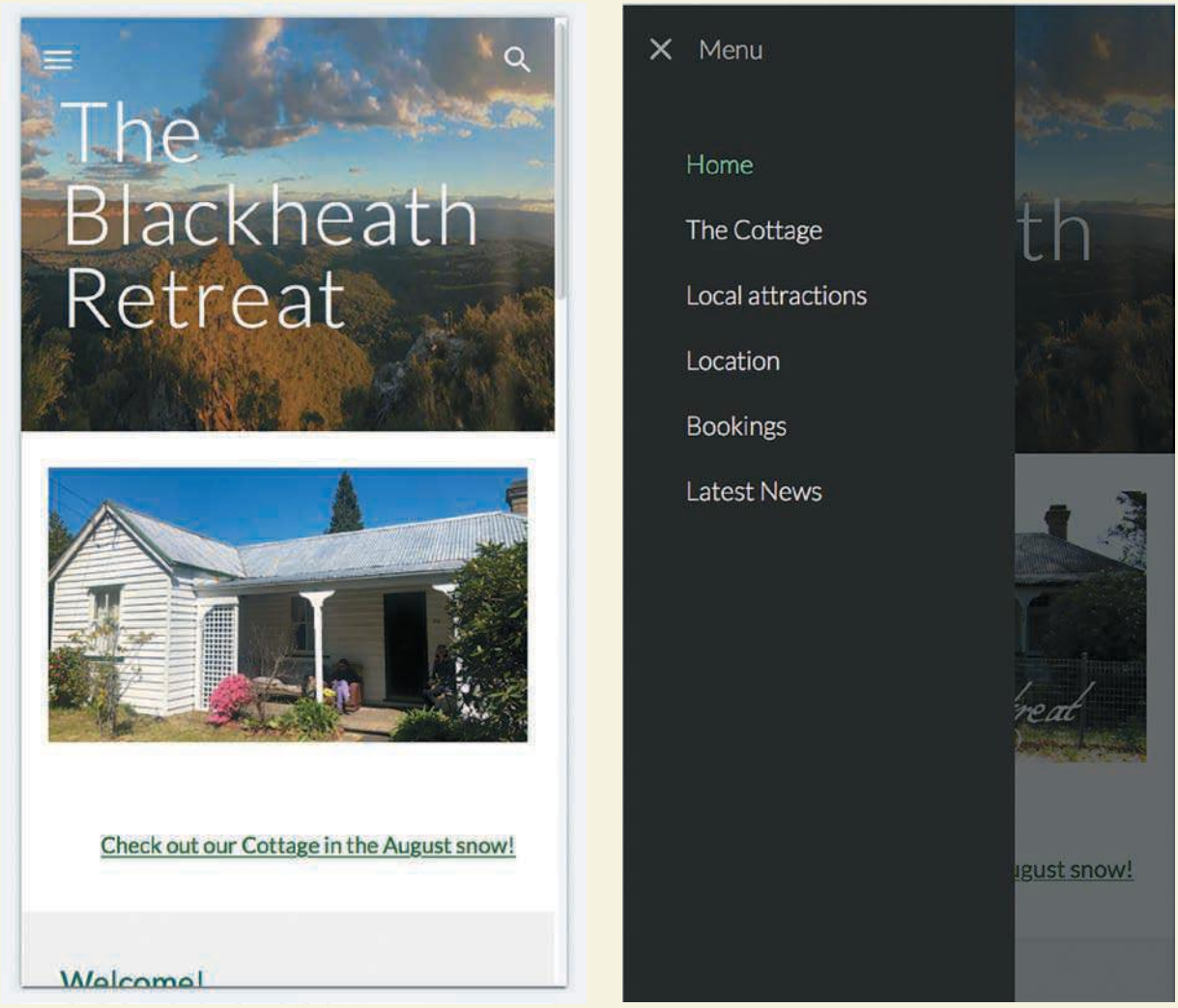


Figure 7.35 Responsive features of the home page for a phone screen format. The horizontal navigation in the desktop version in Figure 7.6 has now been changed to a dropdown menu.

ACTIVITIES

Identify

1 Identify five characteristics of good user interface design.

2 What are *control elements*?

Analyse

3 What do you consider the most common UI error in website design?

4 In what way does an inbuilt calendar assist when completing forms online?

5 Describe a digital UX design characteristic you find annoying and explain why.

Extend

6 Locate two websites demonstrating effective implementation for each of the following aspects of UI design: balance, flow, white space and control elements.

7 Do you believe a company should have the right to protect the UX of its products? Explain your response.

7.7 Audience and analytics

The target audience

Your interface design will be heavily influenced by the type of audience you are trying to reach with a publication, website or application. This is where user stories are useful (discussed earlier). These stories will allow users to tell you how they will use your site and what their expectations are.

SKILL BUILDER

Identifying and understanding a target audience

Copy the following table and for each site (see weblinks opposite), identify the target audience's age, background, abilities and interests as well as any unique cultural sensitivities they may share before establishing an interface design.

Weblinks
Sydney Morning Herald

CSIRO Ngan'gi seasons calendar

Australian War Memorial

ReachOut

	Weblinks	Age	Interests	Genre	UI
1	*Sydney Morning Herald*				
2	CSIRO Ngan'gi seasons calendar				
3	Australian War Memorial				
4	ReachOut				

Questions

1. For each site, complete brief details using the table and comment on the appropriateness of the UI for the expected audience.
2. Does the site cater for those with accessibility needs (elderly, vision impaired, hearing impaired)?
3. Score each site for the overall effectiveness of its UI for their intended audience from 1 (low) to 5 (high).

Heatmaps and eye-tracking

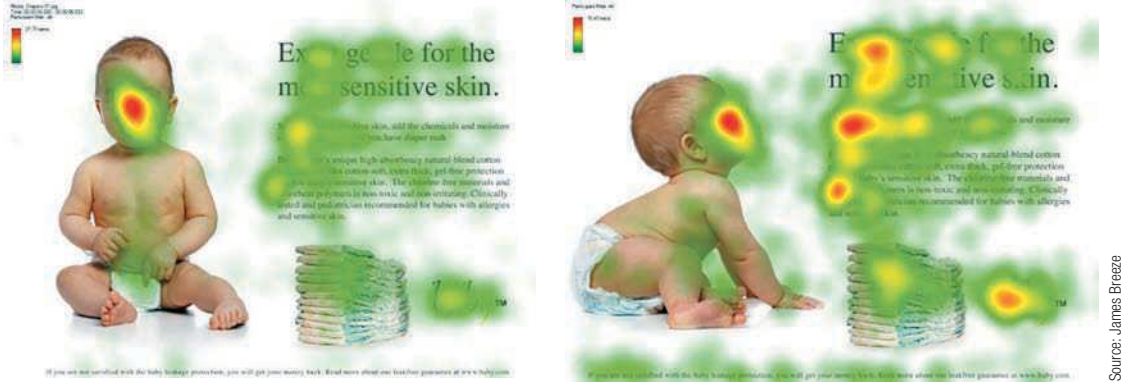
Heatmaps and eye-tracking can identify which areas of the screen (or web page) users look at most and the order in which they do so. This data is valuable to designers in fine-tuning their projects.

Using analytics

You can track a website you own using analytic tools. Such tools are available from Facebook, Squarespace and Google analytics.

Website analytic tools offer a huge amount of data to web managers, including a user's gender, age, country, whether a first-time visitor, which pages were viewed, social media selected, number of hits broken down by period and time spent on each page (Figure 7.37).

As well as this, site analytics offer clickstream analysis. A clickstream reports the sequence of hyperlinks visitors follow in the order they view pages. They may start with a search engine results page and continue through all the pages, including visited sites preceding



Source: James Breeze

Figure 7.36 When users see an ad with the baby facing forward they are drawn to the face. However, when the baby is side-facing the user looks at the ad's headline.

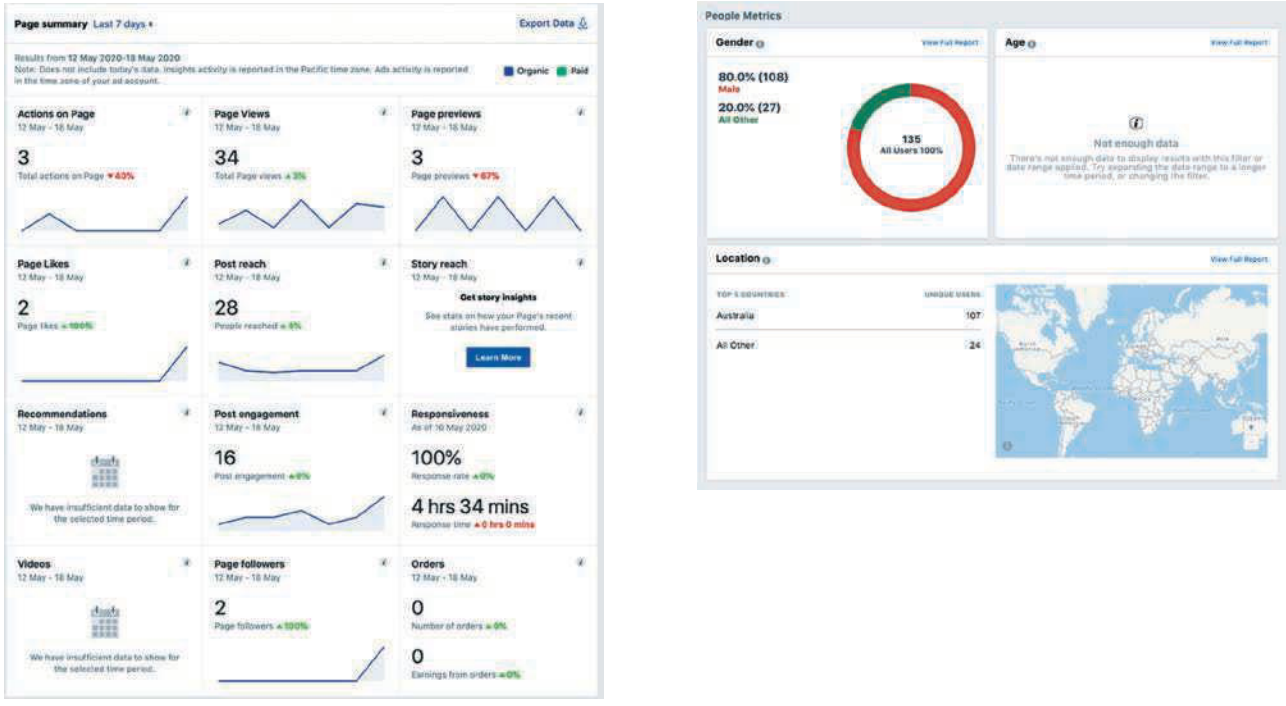
and following the session. Clickstream data can be matched to ads and key words to provide companies with important insights for marketing their products.

Knowing such data for a site provides designers with insight into the best placement of content, which social media links are most used and which layouts result in desired outcomes.

Analytics and privacy

You have probably guessed that analytic data presents significant data privacy issues. If such information becomes available to competitors, a company's operations may be seriously compromised.

In one well-known case in 2018, Cambridge Analytica acquired the personal data of 87 million Facebook users because users of a specialised Facebook quiz app had agreed to share their data. This allowed Analytica access to Facebook friends, most of whom had never given permission for access. The share value of Facebook dropped over $50 billion. The company's datasets have also been used as part of election campaigns in many countries. Cambridge Analytica's executives said in 2018 that the company had been contracted to political parties for more than 200 elections around the world.



Source: Facebook

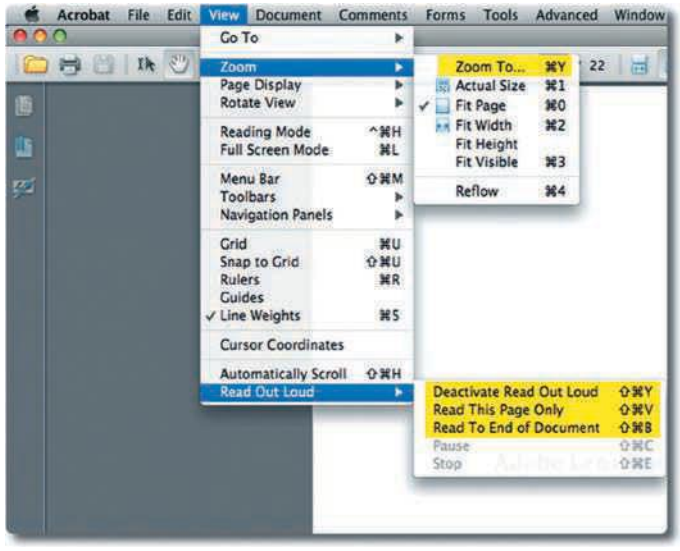
Figure 7.37 Facebook (a) analytics for a website showing numbers of hits and (b) user profiles.

7.8 Inclusive design

For a long time, software was designed to suit only those in the community who were able-bodied and had good eyesight. Over recent years, however, changes have taken place in community attitudes and legislation to ensure that software potentially includes all groups in society.

The move to inclusive design takes into account the needs of the ageing and those with disabilities.

The first step in inclusive design is to adopt an attitude that it is important to provide features to meet the needs of as many different groups as possible. Such principles are known as 'accessibility features' (Figure 7.38).



Source: Adobe Acrobat

Figure 7.38 Many applications, such as Adobe® Acrobat®, allow the user to have the text magnified and read aloud.

Weblinks
WAVE Web Accessibility Evaluation Tools

Apple: Accessibility

Microsoft: Accessibility

KNOWLEDGE BUILDER

Accessibility

'Accessibility' refers to the extent to which a system, environment or object may be used irrespective of a user's capabilities or abilities.

Designing UIs for inclusion

The UX of a device or application can be frustrating for those with special needs. Designing UIs which take advantage of assistive technologies (AT) such as haptic feedback, voice control, magnification, inverted colours and text-to-speech, speech-to-text can have a dramatic impact on the usability of technology for particular audiences.

Open WAVE (one of many free online accessibility testers), enter a URL and see what issues the report picks up.

Using familiar icons instead of words for young children who may not yet be able to read can free them to benefit from a range of educational applications.

1. Take a look at Apple Accessibility and Microsoft Accessibility and list three accessibility features from each.
2. Identify an accessibility issue not yet met by these companies. How might it be addressed?

CASE STUDY

Sony's Project Leonardo

Alamy Stock Photo/PA Images

Source: Sony Interactive Entertainment Inc.

Figure 7.39 (left) Microsoft's adaptive controller and (right) PlayStation's Leonardo adaptive controller.

Accessible gaming

Blind gamer Steve Saylor advises gaming studios on accessibility. He states, 'I am usually only one of many others on the advisory panel as there are so many different types of disability and each one is unique.'

Another user who lost the use of his right arm in a car accident said consoles are designed for people using two arms; however, there are many people who do not have the full use of both hands. He has seen people play using only their feet, others only their eyes. He said Microsoft's adaptive controller was the only option other than going to the expense of having a customised one built, but this was only useful for Xbox™ games.

Sony, maker of the PlayStation™, created Project Leonardo to address accessibility issues for gamers with disabilities. Sony stated that the Leonardo controller was like a canvas for gamers as it uses swappable components. The distance of the analog sticks can be changed, there are different shapes and sizes of buttons, and configurations can be constructed to suit strength, range of motion, and particular physical needs. Any available function can be mapped to a specific button, or two functions mapped to the same button.

It is worth watching users talk about how the Leonardo controller has helped them. Use the weblink to view the YouTube video.

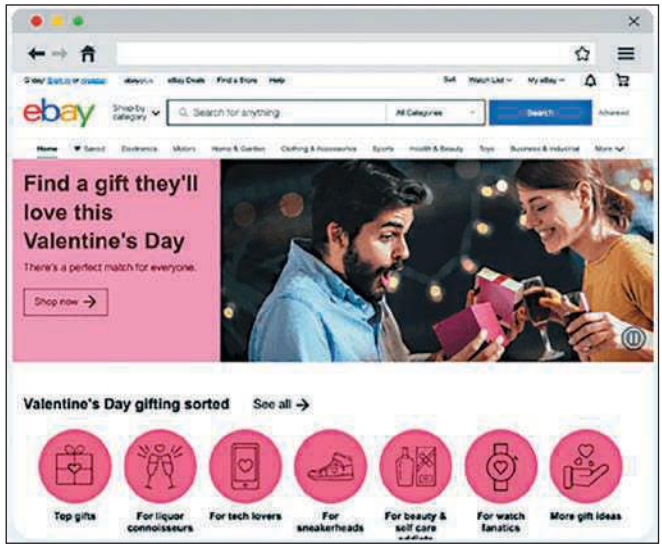
Weblink
Introducing Project Leonardo for PlayStation 5: Perspectives from Accessibility Experts

7.9 Common errors in interface design

Overload

First-time designers often forget that white space is a design element and should be used for effective design. It is not 'wasted space'. By trying to fit too much information onto a screen the user is given no focus and finds it difficult to interpret the excessive information before them (Figure 7.40).

Google the phrase 'worst websites' and examine the many examples!



Source: Indago Digital

Source: Indago Digital

Figure 7.40 (left) ebay home page, 2000; (right) ebay home page, 2019. Can you identify five design improvements that could improve the UI of these sites?

Distracting animations

When animated GIFs became available, many websites overused them. Animations are important but they should always have a purpose. They should never be included just because they move! Providing a kill switch is often a good approach on screens if looping animations must be used.

Links without labels

Using a URL without a keyword or using an unlinked hyperlink requiring the user to copy and paste the URL can be frustrating. Avoid using 'click here' rather than adding a hyperlink to a keyword.

Text that shouts

Overuse of uppercase letters, bolded text, multiple fonts and irregular justification of text are common errors. Inexperienced designers often lower readability by overusing centred justification for large sections of body text.

Too many fonts, too many boxes

A good guide is not to use more than two different fonts on a screen. Other errors are using fancy fonts for body text, having too many text boxes which give the impression of clutter, and using centred text in large blocks which reduces readability.

Pop-up windows

In the case of website design, the user experiences a lack of control when windows suddenly appear and hide what they are reading.

Image resolution

When web pages contain images at full resolution, often many megabytes, loading the page can take be tediously slow. If your images are large and compressed then a CODEC on the receiving computer needs to decompress them and this can slow down the display of a web page. You may have seen images that start out blurred then gradually become sharper, or others than slowly build line by line. Your images should not need to be much larger than 100Kb. You can learn about image compression techniques in Project 2 at the end of this chapter.

KNOWLEDGE BUILDER

Comparing UIs

Creating UI and UX

1 First select one of the following genres:
- games
- applications.

Select two similar products from your chosen genre. Use each product and become familiar with each. Comment briefly on your user experience.

2 Copy and complete the following table. Score 1 to 5 for each. (If the feature listed is not relevant, enter a score of 3.)

3 Comment on the scores. Were they what you expected?

4 Compare the overall user experience of each product in the form of a brief written evaluation.

Genre:		
Assign an effectiveness score from 1 (low)–5 (high) for each UX feature		
UX feature	**Product 1: <name>**	**Product 2: <name>**
Ease of use		
Input		
Colour palette		
Layout of elements		
Consistency		
Behaviour of controls – drop-down lists, radio buttons, checkboxes		
Functionality (and gameplay)		
Navigation		
Borders		
White space		
Instructions		

Help		
Inclusive design		
Speed of loading		
Speed of saving		
Latency		
Additional features		
Audio		
Cyber safety and security		
Supported file formats		
Overall score		

Comparing webpage designs

CSS Zen Garden

You have learned that UX is affected greatly by the design of the UI. CSS code is used on a webpage to control its appearance. The actual content of the page is unaffected by CSS. The long-running Zen Garden CSS 'challenge' demonstrated how the same HTML page, containing the same words and hyperlinks, can be made to look dramatically different simply by changing CSS code (Figure 7.41). Although this site is no longer maintained (the 'view all designs' button is no longer functional), you can still view hundreds of different designs that were submitted over the years.

5 Visit the weblink provided and click on at least five of the designs featured. Note that every page featured has the same content and yet the different designs have a dramatic effect on the UX!

Class activity

6 Select any integer from 100 to 200 and replace the 'X' with this integer in the URL: http://www.csszengarden.com/X and visit that site.

Display a chosen design on your computer in your classroom.

Source: http://www.csszengarden.com

Figure 7.41 A CSS design from the long-running Zen Garden collection.

7 Each student has three coupons (e.g. counters or paper slips) and votes for three favourites by placing one at each. You may not vote for your own.

8 Return to your computer and add up your votes (if any!).

9 Participate in a class discussion: what made some more popular (e.g. ease of navigation, readability, consistency, layout, fonts, colour palette, effective images, use of white space, focus audience)?

Using HTML and CSS

In this unit it is not necessary for you to code a website from scratch. It is possible to design and build highly effective websites using online web building apps such as Google Sites, Wix, Squarespace and WordPress without needing to know code. These sites use pre-built templates with built-in CSS code to achieve their designs.

Despite this, knowing the basics of HTML, CSS and Javascript are important skills and allows you to set yourself free from other people's layouts.

It is recommended you complete beginner tutorials for these three languages. Later in this text, we introduce these basics. Other resources are the free tutorial courses offered by Grok learning and W3 Schools.

Weblink
CSS Zen Garden

Weblinks
Grok Academy

W3Schools

PEOPLE

Careers in UI/UX: Millie Padmanabhan

Micaela Chutrau

Figure 7.42 Millie Padmanabhan.

How did you first become involved in digital design?

I used to work for a commercial VR start-up company doing production and video editing work. We were a close-knit team actively sharing our skills and knowledge. This really exposed me to other areas of the creative industry, such as digital design and media. I was lucky enough to be mentored by the creative lead of the team and coached in many aspects of design.

What projects have you worked on that pleased you most?

There are so many projects to this day, it is very hard to pick one! I'd say the first project that really had a huge impact on my design journey was at the start-up company. I worked with the team on the Woolmark VR app. The team was developing the video content, footage, VFX and some design elements for the project. I learnt a lot and it helped ignite my passion for design.

What do you enjoy most about your work?

I love the everyday design and customer challenges that come to my desk. A lot of the time, we find ourselves defying time and delivering creative work quickly. I enjoy elevating a piece of work that lacks design thinking principles. However, we often deliver at lightning speed, adding pressure on the team and the output. This race against time sometimes takes away from the simple joys of design and creativity.

What training did you do for this work?

I did a degree in 3D Art. As specialised as it sounds, the degree was quite generalist and touched on various aspects of production, colour-grading, compositing, 3D modelling, animation, design, VFX and art. Although I didn't pursue 3D as a long-term career, the degree gave me a good foundation in software and a grounding in design. I also did short courses online that elevated my skill set.

What software do you use most in your work?

I use a mixture of software. On any given day I use Adobe Photoshop, Illustrator, InDesign and Figma. I also occasionally use After Effects and Premiere Pro for any video related content.

What is the basic workflow for a typical project?

My basic workflow is to dissect and understand a brief first. In my line of work the brief has been previously worked out with the client and finalised before coming to our team.

I distill the brief and the target audience and start to work towards the end visual result.

I create and draw rough sketches or scamps of the visual. Then I start designing using any of the Adobe programs that are appropriate. The last step is to finesse the structure and layout of the design before it is sent off to the client.

My contribution can usually range from one to three days depending on the breadth of work required for the project.

What would you say is most important to keep in mind when designing a UI?

I am mainly responsible for creating the visuals for campaigns and marketing material for social and digital media. The most important thing to keep in mind is consistency in design. This includes the colour scheme across the designs, the typography and the composition. It is also vital to keep the design simple, on-trend and use a strong foundation of design principles.

What should a high-school student do to develop their skills in digital media?

As a high-school student I recommend becoming familiar with Adobe software, Figma and similar software by doing crash courses. These are widely available online and will give you hands-on experience using industry standard software.

I recommend creating a project of your own to push your creativity and help you learn as you go.

ACTIVITIES

Identify

1 What is the purpose of website analytics?
2 What is eye-tracking used for?
3 What is 'accessibility'?
4 Identify three features that should be avoided in website design.

Analyse

5 Describe three UI features that address accessibility needs.
6 Explain why consistency is an important UX feature in UI design.

Extend

7 Commercial news websites have tried many UI techniques for delivering advertising in an attempt to make their websites more profitable. Identify three approaches and evaluate each of these.

Chapter review quiz

Weblinks:

- T_Visionarium Project
- Pointing to the future of UI
- Using Mac OS X for the First Time
- Mac Plus emulator running Mac OS System 7
- Software Library: Windows 3.x Games
- Software Library: Macintosh
- Old Web Today
- Adobe Color
- Human Interface Guidelines
- Getting Started Developing User Interfaces for Windows Applications
- Google Books Ngram Viewer
- Unity: Principles of Design
- Canva
- Google Sites
- Sydney Morning Herald
- CSIRO Ngan'gi seasons calendar
- Australian War Memorial
- ReachOut
- WAVE Web Accessibility Evaluation Tools
- Apple: Accessibility
- Microsoft: Accessibility
- Introducing Project Leonardo for PlayStation 5: Perspectives from Accessibility Experts
- CSS Zen Garden
- Grok Academy
- W3Schools
- Glide
- Figma
- Get started with Abode XD
- Designing Adobe XD

Project resources:

- Designing, building and prototyping a mobile travel app

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

7.10 Guided project 1: Building an app with a functional UI

The importance of a positive UX is revealed in this interface design activity.

After exploring two spreadsheet-based interfaces, you will be using a great online app called *Glide*, with which you can create your own app that accesses live data from Google Sheets and design a custom interface for your phone or tablet!

Identifying and defining

The data

Imagine you regularly shop at your local supermarket. You scribble slips of paper to take shopping (Figure 7.43) but you lose them regularly and you are frustrated you can never remember which aisles your favourite items are in.

Figure 7.43 Shopping lists get lost, don't retain data and provide little information!

You decide you need to have permanent access to this information while shopping. As you add items this data will grow very large, so you will work here on a sample subset of data. If you like, you can replace this once you have finalised your design (Table 7.3).

Table 7.3 Data for supermarket aisle locator.

Supermarket item	Name of aisle
apples	produce
bananas	produce
bread	bakery
breakfast cereal	spreads, breakfast, baking and desserts
chocolate bar	confectionery, snacks and drinks
cream	eggs, dairy and fridge
oranges	produce

Requirements

1 Write a user story based on the information above. Use the template *As a [user], I want to [capability], so that [receive benefit]*. Make a list of requirements for such an app by creating the following table on your computer and listing three functional and three non-functional requirements.
Consider such user experiences as:
- ability to search for items
- ability to add new items from the app
- an image for each item to assist identification.

Add any others you can think of.

	Functional requirements			Non-functional requirements		
supermarket aisle locator						

Researching and planning

A spreadsheet

You decide to create a spreadsheet using Google Sheets and name it Supermarket.

2 Open Google Sheets (other spreadsheet apps will not integrate with your app). Open a new spreadsheet, name it Supermarket, then rename Sheet1 to *Items*. Name the columns as shown in Figure 7.44.

	A	B	C
1	**Supermarket item**	**Aisle**	**Emoji**
2	apples	produce	
3	bananas	produce	
4	bread	bakery	
5	breakfast cereal	spreads, breakfast, baking and desserts	
6	ice cream	eggs, dairy and fridge	
7	chocolate bar	confectionary, snacks and drinks	
8	cream	eggs, dairy and fridge	

Figure 7.44 Your items spreadsheet with aisle and emoji icons.

9780170442787

3 Make sure you have named this Sheet1 *Items*. This is important for later. Populate the spreadsheet with data as shown in Figure 7.44:
   - Enter supermarket items into Column A.
   - In Column B, enter the names of the aisles in which each item is located. You can easily make changes later.
   - You decide to improve the UX by adding emojis in Column C to represent each item. First you must double-click to make the cell active. To access emojis using Windows, press the Windows key + fullstop and on a Mac, press command + control + spacebar. Type the emoji into that cell.

Accessing this data using your laptop computer is very inconvenient in a supermarket! An app, however, would be in the cloud, and could be accessed easily using a tablet or smartphone while in a supermarket.

4 Evaluate this aspect of the UX by comparing advantages and disadvantages of using a laptop, tablet and smartphone to access data in the spreadsheet while shopping. Construct and complete Table 7.4.

Table 7.4 Comparing the user experience for a shopping app.

Laptop		Tablet		Phone	
+	–	+	–	+	–

5 Using the Google Sheets app, try searching for *bread* (imagine you are using a very long list and *bread* cannot be seen on the first screen).
   - On the Sheets app you must use *Search and replace*.
   - Do you agree this was very awkward and inconvenient? Imagine if you had a full list of supermarket items and you needed to do this each time!
   - Comment on your overall UX using this interface for this purpose by completing Table 7.5.

Table 7.5 Evaluating UX of Google Sheets app.

User experience: *Items* spreadsheet	
+	–

Even if you sorted rows alphabetically, you would still need to constantly scroll to locate them as you shopped (who shops in alphabetic order?!) or use search and replace constantly.

You will agree that a spreadsheet is not very convenient for shopping in a supermarket! However, although carrying a tablet while shopping is inconvenient, using a phone is not.

These are all UX issues for an app designer to consider. The ways in which people use an app should always be considered along with its functions.

6 Sketch a design for a mobile phone UI for this application. Show clearly all search fields, data fields and buttons. You will need more than one screen. Indicate how users will navigate between these.

Producing and implementing

Building an app with a customised UI

Design thinking

In this stage of the project, you will begin to use a special tool to build a mobile app to help locate the correct aisle for each item in a shopping list. You could do this by designing a database layout, however, this would be difficult to set up on a phone. Here we will build a phone app using the free online tool *Glide*.

Your app will present data stored in a spreadsheet in a far more convenient form for a supermarket shopper. You will build your own custom interface, suitable for the small screen on a smartphone, but you could also design this for a tablet.

Weblink
Glide

7 Now open the Glide website using the weblink provided and sign up with your existing Google account. This is necessary for Glide to automatically access your Google spreadsheet.

8 First, view the Quick Start video tutorial in the Support section. Glide apps are easily created using data in a Google Sheets spreadsheet.

9 Next, create a team name (you might want to collaborate later), then click the large black '+' icon to create your first project on the Glide app. Name it, select mobile and select its source as the Google Sheets supermarket spreadsheet you created earlier (Figure 7.45).

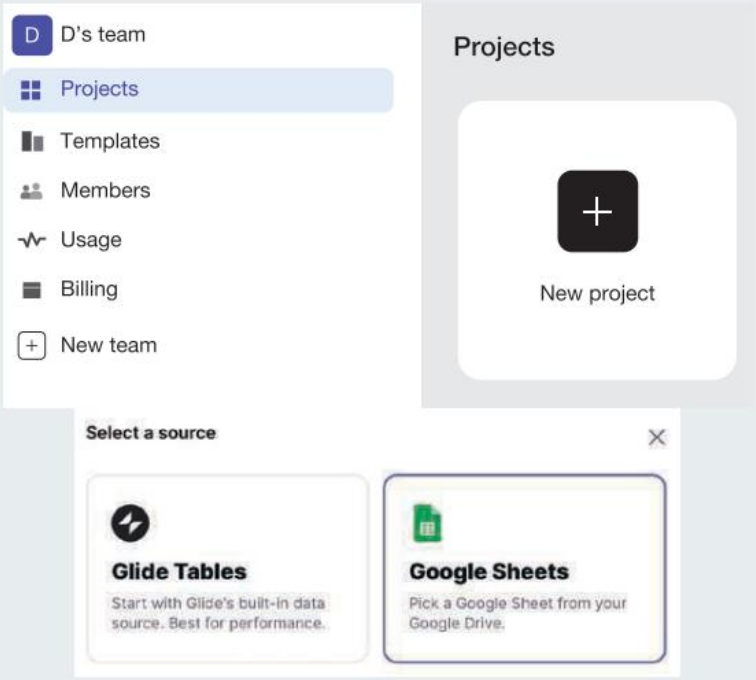


Figure 7.45 Create a Glide team account and create a new project linked to your Google Sheets account.

After linking, your new app is created automatically.

10 Choose the *Table* format from the *Style* choices in the right-hand panel. Your app will appear, but without emojis. This is because the layout is showing only the first two columns. Select Glide's *Data* tab and drag your emoji column to the left to occupy the first column position (Figure 7.46).

11 Refresh the view by selecting the *Data Grid* format from the *Style* choices in the right-hand panel, then click back to *Table* view again. In the right-hand panel, under Items Data, make sure Emoji is selected beside Image.

12 Explore the effect of other formats from the *Style* choices. Return to *Table* style. See Figure 7.47.

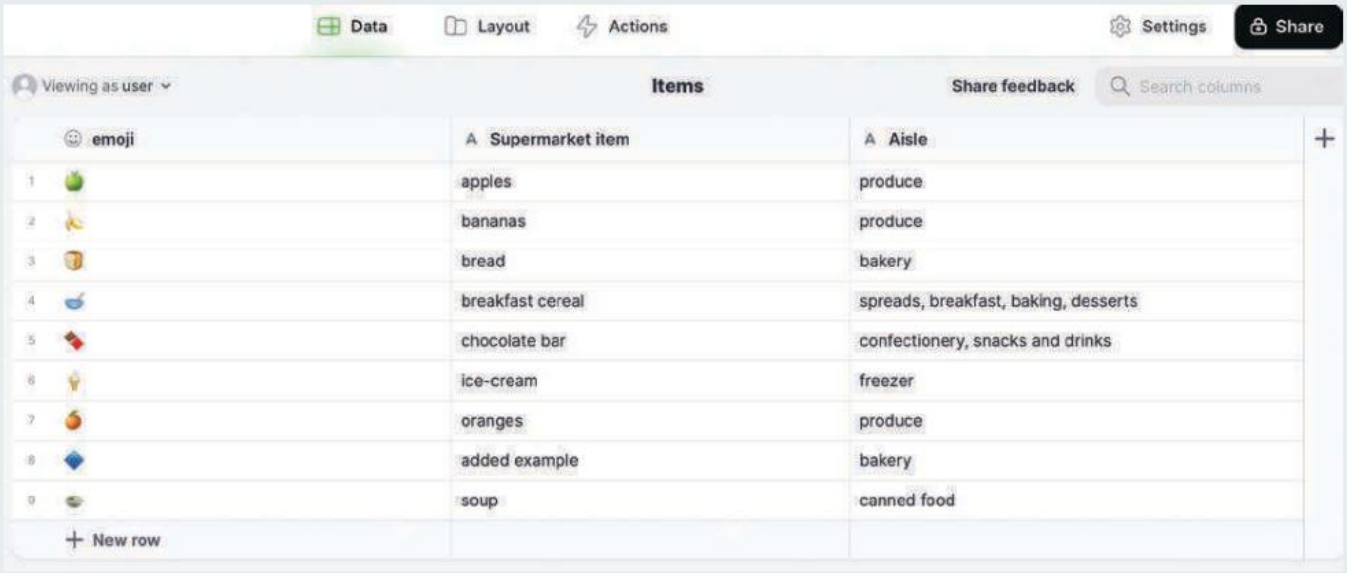


Figure 7.46 The data tab in Glide showing the emoji column dragged into first position.

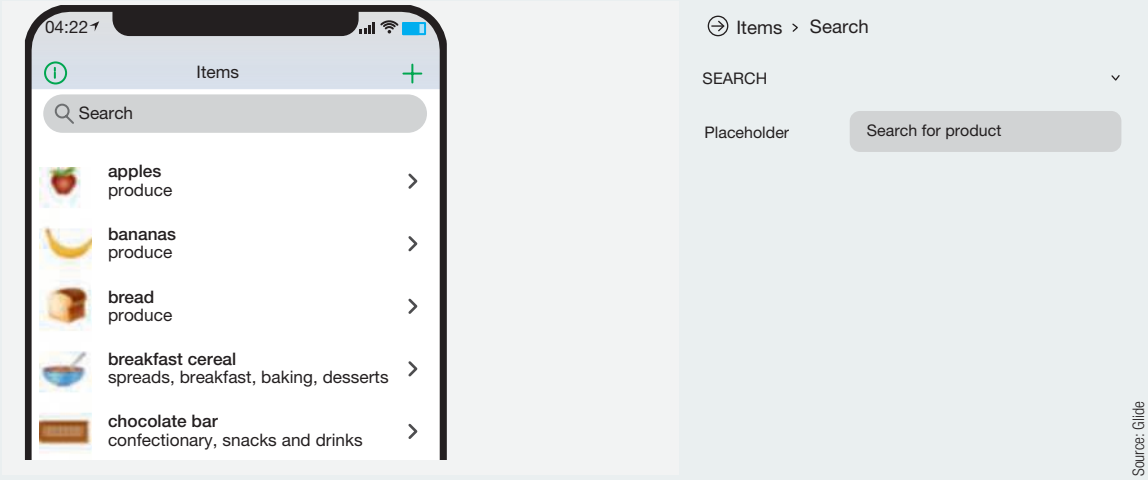


Figure 7.47 Spreadsheet data now seen using the app's interface on a mobile device.

Adding functionality

13 Test the search function by typing 'apples' in the app's search field at the top of the simulated phone screen.

14 A green '+' allows users to add items. If this is not already showing, turn it on using the right-hand panel. Select *Table* style then scroll down to Actions. Enable 'Allow users to add items' (see Figure 7.48).

15 Disable 'Allow users to edit items'. This will then enable the right pointing selection arrow for all items.

16 Add a new item (we called this *added example*, an aisle and a blue diamond emoji) as seen in Figure 7.48.

The new item will have automatically appeared in your Google spreadsheet as a new row without you even opening the spreadsheet!

17 Select the *Data* tab at the top of the Glide screen and check this, then check your actual Google spreadsheet.

Design thinking

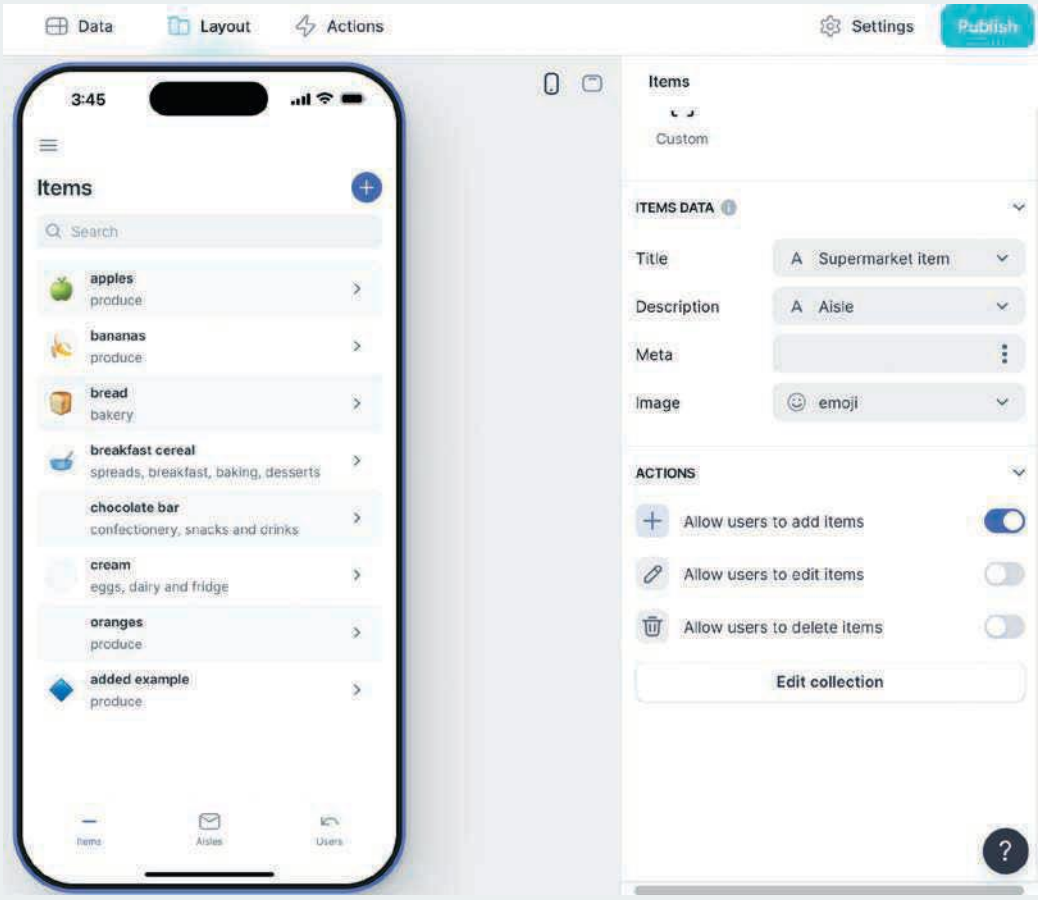


Figure 7.48 Users can add items to their shopping list using the + symbol on their phone screen.

Design thinking

18 Select the Settings icon in the top navigation panel to view the panel shown in Figure 7.49. Click the default icon and select an icon for your app (a shopping cart?) or upload your own, decide on a name and define a theme colour, accent colour and typography by selecting *Appearance*.

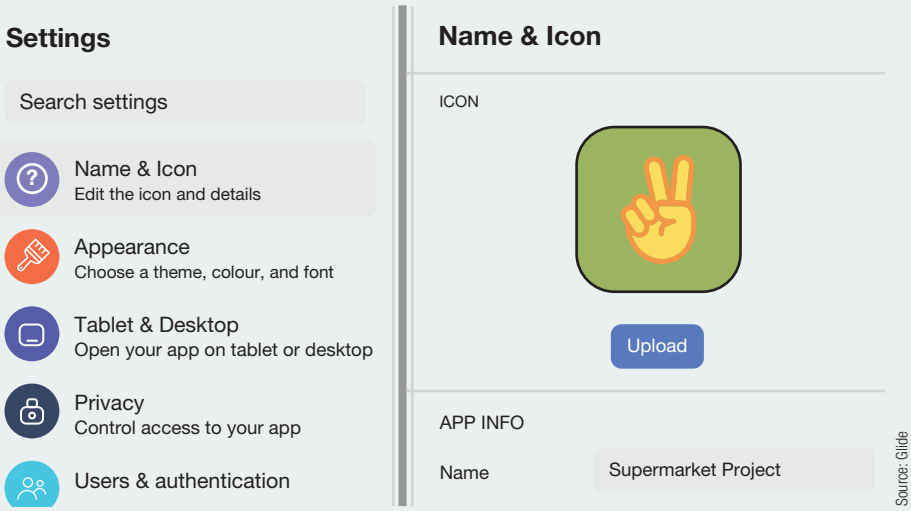


Figure 7.49 Create an icon for your app.

Test your app

19 Select Publish (top-right) and create your own URL (Figure 7.50)!

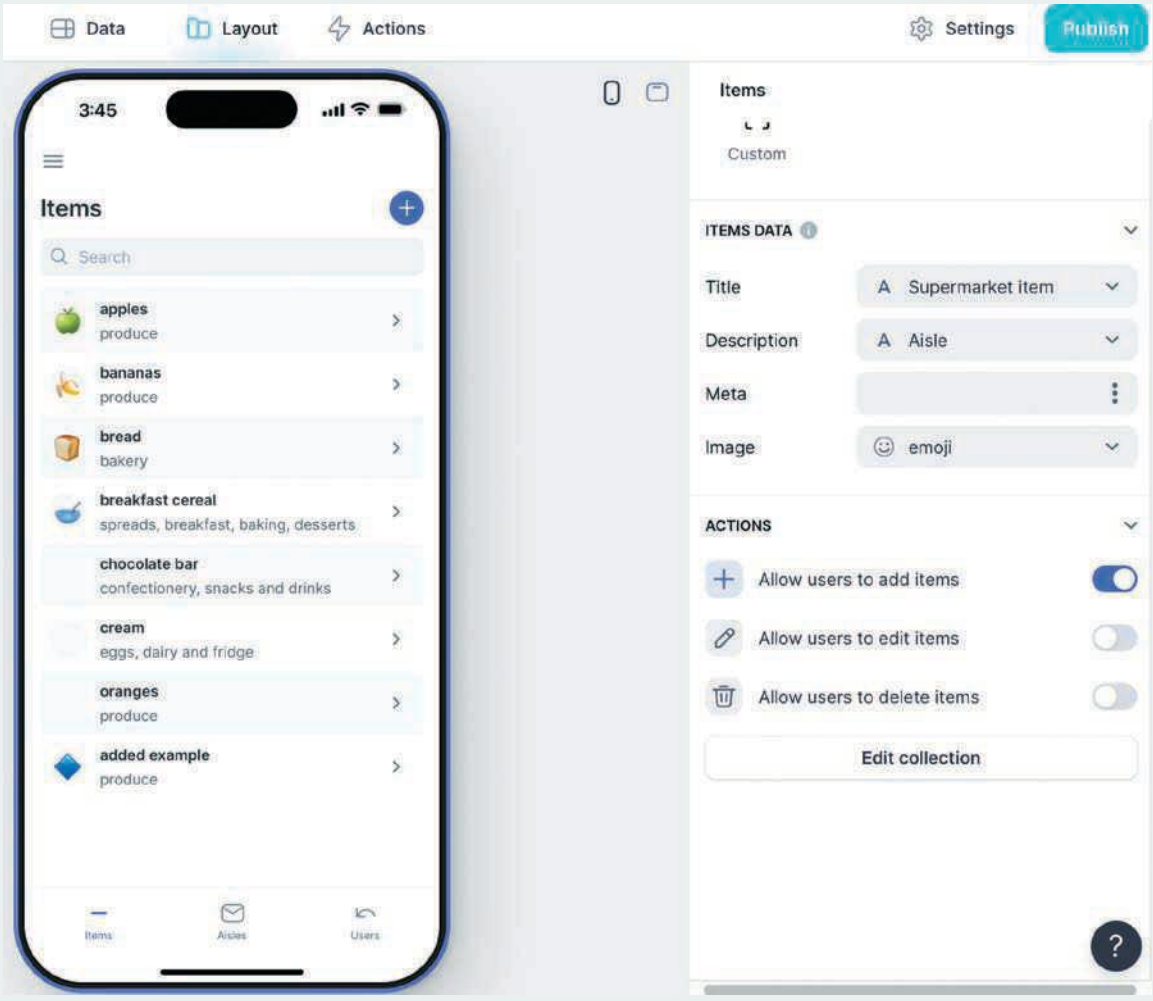


Figure 7.50 Turn off 'Allow users to edit items' and select *Publish*.

20 Scan the generated QR code using a mobile phone or tablet to open the app (Figure 7.51). Enter the email you used to register then add this app to your mobile device's home screen in the usual way for your device. It will appear using the icon you chose, open it and authorise it using the PIN sent to your email. You only need to do this the first time.

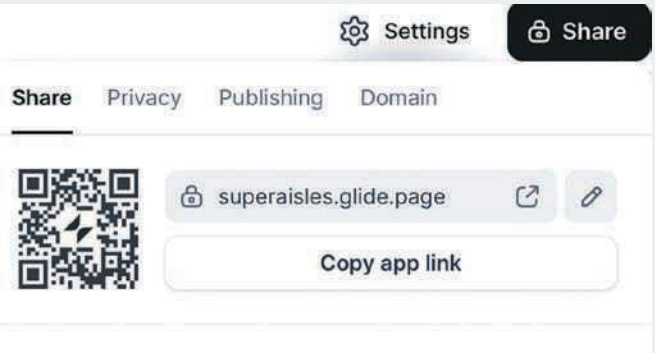


Figure 7.51 After pressing *Publish*, scan the QR code to install your app on your phone or tablet.

21 Test your app using your mobile device (phone or tablet) by:

- searching for an item
- adding an item.

22 Comment on your overall user experience using this interface by completing Table 7.6.

Table 7.6 Evaluating UX of Glide app

User experience *Items* mobile device app	
+	−

Extending functionality

You have created a well-designed app with a much-improved UX (with searching now a lot easier than using a spreadsheet!). It meets all but the last of our requirements from Step 1 in the first section.

It would be very useful to see a list of *all* items in any *chosen* aisle. This is important when shopping, as it is frustrating when a shopper needs to visit the same aisle many times.

To add this new aisle feature to your UI you need to add another sheet to your Google spreadsheet. This will add a new screen to your app in Glide and a new icon at the bottom of your app to display it.

23 Open your Google spreadsheet. Add a second sheet to this spreadsheet and name it *Aisles*.

- Add a heading, *Unique aisles*, to cell A1 on this new sheet.
- Open your Glide webpage and reload your Google spreadsheet by selecting the Data tab, then clicking the *Reload* circular arrow in the bottom left.
- In Glide, change to Layout view. In left panel under Navigation, press + then Screen from data and add Aisles.
- A new *Aisles* icon will now appear in your app at the bottom of its screen, alongside the icon representing the first Google sheet, *Items* (Figure 7.52).
- Note that you can change the default icons appearing at the bottom of your app for these using the right-hand panel under *Screen*. There is even a shopping trolley icon there!
- Hide the *Users* screen by clicking off the eyeball visibility icon beside it.

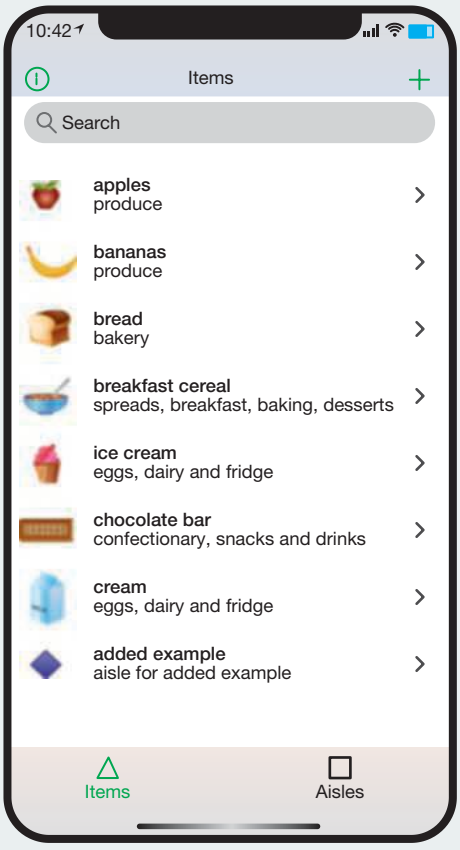


Figure 7.52 Updated app showing new icons, Items and Aisles, one for each of the Sheets in your Google spreadsheet.

Google Sheets has a very useful formula called UNIQUE, which looks through all entries in any given range and lists each once only. Its formula is `=UNIQUE(range)`.

Here is an example: Figure 7.53 shows this formula in action in a spreadsheet where 10 people have chosen a favourite colour. We want a simple list of these colours

Systems thinking

without repeats. Unlike most spreadsheet formulas, this one is placed in one cell only. Here we placed it in C2 with its range set to B2:B10. It is not visible as it is actively at work in the background. It can be seen in the formula bar at the top, however. It produces a list in that column of unique colours appearing in the range.

fx =UNIQUE (B2:B10)

	A	B	C
1	**Person**	**Favourite colour**	**Unique colours only**
2	Derek	blue	blue
3	Cassandra	salmon	salmon
4	Trinh	blue	red
5	Freya	red	orange
6	Lillith	orange	purple
7	Lorraine	red	
8	Aiko	purple	
9	Chika	salmon	
10	Phillip	blue	
11			

Figure 7.53 Example showing the use of the powerful `UNIQUE(range)` formula. Column C creates a list with each colour listed once only.

In Google Sheets, to refer to a cell range for the individual sheets of a spreadsheet you use the sheet name followed by an exclamation mark: `Items!`

In `=UNIQUE(range)`, our range will therefore be the B column of the first sheet named Items, containing the names of the locations of each shopping item. Place `=UNIQUE(Items!B2:B8)` in cell A2 of the second sheet, now named *Aisles*. Because we may wish to add more items beyond cell B8, we change this to `=UNIQUE(Items!B2:B)` Now no matter how many items you add, your formula will operate for the whole of column B. Your spreadsheet now appears as shown in Figure 7.54.

fx =UNIQUE (Items! B2:B)

	A
1	**Unique_aisles**
2	produce
3	bakery
4	spreads, breakfast, baking, desserts
5	eggs, dairy and fridge
6	confectionary, snacks and drinks
7	aisle for added example
8	

+ ≡ Items Aisles

Figure 7.54 Supermarket spreadsheet with a second sheet Aisles added and a column listing each aisle location once only.

We need to connect these unique names for each aisle in the *Aisle* sheet with the *Items* sheet, where we named the aisle in which each item can be found. This will create a one-to-many relationship between the unique aisles named in the *Aisles* sheet with their many occurrences in the *Items* sheet.

24 Select the *Data* tab in Glide. In the left panel you will see each of your sheets: *Items* and *Aisles*. Select *Aisles*.

25 Click the right hand '+' symbol to add an additional data column. Name the column *Aisle_items*.

26 From the Text drop-down field, select *Relation* for the type of data, which will populate this new column. Complete the relationship description in Configuration as shown in Figure 7.55 (*unique aisles* is matched to Aisle in the Items Sheet: *Items* > *Aisle*).

27 Make sure you check the box 'Match multiple'.

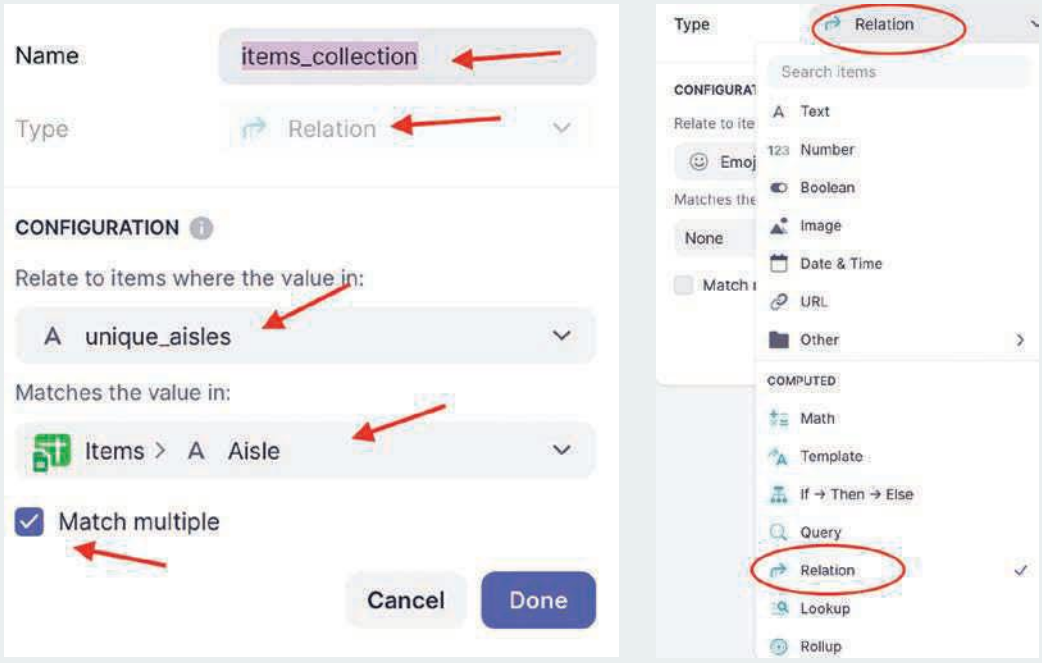


Figure 7.55 Adding an additional one-to-many relational column to the Aisles data in Glide.

28 Return to Glide's *Layout* tab and select *Aisles*. Make sure to turn off *Allow users to edit items* from the right-hand panel under *Actions*.

29 Click on the arrow next to the word *produce* in your app to travel to a new screen. It will be empty, as we haven't yet defined a component to hold the data.

30 In the left-hand panel under Components, add a *List* component. All the items listed as being in the Produce aisle should then magically appear!

31 Return to your app on your mobile device and test it thoroughly. Click on the *Aisles* tab, then on *produce* to see if *apples* and *bananas* are listed.

32 Return to the *Items* tab and add a new item, 'oranges', and enter an emoji of an orange and its aisle as *produce* (important, must be lowercase!). Select the *Aisles* tab and look at *produce* again. You should see the item *oranges* now added to *apples* and *bananas* (see Figure 7.56). Note that as we turned off the ability for users to edit items, if you mistakenly enter Produce with upper case 'P' you need to go to the Data tab in Glide and correct it there.

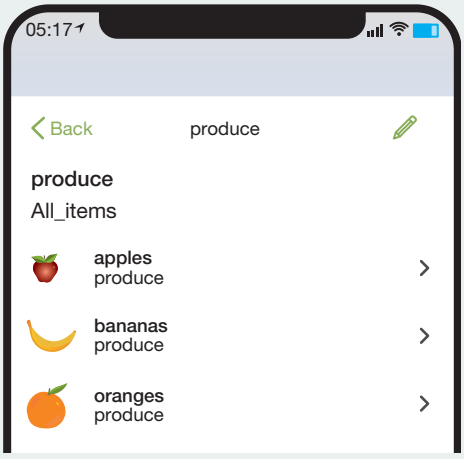


Figure 7.56 The inline list component collects all items belonging to a chosen category.

Testing and evaluating

The time has come to thoroughly test your app. Try it out yourself with a small shopping list and have other people use it. Record their suggestions and arrange these as either essential, desirable or possible. Update any documentation.

Taking it further

You have taken a basic spreadsheet and turned it into a useable app that can be used with any mobile device via an intuitive and well-designed interface. The UX is now many times better!

Glide has many more features for those who wish to explore them. There are many video tutorials that demonstrate these:

- Create a third sheet in your Google spreadsheet (a third tab in your Glide app) to allow users to add items from a shopping list (see Figure 7.57).
- Add the ability for users to delete items from the shopping list tab.
- Change the Aisles to drop down names instead of requiring users to accurately type the names.
- Under Components in Glide's left-hand panel, click the '+' icon and add the Kanban component to your app's *Items* screen to see what it does.
- Replace emojis in your app with actual images by storing the images in the cloud (such as Google Drive or Dropbox) and listing their URL in a spreadsheet column (see Figure 7.57).
- Add the ability for users to upload their own images into your app. The process is outlined on the Glide website.
- List other features that would improve the UX of this app.
- Advanced challenge! See if you can develop a formula for the Shopping list sheet of your spreadsheet that automatically attaches aisle and product emoji to items in the shopping list tab of your app. Hint: You will need the powerful `ARRAYFORMULA` function in Google Sheets. (Solutions to the above are provided on Nelson MindTap.)

	A	B	C
1	**Product**	**Aisle**	**Images**
2	apples	produce	https://www.dropbox.com/s/fq6toh21c375uc8/apples.jpg?dl=0
3	bananas	produce	https://www.dropbox.com/s/02pe97mk4dwzci5/bananas.jpg?dl=0
4	bread	bakery	https://www.dropbox.com/s/nf648wa4apx4pvo/bread.jpg?dl=0

Figure 7.57 Images can be stored on the cloud and linked via the spreadsheet.

7.11 Project 2: Designing, building and prototyping a mobile travel app

Project resources
Designing, building and prototyping a mobile travel app

Identifying and defining

Your task

In this project you are guided through the steps for creating an interactive prototype of a website for tourists in the NSW Blue Mountains for a mobile device using assets we provide.

You will build the application on a desktop computer and be able to view it on your mobile device, or via a simulated phone screen on your desktop computer.

Adobe offers a free tool called *Adobe XD* (XD) for designing and prototyping web sites and mobile apps. It allows you to create mock-ups and interactive prototypes which simulate the final UX for your project.

We will use this in the steps that follow.

Weblink
Figma

Another tool that can be used here is Figma, which has advanced features and is also free but only for a limited number of pages. Figma works in a very similar way to Adobe XD and this tool may be used if you are unable to access Adobe XD.

When you have completed this project, you will know this tool well and be able to use it to create a working prototype of your own from scratch.

Design thinking

Imagine you have been asked to design a website and mobile app for a tourist information centre in the Blue Mountains in NSW. After your site has been designed and coded you show it to your client, the manager of the tourist centre, who suggests lots of changes. Your coding and design will all need to be redone. You are frustrated because you have wasted time and resources. If you had known earlier, you could have fixed it before it was too late. You regret not spending more time with the users to understand their requirements better!

XD is great for showing the user your design plan and it allows users to suggest changes to your design before costly building and coding starts.

After learning to use Adobe XD, you will then design, prototype and build an app for a project of your own. Chapter 1 of this text provides an introduction to project management. Make sure you apply the important principles outlined in that chapter to your project, including addressing functional and non-functional requirements, considering social and ethical issues and documenting your project's development.

Researching and planning

Wireframe and mock-up

You learned in an earlier Skill builder that UI developers often create a sketch first, then a wireframe (showing the rough layout of elements), then a mock-up (where images and text are shown in position) and finally a working prototype which is close to the final product.

Follow the steps you have learned in this chapter to develop your design. Ask others for suggested features, test it with users and make improvements.

Document your desired features, your UX design, the problems you encounter and your solutions to them.

KNOWLEDGE BUILDER

Image compression

An important factor in designing a product with many images is download times. You should ensure the images are not overly large and are in a suitable format. If images need to be compressed then the choice of CODEC is important. There are two main compression techniques for images.

Lossy image compression

Lossy compression relies on reducing the number of colours for a particular image. JPEG (named after the Joint Photographic Experts Group) does this by relying on the characteristics of the human eye. The JPEG algorithm analyses and ranks pixel data on importance to human visual perception and throws away the less important information. The resolution of the image is reduced but we often do not notice it.

Lossless image compression

With lossless compression the decompressed file is restored exactly with no loss of data. A compression algorithm called run length encoding (RLE) looks for recurring patterns in the sequences of ones and zeros, and replaces these with shorter codes.

GIF and PNG formats both use this approach and so are lossless methods.

Figure 7.58 illustrates this method.

You need to balance image quality and image size. PNG format is a popular choice for images as it can still be compressed but is lossless in quality.

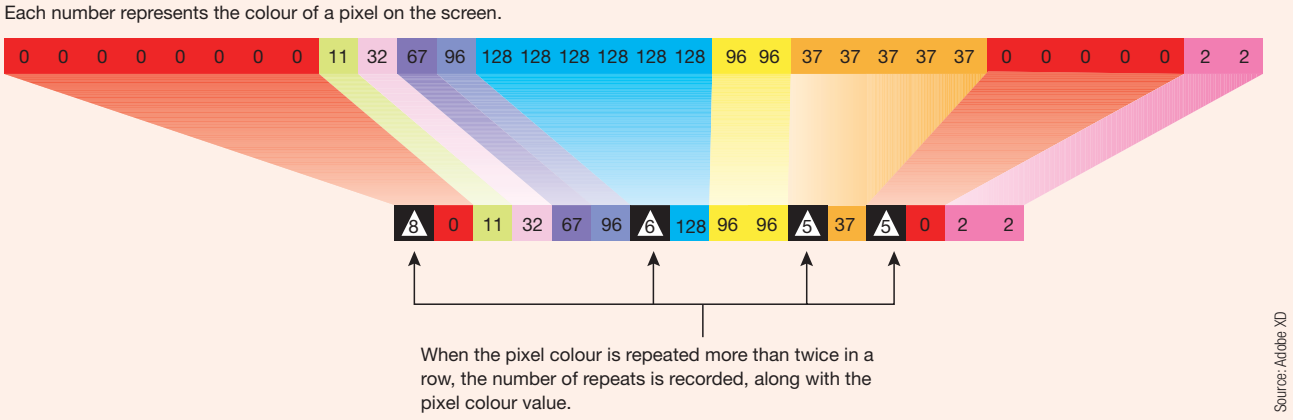


Figure 7.58 Lossless compression in a GIF image file explained.

Producing and implementing

Creating a landing screen

Let's get started!

1 We suggest you first complete the excellent introductory tutorial via the weblink here. Open Adobe XD. The welcome screen provides a set of standard-sized screen templates (Figure 7.59). Select a suitable phone one.

2 Adobe XD has two modes for viewing work: Design and Prototype. Design mode is where you create your layout and import resources (logos, images etc.). Prototype mode is where you create navigation links and test user interaction.
   - Select a suitable phone screen size to create a new project for this screen size (available as a dropdown choice) (Figure 7.60). XD will open in Design mode.

Weblink
Get started with Abode XD

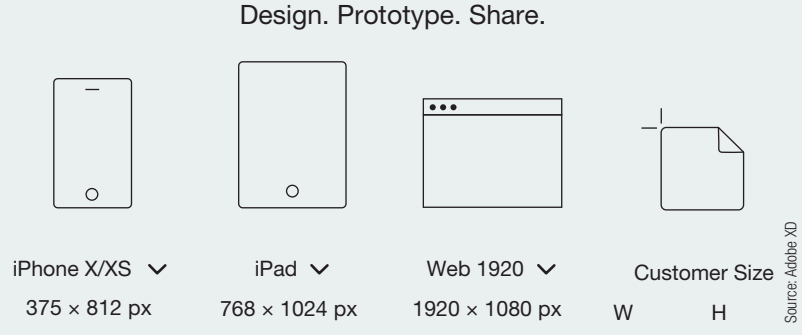


Figure 7.59 Adobe XD's welcome screen showing choices for device dimensions and an introductory tutorial.

3 Familiarise yourself with XD's tools (Figure 7.60) by hovering your mouse over each. You create new screens using the artboard tool (second from the bottom).
Build a home page or landing screen on the first artboard already created. Use the following along with Figure 7.60 to guide you.

- Click on the artboard and notice its properties appear to the right. Click on *Fill colour* under *Appearance* and select a green background.

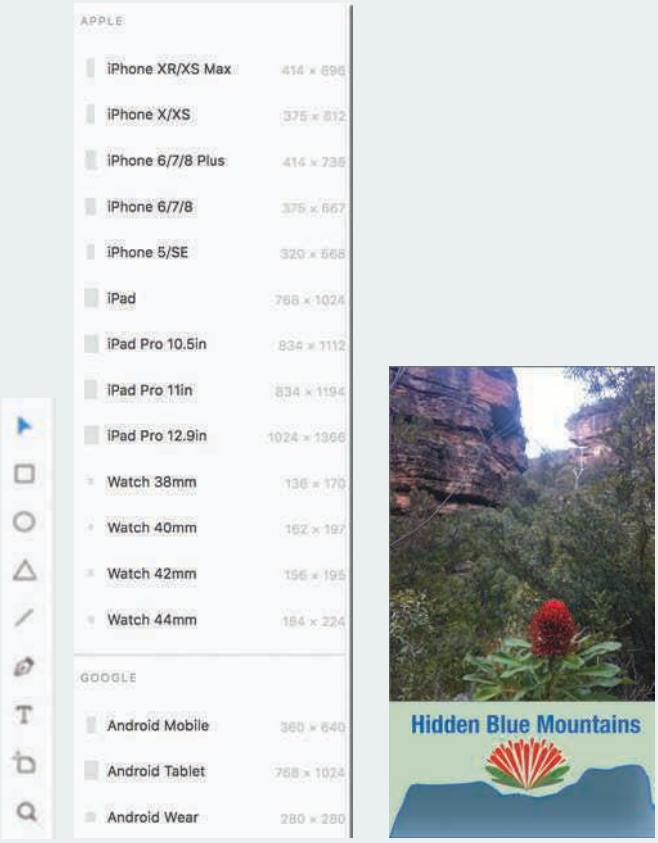


Figure 7.60 L to R: XD tool palette showing the tool palette, screen templates and final home screen.

- Drag *waratah.png* from your online Assets folder for this Skill builder and resize and position this image on your artboard.
- Click on the artboard and rename it *Home* by double clicking its name.
- Add the words 'Hidden Blue Mountains' and change the size and change the text colour to blue. Reposition using the pointer tool.
- In the project resources available via Nelson MindTap, locate the image file *0_landing_page.jpg* as seen in Figure 7.60 and drag it above your text.

Design thinking

INFOBIT

'Responsive design' refers to the ability of the layout of elements in a webpage design to automatically adjust to different screen dimensions. In the past designers produced a separate design for each screen size.

4 Once you have accurately positioned items, using responsive design, XD automatically positions selected objects by analysing their grouping.

- Turn on responsive design for an artboard using the 'slide switch' in the *Properties* panel then try resizing it. Change it back when finished.

5 XD has the ability to turn objects into *components*. These have the useful feature that any changes in the original are reflected in all copies of it. This can save designers lots of effort and time.

- Turn the waratah into a component by selecting it and right clicking then selecting *Make Component*.

6 We now add a mountain design as seen in Figure 7.61. XD allows simple designs to be created in the app itself.

- Select the pen tool (Figure 7.60) and in an empty space, draw the mountain shape shown in Figure 7.61. If you haven't used a pen tool before (such as when using Adobe Illustrator), you *click* to create points and *click and drag* to form curves. You must close the shape by ending on your start point. Adjust stroke, fill and transparency.

Creating scrolling images

We now create a second screen for thumbnail images of the locations. This will be where users choose the tourist attraction that interests them. We use a powerful feature XD calls a *repeat grid*.

7 Click the artboard tool (at left side) to add a page. Reposition by using its title to move it.

8 Click the top title of this screen and drag the bottom edge down to extend it, stretching it longer than the Home screen. A dotted line appears at the bottom edge of the visible screen area and the area below is the scrollable

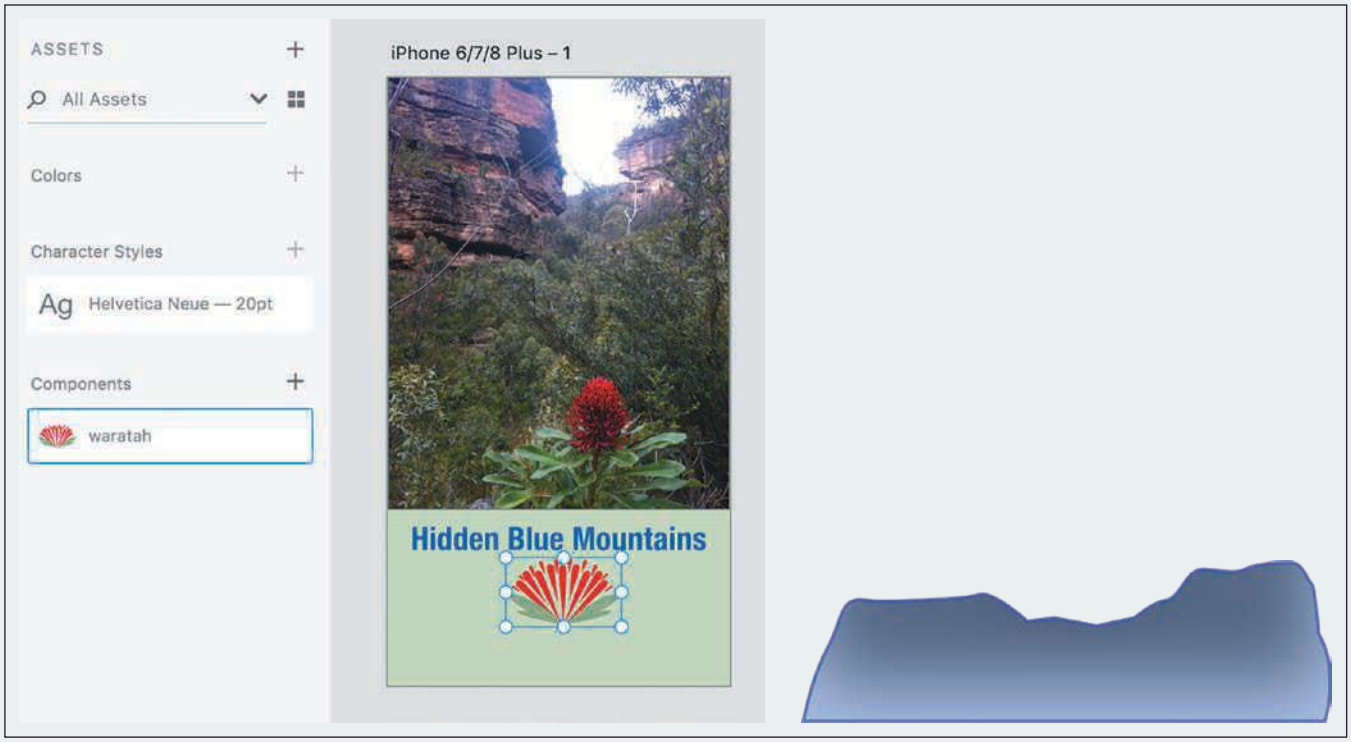


Figure 7.61 Adobe XD allows us to draw simple shapes and turn them into reuseable components.

hidden area. Dragging this = sign at the left changes the scrollable area seen by the user. Leave it at its position now (Figure 7.62).

9 Using the tools palette, draw a small rectangle top left of the artboard and below it create a textbox. Enter a line of text below it (you can change the text later). Select *both* rectangle and text box and switch on *Repeat Grid* in the Properties panel. Green handles appear right and bottom. You have created a repeat grid (Figure 7.62).
Drag these handles down and to the right and watch what happens! Create 12 placeholders for images.

10 By hovering over the gap between elements, a pink column indicator is displayed (see Figure 7.62). Drag these pink gutters to adjust spacing between cells and the size of the Repeat Grid until you have the right number of elements to fit your artboard.

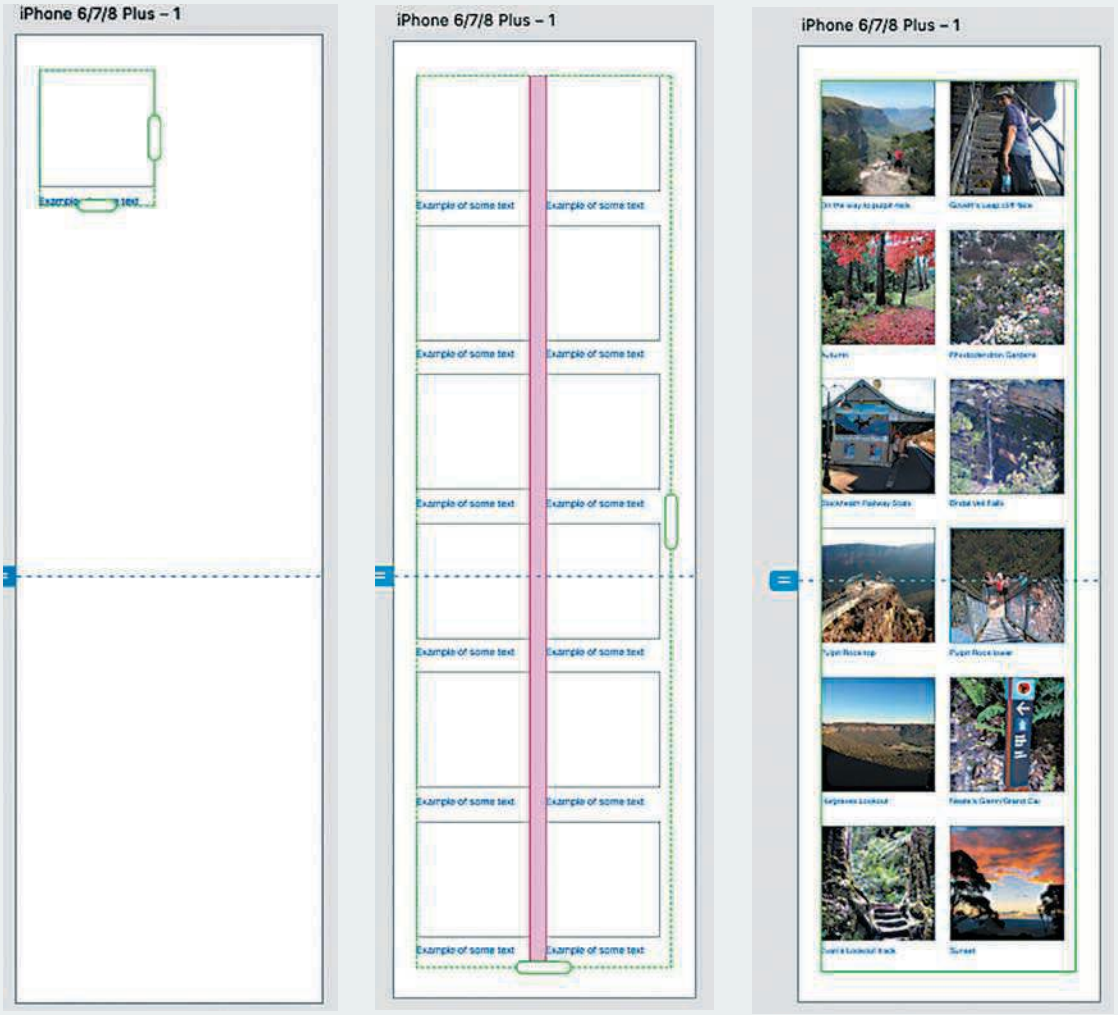
11 Drag all 12 images provided on Nelson MindTap numbered 1 to 12 into the *first* placeholder (even if it has an image already). All rectangles fill with different images (Figure 7.62c).

12 You can also populate text boxes automatically if you have a text file using separate lines for each title. We have created one for you. Try dragging *thumbnails.txt* (provided) onto your first textbox. Resize font using the right properties panel to 10pt.

13 Select the thumbnail images artboard and click the *Play* button top right. Test out scrolling. If there are black edges on the sides, adjust your design to eliminate them.

14 Try changing the shape of any rectangle. Click Command/Ctrl while selecting a rectangle to make it editable and then drag one of the four small circles *inside* the rectangle to round the edges or make circles.

15 Staying on this second screen, use the eyedropper colour picker tool next to *Fill* to copy the exact green fill by clicking on the green of the Home screen to change the background and then add our app's waratah logo at the top. Components can be accessed using the Libraries icon bottom left corner. See Figure 7.63.



(a) (b) (c)

Figure 7.62 (a) Selecting the artboard and dragging down created a scrolling area. Here we have added a shape and a textbox and created a Repeat Grid (see green handles). **(b)** Dragging the handles creates 12 repeating placeholders. **(c)** After images and text are added.

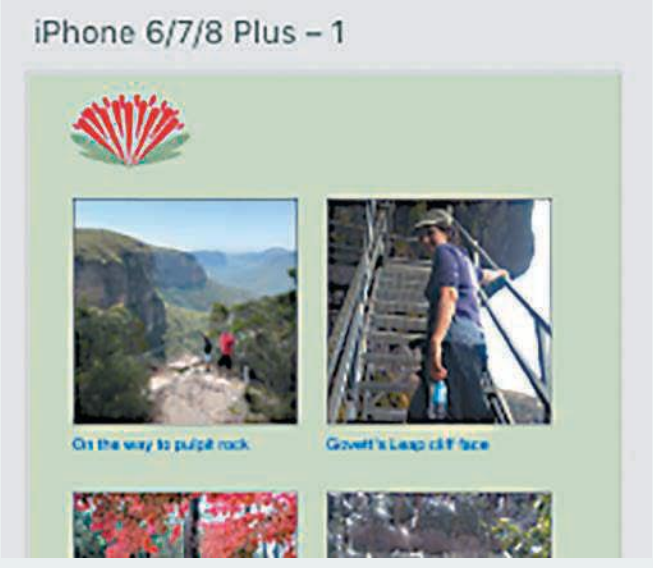


Figure 7.63 The waratah will be used to return to the Home screen.

Adding navigation

You have now designed two screens: a landing screen and a scrolling screen of thumbnails. You are ready to link them to create your first prototype – a working model of the final product.

16 Change XD into Prototype mode (top of the XD app window). Select your first artboard. A small blue arrow appears. Drag this out to the second screen to create a link. ***Note:*** you can also select the *Type* of transition. Select the waratah and drag its small blue arrow back to your home screen. See Figure 7.64.

You can remove any link by dragging its end to any open area of the XD app.

17 Click the *Play* button (top right) to view your working prototype. Imagine this as your phone's screen – clicking on items allows you to navigate.

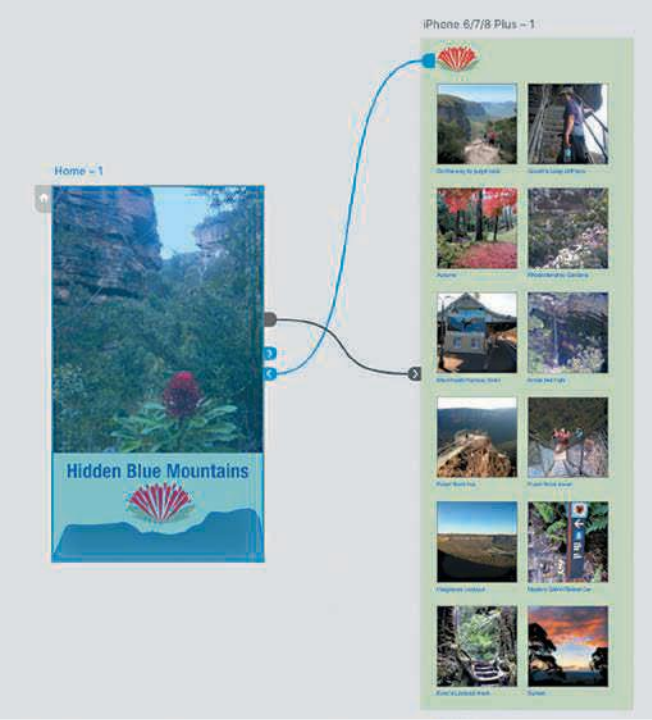


Figure 7.64 Two linked screens. Tapping the first moves to thumbnails, tapping the waratah returns to Home screen. Interaction is tested using Prototype mode and pressing play button.

Explore other great features: select different Triggers (even voice commands) and choose different Actions (other than transitions) – such as text-to-speech.

How could this UI feature improve the UX for those needing accessibility features? (See panel.)

INFOBIT

'Accessibility' refers to the design of products, devices, services, or environments for people with disabilities. *Equity* and *access* refer to the principle that all people should have equal rights and access to the benefits that software technologies offer.

Often people with disabilities experience difficulties which could be eliminated simply by changes in design.

What are software design features that can assist those with disabilities?

How could your prototype design in this Skill builder address the issue of accessibility?

Creating the remaining screens

You will now link separate screens to their matching thumbnails.

18 Return to *Design* mode. Click the artboard tool to add a new screen. Reposition using its name to select it.

19 Drag the waratah logo from Components and position it on this artboard (Figure 7.65). Use the eyedropper tool to choose the same green as previously for the background colour. Add a white rectangle as a placeholder for the image.

Figure 7.65 Designing the 12 location screens.

20 Make a copy of this artboard by dragging its name (blue writing at top) to the right while holding down alt/option key. Use this method to create 12 screens – one for each thumbnail image. Arrange these conveniently. You can adjust by using the zoom at the top of the XD window.

21 Locate the 12 images in your Assets folder and drag them onto the separate screens. See Figure 7.66.

22 Locate the file *images_text.txt* in Assets which provides text for each screen and add these to appropriate screens as shown in Figure 7.66. Turn on *multiline auto height* text in *Properties*. Apply bold to the headings.

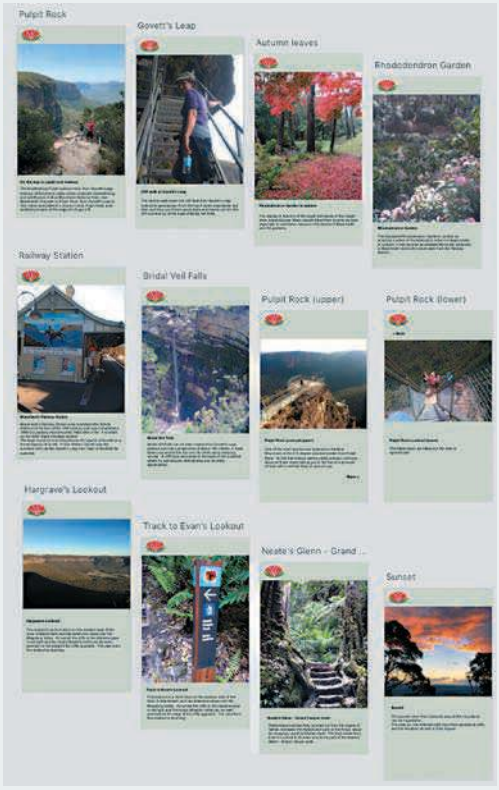


Figure 7.66 All 12 screens with text added.

23 Appropriately rename each of your 12 screens.

Completing all links

24 Change to Prototype mode. Right-click on the thumbnails and Ungroup the grid.

Note that a linking choice exists in Properties under Action > Type of returning to the 'Previous Artboard'. You can use this for the reverse link from these 12 new screens instead of the waratah link if you wish as it is good to avoid a lot of link lines in your prototype.

Create all 12 links from each thumbnail image to its details screen then back again (Figure 7.67). You can use *Animation* under Actions in the Properties panel to set the type of transition, such as a sliding one.

Add a link via the word 'more…' from the Pulpit Rock image to the further text in Pulpit Rock (lower) then back again and make it a slow sliding transition.

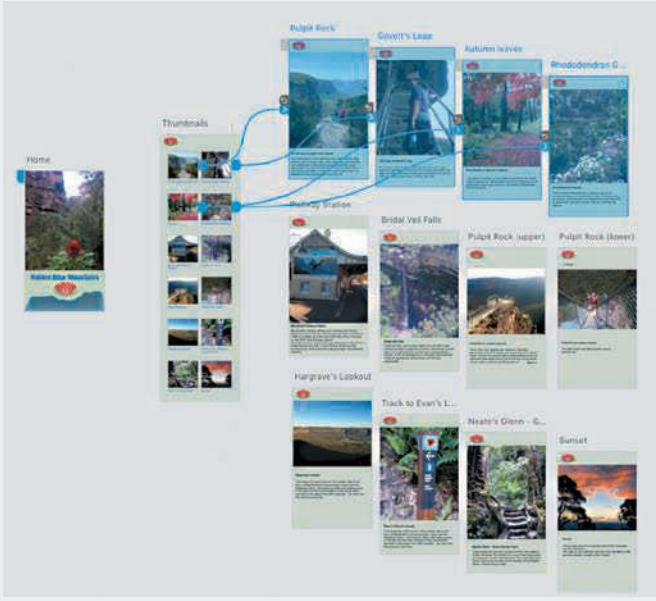


Figure 7.67 Final prototype, showing links from thumbnails to the first four images. Note the 'return' link (grey arrow) on each.

25 Complete these navigational checks:

- tapping a thumbnail should link to the correct details
- tapping a waratah should always link to the Home screen
- tapping any of the 12 image screens should always return to the 'previous screen' (which will be the thumbnails screen)
- the Pulpit rock (upper) screen should provide a word (More >). Link it by a sliding transition to the Pulpit Rock (lower) screen and allow it to return.

Testing and evaluating

Final testing and phone simulation

26 Make sure you are in Prototype mode, press the *play* button and *Preview* your navigation. Correct any errors.

27 XD allows you to simulate your project on an actual mobile device! Download the XD app to your mobile device. Connect your device via USB cable and click the phone icon at the top of the desktop app window. Thoroughly test your app.

28 The Share button top right allows class member(s) to edit your project. This would be very helpful in a company environment.

This app covers one Blue Mountains town only. How would your design change to add other towns?

Taking it further

Now you have mastered this design and prototype tool, create a project of your own.

- Develop a sketch, wireframe for a concept for an app of your own and develop a working prototype.

Ask people to add suggestions for improvements:

- Allow a range of users to try out the app, ask them to comment and to rate it for design, navigation, ease of use and information, and to suggest new features. Choose different age groups, genders and backgrounds.
- Collect all the suggestions and list them in a table. Rank each suggestion by ticking in three columns to indicate urgency: immediately, soon and later.

Going further:

- All software undergoes further development and features are added continually. XD is an example of this. You are now an experienced user! What additional features would you suggest that Adobe could add to this prototyping tool?
- Try creating the same prototype using Figma (see start of project) and compare this tool to XD.
- XD can be used to create cool short animations and record them as videos. See the weblink outlining the technique.

Weblink
Designing Adobe XD

8 Designing for user experience in AR and VR

FOCUS AREA: Designing for user experience

Chapter outcomes

In this chapter, you will learn:

- about the extended reality continuum
- about AR, VR, triggers and markers
- about the types and uses of AR and VR apps
- to use AR and VR apps
- to design and build your own AR app
- to design and build your own VR app
- about the evolution of VR headsets
- to use Unity to design a VR user experience.

Key terms

augmented reality (AR)
extended reality (XR)
haptic
mixed reality (MR)
overlay
trigger
virtual reality (VR)

IN ACTION

A UX for all senses

Alamy Stock Photo/PictureLux/The Hollywood Archive

Figure 8.1 Disney's CGI *The Lion King* can be a multi-sensory experience in a 4DX cinema.

The ultimate movie UX?

Is watching a movie in 4DX the ultimate movie UX (user experience)? Movies offered in 4DX cinemas feature not only 3D but also moving seats and special effects like wind and rain, smells, fog and lightning.

Attention to detail

Disney's newest version of *The Lion King* (Figure 8.1) already includes hyper-realistic 3D CGI (computer-generated imagery) – every hair on Mufasa's mane is individually animated. Dust hangs in the air. You may know you are not watching real life, but the movie looks so real you can believe you are.

4DX adds to this realism by allowing you to experience the movie using all your senses: sight (3D imagery, fog and lightning), smell (of the savannah and the jungle), sound (Dolby surround sound), touch (seat vibration, tickling, puffs of breeze, wind, rain, mist) and temperature (hot and cold).

Sensory effects

The expansive panning of the camera across Pride Rock is matched by a gentle pitching of the seats and you are immersed in the fight scenes as your seat jolts to match the movements of the animals. A thunderstorm is enhanced by seat vibration, flashing strobe effects, wind blowing and rain falling in the theatre. Shots of air on your face capture sudden directional changes in galloping scenes. The smells of the elephant graveyard fill the air. The seats bounce gently to match the beat of the *Hakuna Matata* song and spray water and fresh smells when the animals meet the waterfall.

Daniel Yi is the art director at CJ 4DPLEX, the South Korean company responsible for 4DX theatre experience. Yi says he is focused on one thing every day: How would a person experience that in real life?

Production

For any 4DX-destined movie, the first thing the team does is watch the whole film once to get a general feel for it. Then they discuss what sort of movie this is – is it highly emotive or is it mostly action? What are the characters like? How can 4DX best serve this type of film?

They then break the movie down, section by section, and add the various 4DX elements (Figure 8.2). To avoid roller coaster-like gimmicks, Yi's team needs to make sure the final product is high-impact but not too extreme. For instance, when there's a busy action sequence, the team needs to pick and choose what to focus on. 'You can't put in chair vibration and a shot of air and fog and everything else that's going on', said Yi. 'It becomes too much for our bodies to understand.'

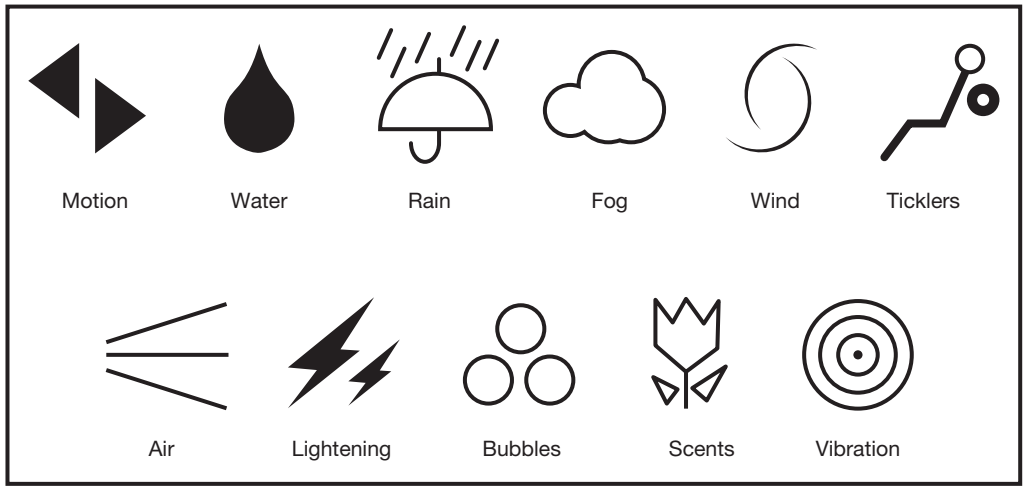
'It's our job to figure out how to highlight what the director intends for audiences to experience', he said. That means selecting what effect will accentuate that part of the story best.

The future of 4DX hangs on the assumption that movie-goers want to actually experience movies, not simply watch them. It's an expensive assumption, with each auditorium costing $1.2 million to $1.4 million to build.

'We are giving audiences something they can't get at home – a way to watch that's a visceral experience', he said.

Source: Sarah Toy, 'Punches, kicks and gunpowder: What goes into making a 4DX movie', MarketWatch, 14 October 2018.

Alamy Stock Photo/Belga News Agency



Source: 4DX

Figure 8.2 The sensory effects advertised in 4DX theatres.

Weblink
Introducing 4DX

Questions

Watch the video *Introducing 4DX* by 4DX on YouTube and answer the following questions:

1. It has been said that the key to the success of a great movie is a great story. How much do special effects contribute to the UX? Can they detract from that success?
2. Can you suggest any other effects that might be possible in a theatre?
3. Do you believe visits to theatres might one day be replaced by other forms of movie experiences?
4. If you have seen a movie in 4DX, write a short review of your experience.
5. Suggest the kinds of expert skills that might be required to produce an immersive 4DX experience.

8.1 Extended reality user experiences

In the previous chapter you discovered what makes an effective user experience (UX) and the principles of effective user interface (UI) design. You have built interfaces which engage users, learned about the evolution of modern user interfaces and created a responsive website in the form of a mobile app which displayed data interactively.

In this chapter you will extend your knowledge of UX design by exploring **extended reality (XR)** user experiences including **augmented reality (AR)**, **mixed reality (MR)** and **virtual reality (VR)**, learning about UX careers and designing your own virtual 3D world using Unity 3D.

The extended reality continuum

AR and VR are ends of the continuum between the real world and the virtual world as shown in Figure 8.3.

When a gamer participates in an entirely artificial world created on a computer, they are part of a VR environment. But this is just one end of a spectrum known as XR. At the other end is the real world in which we all live. And somewhere in the middle is AR, which combines the real world with a virtual world.

We take a look here at both AR and VR as they generate unique user experiences and provide information that would otherwise not be available.

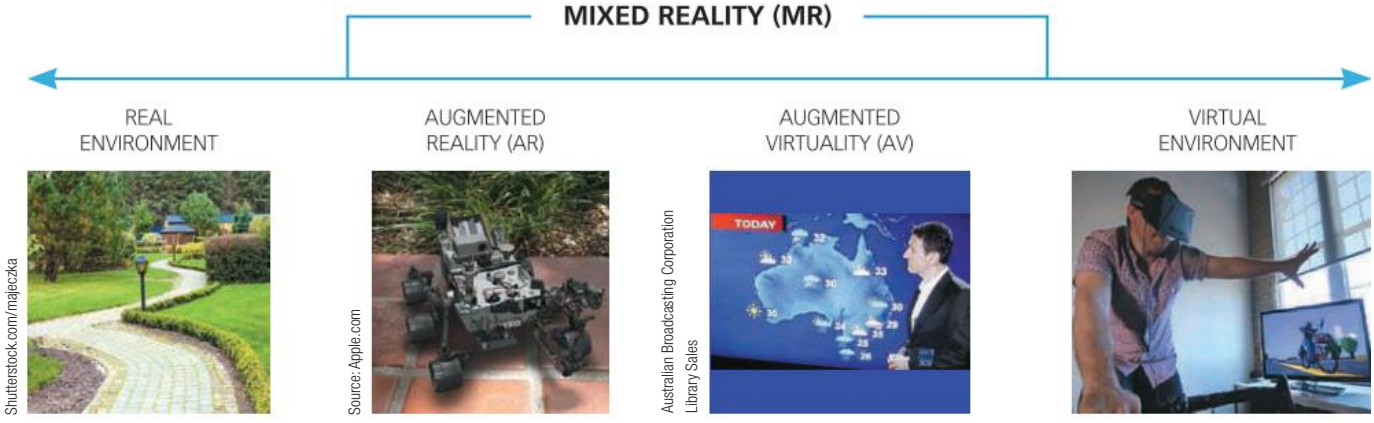


Figure 8.3 Milgram's reality (virtuality) continuum: (from left to right) The real world; augmented reality – here the spacecraft is seen on the mobile device as if really there; augmented virtuality – the weatherman is superimposed over a virtual map; virtual reality – the user participates in a virtual world as an avatar. The wide centre band is mixed reality (MR) combining elements of both AR and/or VR in ways which involve user interaction.

8.2 Augmented reality (AR)

Augmented reality (AR) is a real-time view of a real-world environment overlaid by computer-generated media such as text, audio, animation, video, 2D or 3D graphics.

AR integrates real-world data with computer-generated images to deliver a convincing interactive 3D environment. Users hold a printed page, containing an image, up to the camera in a device or select a flat surface in its live image and software superimposes CGI (computer generated imagery) in a layer over it. Advertisers have been aware of its potential and applications using it have been developed for mobile phones. An example from IKEA is shown in Figure 8.13.

Weblink
Meta Spark

Triggers

There are three main types of **'triggers'** used in AR.

Marker and image triggers

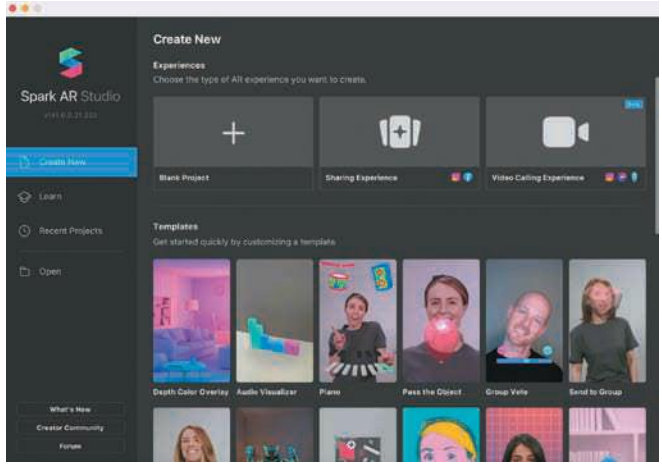
To display an educational animation over a relevant page of a book, it is important that the user is looking at that page. This is especially important when the AR is to be layered over an existing printed scene. The AR software must recognise the image at which the user is pointing the device camera. This is achieved using a special marker, either a picture or a shape printed on the page.

In Figure 8.4 a student has produced a picture book of origami figures containing instructions on how to fold them. To show the final folded figure, she created them using the 3D design application SketchUp and used markers to trigger the 3D model.

Figure 8.4 Augmented reality: a 3D origami model designed using Sketchup appears over a printed marker in a student's senior school multimedia project.

When markers are used, the 3D image 'anchors' onto the marker and moves when the marker is moved. The user can move a physical book and see the augmented image 'stick' to the marker on the page.

Instead of specially designed markers, image recognition (live or static) can also be used to trigger AR overlays.



Source: Spark AR Studio

Figure 8.5 Meta's *Spark AR* app specialises in a wide range of face tracking AR effects.

Meta's Spark AR application was especially designed for faces and provides many tools for lighting, particle and object overlays (Figure 8.5).

Check out *Spark AR Studio*.

Planar surface triggers

Imagine an AR application that can place virtual furniture in your house. You can try different combinations of items but you need to tell the AR application where to place the virtual object. This is a markerless surface trigger. The software recognises a flat uncluttered area and places the object. See the example in Figure 8.13.

9780170442787

This is used in some AR games where it is not important where the character actually stands, but it is important that they lock in that position and not randomly jump around. The gameplay is the important interaction.

In this type of AR virtual objects appear to 'float' but may also be automatically placed onto a flat surface, along with generated shadows to increase realism (for instance, placing a lamp onto a table). See Figure 8.8 and try out this excellent example in the BBC's *CivilisationsAR* app by downloading it to your phone or tablet.

Geolocation triggers

The GPS coordinates recognised by mobile devices can also be used to trigger an AR experience. This feature was the basis for the *Pokémon Go* craze, one of the most profitable apps in 2016 with over 1 billion downloads.

The app uses mobile GPS to locate, capture, battle, and train virtual creatures, called Pokémon, which appear as if they are in the player's real-world location.

Pokémon Go popularised this third type of AR trigger (Figure 8.6).

The game promoted physical activity and social bonding as well as helping many small businesses grow as players swarmed to particular locations to capture virtual 'objects'. The gameplay had unintended results as well. One man's physical house was designated as a 'gym' and Pokémon players showed up through the night.

The game also attracted controversy for contributing to accidents and creating public nuisances and various governments expressed concerns about security.

Overlays

The output from AR is in the form of **overlays**, also known as *expressions*, which are triggered by the types of triggers described in the last section. The basic forms of output are visual, auditory, **haptic** (the sense of touch) and hyperlinks.

AR tools have become very sophisticated and even include physics effects for objects. Such effects recognise collisions and gravity, for example.

Apple's *Reality Composer* lets developers prototype and produce AR experiences with no 3D experience providing a drag-and-drop interface and a library of high-quality 3D objects and animations.

Visual expressions

Text, animation, video and images (2D or 3D) are overlaid on the natural view in the form of layers.

Groundbreaking advances in AR technology have enabled AR to be far more immersive than previously, with overlays integrated into scenes using hyperrealism, lighting effects acquired from the scene as well as occlusion (objects can move behind other objects).

You will witness these tools at work with the apps featured in this chapter.

Auditory and haptic expressions

Headphones or speakers are used to deliver audio information to the user.

Spatial audio effects are possible to allow sound to emerge from different directions on moving the head.

Haptic feedback involves vibration which can be used as a feedback mechanism and is most common in gaming applications.

Hyperlinks as expressions

Many AR apps provide the triggering of hyperlinks to websites, messaging, social media posting, email sending or trigger a phone call.

Shutterstock.com/Rasdi Abdul Rahman

Shutterstock.com/KeongDaGreat

Figure 8.6 *Pokémon Go*, with over 1 billion downloads, uses geolocation as the augmentation trigger where the goal is for players to collect objects. Travel companies even offer *Pokémon Go*-inspired trips.

SKILL BUILDER

Exploring AR

AR is best understood by experiencing it. There are hundreds of AR apps available. We suggest downloading the following examples to your phone or tablet and trying them out.

- *CivilisationsAR* from the BBC
- *Spacecraft 3D* from NASA
- *Skyview* and *Transparent earth*
- *Monster Park*
- *PaintSpaceAR, Quiver* and *ARSoccer* (just for fun).

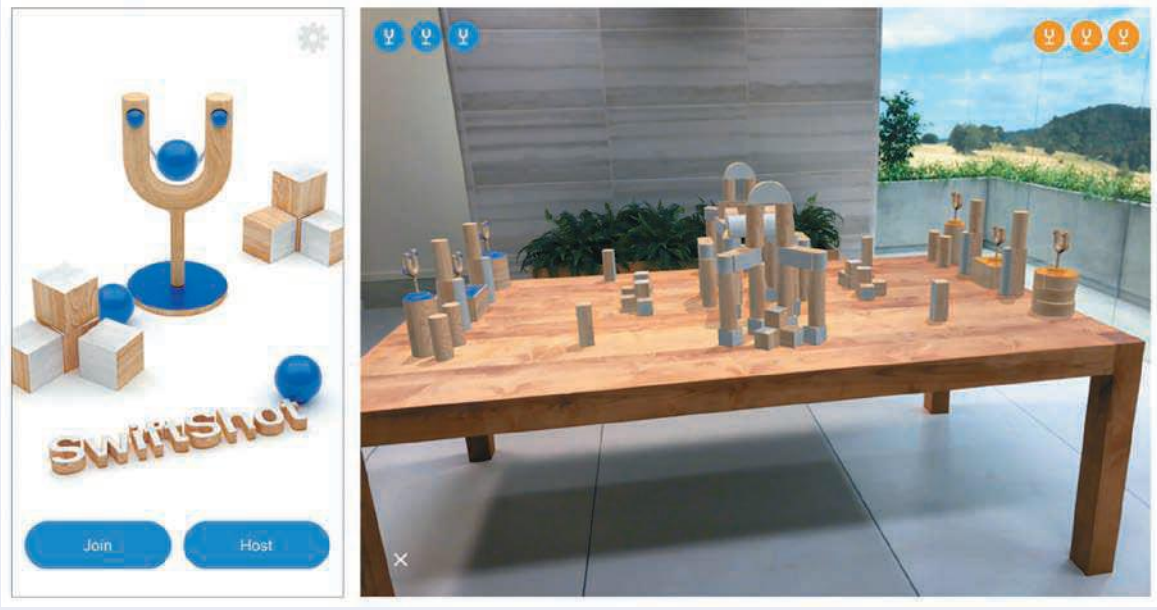
AR has been around for many years but has been very slow to be widely adopted. Both Apple and Android have recently made AR kits available, providing developers with tools to create very high-quality AR experiences.

Questions

1. Compare the UX of three AR applications you tried.
2. Compare the technical features of three AR applications you tried.
3. Select one of these apps and suggest improvements for its UI and UX.
4. Why do you think AR adoption has been so slow?

CASE STUDY

SwiftShot



Source: Apple.com

Figure 8.7 *SwiftShot* as seen from outside and inside this AR game. See it in action using the weblink below.

The game *SwiftShot*, in which players use virtual slingshots to demolish a city built of blocks (Figure 8.7), was developed to demonstrate the capabilities of Apple's *ARKit*.

When the developers were asked what they had learned they said that the AR experience does not need to be a long one and play is better slow rather than fast. They discovered there can be a social element to the gameplay – watching others play was as much fun as playing it yourself!

They said too, that they realised how important it was to give both audio and haptic feedback in their game and that the device acted like a camera lens, through which the user sees the world.

Players needed a network connection and a shared coordinated space in which the same object appears in the same place on both devices.

They said they discovered one thing above all – that the central concept of the gameplay is the thing and that you can't just sprinkle fun on top at the end.

Take a look at a game of *SwiftShot.*

Weblink
Apple SwiftShot Demo: First multi-player game created with ARKit 2

A selection of AR apps

Historical apps

The BBC's *CivilisationsAR* app (Figure 8.8) puts history into the user's hands. The virtual collection of over 30 AR items include an Egyptian mummy, masterpieces by famous artists and historic archaeological items.

Its interactive UI allows users to 'rub' through layers of history to revive a faded sculpture to what it would have looked like when it was first made, a magic spotlight reveals special features, there is an audio guide, an X-ray feature and a tool to translate hieroglyphics. A world globe can be navigated and objects shared with friends. Londinium (Figure 8.9) was another historical app.

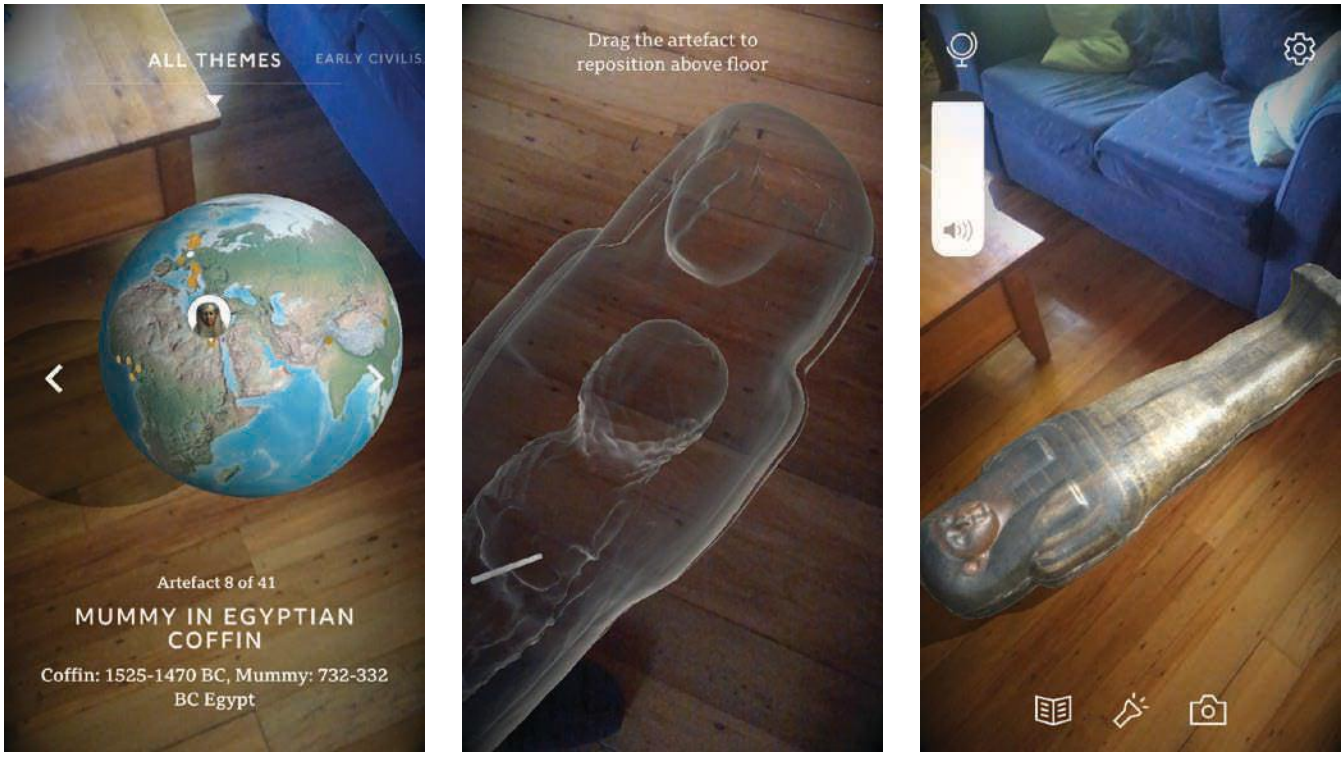
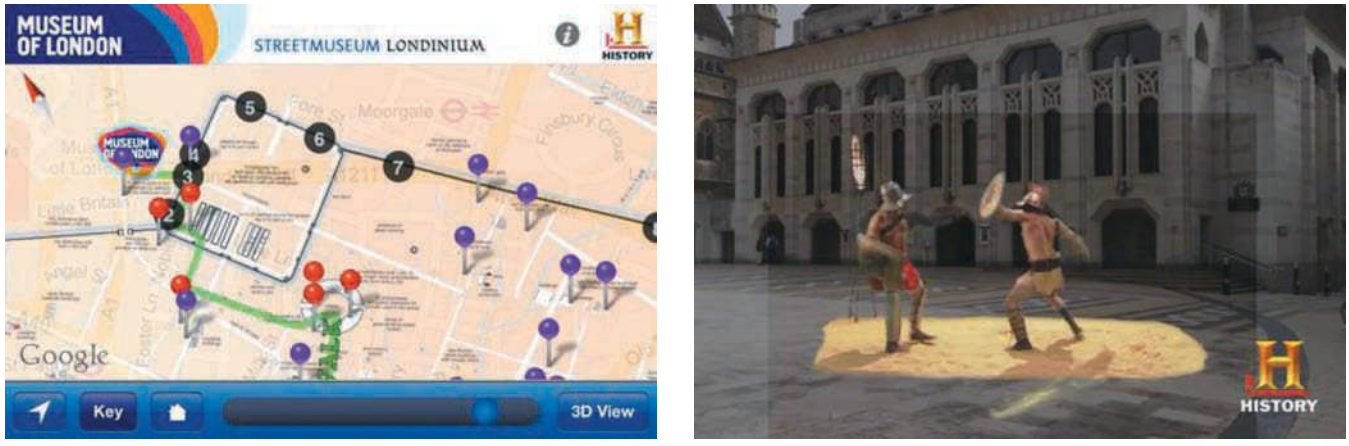


Figure 8.8 The BBC's *CivilisationAR* app allows users to select a period of history, a location on the globe, an item to view and special tools such as X-ray vision or a translator.

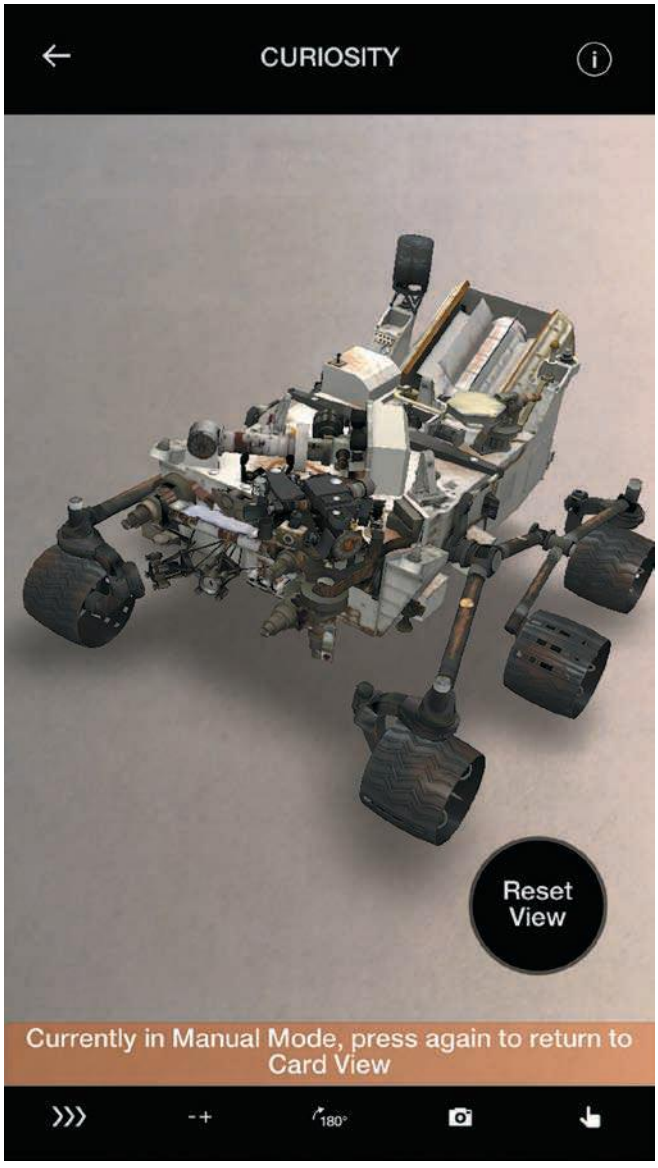


Source: Apple.com

Figure 8.9 *Londinium* allowed users to excavate ancient London using an interactive map by 'digging' (rubbing) on the screen. Updates to operating system software, however, became too great a challenge for the Museum of London and the History Channel to maintain it – a common issue for app developers.

Space apps

In NASA's *Spacecraft AR* app (Figure 8.10), highly detailed interactive models of actual spacecraft are triggered by an image of the surface of Mars, can be rotated in 3D space and parts of the spaceship can be animated and examined.



NASA

Figure 8.10 NASA's *Spacecraft AR* app provides many choices of spacecraft and a marker to trigger them. These can also be viewed manually as above.

Educational apps

Osmo, winner of *TIME*™ *magazine*'s Invention of the Year for 2014, recognises letters forming a word for the displayed image. The front-facing camera uses a mirror to view letters placed in front of the tablet and AR to interpret their shapes instantly. Players in red and blue teams compete to be the first to place their letters, and teachers can design their own customised word and picture libraries online (Figure 8.11).

Source: Osmo

Figure 8.11 *Osmo*, winner of *TIME magazine*'s Invention of the Year for 2014, recognises letters in a word for the displayed image.

KNOWLEDGE BUILDER

Froggipedia

Figure 8.12 shows three screens from *Froggipedia*, voted an app of the year, which enables a frog's anatomy to be examined and then to be dissected while appearing as if on your actual bench.

Questions

1. Download and experience *Froggipedia* or another app of your choice.
2. Evaluate the UI of the app using the principles in the previous chapter. Evaluate sections separately.
3. Discuss the app's UX using the principles introduced in the previous chapter.
4. If you used *Froggipedia*, research online why reviewers thought this app deserved to win app of the year.

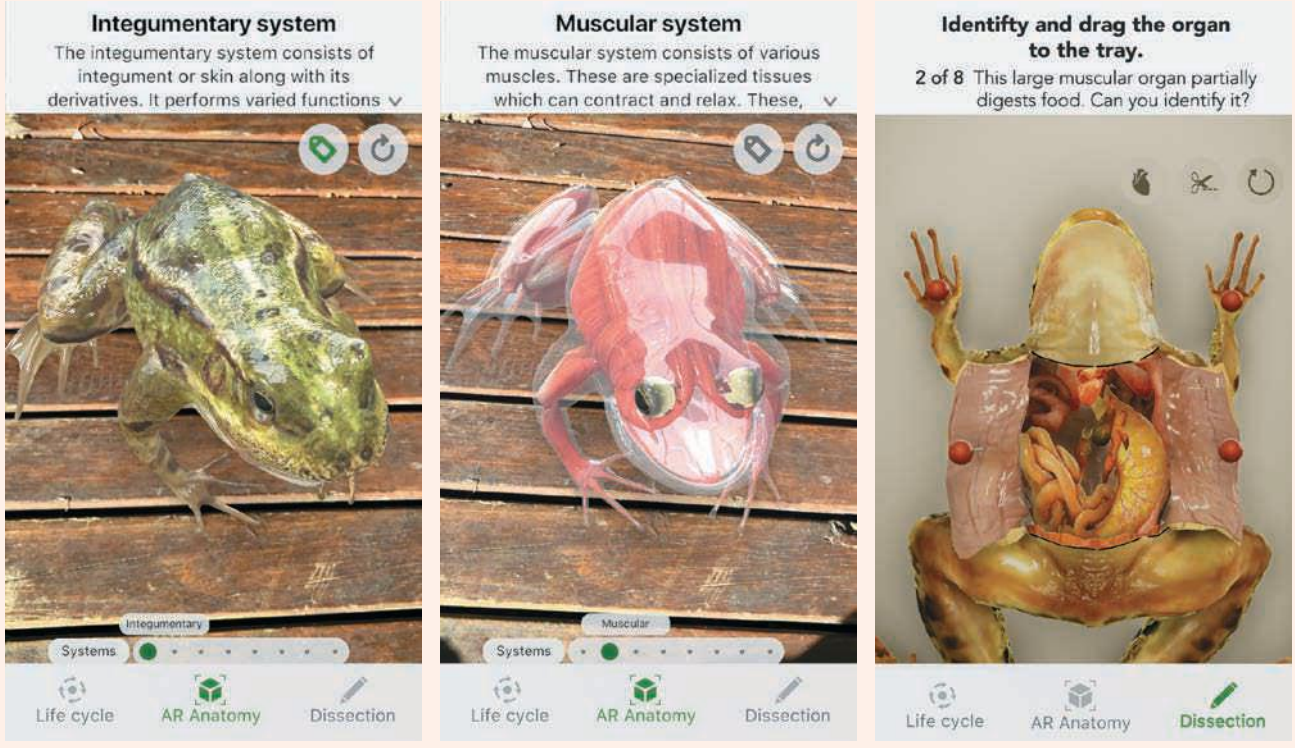


Figure 8.12 In *Froggipedia*, the frog can be handled, examined and dissected. Here it sits on an outside table using AR. Download and try out this app.

Business apps

Furniture maker IKEA has adopted AR, and using their IKEA Place app, chosen furniture appears as if located in the room (Figure 8.13).

IKEA's first AR app used the printed catalogue as the trigger for the AR. Users placed the catalogue somewhere on their floor and the 3D product appeared in that location. The app now uses a planar surface as a trigger. Can you suggest why the developers made this change to its UI?

Guerrilla marketing is an advertisement strategy in which a company uses surprise to promote a product or service. In one Christmas season, a Westfield Shopping Centre installed AR-triggered digital billboards where products were creatively animated around images of customers filmed while shopping.

Drink company Pepsi(Co) even constructed their own AR-powered bus shelter as a form of guerilla advertising! This YouTube video then turned viral with around 9 million views at last count. Watch the video.

Figure 8.13 Download the *Ikea Places* app to your mobile device and try it out. In this image, the dog is real but unaware of the chair as it is 'not there'. Notice the subtle shadow cast by the chair.

Weblinks
QuiverVision: Michael Bonner - Embrace It

Pepsi Max's Bus Shelter commercial explained

Just for fun! apps

Monster Park provides a highly immersive experience where users select an extinct creature and place it in their house or yard (see Figure 8.14). By selecting the portal option, users enter it then look back to their real world!

New Zealand company *QuiverVision* has for many years produced amazing AR experiences for younger children. After colouring a scene, the app *Quiver* brings the scenes to life clothed in the colours the children used (Figure 8.15). Many of their titles have an educational purpose as well. In one example, a children's book about a basketballer is brought to life and the user can try their hand at shooting hoops.

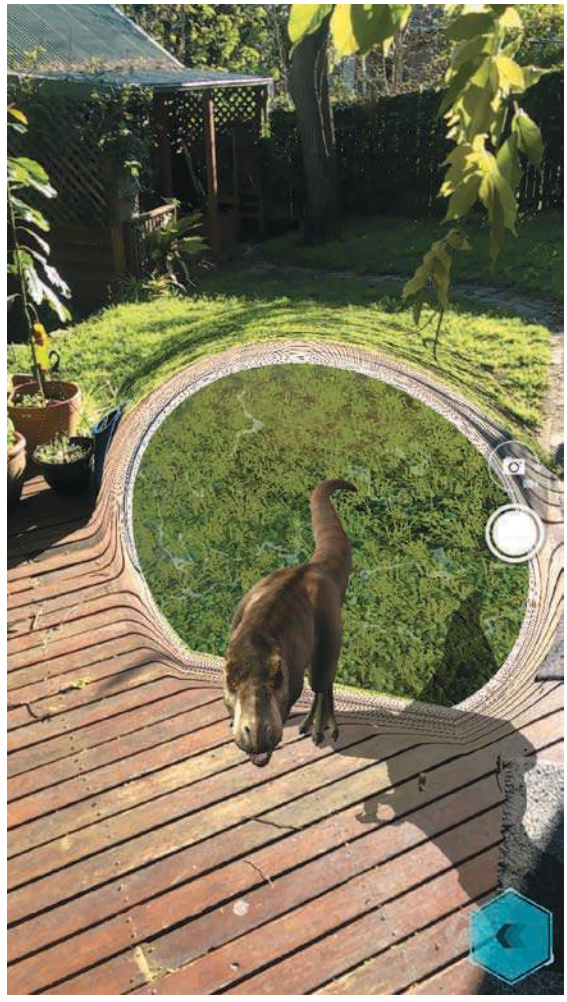


Figure 8.14 The *Monster Park* app provides a highly immersive prehistoric experience.

Source: Quiver

Figure 8.15 AR-based children's colouring using the *QuiverVision* app.

SKILL BUILDER

Use a *Merge cube* to create your own 3D creations

Alamy Stock Photo/Hugh Threlfall

Merge Labs, Inc.

Figure 8.16 *Merge cube* is an inexpensive soft foam cube, which triggers compelling AR experiences as well as allowing you to view your own 3D creations.

In this Skill builder, you explore the *Merge cube* with an associated app and create a 3D object of your own to manipulate using the cube (Figure 8.16). *Merge cubes* can be ordered online. You will need one to complete this Skill builder. You will be surprised at how compelling this UX can be.

Merge cube is an inexpensive soft foam cube covered on all sides with triggers for 3D object manipulation. The amazing thing about the *Merge cube* is the seamlessness of the UX. 3D objects are manipulated as the cube is rotated and viewed using apps on a mobile device or AR glasses (such as Google Cardboard).

1. Download these free Merge apps to your mobile device using Apple's App store or Google Play (for phone or tablet): *Merge Object Viewer, Merge Explorer* and *Merge HoloGlobe*. You do not need to subscribe for basic use. Explore these three apps using your *Merge cube*.
2. Use a 3D creation app such as *SketchUp* to design your own 3D object or download one from 3D warehouse or Thingiverse.
3. Upload an .obj or .stl version of your model to your phone or tablet.
4. Use *Merge Object Viewer* (downloaded previously) to view your object using your *Merge cube*.
5. Compare your UX of using the *Merge cube* with on-screen methods for viewing 3D.
6. Comment on the UI of the *Merge Object Viewer* app and the website.

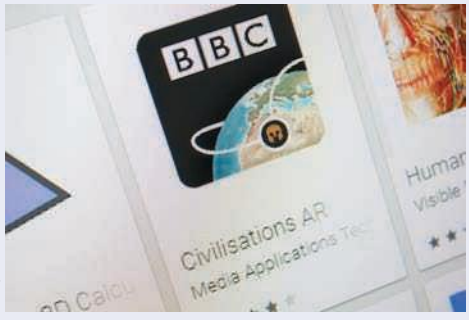
Weblink
Merge EDU

CASE STUDY

AR apps and user interface research

The research and development (R & D) division of the BBC wanted to understand how real users interact with mobile AR applications.

The BBC had released *CivilisationsAR* (Figures 8.8 and 8.17) and now needed to understand how well the various elements of the AR experience worked in being discoverable, useable, useful and to translate this knowledge into a set of design guidelines for future mobile BBC AR apps.



Alamy Stock Photo/Sharaf Maksumov

Figure 8.17 Items in the BBC's AR app *CivilisationsAR* can be placed anywhere to augment the real world.

UI research

The BBC engaged the product development and research company *Three sixty,* who studied 12 target Android and iOS users downloading and trying BBC's *CivilisationsAR* for a week then coming in for sessions to run through each part of the app in detail.

The research plan was designed to understand how the app was used in real life. They chose users who were all experienced using digital devices but new to AR.

The sample also included users with access needs, including one who was blind, but interested in trying AR. Although, many mobile AR apps today are not designed to be accessible for blind users, they uncovered ideas about how these immersive experiences can become more inclusive in the future.

Results

Following the research, *Three sixty* produced a detailed set of guidelines stating what does and doesn't work well with mobile AR.

Weblinks
threesixty: BBC Civilisations AR

Apple: Augmented reality

Zapworks

Introduction to Zapworks

Systems thinking

The guidelines cover important UX topics for AR such as:

- detecting surfaces
- guidance for first-time users
- manipulating 3D objects
- optimising
- use of audio and haptic feedback
- overlaying interactive elements in AR
- balancing AR and traditional 2D interactions
- AR accessibility.

One outcome was the discovery that the user must be told if a surface has to be horizontal or not before placing the AR object. Another was the importance of allowing the user to reposition, rotate and resize objects.

These guidelines are now being used by the BBC to guide the design of more mobile AR experiences.

Questions

1 Use the adjacent weblink to look at Apple Computers' UI guidelines for using their AR kit. Are there any principles in this article that are not mentioned in this Case study?
2 Try the BBC *CivilisationsAR* app. List features of the UI in the app that give the user effective feedback.
3 What did the research discover?
4 Suggest another improvement for the UI.

SKILL BUILDER

Build your own first AR experience

In this Skill builder you create your own AR experience using *Zapworks*.

World tracking, image tracking and face tracking are all possible using *Zapworks* (Figure 8.18).

Source: Zapworks

Figure 8.18 In *Zapworks* face tracking can be used. Here, a helmet (or face paint!) is matched to head movements.

You create a trigger, then overlay and view your creation using a mobile device.

1 Explore *Zapworks* (choose Designer). See weblink.
2 View the *Zapworks* introductory video at the weblink provided.
3 Select *Zapworks* Designer and view the introductory lesson tutorial.
4 Build your first simple AR experience.
5 Test the AR experience using the app on a mobile device.

ACTIVITIES

Identify

1 What is the MR continuum?
2 What is the difference between AR and VR?
3 Identify types of media that can be overlaid in an AR project.

Analyse

4 Describe an AR app you have used from the point of view of the experience of a regular user.
5 How can companies like Ikea take advantage of AR?
6 Compare the *Merge cube* to other AR products.

Extend

7 Research the way AR technology locks 3D augmented objects to an image or real-world environment.
8 Sketch (on paper) an educational product that could utilise the *Merge cube's* features.

9780170442787

8.3 Virtual reality (VR)

We experience the world through our senses. You have probably heard we have five senses: taste, touch, smell, sight and hearing. The truth is, humans have more senses than these, such as a sense of balance. All our sensory inputs are processed by our brains and provide a flow of information from our environment to our minds.

If we present our senses with invented information, our perception of reality changes in response. With VR, users are presented with a version of reality that isn't real, but it is perceived as real.

Using just a piece of folded cardboard and a mobile phone (Figure 8.19), it is possible to visit an ancient Mayan city, perform a chemistry experiment, travel through the human body, explore a star system in the galaxy, tour ancient Jerusalem or tour a nuclear reactor. VR opens a whole range of realistic experiences.

VR creates and presents an artificial 3D environment using software in such a way that the user's brain suspends disbelief and accepts the experience as a real environment.

A worthwhile VR experience can be created using *Google Cardboard* viewers for under $20; however, using the more sophisticated *Oculus Quest*, the experience of VR can be even more compelling (Figures 8.19 and 8.20).

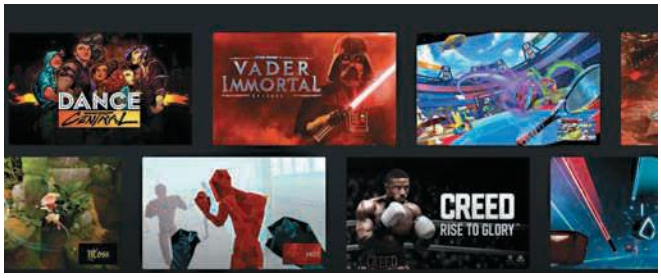
Shutterstock.com/misszin

Getty Images/T3 Magazine

Figure 8.19 Two VR headsets: the inexpensive Google Cardboard (around $20 using the user's mobile phone) and the Oculus Quest self-contained headset (around $600).

Source: Oculus

Figure 8.20 The lobby area and UI of the Oculus headset, showing the user library.



Source: Oculus

Figure 8.21 A sample of VR titles available for the *Oculus Go*.

You can explore the secret rooms Anne Frank and her family used to hide themselves from the Nazis during the war and hear her story in *Anne Frank House VR*. Dust realistically hangs in the air as you move around and 'pick up' objects.

You hold your breath while Alex Honnold climbs the vertical rockface of El Capitan without ropes or safety harnesses in *Free Solo 360* (Figure 8.22) and follow his years of preparation.

Alamy Stock Photo/TCD/Prod.DB

Figure 8.22 View National Geographic's VR app Free Solo 360 and accompany Alex Honnold throughout his heart-stopping climb of the vertical rockface of El Capitan, without ropes or safety harnesses.

VR offers unique experiences that are impossible to deliver using other methods of multimedia.

CASE STUDY

A VR time machine

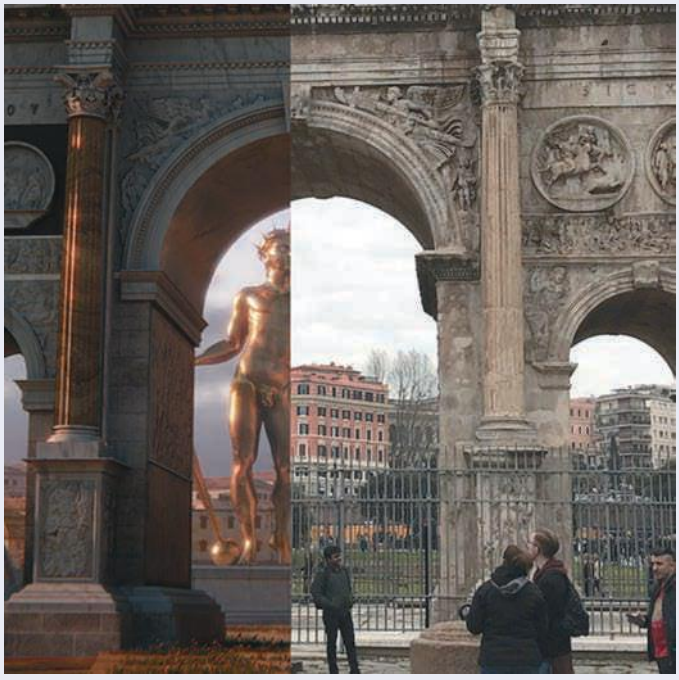
Have you ever stood in front of a historic ruin and closed your eyes, imagining what the scene before you would have looked like centuries ago?

Thanks to VR, seeing ruins as they looked in their heyday is possible. It may even be a game changer for how we visit ancient cities like Jerusalem or Paris.

Australian archaeologist Simon Young would like to show people what Rome looked like nearly 2000 years ago by fitting them with VR headsets (Figure 8.23). His company, *Lithodomos VR*, creates immersive virtual recreations of iconic ruins. The recreations can be used onsite with a smartphone headset, or from home or school using a commercial VR system like *Oculus* devices.

'It's 360-degree 3D virtual reality', Young says. 'It really helps you to place yourself back in time.'

Today, Rome's Temple of Venus lies split in half, most of its columns gone, ravaged by centuries of fire, earthquakes and pillaging. But with a VR headset and *Lithodomos*' app, suddenly it's a June afternoon in the 1st century AD. The temple is whole again, its vanished columns are there again, its façade of white marble and the intricate relief sculptures of its pediment have been restored. The app maps your physical location onto the temple, allowing you to look around from various angles.



Source: Lithodomos VR

iStock.com/selimaksan

Figure 8.23 *Lithodomos VR* creates apps for historic tourist locations which recreate scenes from the past using geolocation triggers.

Young observes: 'It's low season at the moment in Italy, but there are still hundreds and thousands of people wandering in the streets and looking at ruins'.

Young sees his software being used by tour groups who would provide their guests with headsets, or by individuals using cheap, portable viewers like Google Cardboard. He also hopes to partner with museums and universities to create other historical VR experiences, such as allowing museum-goers to view artefacts up close and in 360 degrees.

As an archaeologist, Young worries that some companies offering ancient world VR experiences aren't serious enough about accuracy.

'Some game developer in Silicon Valley who has no idea thinks, "Oh, a column would look great there"', he says. 'The real danger is that, because VR is such a powerful medium, if someone visits the Colosseum, they walk away with the belief that this is what it was like.'

Source: Emily Matchar, 'See the Ancient World Through Virtual Reality', *Smithsonian Magazine*, 22 February 2017.

Sample a showcase of historic VR recreations by downloading the *Lithodomos Engage* app for iOS or Android from the *Smithsonian* using the weblink provided and trying out one of the free examples.

Weblink
See the Ancient World Through Virtual Reality

Applications of VR

Educational apps

The range of VR educational applications has become far greater with the availability of low-cost viewers such as Google Cardboard and cheaper high-quality viewers, such as those from Meta. VR allows a student to experience the closest thing possible to handling an archaeological object, visiting another country or a compelling encounter.

Following are just a few examples we suggest you download and try out for yourself. Make sure audio is turned on!

Helping those with autism

Shutterstock.com/wavebreakmedia

Figure 8.24 *Floreo VR* uses VR to help those with autism improve social and communication skills.

Autism spectrum disorder (ASD) affects how people communicate and interact with others. It affects how people make sense of the world. People with ASD can experience difficulties with communication, social interaction and sometimes show restricted or repetitive interests and behaviours. They often experience oversensitivity or undersensitivity to sounds, smells or touch. All these elements can affect their behaviour.

Floreo VR (see Figure 8.24) uses a library of fun VR lessons for the learner and provides data for a supervising adult in the form of graphs that monitor and track the learner's progress. Lessons include topics such as attention, gestures, imitation, breathing, school life and general social skills. The results of research following its use have shown positive changes in communication and social interaction for users.

Chemistry

MEL Chemistry has cost more than $12 million to develop and uses VR alongside physical kits delivered to the door to allow students to enter into the structures of solids and gases at the molecular level and to mix and construct atoms to learn about electron configuration (Figure 8.25).

You can test run the VR app yourself for free.

MEL Science

Figure 8.25 The *MEL Chemistry* VR program has been described by the Royal Society of Chemistry as one of the most exciting and ambitious home-chemistry educational projects ever made.

Entertainment apps

The immersive nature of VR lends itself to many exciting entertainment applications. Realistic roller-coaster rides, first-person adventures, carnival-style games and puzzle-solving are some of the many products that have been developed.

The most popular early computer screen-based adventures, such as Myst and Riven, involved explorations of realistic artificial worlds which combined rich graphics, ambient sounds and puzzle-solving roadblocks. Now with the added technological VR features of 3D and interactive navigation, these adventures are even more compelling.

Vehicle manufacture, sales and service

By wearing a VR device to examine 3D models of vehicles, employees can understand their features better and interact with them. Sales personnel can become aware of special features that might generate interest and customers can compare models and view different colours.

Customers can also explore a physically present car, while virtually activating its sensors and seeing them in action. For example, in Figure 8.26, a virtual cyclist is entering a car's detection zone and causes the vehicle to signal.

Getty Images/Bloomberg

Figure 8.26 Mercedes gives employees and customers virtual experiences of their vehicles using VR.

9780170442787

Space apps

NASA has used VR technology for many years. Astronauts are trained using simulated VR environments on Earth. This is far less costly than having to create physical recreations and the constant simulation of weightlessness. Immersive VR is used to train astronauts before flights. These simulations include exposure to zero-gravity work environments, training for spacewalks and the use of various specialised tools using low cost tool mock-ups (see Figure 8.27).

NASA, Public domain, via Wikimedia Commons

Figure 8.27 NASA astronaut and Flight Engineer Christina Koch wears a VR headset for exploring how microgravity affects an astronaut's motion, orientation and distance perception.

SpacewalkVR invites users to experience floating above the Earth by setting off from the cargo bay of the Space Station's shuttle, playing the solar panel challenge and choosing the shuttle orbit path and time of day (Figure 8.28).

Source: Forth Interact

Figure 8.28 *SpacewalkVR* gives users the option to experience floating above the Earth.

Medical apps

Using VR, emergency doctors learn how to treat children who have suffered a seizure or anaphylactic shock. In this way, it is safe to make mistakes and learn from them. In this controlled environment, doctors can test out options that would be unrealistic in the pressured situations faced in a real emergency.

The doctor's performance can be assessed as well. In medicine, VR training saves lives (Figure 8.29).

Getty Images/MediaNews Group/Orange County Register

Figure 8.29 VR can also be used to lower risk in life-threatening situations in medical procedures. Surgeons at the Children's Hospital in Los Angeles (CHLA) were funded by Meta's *Oculus VR* division to create training in treating children in emergencies.

Retail apps

Inside a simulated VR environment (Figure 8.30), employees might find themselves standing in a busy aisle facing multiple problems (e.g. a product spill, questions about product locations and an angry customer) while being given 30 seconds to figure out how to solve them.

Managers can identify how well the employee knows their store and how well they respond under pressure.

Shutterstock.com/FrimuFilms

Figure 8.30 Retail employees receive training using VR simulations.

Drones and VR

VR headsets are used in first-person view (FPV) drone control, where participants control quadcopters equipped with cameras (Figure 8.31). Head-mounted displays stream the live video feed from the drone's camera. Most countries require the user to have line-of-sight while flying a drone. This is not possible while using VR goggles and thus another person is required to keep the drone in sight.

The remote control, the drone and the VR goggles are all connected via radio but must transmit with sufficient speed and reliability to allow control without significant delay (latency) in the signal.

VR can be used to view the live video as well as in drone racing where the goal is to complete a set course as quickly as possible. Drone racing began in Australia in 2013.

Shutterstock.com/Lukas Gojda

Figure 8.31 VR headsets provide the user with a view from the drone's camera.

8.4 Ethical issues in VR

It is possible for a surgeon to conduct an operation on a patient from a remote location using interactive digital media. Few people would raise ethical objections to such an approach, provided the technique is reliable and accurate.

However, it is also possible to simulate human behaviours, including strong violence, so realistically that it could be argued that the only difference between it and the real thing is that human beings are not physically injured. It is less certain whether the user is being psychologically injured. Many people argue that exposure to such realism, and participation in it, desensitises the user to violence in real life.

VR clearly raises the question of whether an act committed in our imagination should be made illegal if it does not physically affect another person. Constant exposure to certain images, thoughts, experiences and ideas may affect the way in which we see the world, and perhaps the way we act in it.

INFOBIT

In 2019 Oculus (Facebook, now Meta) bought a Danish eye-tracking company which developed 'foveated rendering' – a technology that only renders high resolution in that section of an image where the user is looking. This results in lower bandwidth requirements and is a form of lossy data compression.

Our eyes have two types of light receptors: rods and cones. Rods work in low light but do not recognise colour whereas the cones are the cells that recognise colour and give us sharp images. The fovea is a small depression at the back of our eyes where the layers of the retina spread to let light fall directly on the cones. Lower resolution is used further away from the centre of the visual field, which reduces greatly computational load on the CPU.

Question for discussion

Should the law judge only those who perform such actions or should the digital designers and programmers of this material be liable as well?

Getty Images/The Asahi Shimbun

Figure 8.32 Should we be concerned if large corporations control popular VR devices?

8.5 The evolution of the VR user experience

From the earliest times artists and sculptors have devised ways to make the UX more realistic with an ultimate goal of creating experiences indistinguishable from real life.

In June 1838, the British scientist Charles Wheatstone discovered that if he drew two pictures from two slightly changed perspectives, and delivered each one to a different eye, the brain saw them as three-dimensional. He noted this is exactly how our brains use our two eyes to interpret our 3D world (Figure 8.33). Wheatstone used mirrors to build the first stereoscope.

A popular early effort at VR was the *diorama* (Figure 8.34). Dating from the late 1800s, these exist in most natural history museums and continue to provide an immersive experience of historical events and natural landscapes. A curved painted panorama acts as a backdrop for recreated scenes in which human figures and models are placed.

Shutterstock.com/Danny Smythe

Alamy Stock Photo/ART Collection

Figure 8.33 This 1905 stereoscope (left) gave users their first experience of simulated 3D scenes; (right) the Viewmaster (still available) was a popular 20th century version.

Alamy Stock Photo/Bjorn Svensson

Figure 8.34 This 1924 diorama of Lone Pine in the Australian War Memorial in Canberra shows the men of the 1st Australian Infantry Brigade attacking Turkish trenches. (The Australians are wearing white patches to prevent them being fired on by their own men.)

Immersive virtual technology

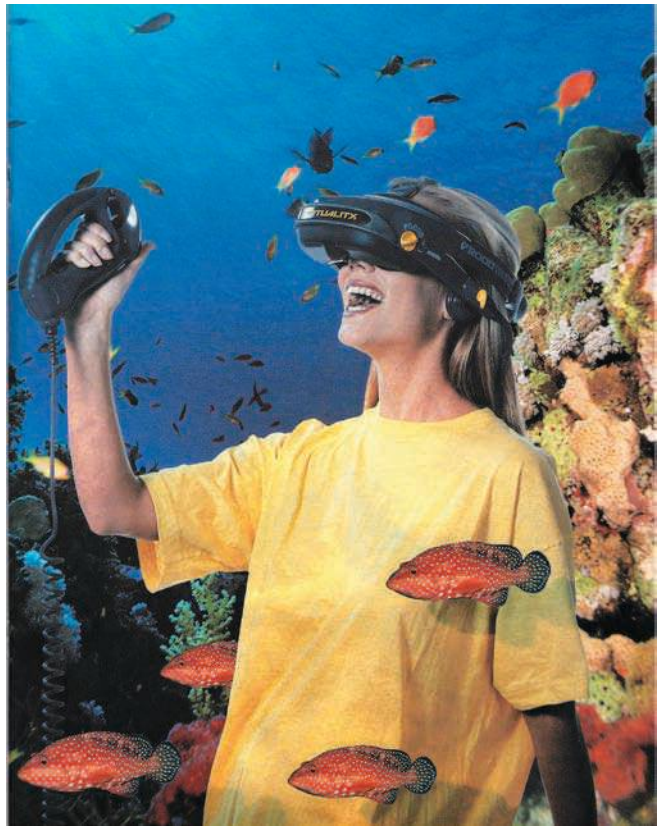
One of the earliest examples of immersive virtual technology was Morton Helig's 1962 *Sensorama* – his description is remarkably similar to that of the 4DX theatre experience at the start this chapter.

The Sensorama displayed stereoscopic 3D images along with body tilting, stereo sound, wind and smells triggered during the film (Figure 8.35). In fact, Helig described it as a view of the future of cinema.

Courtesy of Morton Heilig Collection, HMH Foundation Moving Image Archive, USC School of Cinematic Arts

Figure 8.35 Morton Helig's 1962 *Sensorama*.

In 1991 The Virtuality Group launched its VR arcade equipment (Figure 8.36). These machines could only be afforded by large venues. Its games were stored on huge discs and used 3D drawn graphics and animated objects. Players wore a headset, held a joystick and stood inside a podium. One popular game involved intercepting attacking pterodactyls.



Wikimedia Common/Dr. Waldern/Virtuality Group/CC BY-SA 4.0

Figure 8.36 The Virtuality Group's Cyber VR arcade podium.

Cardboard VR devices

In 2014, two French Google engineers developed a low-cost cardboard VR device into which a mobile phone was inserted. They came up with their *Google Cardboard* concept (Figure 8.37) during 'innovation time off', during which Google encourages its employees to try out new ideas. Many millions of the device have been sold. You can even construct one yourself for the cost of $1 plastic lenses!

In 2015, Samsung, in collaboration with Oculus, marketed its *Gear VR,* also requiring an inserted phone, an advanced version of Google Cardboard.

Oculus developed the *Oculus Rift* using a Kickstarter project, marketing it in 2016; however, its cost and the need for it to be connected to a fast computer was an obstacle for many.

Then in 2019 it marketed the *Oculus Go*, the first VR device that was self-contained, affordable and did not require a phone.

For his major year 12 project, the student in Figure 8.38 designed a virtual Jurassic-era world in VR and populated it with dinosaurs created using Blender to create 3D animations as shown in Figure 8.44. One issue faced by VR developers is that using VR goggles for extended periods can become tiring.

Shutterstock.com/AlexLMX

Shutterstock.com/rootstock

Shutterstock.com/Alexandru Nika

Figure 8.37 Google Cardboard's smartphone-powered VR goggles.

Figure 8.38 *Oculus Go* standalone VR headset.

Chapter review quiz

In 2023, Apple showed off its first VR headset, the Apple Vision Pro, which initially cost many thousands of dollars, but is likely to be the forerunner of cheaper models in future. Enhanced UX of both AR and VR are prominent in its design features. It supports both AR and VR applications and Apple has expanded the features of a typical headset with users controlling their VR experience using their eyes, hands and voice, eliminating physical controllers altogether. An external sheet of wrap-around glass acts as a display to show others what your eyes look like at that moment, so they are not disconnected from you. When you are fully immersed in VR, a cloudy view appears on the front of the Vision Pro to let the people around you know that you're not able to see them at that moment. Audio ray tracing is used to create a sound profile of your space.

KNOWLEDGE BUILDER

The first AR?

Wikimedia Commons/George Steckel - Los Angeles Times photographic archive, UCLA Library, Public Domain

Figure 8.39 L Frank Baum, author of *The Wizard of Oz*.

In 1901, L Frank Baum (Figure 8.39), author of the *Wizard of Oz*, had a concept of AR 112 years before the invention of Google Glasses or AR apps. What follows is the earliest recorded reference to an AR device:

> 'On the other hand', continued the Demon, 'some people with fierce countenances are kindly by nature, and many who appear to be evil are in reality honorable and trustworthy. Therefore, that you may judge all your fellow-creatures truly, and know upon whom to depend, I give you the Character Marker. It consists of this pair of spectacles. While you wear them every one you meet will be marked upon the forehead with a letter indicating his or her character. The good will bear the letter "G", the evil the letter "E". The wise will be marked with a "W" and the foolish with an "F". The kind will show a "K" upon their foreheads and the cruel a letter "C". Thus you may determine by a single look the true natures of all those you encounter.'
>
> 'And are these, also, electrical in their construction?' asked the boy, as he took the spectacles… 'All character sends out certain electrical vibrations, which these spectacles concentrate in their lenses and exhibit to the gaze of their wearer, as I have explained.'
>
> 'It's a fine idea', said the boy; 'who discovered it?'
>
> 'It is a fact that has always existed, but is now utilized for the first time.'

Source: Johnson, Joel. "The Master Key": L. Frank Baum envisions augmented reality glasses in 1901, *Mote & Beam*, 10 September 2012. (https://wiki.mq.edu.au/display/ar/Augmented+reality+history%2C+background+and+philosophy - site developed with Macquarie University by author)

KNOWLEDGE BUILDER

Imagining the UX of the future

In recent years, applications, including games, offer increasingly immersive experiences. Content is more realistic. Images are more detailed. Animation is more believable and surround sound more compelling.

Photorealistic, real-time CGI images are here now, and sound is now close to true 3D sound – where a noise seems to come from the appropriate point in space, relative to the listener.

Perhaps we can imagine what future user experiences might be like.

Displays might provide true peripheral vision, with microphones and earphones built in. We can assume systems that generate smells, and others that will give users sensations of texture and temperature (such as fur, sandpaper or water) will be common. Such features are already part of 4DX cinemas.

After this we can begin to imagine users suspended in 3D fields so their minds are fooled into seeing the world differently, as imagined by the film *Minority Report*.

And, sooner or later, someone will come up with a safe way of making digital connections to human nerve cells. Instead of fooling the senses, this will bypass them completely by telling our brains exactly what sensory input it is receiving, right down to the last detail.

What if instead, we were to bypass the senses completely and transmit data directly to the brain? This would be the ultimate in virtual reality – a system so powerful that it would let us interface directly with the human imagination!

Is such a final vision for technology ever likely to be developed? Well, yes, it is. In fact it was invented in Mesopotamia some 5000 years ago. It is called 'text'!

We may have access to the most intensive user experiences imaginable, but as long as we can read there's always something better.

Question for discussion

Do you agree with the idea portrayed in Figure 8.40? If VR can provide us with experiences indistinguishable from reality, and provide access to huge databases of the world's knowledge, might books become obsolete?

Alan Moir, Sydney Morning Herald

Figure 8.40 Children looking for the on switch on a book … in your lifetime?!

ACTIVITIES

Identify

1 Describe how the *Google Cardboard* device creates a VR experience for users.

2 Identify two differences between a *Cardboard* viewer and the *Oculus Go*.

3 How can VR enhance the experiences of visitors touring historic sites?

4 Identify three areas where VR would be useful.

Analyse

5 Describe your earliest experience of VR.

6 Think about your most compelling VR experience to date. Describe what was effective about the UX and what could be improved.

7 Identify two UI challenges interactive VR devices present to designers and describe solutions to each of these.

Extend

8 Research and report on challenges faced by Oculus in producing VR experiences.

9 Apart from those described in this chapter, find one VR training application used in industry and describe the benefits it offers over other training methods.

10 There is a popular saying, often wrongly attributed to Confucius: 'I hear and I forget. I see and I remember. I do and I understand.' Respond to the statement with reference to VR UX.

SKILL BUILDER

Build your own VR experience using *CoSpaces*

This chapter provides two possibilities for building your own VR experience. The first is this Skill builder, which uses *CoSpaces*. The second is the Project for this chapter, which uses the industry standard development tool, Unity.

In this Skill builder, you create your own VR interactive world using *CoSpaces*. You then view it through a VR device such as *Google Cardboard* or similar.

You start by using *CoSpace*'s prebuilt library of backgrounds and objects then use your own panoramic image.

The Unity project at the end of this chapter provides the opportunity to create your own completely realistic 3D experience.

You can view our simple *CoSpaces* example, which demonstrates a 360-degree background we created using *Google Streetview* (see step 6 below), including the use of paths, animation, music and clickable objects (can you find two?) by using the weblink provided.

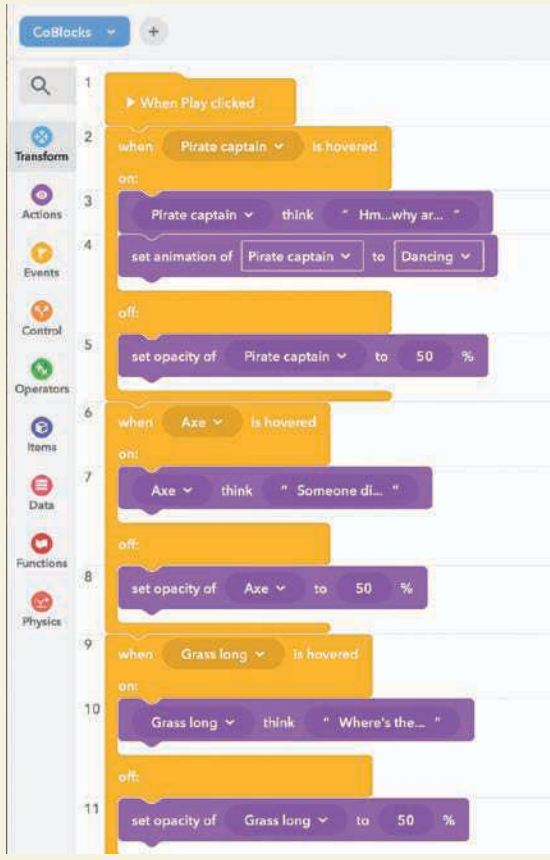
Weblink
CoSpaces Edu: Paths and Code Demo

Design thinking

1. View the introductory video Creating with Cospaces EDU (see weblink provided).
2. Design your world on paper. Think about UX by determining your purpose, audience, UI and interactions.
3. Create your first VR world. Select from the following or use your own idea:
   - scene from history
   - room in a museum of objects
   - tutorial for identification of features of some device
   - eco-friendly house (Figure 8.41).
4. Use *CoSpace's* library to select your background, objects and pre-defined animations. (Optional) View the *CoSpaces* video tutorial on coding for further skills, then code object interaction using the *CoSpace* coding toolbox.
5. View your finished product in VR glasses mode using the free *CoSpaces* mobile app (click the VR glasses icon after opening your project).

Figure 8.41 Eco-friendly VR house.

Create a virtual tour using your own 360-degree image



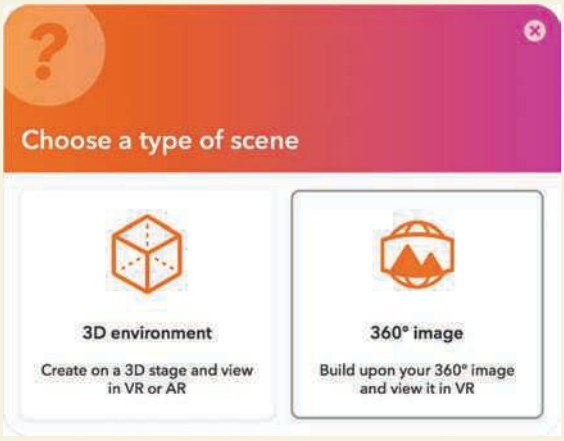
Source: CoSpaces Edu

Figure 8.42 (Top) In this VR scene, a student loaded a 360-degree historic image and added objects from the *CoSpaces* library to accompany questions on a student worksheet. Can you identify some? (Middle) Code was added so speech bubbles appear when items are clicked and opacity is reduced to show the object has been located by the user. (Lower) The code that controls the interaction.

Weblink
Creating in CoSpaces Edu - Beginner tutorial

6 You will need to either obtain or make a spherical 360-degree image.

7 Login to *CoSpaces* and create a new space and select the 360-degree image button (Figure 8.43).



Source: CoSpaces Edu

Figure 8.43 Select the 360-degree image button.

8 Click *Environment* and select *Edit* to replace the default image by uploading your own 360-degree image.

9 Add objects and position them. Note they can be rotated and scaled. Right-click to include them in code.

10 Have others evaluate your final version and suggest improvements and additional features.

Weblinks:

- Introducing 4DX
- Meta Spark
- Apple SwiftShot Demo: First multi-player game created with ARKit 2
- QuiverVision: Michael Bonner - Embrace It
- Pepsi Max's Bus Shelter commercial explained
- Merge EDU
- threesixty: BBC Civilisations AR
- Apple: Augmented reality
- Zapworks
- Introduction to Zapworks
- See the Ancient World Through Virtual Reality
- CoSpaces Edu: Paths and Code Demo
- Creating in CoSpaces Edu - Beginner tutorial
- Unity Learn

Project resource:

- Create your own interactive 3D world using Units

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

8.6 Project: Create your own interactive 3D world using Unity

Your task

Have you ever wanted to create and control a world of your own? Here you use the industry standard app creation tool *Unity*.

In this project you will imagine your own world – adding trees, rippling lakes, buildings, the passage of day and night. When you have finished, you and your users can begin to explore it!

You can discover what makes a compelling UX in a virtual world by thinking about the features you enjoy in the real world. How realistic and interesting is the environment? What activities are available? Are there things to learn? An ancient culture or a new world to explore? Is there a challenge or adventure to be had?

Students have successfully used the following approach to recreate such worlds as Ancient Athens, the ruins of Pompei and a Jurassic experience (Figure 8.44). In these environments, players encounter historical objects and learn about the past. The world you create could be anything from an imaginary planet to a mysterious island. You are limited only by your imagination.

Project resource
Create your own interactive 3D world using Units

Figure 8.44 Student interactive projects of ancient Athens, Pompei and a Jurassic virtual world. Dinosaurs were modelled and animated using *Blender*, structures were modelled in *Sketchup* and imported into *Unity*.

A positive overall UX will be critical in such a game environment. You will want your users to 'feel at home' and to know where they need to go to achieve the goals you set. You need to provide visual clues to assist them.

Chapter 1 of this text provides an introduction to project management. Make sure you apply the important principles outlined in that chapter to your project, including addressing functional and non-functional requirements, considering social and ethical issues and documenting your project's development.

Identifying and defining

Background to Unity

Unity is a powerful game engine designed for creating both 2D and 3D environments. Many popular video games have been made using it. The personal edition of *Unity* is free and cross-platform. One of *Unity's* great strengths is that it allows you to build your game to run on most devices.

You start by designing in the *Scene View* where you create your components and test them in *Game View*.

Unity has drag and drop capabilities allowing it to be used without scripting, but for those who wish, it offers scripting control via the C# programming language. (This project will also suggest some scripting as an optional extra.)

Weblink
Unity Learn

Unity allows you to create your own world, build objects or import them from other software, such as *Blender and SketchUp*. By creating a player character, users interact with your world.

Unity even provides you with an assets store where you can find free and paid objects for your games.

Briefly research *Unity's* interesting history via the web before starting the project.

INFOBIT

If I had my way…I'd like to see high school and college kids using it, people outside the core industry. I think it's sad that most people are consumers of technology and not creators. The world's a better place when people know how to create, not just consume, and that's what we're trying to promote.

John Riccitiello, CEO of Unity

Install Unity and Unity Hub

Install Unity

1. Download your free personal edition of *Unity* for Mac or Windows from the *Unity* website and register as a user.
2. Install *Unity* and then use *Unity Hub* to create a new project.

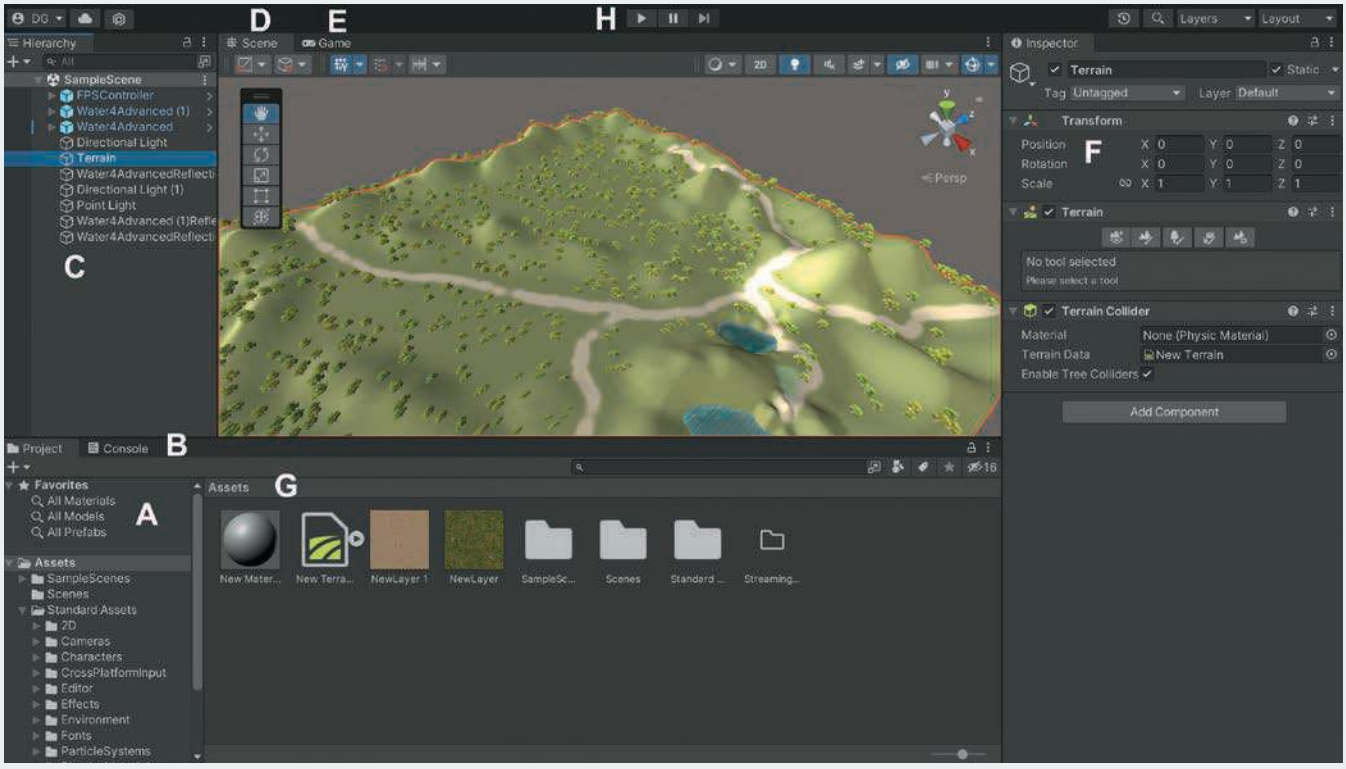
Get to know Unity

3. We suggest first completing *Unity*'s excellent 'Getting Started with *Unity*' tutorial, which introduces you to *Unity* basics (weblink provided).

Beginning your first project

4. If not already displayed, choose the default layout for *Unity*: Window > Layouts > Default.

Refer to Figure 8.45 and identify each feature of the Unity workspace.



Source: Unity

Figure 8.45 Components of the Unity Interface (shown in Dark mode).

A The Project panel shows all the assets you have imported. When you import assets into your project they appear here. These include 3D models, 2D images, scripts etc.
B Console panel is where errors are reported.
C The Hierarchy panel shows all game objects that relate to your current scene and the relationships between them. Clicking on items listed here allows you to locate them in your scene view and double-clicking zooms in on them.
D The Scene view is the editable visual representation of your current world.
E Game view allows you to run the game using the play button.
F The Inspector panel shows all the properties of the selected object. This changes as different game objects are selected.
G The Assets panel shows all the files available for your project. These include 3D models, 2D images, code files etc. Imported assets appear here.
H Toolbar contains controls for running the game to test it.

Toolbar

The most used tools are in a default floating tool panel which provides navigation within your scene and transformation tools to manipulate objects (see Figure 8.46). It contains the Hand, Move, Rotate, Scale, RectTransform (mainly for 2D projects) and Transform tools. Keyboard shortcuts are Q, W, E, R, T, Y respectively!

The last icon (Y) is recent and is especially useful as it provides access to all object transformations (move, rotate or scale) in one tool. A three-button mouse allows speedy access to these tools.

Note: The F key is also very useful. It will focus the scene on any item the mouse is pointing at.

Source: Unity

Figure 8.46 Unity app toolbar.

Try something simple first

Create a cube object to test out the *Unity* interface.

5 To become familiar with *Unity*, add a cube object to your scene using the menu bar as follows.

GameObject > 3D Object > Cube (or click *Create* under the *Hierarchy* tab)

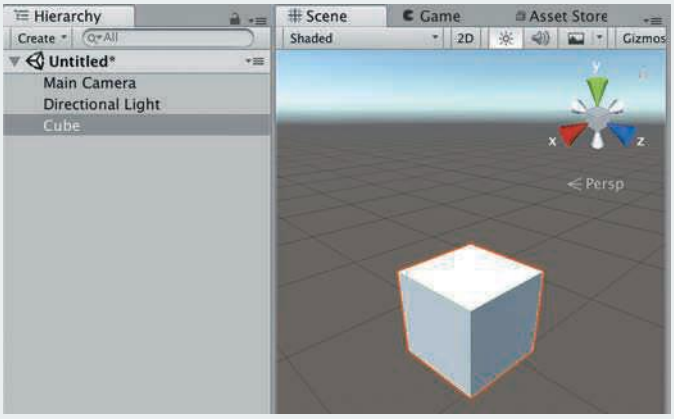


Figure 8.47 Your first creation – a cube.

Select the cube in the hierarchy panel (Figure 8.47). When you select a Transform tool (such as scale and rotate), coloured controls will appear on the cube. Use these to:

- *scale* the cube along an axis as well as scale proportionally from its centre
- *rotate* the cube
- try the Transform gizmo to do all at once!

6 Create a sphere (GameObject > 3D Object > Sphere). If you haven't moved your cube along an axis, the sphere will be created at the same position and may be hiding inside the cube! Move it upward using the move tool (Figure 8.48).

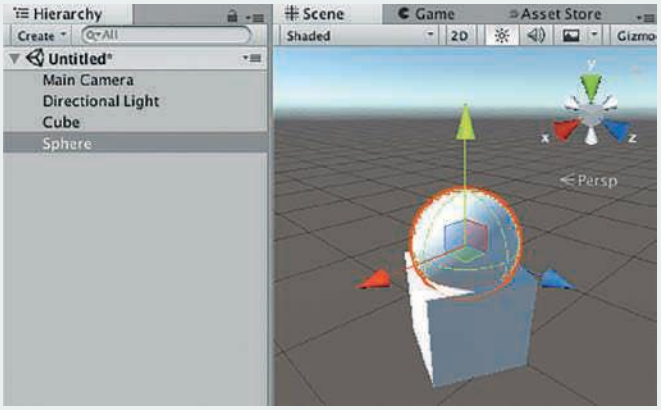


Figure 8.48 The sphere moved up on top of the cube.

7 Join the sphere to the cube by dragging its name in the hierarchy panel on top of the name of the cube. (Figure 8.49 top) The two will now be joined in what is known as a *parent-child* relationship. The sphere you dragged is called the *child* of the parent cube object on which it was dragged. The combined 'family' can be moved as one. Try doing this (Figure 8.49 bottom).

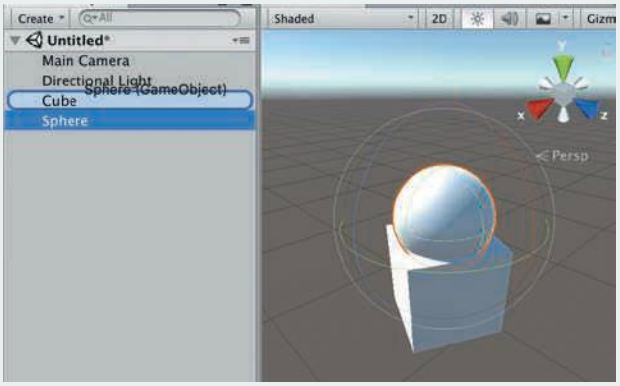


Figure 8.49 (Top) The sphere's name being dragged on top of the cube's name in the hierarchy panel. (Bottom) The result: A parent-child relationship is formed. The cube becomes parent of the sphere.

8 If you wish to reuse any object you create (including a parent-child combination), you can drag it from the Hierarchy panel into the Assets panel. It will now become what Unity calls a *prefab*. This is an item that contains all the objects, materials, textures and scripts in a single package for use in a project. It will turn blue in Assets to indicate its new status. Try this for your sphere-cube object.

INFOBIT

Shaders, textures and materials

Textures are flat bitmap images which are wrapped onto a 3D object to represent its surface and roughness.

The term *shader* was originally invented by Pixar to describe the feature of its early animation software which calculated the shadowing (correct lightness and colour of parts of an image). It has now evolved to handle the colour of each pixel to be displayed. If we had a *Unity* object wrapped with a wood texture, then if it got wet, a shader would be used to change the appearance of its texture; or if it was polished, how shiny it would appear.

Materials are like the containers for textures and shaders.

Changing material texture

Create a new material of your own: Assets > Create > Material. The material will now appear in your *Assets* panel. Rename it *mymaterial*.

9 Make sure your new *mymaterial* is selected in the *Assets* panel. Change its colour in the Inspector panel using the white colour patch selector near the top. Change it to, say, purple (see Figure 8.50).

Drag this *mymaterial* from *Assets* on top of your sphere in *Scene*. Your object will change to this material's colour.

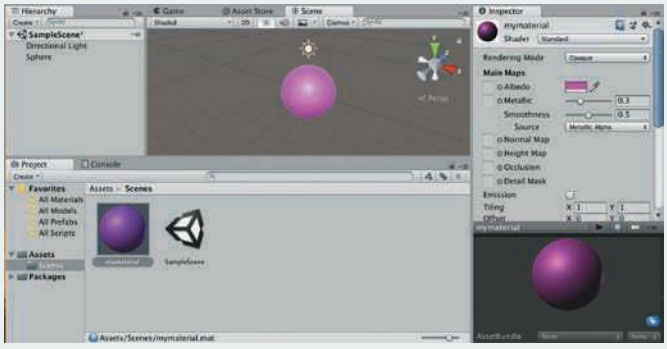


Figure 8.50 Creating materials.

10 Type *Textures* in the Assets search box and select an interesting one.

11 Keep your assets organised by creating a new folder named 'Textures' and place your wood texture file inside this (Figure 8.51).

Figure 8.51 Selecting a texture.

12 Drag this texture onto the sphere and *Unity* will automatically create a new material and add it to the *Materials* folder in *Assets*. The sphere will now acquire this new surface texture (see Figure 8.52).

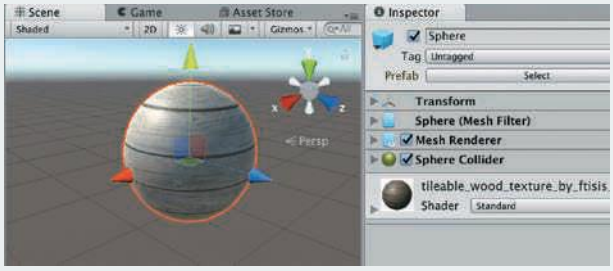


Figure 8.52 Sphere with texture material added. Note Inspector now lists this as the material for the Sphere.

13 Click on your sphere to see, in the *Inspector*, that your new textured material has been added as the sphere's material in its Mesh Renderer with further material details appearing further below in the Inspector panel.

14 Click the triangle to open these details. Observe the texture pattern thumbnail and change its assigned colour.

15 Experiment with the x and y tiling values to discover their effects. Experiment with different shaders.

Creating your world

Import assets

Design thinking

Unity has an asset store, where you can find free and paid assets for building your virtual world: trees, grass, water and controllers such as first person controller which becomes the player and moving camera. For some time it contained a free collection of standard assets, which at the time of writing are no longer in the store. In the Nelson MindTap resources for this chapter we have supplied a standard assets folder, modified for use with current versions of Unity. These resources will allow you to complete this project.

16 Download this file and import it into your project simply by dragging it into the Assets panel of your project. A dialogue box will eventually appear asking which files to import.

17 Choose to import all. Ignore any yellow warnings in the console.

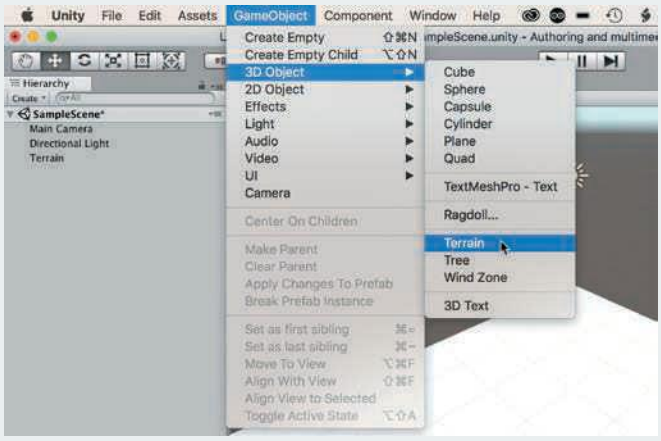
Other free alternative assets can be used and are described in the *read me* file on Nelson MindTap.

Create terrain

This is where you begin to create your own world!

18 Open a new scene: File > New Scene.

19 Select Game Object > 3D Object > Terrain (Figure 8.53). A new empty terrain plane will now have been placed in your world.



Source: Unity

Figure 8.53 Creating a terrain game object.

20 The terrain icons near the top allow you to 'paint' mountains on your world (Figure 8.54).

Figure 8.54 The terrain tools and the Terrain Settings cog (selected here).

21 Select the terrain object. It appears in your scene's hierarchy. In its Inspector the last Terrain icon is a mountain with a cog. Click on the cog to access Terrain Settings. Change the Terrain Width and Length to around 1000 (Figure 8.55).

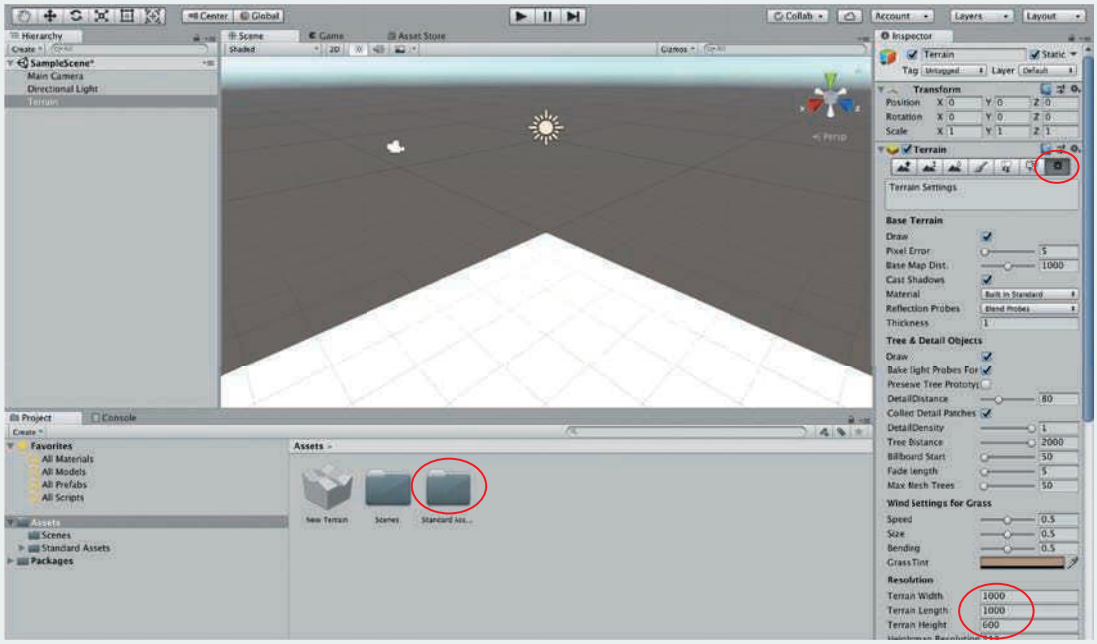


Figure 8.55 Newly created terrain now appears in scene window and is listed in the hierarchy panel. Note that the cog icon is selected in the terrain's Inspector panel top right. Width and height of the terrain have been increased to 1000. The Standard Assets folder, imported in Step 4 under 'Changing material texture', appears in the Assets panel of your project.

22 The second icon from the left gives you the power to raise mountains and lower them (while holding Shift down). The longer you hold your mouse button, the higher the mountains will grow.

Make sure Raise or Lower terrain is showing in the drop-down menu immediately below the terrain icons.

Select your new terrain object in your project's hierarchy then follow this suggested sequence to create your own mountainous terrain:

- Zoom out so you can see most of your terrain. Set *Brush size* to 100. Set *Opacity* (the strength of the brush) to 20. Create mountains by 'painting' with the first icon using the standard (first) brush. You may need to zoom out more using your mouse to control this properly.
- *Hold Shift* to lower mountains.
- *Select* Set Height from the drop-down menu and click as stated to left-click sample an existing mountain.
- *Select* Smooth Height to smooth out the terrain.
- *Select* Stamp Terrain and set the stamp height to create unusual formations.

You are offered various paint brushes for achieving different effects. Experiment using different brushes to create your world, remembering you can always undo any action. A suggestion: encircle the perimeter of your world with high mountains to prevent the player 'falling off' the edge when playing your game (Figure 8.56).

23 Create some completely enclosed areas between mountains to hold water later for lakes.

Save your work!

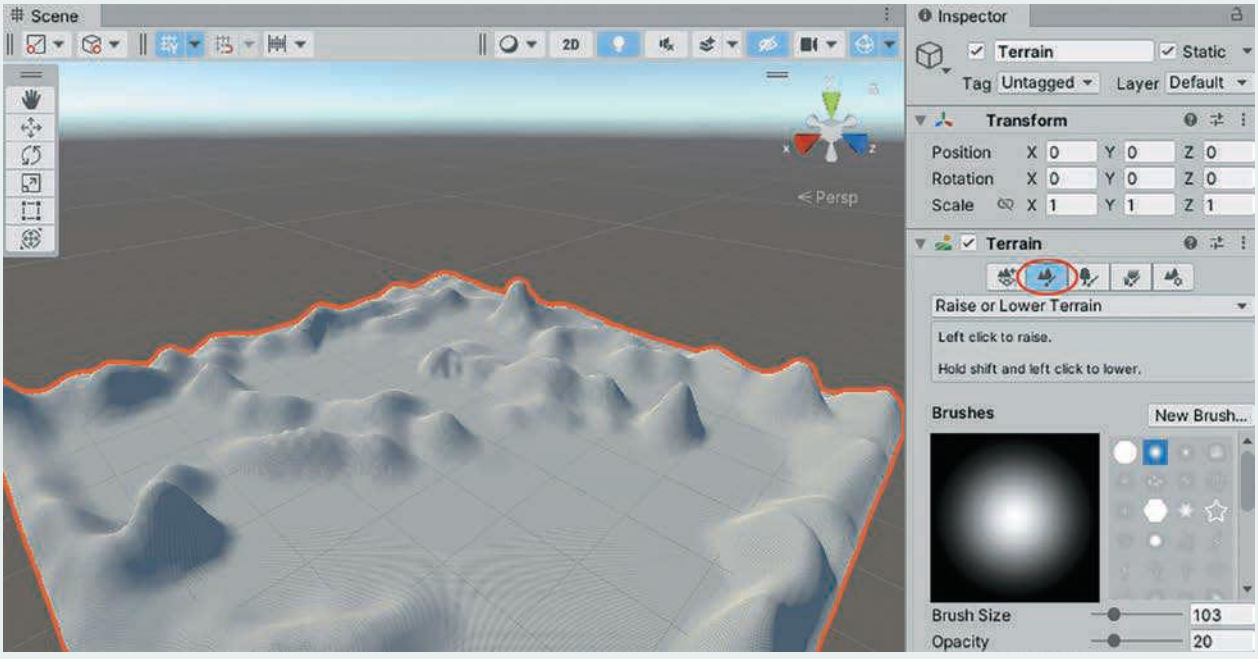


Figure 8.56 Mountains begin to appear as you use the second terrain tool.

Add environment assets

At the start of this project you imported some Standard Assets. These contain textures for you to clothe your terrain with grass, sand, rocks, trees and water.

Grass, rocks and sand

24 Select your existing Terrain object in the hierarchy and in the Inspector keep the second icon selected. Change the dropdown selection to Paint Texture (Figure 8.57).

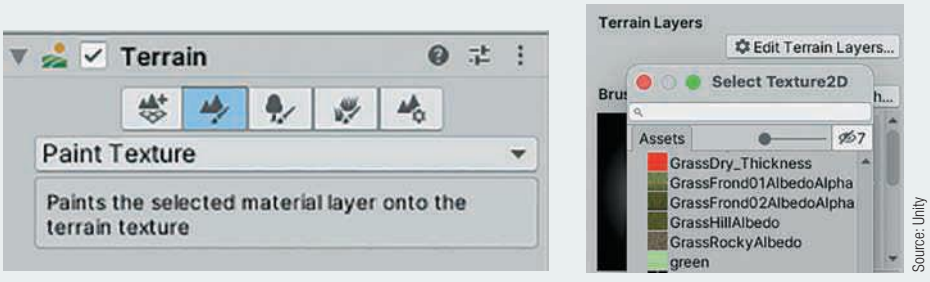


Figure 8.57 (Left) Adding a new texture from those provided in your project's Assets. Here select *Standard Assets* > *Environment* > *TerrainAssets* > *SurfaceTextures* by selecting *Edit Textures…* (Right) Make a texture selection for your first terrain layer from the panel after clicking *Create Layer…*

25 Click the *Edit Terrain Layers…* button in the Inspector and choose *Create Layer…* A *Select Texture 2D* selection panel appears (Figure 8.57 right).

26 Double-click on your selected texture to add it. We suggest *GrassHillAlbedo*. The first layer fills the whole terrain. Additional layers will instead allow you to 'paint' textures like paths and grass.

Your terrain will now be clothed in beautiful green grass (see Figure 8.58).

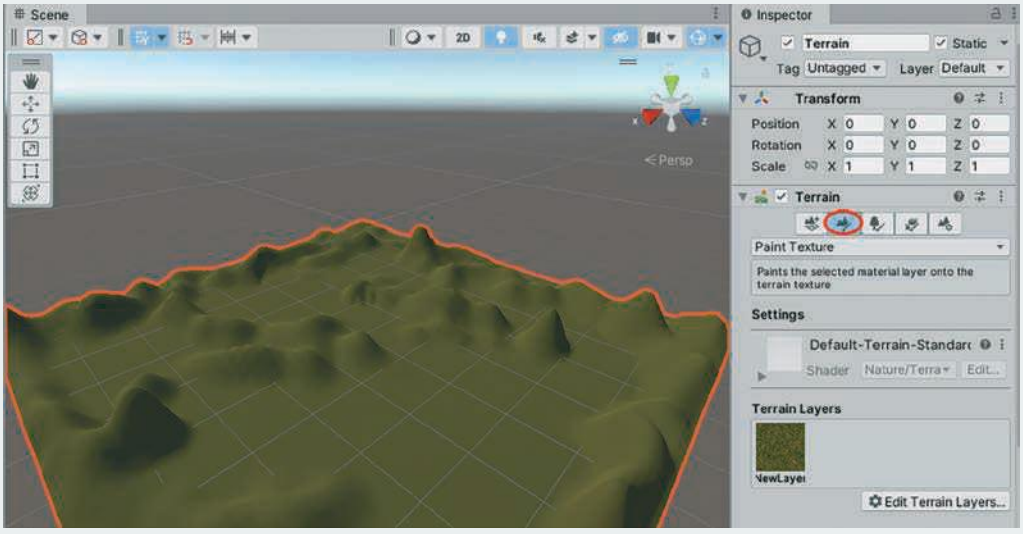


Figure 8.58 Terrain now covered with grass using one of the three loaded textures shown.

27 Click the *Edit Terrain Layers…* button again in the Inspector and this time choose *Add Layer…* A new selection panel appears.

Double-click on a sandy coloured texture to add it to your terrain layers panel. Now use the paint brush to paint on your terrain. You must select their thumbnail first – it is like the paint for the brush. Create paths and rocky patches (Figure 8.59).

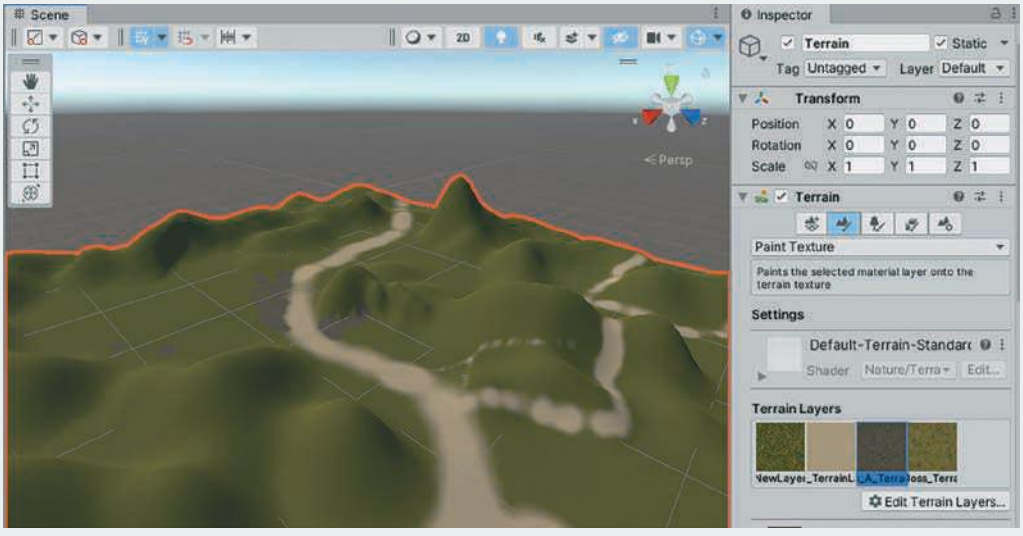


Figure 8.59 A new sandy texture has been added and paths drawn over the original terrain.

Many other free assets are available from Unity's Assets Store, which can be accessed from the Assets Store tab above your Scene view.

Add more environment assets: Trees

28 Select your Terrain again in the hierarchy and now select the third Terrain tool shown in Figure 8.60 (Paint Trees).

29 Click the *Edit Trees…* button in the terrain's Inspector, click *Add Tree* and from the selection panel beside the field for *Tree prefab*, click the small right triangle and select a tree. We suggest *Broadleaf Desktop*. Click Add at the bottom of the panel.

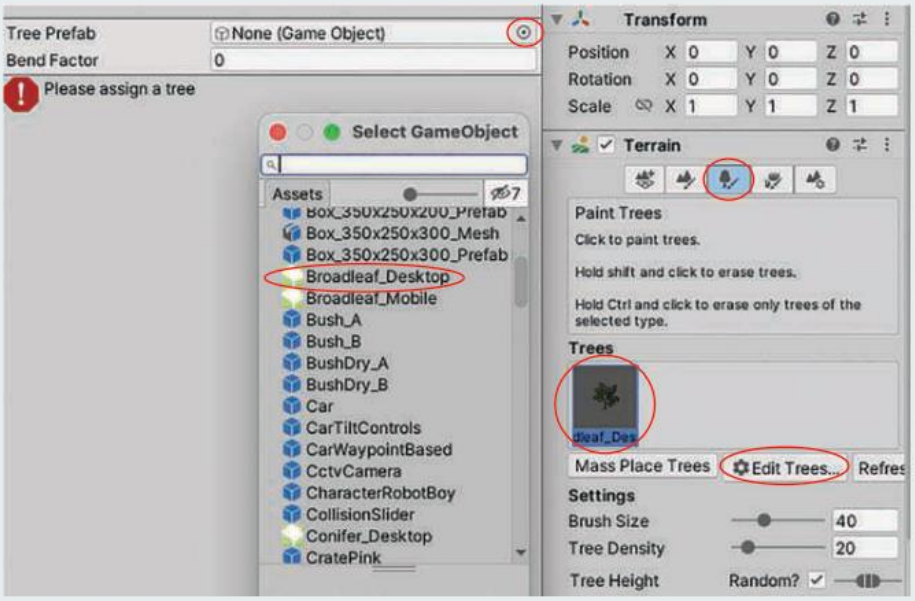


Figure 8.60 The third icon is the Paint Trees tool. Click the Edit Trees button in the Inspector, choose a Tree Prefab and use the panel that appears to select a tree object. It will now appear under Trees in the Inspector.

30 In Figure 8.61 we have chosen the broadleaf tree. Select the tree thumbnail and paint your terrain with trees! You can control how many, and even *Mass Place*, trees using the button seen below your tree image.

31 Trees can be removed by holding shift down while using this Paint Trees tool.

Save your work!

Add swaying grass

32 Select the fourth terrian icon *Paint Details*. Under *Edit Details* select *Choose Grass Texture* and select *GrassFrond01Albedo*. Adjust the brush parameters and paint grass in your scene that will sway in a (virtual!) breeze when you run the scene. If you can't zoom close enough, press F to focus.

Add more environment assets: rippling water

Add realistic rippling water to your scene!

Having reached this stage of your project, it is useful to know you can hold down alt/option and directly zoom in on a particular item in the scene.

33 Directly from your Assets panel, drag a water prefab from *Standard Assets* > *Environment* > *Water*> *Water4* > *Water4Advanced* (which is very realistic) to form a lake positioning it in an enclosed depression in the terrain created for your lake (Figure 8.61). At first you may need to lift the water prefab up or down and scale it to position it correctly. Notice as you scale it up its edges will contour around the curves of hills. You may need to add more mountains.

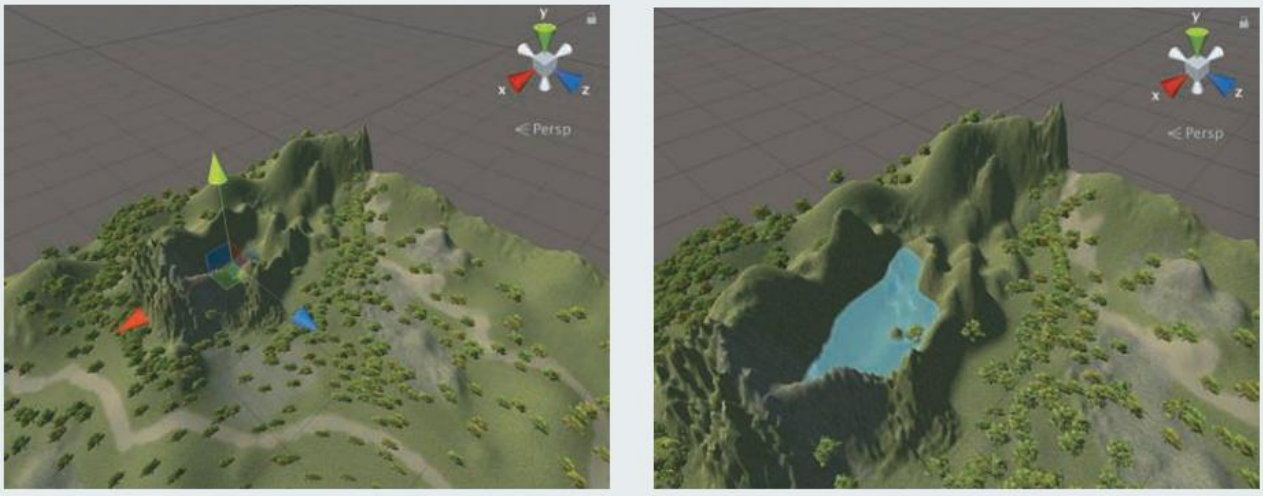


Figure 8.61 (Left) Water prefab before adjustment. (Right) Water prefab after lifting and scaling.

34 If you can, position the Main Camera so that it points towards the lake and its shoreline. Select this camera and press Play, then go to Game view to see how impressive your lake of water is.

35 Add a few cameras (*GameObject* > *Camera*) and point them at different places in your world using the usual transformation tools. Select which camera is used by turning all others off as you run your game.

36 We have provided a video in the Nelson MindTap resources for this project so you can follow the steps above as well as seeing the final result – an amazing lake of rippling water.

Unity project_ adding lake video

Save your work!

Add objects to your world

Now is the time to add some 'man-made' structures to your world. This is your chance to show your environmental sensitivity by retaining a natural environment while making your world liveable!

Add simple 3D objects

Unity is a sophisticated game engine but it is not designed to be used to create complex objects. It is designed instead to import such objects. However, it is useful for creating simple objects.

37 Add a cube to your scene: *GameObject* > *3D Object* > *Cube*. Position it at ground level. Remember you can zoom in on the cube by holding down alt/opt.

38 Add a sphere to your scene: *GameObject* > *3D Object* > *Sphere*

39 Position the Sphere on top of the cube. Drag the sphere on top of the cube in your Hierarchy. You will now see the Sphere name indented under Cube. This action has made the sphere *a child of* your cube. In other words, the sphere now belongs to the cube and they now form a group that moves together. Try resizing this – they will scale as one object.

40 Drag one of the materials you have created from a Materials folder in your Assets on to the object.

Next, create a new material by right clicking in the Assets panel and selecting *Create* > *Material*. Give it a name. Change its colour from white in the *Inspector* (select colour from outside ring then select its shade from the centre square) and drag it onto your cube.

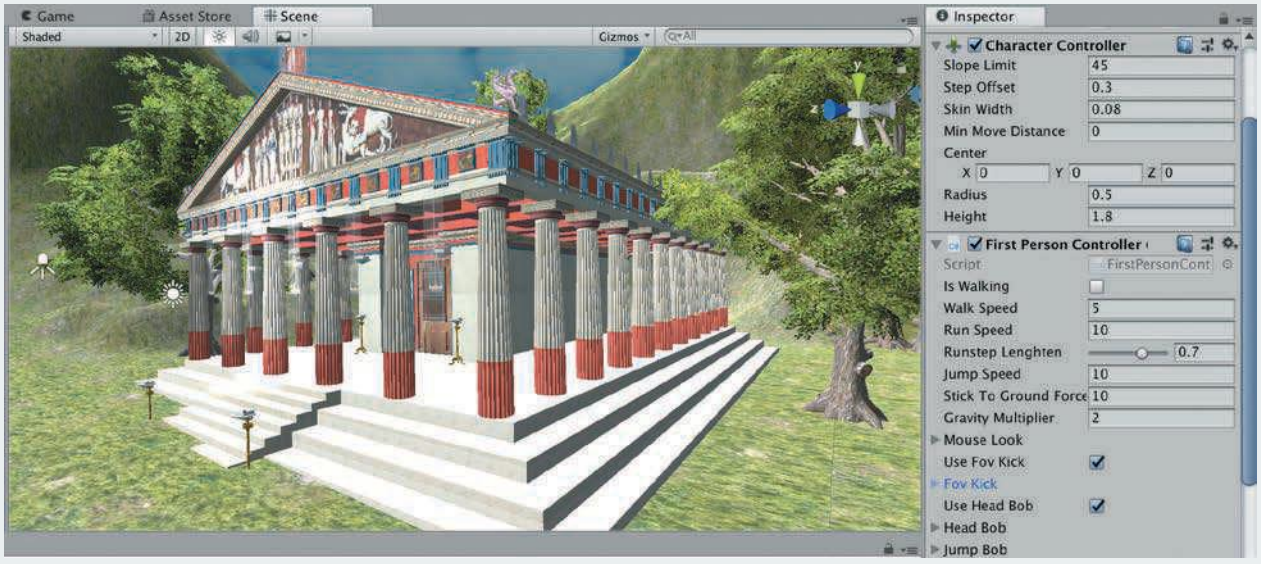
Save your work!

(Optional) Add complex 3D models using SketchUp and Blender

41 Create an object in *SketchUp* and import it into *Unity*. You will need to be able to use the free Schools version (sign in with your Google account) to be able to export it in .fbx format or own a paid version.

42 In *Unity* select *Assets* > *Import New Asset* and import your file. It will be added to your Assets folder. Select it in the Assets folder and drag it into your scene. Depending on its scale in *SketchUp*, you may need to scale it up to see it!

43 *Unity* can import 3D models produced by outside apps. *Blender* is a popular tool for creating models. Watch the Jurassic example designed in *Unity* by a student provided in the Nelson MindTap resources for this project. Here, the dinosaurs were created using *Blender*. In addition, you can import existing models from *SketchUp*'s 3D Warehouse (see Figure 8.62).



Source: Unity and 3D warehouse

Figure 8.62 Imported *SketchUp* model of a temple imported into our world, along with its Textures.

It gets real! – adding interactivity

It's time to add interactivity to our scene and enter the world we have created.

44 Unity's Standard Assets contain ready-made *Character Controllers* which are very easy to use, both a First Person Controller (FPC) and a Third Person Controller.

If you don't know these terms, use the web to research the difference.

45 *Important!* Delete or turn off (via the Inspector) *all* cameras you have in your game hierarchy to allow this character to become the default camera.

46 Look inside *Standard Assets* > *Characters* > *FirstPersonCharacter* and inspect the files that control behaviour.

47 Locate the *Prefabs* folder and drag the *FPSCharacter* prefab into your scene. Make sure it is above your terrain or it will fall into endless space!

48 Play the game using your mouse and arrow keys or the the W, A, S and D keys to navigate your scene. You will notice there are footstep sound effects and a head bob! Press spacebar to jump. Change these in the *Inspector* to suit. Speed up by holding the *Shift* key down. Experiment with changing parameters in the *Inspector* for this FPC (see Figure 8.63).

Save your work!

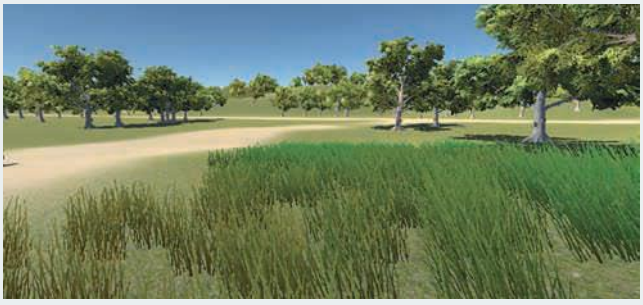


Figure 8.63 Your view as a first person character moving around in your virtual world.

Add lighting

Directional lights

These are *Unity*'s default lighting types. One will always be present in any new scene. Directional lights can be placed anywhere as their source is imagined as being very distant. They are like our Sun and as a result, the move tool does not affect these lights.

49 Directional lights can be used to simulate day and night by rotating towards or away from your *Unity* world and can simulate sunrises and sunsets by being pointed horizontally towards to your horizon. Double-click on your *Directional Light* in the *Hierarchy* and rotate it to see the effect on your world.

50 Change its properties in the *Inspector*, such as its colour and intensity.

Point lights

A point light is similar to a lightbulb – it shines out in all directions but has limited range.

51 Turn off the directional light by unchecking its box in its Inspector to explore the point light.

52 Add a point light (*GameObject* > *Light* > *Point Light*). Set its Range to 100 and its Intensity to 15. You will need to make sure it is above your terrain and close to it. If you have difficulties positioning it, set its Transform x,y,z coordinates in the Inspector to 0,0,0 then move it in the xz-plane to the centre of your terrain then try lifting it slowly above your terrain. Experiment with moving it around your scene (Figure 8.64).

53 Experiment with changing its properties in the Inspector.

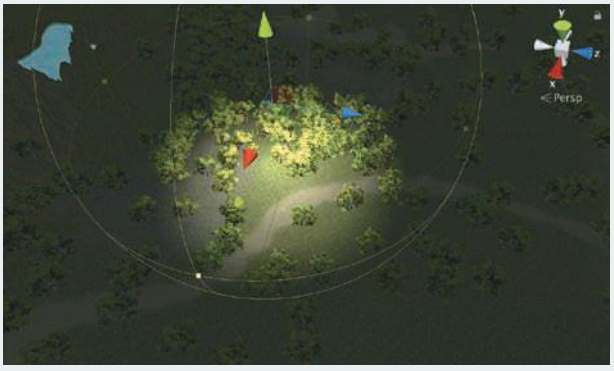


Figure 8.64 The point light in action with other lights turned off.

Spotlights

A spotlight is similar to a point light but is restricted to an angle. This produces a cone-shaped limited illuminated region and can produce dramatic effects. It is like a torchlight or headlights.

54 Add a spotlight and try moving it around your scene.

55 Experiment with changing its properties in the Inspector.

Add a skybox

A skybox is a collection of six images which you can imagine as a huge cube surrounding your world. The walls of this cube will be so distant that they act like distant clouds. We will use a free one available in the *Unity* Assets Store in a collection called *Classic Skybox*.

56 Search for the free *Classic Skybox* in the Assets Store (not in your Standard Assets folder) and download it. It will appear in your Assets. Remember you can simply drag folders into Assets, too. We selected folder 05 (Figure 8.65).

57 As with other textures, *Skybox* images are assigned to a material. Create a new material: *Assets* > *Create* > *Material*. It will appear in Assets. Do not yet drag it into your scene. Call it *MySky*.

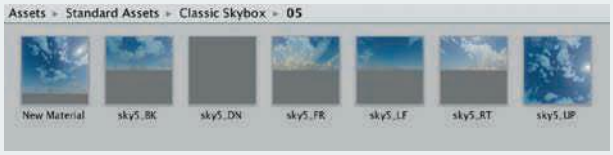


Figure 8.65 A screenshot of the *Unity* Assets Store with seven different sky files to choose from.

58 Select the new material in Assets and at the top of its Inspector select *Shader* > *Skybox* > *6-sided* (Figure 8.66).
Click the lock at the top of the Inspector to stop it changing.

Source: Unity

Figure 8.66 Selecting the *Skybox* 6-sided shader.

59 Now drag each of the six images from your selected (ours is Skybox *Assets* > *Classic Skybox* > *05* folder) in the Assets folder into slots in the shader. Match ours as shown in Figure 8.67.

The thumbnail shows the final result (and is interactive!). If these *Skybox* images do not match up correctly, swap *back* with *front*, and *left* with *right*.

Having completed this you can now drag the new material on to your scene into the empty space outside the terrain area.

Testing and evaluating

Publish as a standalone app

Save your work and thoroughly test it. Have other users test and comment. Update your documentation. When you have added all the features you want, it is time to publish your game as a standalone app for the computer platform you are using. Check *File* > *Build Settings…* and make sure you have selected the correct platform(s).

Figure 8.67 The six sides of the classic Skybox cube correctly assigned.

Select Build and Run, select a location for the app and press the Save button.

Be inspired!

We have provided an example on Nelson MindTap by a Year 12 student of a dinosaur Jurassic World project, which used this worksheet concept along with dinosaur animations created in the 3D *Blender* application and imported into *Unity*. Download, explore and be inspired by some of the ideas in the project!

Extend this project

Adding scripts for two great behaviours

Moving clouds

If you would like your clouds to move (who wouldn't?!) and are willing to add around 10 lines of prepared code to your project, it is easy to do so (Figure 8.68). We have provided the details in this project's Nelson MindTap resources.

Rollover text

Imagine if users could read information appearing when their mouse pointed at objects! Using this idea, you could create a checklist for users to complete information while they explored your world. This would make a useful feature in an educational app. We have provided the details in this project's Nelson MindTap resources.

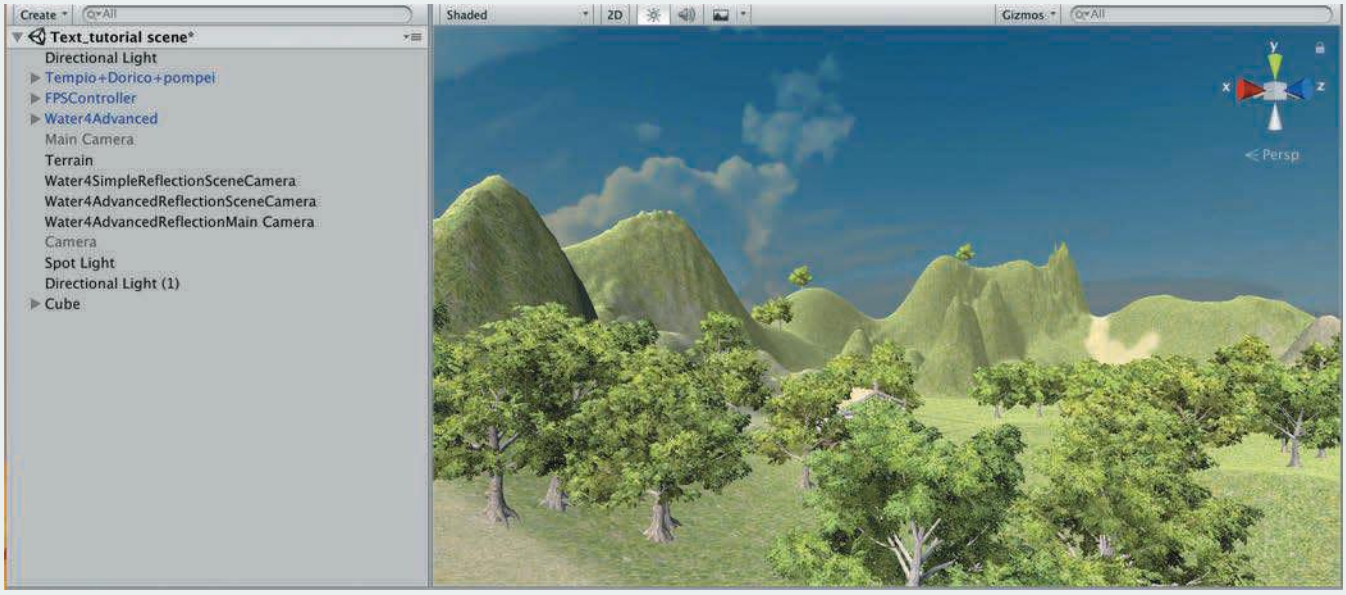


Figure 8.68 Our completed world, complete with moving clouds.

9 Analysing data using spreadsheet systems

FOCUS AREA: Analysing data

Chapter outcomes

In this chapter, you will learn:

- about the purpose and structure of spreadsheet systems, understanding how they are used in various industries for data management and analysis
- how to design formulas and use functions to perform calculations and manipulate data, gaining skills that can automate complex tasks within a spreadsheet
- about cell referencing techniques, including absolute and relative referencing, as well as the use of named ranges for more efficient and understandable spreadsheet design
- about data visualisation techniques, such as charts and dashboards, and how they can help convey complex information in an easily digestible format
- the importance of data validation, data types and data cleaning in improving the accuracy and reliability of your analyses and visualisations
- to use advanced functionalities such as lookup tables, goal seek analysis, pivot tables and query functions, which offer deeper insights into your data and enhance your analytical capabilities
- how application programming interfaces (APIs) and system integration can extend the capabilities of spreadsheet systems, enabling automated data collection and sharing between different software applications.

Key terms

absolute referencing
cell
chart
csv
data dashboard
data validation
export
fill
filter
formula
function
goal seek
infographic
parameter
pivot table
relative referencing
spreadsheet
what if

IN ACTION

ABS data analysis

The Australian Bureau of Statistics (ABS) describes itself as 'Australia's national statistical agency, providing trusted official statistics on a wide range of economic, social, population and environmental matters of importance to Australia'. The ABS collects data from a range of sources, including schools, and performs numerous analyses in order to produce this information. The ABS also makes a lot of the data available online for people to download and analyse for themselves.

One very large set of data collected each year is about school students and includes variables such as gender, whether they are Aboriginal or Torres Strait Islander, age, and so on. From an analysis of this data set, the ABS releases an annual report with key statistics. The 2022 report included the following:

- there were 4 042 512 students enrolled in 9614 schools
- 64.5% of students were enrolled in government schools, 19.7% in Catholic schools and 15.9% in independent schools
- the Grade 7 to 12 Apparent Retention Rate (ARR) for Australia was 80.5%
- the student to teaching staff ratio for all schools was 13.1.

The ABS also distributed media releases such as *'Fewer overseas school students as COVID-19 hits'*. This report summarised some of the key statistics relating to the drop in enrollment growth rate due to a fall in the number of overeas students with COVID restrictions in the previous years. A drop in ARR was also noted.

There is also summary data provided that allows the general public to perform their own analysis. In 2017, the ABS provided data on the retention of Aboriginal and Torres Strait Islander students in schools up to Year 12 over time. To access the more recent equivalent data, the public can download the full data set from the latest release and use filters in a spreadsheet software to focus on the data of interest.

The spreadsheet software can then be used to generate a graph to show a visual representation of the data (Figure 9.1).

The initial upward trend of both series clearly shows that there has been success in keeping more students at school until Year 12. However, a trend down in recent years, possibly related to COVID, could be concerning and should to be addressed. There is also still a large gap between Aboriginal and Torres Strait Islander students and other students (Table 9.1).

Table 9.1 Summary data showing the retention of Aboriginal and Torres Strait Islander students and non-Aboriginal and Torres Strait Islander students in schools up to Year 12 from 2011 to 2022.

Year	Non-Aboriginal and Torres Strait Islander students ARR (%)	Aboriginal and Torres Strait Islander ARR (%)
2011	80.7	48.7
2012	81.3	51.1
2013	82.9	55.1
2014	84.8	59.4
2015	85.2	59.4
2016	85.5	59.8
2017	86	62.4
2018	85.8	60.9
2019	85.4	58.7
2020	85	59.6
2021	84.5	59
2022	82	56.9

Source: ABS, *Schools, Australia, 2022*, cat. no. 4221.0

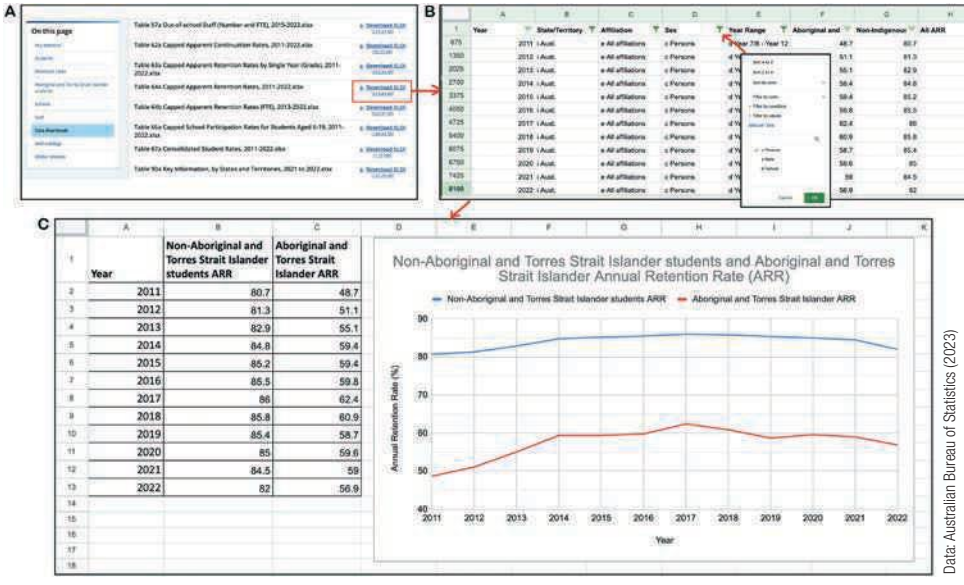


Figure 9.1 Data on ARR downloaded from ABS as a comma separated value (csv) file (A) then filtered in spreadsheet software to only display data on all of Australia, all schools affiliations, all persons and all high school year range (B). Selected data then copied into a new sheet and graphed using a line chart (C).

Computational thinking

Questions

1. Identify the data that would have been collected in order to generate those statistics.
2. For each of the data identified above, what would be its data type?
3. Why is it important for the Australian Government to collect data on schools over time?
4. The graph in Figure 9.1 has the y axis starting at 40 rather than 0. Why has this been done? What are the issues with presenting data in this way?
5. Suggest any other information about schools that would be useful to calculate.
6. Visit the ABS website and explore another report based on collected environmental data. List some of the statistics presented.

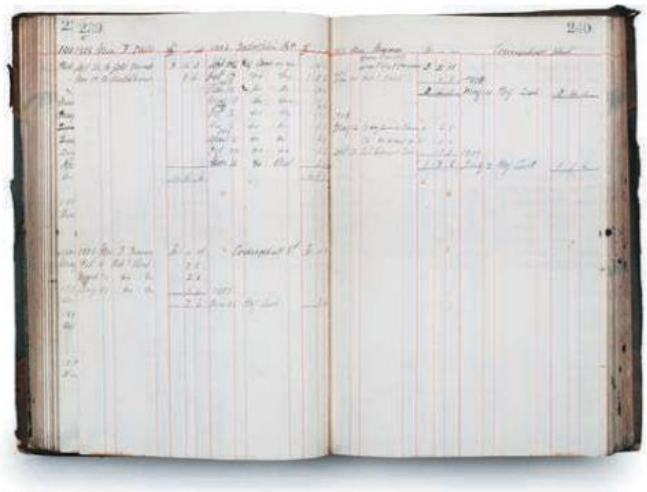
Weblink
ABS

9.1 Introduction to spreadsheet systems

A little bit of history

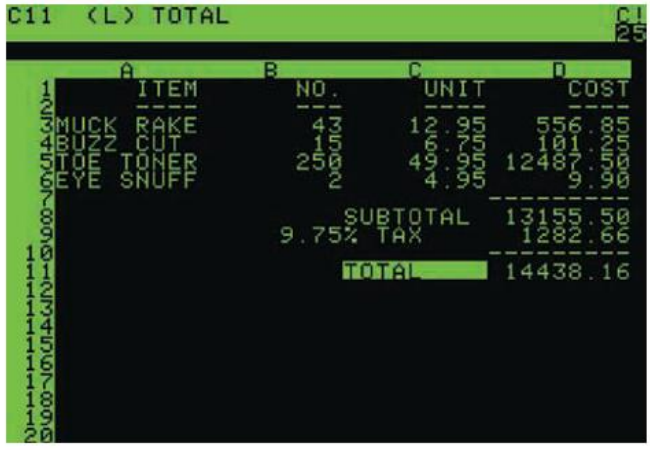
In the late 1970s, Dan Bricklin was attending a lecture at the Harvard Business School. The professor had recreated a table of rows and columns on the board like the ones drawn with accounting paper (Figure 9.2). However, rather than just adding the values for a particular calculation, they demonstrated a financial model with **formulas** and data written in the **cells**.

Bricklin noticed that when there was an error in the formula or change in data the professor erased and rewrote many entries in sequential cells of the table. This inspired the idea of an 'electronic **spreadsheet**' that could repeat this process digitally. Soon after, Bricklin and Bob Frankston developed the first spreadsheet software program called VisiCalc in only two months (Figure 9.3). VisiCalc's functionality, along with its user-friendly design and excellent documentation, revolutionised the way businesses did their finances. It was a huge success and many people bought new computers just to run VisiCalc.



Alamy Stock Photo/John Gollop

Figure 9.2 Accounting ledger for a restaurant. Each page is divided into rows and columns with raw data added and the results of the calculations, such as a sum of all the values.



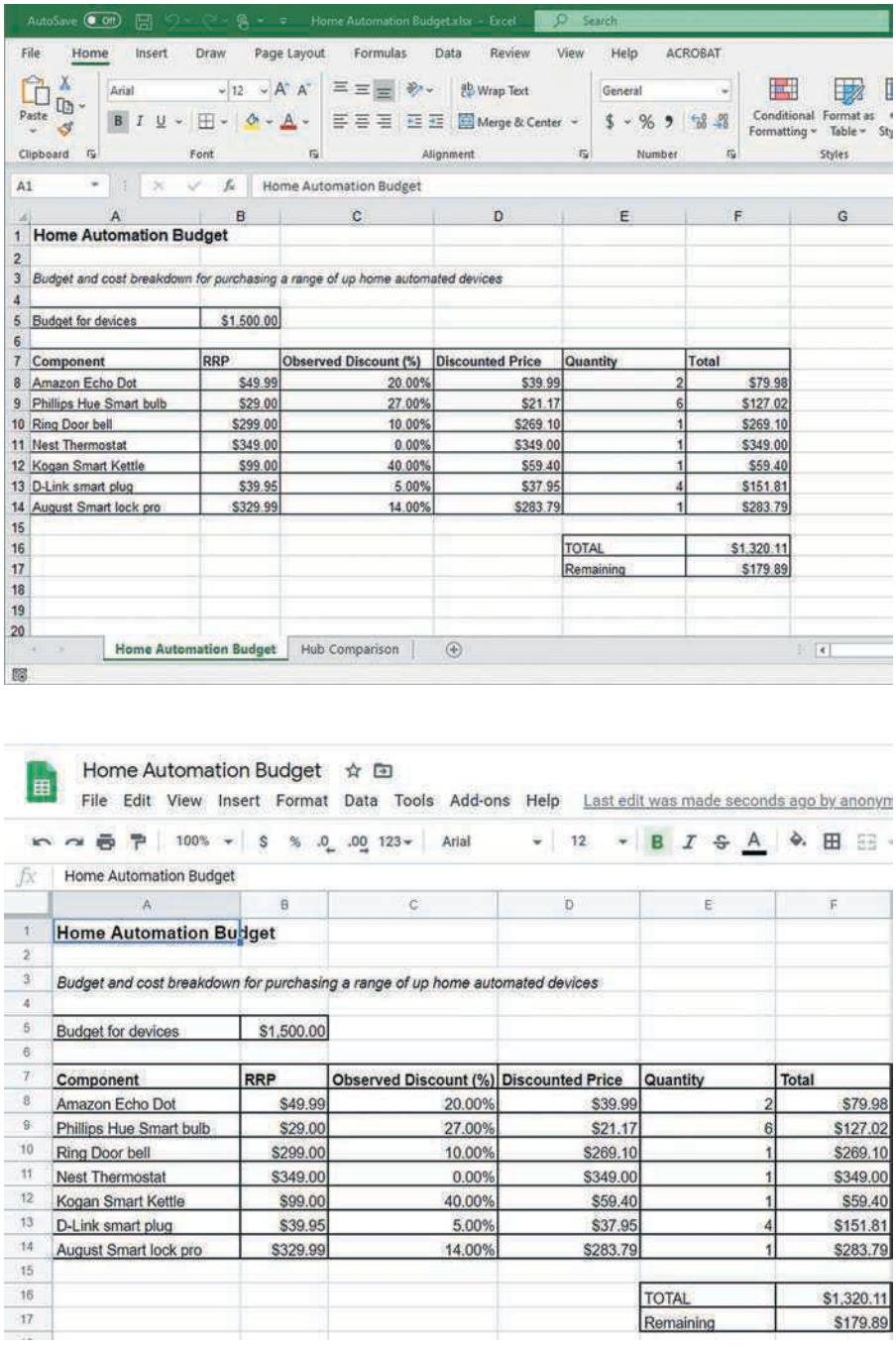
Source: apple2history.org

Figure 9.3 Screenshot from an example spreadsheet in VisiCalc.

Spreadsheet software used today

Fast-forward more than 40 years to today, and spreadsheet software has evolved from the simple table and calculator **functions** in VisiCalc to an extensive range of powerful and customisable analysis tools able to process huge data sets. The spreadsheet software an individual or organisation may choose for their data processing will depend on the platform they will be working on (web, desktop or mobile), the type of data processing they intend to do, their budget for purchasing or subscribing to the software and whether they are collaborating with others on a project.

Smartsheet is spreadsheet software with the main purpose of providing project management functionality, while Quip is spreadsheet software that also includes collaborative document editing, a presentation creator and chat features. LibreOffice Calc is also a very popular general-purpose spreadsheet software, as it is free and open-source. However, the two main players in the spreadsheet game are Microsoft Excel and Google Sheets (Figure 9.4). After we briefly introduce these two, you will realise they are surprisingly similar, and you can choose to use either for the data analysis activities you will perform in this unit.



	A	B	C	D	E	F
1	**Home Automation Budget**					
2						
3	*Budget and cost breakdown for purchasing a range of up home automated devices*					
4						
5	Budget for devices	$1,500.00				
6						
7	**Component**	**RRP**	**Observed Discount (%)**	**Discounted Price**	**Quantity**	**Total**
8	Amazon Echo Dot	$49.99	20.00%	$39.99	2	$79.98
9	Phillips Hue Smart bulb	$29.00	27.00%	$21.17	6	$127.02
10	Ring Door bell	$299.00	10.00%	$269.10	1	$269.10
11	Nest Thermostat	$349.00	0.00%	$349.00	1	$349.00
12	Kogan Smart Kettle	$99.00	40.00%	$59.40	1	$59.40
13	D-Link smart plug	$39.95	5.00%	$37.95	4	$151.81
14	August Smart lock pro	$329.99	14.00%	$283.79	1	$283.79
15						
16					TOTAL	$1,320.11
17					Remaining	$179.89
18						
19						
20						

	A	B	C	D	E	F
1	**Home Automation Budget**					
2						
3	*Budget and cost breakdown for purchasing a range of up home automated devices*					
4						
5	Budget for devices	$1,500.00				
6						
7	**Component**	**RRP**	**Observed Discount (%)**	**Discounted Price**	**Quantity**	**Total**
8	Amazon Echo Dot	$49.99	20.00%	$39.99	2	$79.98
9	Phillips Hue Smart bulb	$29.00	27.00%	$21.17	6	$127.02
10	Ring Door bell	$299.00	10.00%	$269.10	1	$269.10
11	Nest Thermostat	$349.00	0.00%	$349.00	1	$349.00
12	Kogan Smart Kettle	$99.00	40.00%	$59.40	1	$59.40
13	D-Link smart plug	$39.95	5.00%	$37.95	4	$151.81
14	August Smart lock pro	$329.99	14.00%	$283.79	1	$283.79
15						
16					TOTAL	$1,320.11
17					Remaining	$179.89

Figure 9.4 Microsoft Excel (a) and Google Sheets (b) are the market leaders in spreadsheet software.

Microsoft Excel

For most people, the word 'Excel' simply means spreadsheet. In fact, many people would never have heard of any other spreadsheet software. Most businesses use Microsoft Excel, and for this reason many workplaces expect employees to know how to use at least the basic features. Excel has this reputation for a reason. It has been around since the late 1980s and is the most powerful spreadsheet software available, able to handle enormous data sets and process data very efficiently. It also includes countless features, many of which Microsoft developed, including conditional formatting and **pivot tables**, which we will cover later in this chapter. Excel is mainly a desktop software application, available for Windows and Mac. However, with the rise of web-based applications and collaboration, it also moved into the web-based platform, now competing with the slightly less powerful, but free and highly integrated Google Sheets software.

Google Sheets

Google Sheets is a very popular spreadsheet software choice, particularly for individuals and organisations who do not need some of the advanced features of Microsoft Excel. Apart from being free, Google Sheets collaboration functionality and clean integration with other Google applications, including Google Forms and Google Analytics, are what initially set it apart from Microsoft Excel. Google Sheets looks and acts much like Excel in most ways people use spreadsheet software, making it easy for those considering the transition. It also includes some of its own features, such as a built-in function that uses the Google Translate algorithm. Google has gone one step further to encourage people to use its spreadsheet software, now allowing users to upload and edit Microsoft Excel files using Google Sheets, without any need to convert the file. Google Sheets also has the ability to add functionality not included in the software through third-party applications called 'add-ons'.

INFOBIT

A Microsoft Excel spreadsheet can have a maximum of 1 048 576 rows and 16 384 columns, making a total of 17 179 869 184 cells. If you decided to add data to each of these cells manually and it took you one second to fill a cell, the spreadsheet would take 545 years to complete. A Google Sheets spreadsheet has a limit of 5 000 000 cells, so would only take 58 days to fill manually.

Industry uses of spreadsheets

The most common uses of spreadsheet software are to store and organise structured data, to create budgets and to produce **charts** and graphs. Doing this requires very little processing but can still prove highly useful to many businesses. Beyond these basic uses, businesses mainly use spreadsheet software to do invoicing, manage payroll data, do forecast analysis (for example, for sales and revenue) and perform statistical analysis for reporting. Spreadsheet software is also used extensively in research. Although some researchers may rely on more specialised software to analyse big data sets, such as R or a purpose-built application developed using a programming language, many people find the accessibility and flexibility of spreadsheet software provides all the functionality they need. In this chapter, we will be performing data analyses in spreadsheet software. The list of spreadsheet software uses is endless, with people customising the software to do new functions all the time.

INFOBIT

In 2018, 'C bel' built a three-dimensional (3D) game engine in Microsoft Excel that relies only on formulas. It includes functionality most people would think could only be implemented in advanced software like Unity or UnReal, including a maze map that is infinitely generated, real-time ray tracer and basic illumination rendering, occlusion calculation and a natural displacement engine. Although this is not the intended use of spreadsheet software, it was developed to show just how powerful these applications can be. To investigate this further, search online for MS Excel 3D engine.

SKILL BUILDER

Analysis of video game sales data – Part 1

Introduction

Data is numbers and other values that by themselves have no meaning. When we process and analyse data, we can turn it into information, making it useful. For example, if we perform calculations and graph data on the weather, we can make forecasts and predictions.

In this series of Skill builders, you will be taking a large data set and performing processes on it using Microsoft Excel or Google Sheets to find out key statistics and to create informative graphs. You will then be communicating your information in the form of an infographic and data dashboard.

Spreadsheet processing will be demonstrated on a data set of video game sales from 1970 to January 2023 using Google Sheets. You can download this data set from Nelson MindTap or you may work with your own data set.

You will notice when you download the file that the file format is a '**csv**' or comma separated value file. This is a generic file format used for spreadsheets and can be imported into nearly all spreadsheet software applications. To analyse the data, you may also use Microsoft Excel, if you prefer, as most processes are the same or very similar.

A simple evaluation of the data set can be done by considering the three Vs: volume, velocity and variety.

- Volume: This large data set contains 60 000 entities (rows) with 14 attributes/fields (columns). This means there could be 60 000 x 14 = 840 000 possible pieces of data. Excel has a limitation of just over 1 million rows and 16 000 columns, so can easily handle this data set. There are, however, many holes in the data set, mostly representing unknown values.
- Velocity: This data set has been captured from an online source that has new entities added all the time, when new video games are released and when previously unrecorded games from the past are added. Video game sales are also always changing, so this data is also being updated in the entities records.
- Variety: This is a highly structured data set with all of the attributes being quantitative. This makes it very easy to perform calculations on the data using standard spreadsheet functions and formulas.

Skill builder resource
Video game sales data set

Masterclass
Video game sales data

INFOBIT

Kaggle is a community of data-loving individuals who contribute data sets that are mostly free for you to download and use for your interest.

Weblink
Kaggle

ACTIVITIES

Identify

1 What did Dan Bricklin observe that led to him seeing the potential in using computing power to solve accounting problems?
2 How is the data in a spreadsheet organised?
3 Identify six different applications of spreadsheet software.

Analyse

4 Looking at the accounting ledger in Figure 9.2, what mathematical functions were being performed?
5 Suggest reasons why a volunteer team running a social football competition might opt to use Google Sheets as opposed to Microsoft Excel to manage their schedule and results.

Extend

6 Visit the Kaggle website and find a data set about movies that includes just one file and at least 10 000 records. Hypothesise how the data was collected and list the column headings (variables).

9.2 Organising data in spreadsheet systems

How the raw data is organised and viewed in the spreadsheet software is a key feature of making the analysis processes accurate and efficient. Understanding the layout of a spreadsheet, how data types are applied and displayed, the benefit of data validation and the use of filters is therefore what we will look at first as we learn about spreadsheets.

Layout of a spreadsheet

The layout of a spreadsheet has not changed since it was first a replication of an accounting ledger. A single sheet is divided into *columns* and *rows* with each section formed called a *cell*. Columns are identified by a letter (or letters) and rows are identified by numbers. A cell is then simply identified as the combination of the column and row that it aligns with. Figure 9.5 demonstrates the main components of a spreadsheet. This spreadsheet is laid out across in columns A to F and rows 1 to 17. For example, the location of the data that is the RRP of the Amazon Echo Dot ($49.99) is cell B8.

Data types

In order for the spreadsheet algorithms to accurately process the data, it needs to define the type of data in each cell. Data types are introduced in Chapters 3 and 4. In spreadsheet systems, the same concept largely applies. In the spreadsheet in Figure 9.5, the data types represented include text (Component), currency (RRP, Discounted Price and Total), percentage (Observed Discount) and number (Quantity) (Figure 9.6). Depending on your spreadsheet, you may need to use other data types such as date or time. Your spreadsheet software will mostly automate the definition of these data types based on the initial data added. However, this can be changed if needed – for example, defining a number into a currency value.

INFOBIT

The earliest date allowed in Microsoft Excel is 1 January 1900. If you enter a date before this, Excel will not treat it as a date data type, meaning you cannot perform date-related formulas and functions on it. What can users do if they need to store dates before this? Check if Google Sheets also has a lower limit for the value of a date.

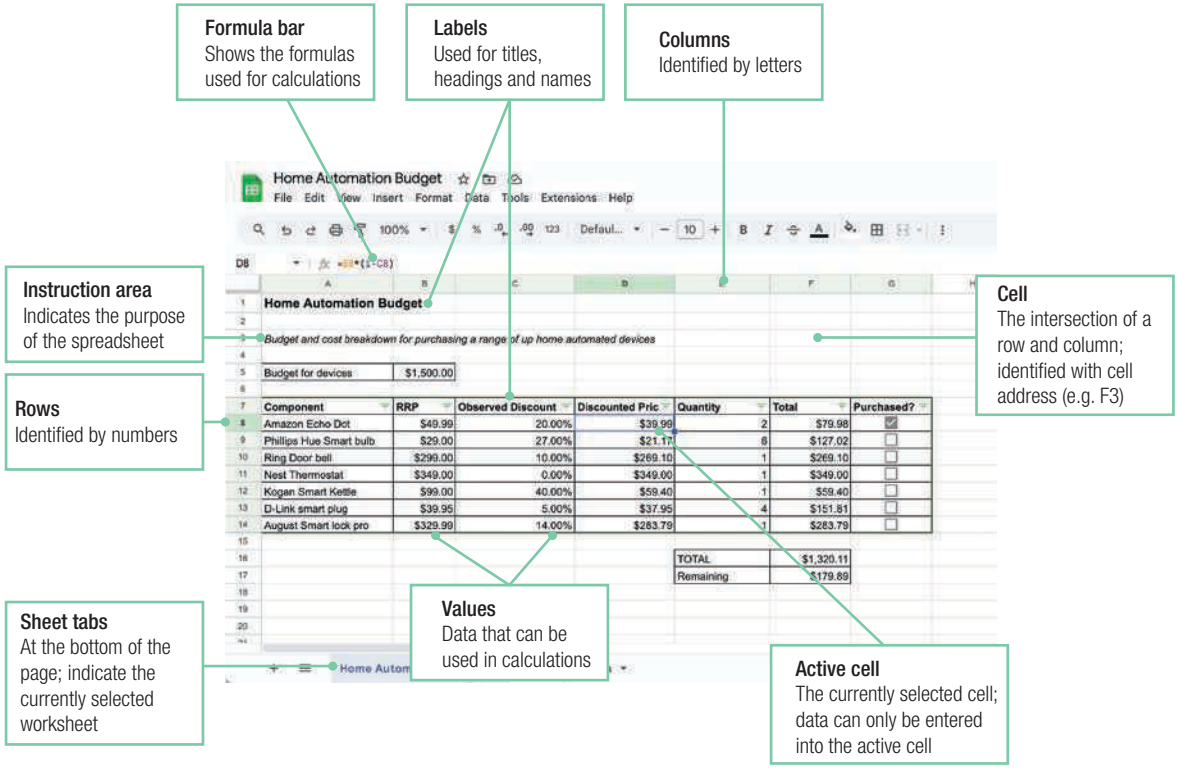


Figure 9.5 The basic elements of a modern spreadsheet.

9780170442787

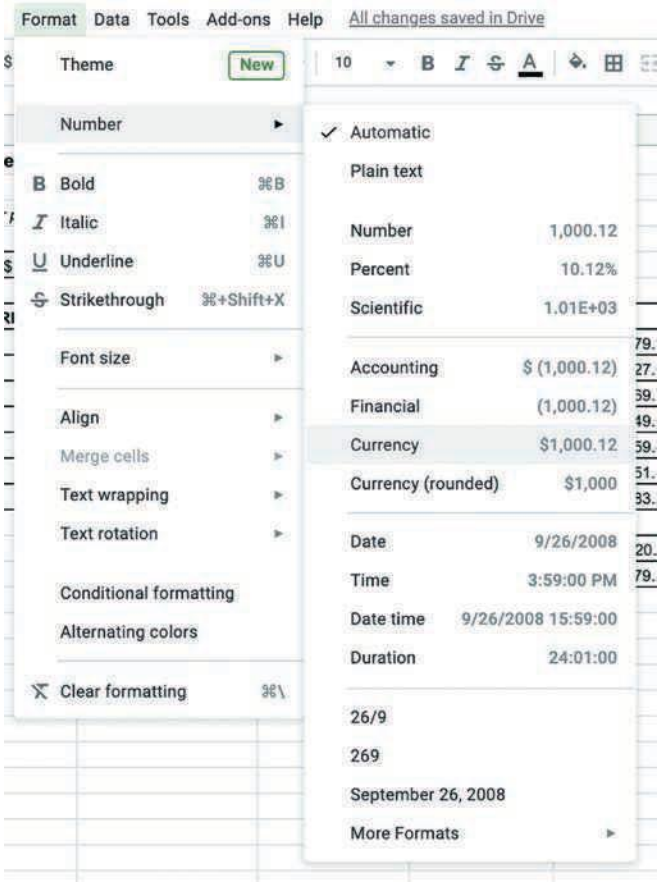


Figure 9.6 Data types for values in a spreadsheet can be defined if the data is not initially interpreted correctly.

Data validation

The spreadsheet is only as useful as the data it contains. If data has been incorrectly added or is not kept up to date, the resulting calculations may be inaccurate. When setting up a spreadsheet, it is useful to put in some **data validation** functionality to ensure only data that makes sense is added, reducing the chance of incorrect data being included (Figure 9.7). In the spreadsheet in Figure 9.5, we can define valid ranges for some of the data. For example, a discount can only be a value between 0 and 100, so any other input would be rejected. You could also define some logical ranges, such as quantity should be between 0 and 20, as it would be very unlikely you would need more than 20 of any of these items in your home. Rather than rejecting these values, you could have the system display a warning message. Finally, you could use a control structure such as a checkbox or drop-down menu so only valid data could be added, like in the 'Purchased?' column (G) of the budget table (see Figure 9.12 in the next section).

Sorting and filters

A common final step when setting up a spreadsheet is to define a **filter**. This will allow you to easily sort and filter data from a section of your spreadsheet. Although sorting and filtering may provide minimal functionality for small data sets like the home automation budget, it is highly useful when dealing with big data sets such as the video game sales data.

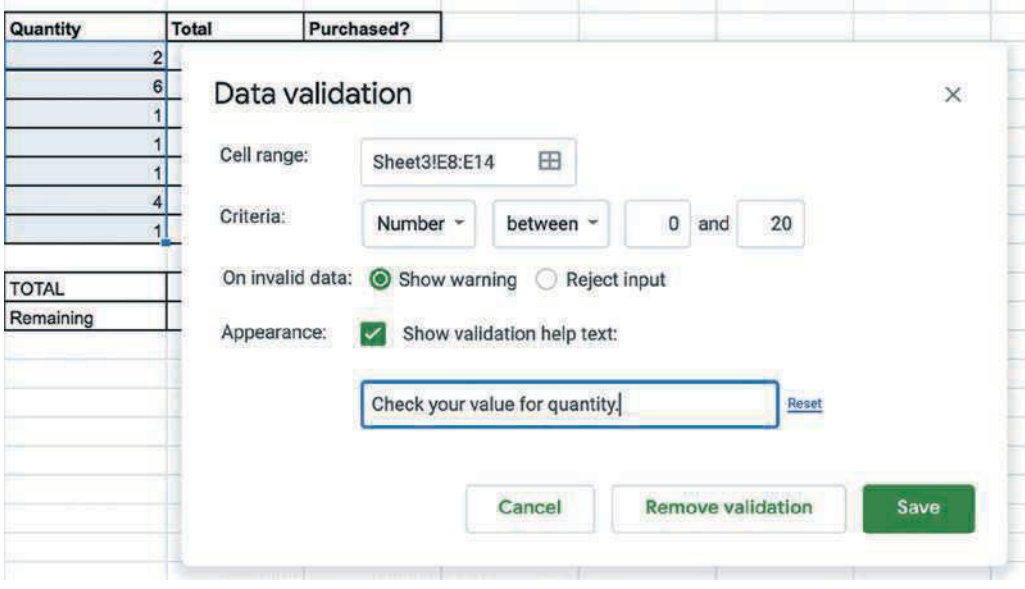


Figure 9.7 Data validation is a useful feature to include to minimise the chance of inaccurate data being entered into the system.

SKILL BUILDER

Analysis of video game sales data – Part 2

Setting up your spreadsheet

This data set contains the sales and scores of over 60 000 video games since 1970. Each row in the data set we can call a video game entity. This is too much data for our brains to process, but a computer can do even the most complex calculations on these numbers almost instantly. We just need to give it the instructions on how to do the calculations.

First, we need to examine the data in this data set so we can identify and define it for ourselves (Figure 9.8).

Note: The data set you will examine will likely include more up-to-date records to the ones represented in these screenshots so the values and calculations you observe may be slightly different.

1 Open the data set and observe the following column headings. These are the 'fields' or 'attributes' in this data set.

- Name – Name of the game
- Platform – Platform of the game (e.g. PC, PS4, Wii)
- Year – Year of release of the game
- Genre – Genre of the game
- Publisher – Publisher of the game
- Developer – Developer of the game
- Critic Score – Critic score of the game from 10
- User Score – Users score the game from 10s
- NA_Sales – Sales in North America (in millions)
- PAL_Sales – Sales in Europe (in millions)
- JP_Sales – Sales in Japan (in millions)
- Other_Sales – Sales in the rest of the world (in millions)
- Global_Sales – Total worldwide sales (in millions)
- Total shipped – Total shipped copies (in millions)

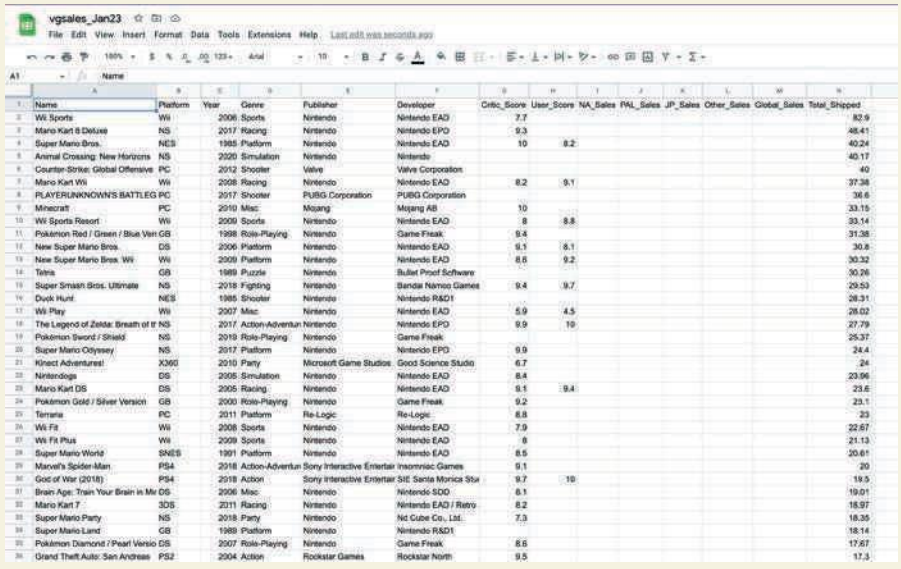


Figure 9.8 The downloaded csv file will open directly in spreadsheet software, with data types automatically determined using the values.

2 When you scroll up and down in such a large data set, you may find it hard to keep track of which data is in which field. A quick view setting can help us. Click on the '1' row heading and go View > Freeze > 1 row. This will keep the first row of the data always at the top of the view (Figure 9.9).

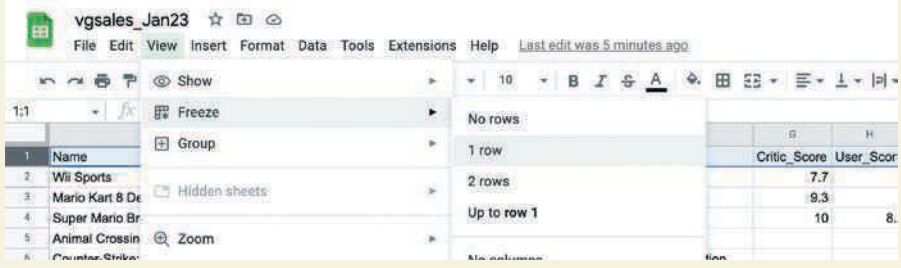


Figure 9.9 Freezing the first row means the variable headings will always stay at the top of the window.

As you inspected the data set, you will have noticed many cells are empty. The reality of many big data sets is that there will be missing data or even data that seems incorrect. One of the first processes you may need to do is clean the data so that your spreadsheet system can process it as intended (Figure 9.10).

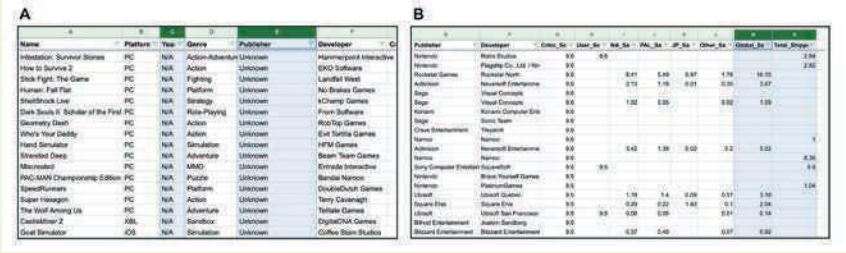


Figure 9.10 Data from external sources may have issues, such as missing values for example the 'N/A' and 'unknown' values for year and publisher (A) or completely blank cells for many of the score, total_shipped and sales data points (B). This may prevent effective analysis. After inspecting the data, you may need to 'clean' it. One way this data set can be 'cleaned' is based on the observation each game record either has a value for total_shipped or global_sales (B). These columns can be combined based on the assumption they represent roughly the same measure.

3 To help us see which data is missing, we will create a *filter* (Figure 9.11).
   a Click the column A heading and drag to select all headings up to column N. Then go to Data > Create a Filter. If we sort by Year in ascending order, we will observe many games do not have a year associated with them. To clean this up, we could manually find the year for each game through online research. This would take a very long time, so we may decide this additional data is not worth including. To not include this data, filter by blank dates then select and delete these rows.
   b One small change we can quickly do to this data set to help with our processing is to assume all the missing global sales values are the collected total shipped values. Filter the data based on Total Shipped. Uncheck 'Blank' values. Select all the visible cells containing Total Shipped and cut and paste them into the Global Sales column. Refilter the Total Shipped to include blanks, then delete the Total Shipped column.

4 a Perform some other sorts and filters on the data in other fields to get a better understanding of the data you are working with. For example, Sort Critic Score Z > A (Descending) to determine which game(s) got the highest rating from critics, or filter the data set to only show platform (genre) games.
   b Experiment with sorting and filtering based on various other fields. Write down 3–5 pieces of information you gained from this first analysis of the data.

Figure 9.11 Sorting and filtering large data sets can quickly show you some data points or key information.

ACTIVITIES

Identify

1 Open a spreadsheet application and list all the different data types it can process.
2 Define data validation and describe one way it can be applied in a spreadsheet system.
3 If you were to sort the data in the spreadsheet in Figure 9.5 in descending order based on RRP, which would be the first component in the table (row 8)?

Analyse

4 What are the risks with sorting one column in a table of data using the sort function rather than using a filter?

Extend

5 Design a spreadsheet table for data entry of player details for registration into a football club. Add validation to your identified fields.

9.3 Formulas and functions

Computational thinking

The power of spreadsheets comes from the user-defined formula and the use of built-in functions. In Figure 9.5, you saw that spreadsheets have cells that contain raw values while other cells contain formulas that perform calculations using those raw values (or calculated values) and display the result. In this section, we will introduce some key principles about defining formulas and outline how to use some common functions in these.

Formula and cell referencing

A formula is a calculation. All formulas begin with an equals sign (=), indicating to the spreadsheet software that what follows should be processed as a calculation and not simply displayed as a raw value. A simple formula could involve the addition (+), subtraction (–), division (/) or multiplication (*) of data. This data could be typed directly into the formula, or more commonly, be referenced from another cell or cells.

For example, in Figure 9.12, the discounted price in cell D8 is calculated using the formula `=B8*(1-C8)`, while the total for each component in cell F8 is calculated using the formula `=D8*E8`. One benefit of using cell referencing rather than copying the values into the formula is that, if we need to change the value – say we found a better discount – we would only need to change the raw value and the formula would perform its calculation using that.

Another benefit of cell referencing comes from the magic of the **fill** tool. This allows a user to automatically copy the formula from one cell into multiple other cells, with the cell referencing adjusted accordingly. So, rather than having to write in cell D9 the formula `=B9(1-C9)`, we simply *fill down* and the formula above is copied and adjusted. Cell referencing that changes to match changes in the formula cell location is called **relative referencing**.

Sometimes we don't want a cell reference to change as we fill. For example, if we were to calculate the percentage of the budget consumed by each item, the denominator in the percentage calculation, the total budget in cell B5, stays the same for each calculation. If we were to use relative referencing to B5, the formula in cell G8 `=F8/B5%` would give the correct result of 5.33 per cent. However, after we filled down the formula for the remaining components, the formula in cell G11 would be `=F11/B8`, with a result of almost 700 per cent, clearly incorrect.

Other results would be displaying errors as we will have instructed the software to divide a number by a value of text data type. In order to stop this auto adjustment from happening, we need to use **absolute referencing**. This is achieved by placing a '$' in front of the column letter if we want that to remain fixed and/or in front of the row number if we need that to remain fixed. Our formula in G8 would become `=F8/B$5%` and after the fill down we would observe the formula in cell G11 would be `=F11/B$5%`.

	A	B	C	D	E	F	G	H	I	J
1	**Home Automation Budget**									
2										
3	*Budget and cost breakdown for purchasing a range of up home automated devices*									
4										
5	Budget for devices	$1,500.00								
6										
7	**Component**	**RRP**	**Observed Discount**	**Discounted Pric**	**Quantity**	**Total**	**% of Budget**	**Purchased?**	**Discount?**	**Amount Paid**
8	Amazon Echo Dot	$49.99	20.00%	$39.99	2	$79.98	5.33%	☑	☐	$79.98
9	Phillips Hue Smart bulb	$29.00	27.00%	$21.17	6	$127.02	8.47%	☐	☐	
10	Ring Door bell	$299.00	10.00%	$269.10	1	$269.10	17.94%	☐	☐	
11	Nest Thermostat	$349.00	0.00%	$349.00	1	$349.00	23.27%	☑	☐	$349.00
12	Kogan Smart Kettle	$99.00	40.00%	$59.40	1	$59.40	3.96%	☐	☐	
13	D-Link smart plug	$39.95	5.00%	$37.95	4	$151.81	10.12%	☐	☐	
14	August Smart lock pro	$329.99	14.00%	$283.79	1	$283.79	18.92%	☐	☐	
15										
16					TOTAL	$1,320.11				
17					Remaining	$179.89				
18										

=B8*(1-C8)
=D8*E8
=F8/B$5%
=IF(H8=TRUE,F8,)
=SUM(F8:F14)
=B5-F16

Figure 9.12 Formula and functions used in an expanded version of the home automation budget.

Common functions

We may want to perform more complicated calculations. Spreadsheet software has many built-in functions that we can use in our formulas. To use functions, we call their specific name and complete the required **parameters**.

SUM and COUNT

Two of the most frequently used functions are SUM and COUNT. If we would like to sum a range of values rather than creating a formula that adds them all using the addition operator (+), we can use the function SUM with the range of cells provided as the parameter. For example, to calculate the total cost of all of the home automation components in F16 you would use the formula `=SUM(F8:F14)`, where the ':' symbol states you are referencing a range or list of cells starting at F8 and ending at F14.

The COUNT function simply counts the number of cells within the provided range of cells that contain numbers. This can be useful for checking the number of data items in a table, especially when the number of rows in a table is changing. For example, to find the number of components listed in the home automation budget, you would use the formula `=COUNT(E8:E)`. This would count the different types of components, regardless of how many components were potentially added to the table in new rows in the future. The headings 'TOTAL' and 'Remaining', also in column E, would not be counted as they are not numbers.

Logical function

We may also want to perform a certain process on data based on a condition. To do this, we need to use a logical function, the most common one being the 'IF' function. The IF function has three parameters: the condition, the result if the condition is true and the result if the condition is false. For the home automation budget, the Amount paid column (J) will contain the amount paid *if* the product has been purchased; otherwise, it will be left blank. The formula for this in cell J8 is `=IF(H8 = "TRUE", F8, )`. Functions can also be a parameter for another function. For example, if our home automation spreadsheet also had a column of checkboxes for 'Discount applied?' in column I, the Amount paid column (J) could be calculated using the formula `=IF(H8 = "TRUE", IF(I8="TRUE",F8,B8*E8), )`. This means, if purchased is true, then if discount applied is true, paid is the total, otherwise paid is the RRP times the quantity.

Statistical functions – AVERAGE, MEAN, MODE

One of the most common statistical calculations you would perform on data is to find the average of a set of numbers. Consider a separate sheet in our home automation budget (Figure 9.13). This sheet is used to compare different home hubs available so the user can select the right product for the core of our system. It would be useful to find the average price for a home automation device to determine whether each of the products has an above or below average cost, or to use this average value as a placeholder in the main sheet, before deciding on the final product to purchase.

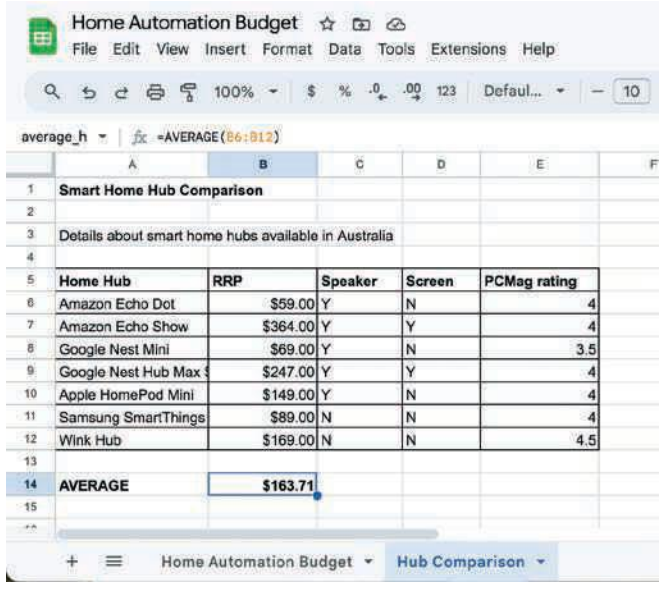


Figure 9.13 The AVERAGE function being used to calculate the average cost of home hubs being compared for the home automation budget.

Table 9.2 Summary of simple useful spreadsheet formula and functions.

Function	Example formula	Explanation
SUM	`=SUM(A2:A10)`	Returns the sum of the values in the range A2:A10
COUNT	`=COUNT(A2:A10)`	Returns the number of cells in the range A2:A10 that contain numbers
IF	`=IF(A2>50,"big","small")`	Returns the word `"big"` if the value in A2 is greater than 50, and returns `"small"` otherwise
AVERAGE (or MEAN)	`=AVERAGE(A2:A10)`	Returns the average of the values in the range A2:A10
MEDIAN	`=MEDIAN(A2:A10)`	Returns the value in the middle position of the range A2:A10 when all the values are sorted
MODE	`=MODE(A2:A10)`	Returns the value appearing most often in the range A2:A10

Linking sheets and named ranges

As our spreadsheet gets more complicated with multiple sheets, large tables and lots of cell referencing, there are some processes we should do to allow the system to work efficiently. This includes linking sheets and defining named ranges.

Linking sheets means we reference cells in other sheets of the same file. For example, if we wanted to copy across the average home hub price into our main table as a placeholder, in most spreadsheet software we can do this by referencing the sheet followed by an '!' then the cell. So, the formula in cell B8 of Sheet 1 would be `=Sheet2!B14`.

When all the letters and numbers of cell referencing get confusing, it is also useful to define a cell or range of cells by a meaningful name (Figure 9.14). This is also useful for others who may be using the spreadsheet and trying to understand how it works. Using named ranges can be particularly useful when using linked sheets, as it is hard to visually check what data is being referred to. In this example, we could define `Sheet2!B11` as `average_hub_rrp` and use this name in the formula for cell `B8: =average_hub_rrp`.

Figure 9.14 Using named ranges can make referencing cells more efficient and understanding formulas easier.

SKILL BUILDER

Analysis of video game sales data – Part 3

Simple formula and functions

We will demonstrate the use of the SUM and AVERAGE functions on this data set by finding the total number of video game sales (Figure 9.15), and the average video game sales per region (Figure 9.16). We will use this latter calculation in our first graphing exercise in section 9.5.

1 Go to the bottom of the data set (a shortcut to do this is to click on cell A1 then press Ctrl/Cmd and the down arrow) and in a cell below the last data row and in the global sales column (M) add the function `=SUM(M2:M60885)` then press Enter.

What result did you get? Remember that the value is in millions!

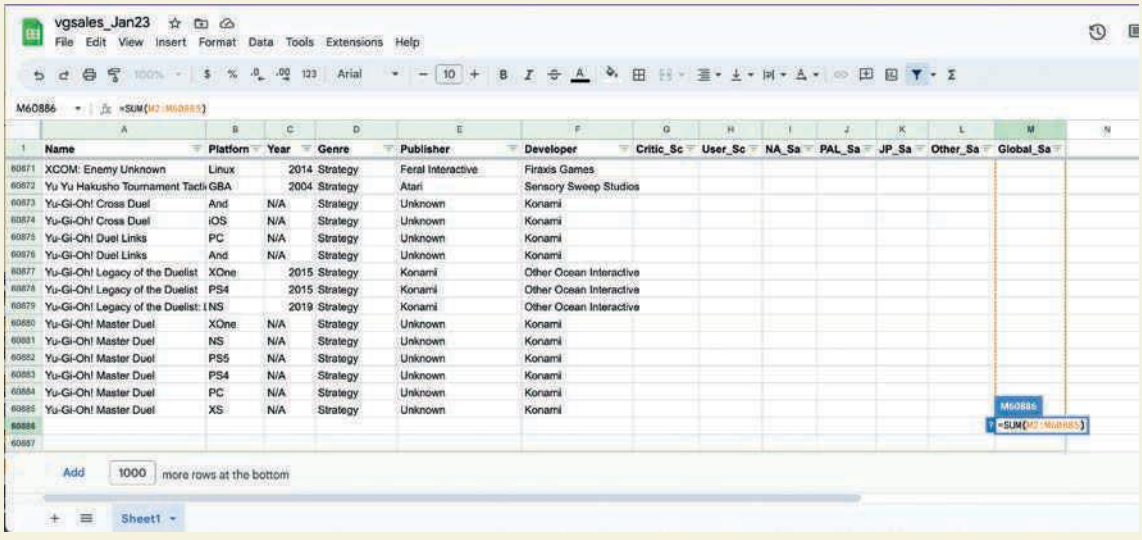
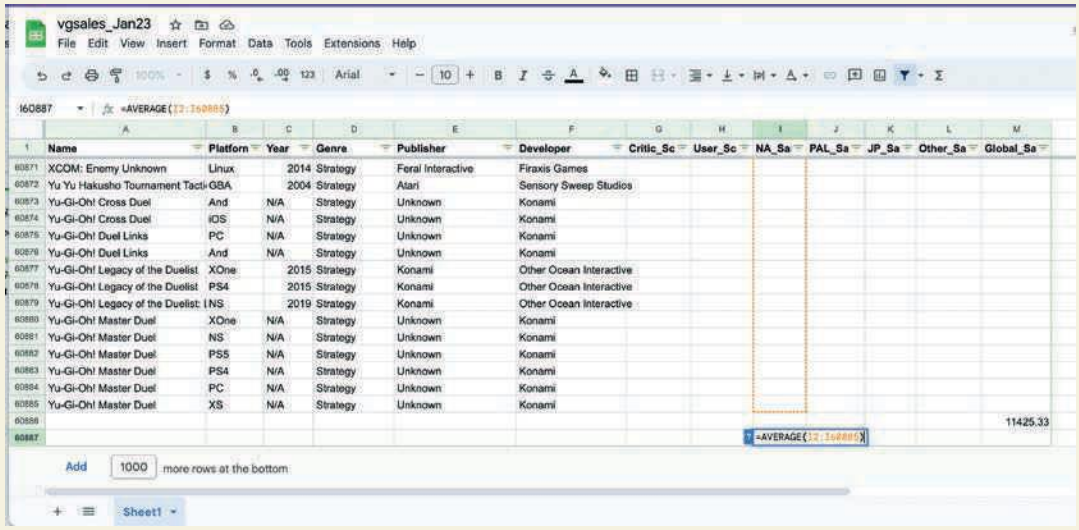


Figure 9.15 Calculating the total video game sales since 1970.



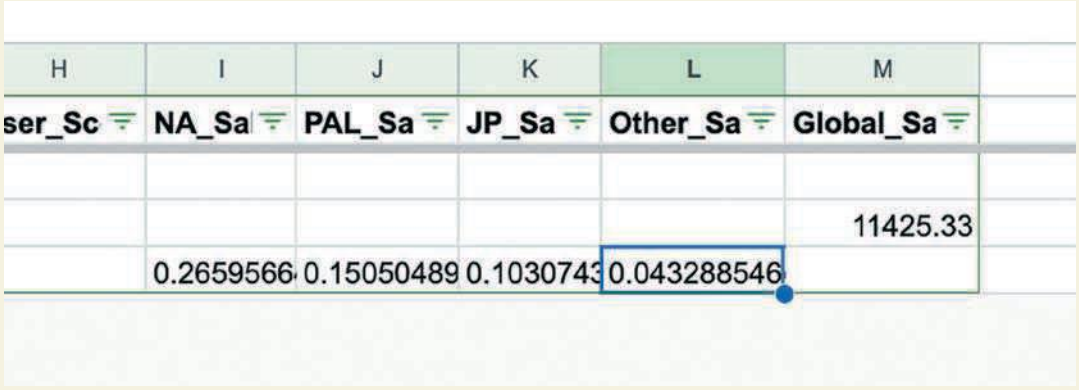


Figure 9.16 Calculating the average video game sales for each region.

2 In the cell two below the last data row in the NA sales column (I) add the function `=AVERAGE(I2:I60885)`. Press Enter. Copy this function into the next three adjacent cells by dragging from the bottom right corner of the cell to 'Fill' across.

   Which region had the highest average sales?

3 We will be using this data a lot more in future calculations, so let's make the cell referencing more meaningful by defining named ranges for the data we just analysed (Figure 9.17).

4 Experiment with performing some other SUM or AVERAGE calculations on data that you think reveals some interesting information. Share this with a peer.

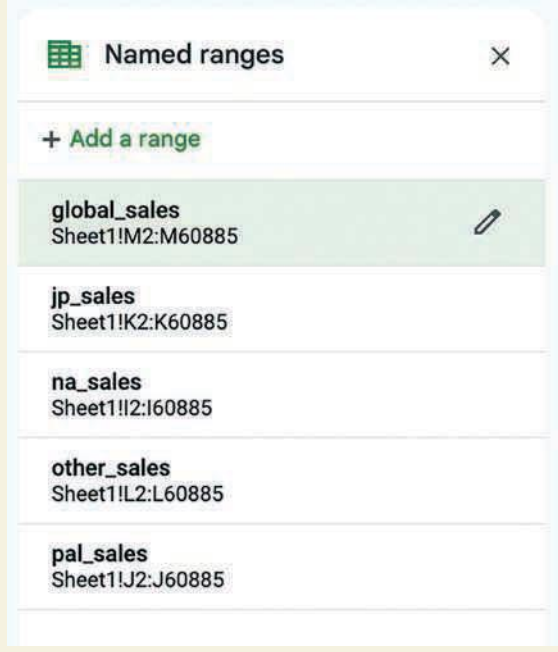


Figure 9.17 Naming ranges for variables we will be analysing.

ACTIVITIES

Identify

1 What are the parameters in an IF statement?

2 What is the difference between a formula and a function, and how are they related?

3 Why are named ranges useful even though they don't change what the software is calculating?

Analyse

4 Compare relative and absolute cell referencing using an example.

Extend

5 Create a simple spreadsheet that contains a list of numbers representing the heights of students in your class to one decimal place. Use the statistical functions AVERAGE, MEDIAN and MODE on them to calculate some statistics about the students' heights.

6 One way to verify your data collection and analysis is to compare it with the data and findings of other people who have also done this same process. Perform a verification, identifying any errors and how these occurred and could be prevented in the future.

7 Design and build a similar spreadsheet to calculate the cost of buying ingredients for your favourite recipe.

9.4 Advanced functions

Lookup tables

Computational thinking

Resource

A lookup table is a matrix of data items that can be searched through. Here we will learn about the VLOOKUP function as it is the most widely used and versatile. VLOOKUP is commonly used to assign a new value based on a map of data or to provide a searchable window for users looking at a large data set. We will explain how this is achieved using a student markbook example that contains the marks from an assessment task for 200 students across six classes (Figure 9.18). A VLOOKUP table will be used to assign a letter grade for each mark that will go on their reports, and another will be used to provide a form view to quickly see a student's data. Letter grades are assigned to numerical marks based on the cut-offs in Sheet 2 of the spreadsheet (Figure 9.19).

The formula used to calculate the grade for the first student is `=VLOOKUP(D2,Sheet2!A$2:B$6,2)`.

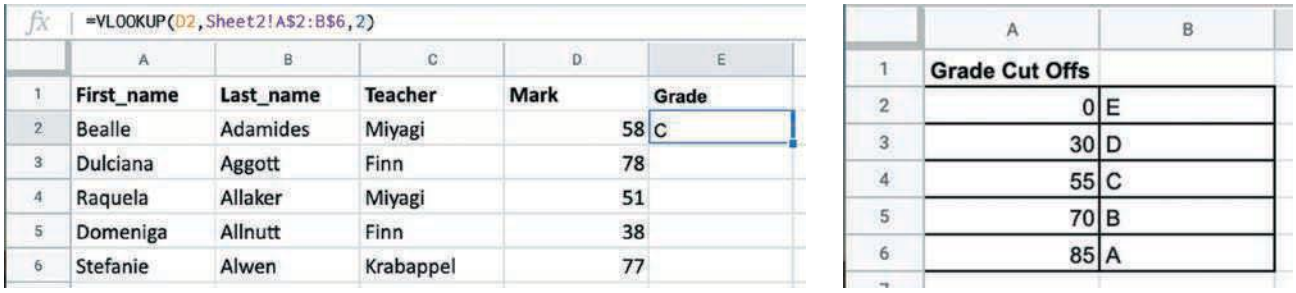
This formula means find the value D2 in the table Sheet2!A$2:B$6 and return the value in the second (2) column of the table. If there is no D2 value in the table, it looks for the value closest to it without going over, in this case, the number 55. Note the use of absolute referencing with the '$' symbol. This means when the formula is filled down into all 200 grade cells, the reference to the grade cut-off table stays the same. An even better option would be to create a named range for the grade cut-off table and simply reference this.

You can also use the VLOOKUP function to create a form viewer or window to allow a quick query into a student's data, much like what you could do in a database system, such as in Figure 9.20. When the user types in a last name to query in cell B3, the Teacher, Mark and Grade data updates.

	A	B	C	D
1	First_name	Last_name	Teacher	Mark
2	Bealle	Adamides	Miyagi	58
3	Dulciana	Aggott	Finn	78
4	Raquela	Allaker	Miyagi	51
5	Domeniga	Allnutt	Finn	38
6	Stefanie	Alwen	Krabappel	77
7	Hobie	Alyokhin	Keating	67
8	Levy	Ambridge	Miyagi	48
9	Husain	Americi	Garrison	40
10	Jerry	Archard	Krabappel	54
11	Claudie	Axton	Miyagi	42
12	Leola	Badman	Kimble	67
13	Lesly	Barense	Krabappel	50
14	Harper	Battram	Finn	65
15	Laural	Bedle	Day	43
16	Albie	Beeres	Day	63
17	Wendi	Bellhanger	Krabappel	47
18	Orazio	Bergeau	Miyagi	69
19	Augusto	Berryann	Finn	80
20	Alida	Bess	Finn	73
193	Barbette	Whiscard	Keating	81
194	Tiffanie	Whitloe	Miyagi	69
195	Antons	Widdicombe	Kimble	96
196	Ethyl	Widmore	Keating	76
197	Hamlen	Wogdon	Kimble	45
198	Philippine	Wontner	Garrison	74
199	Lotte	Wooderson	Keating	61
200	Scottie	Wooding	Finn	70
201	Ibrahim	Woofinden	Krabappel	88

Figure 9.18 Sheet 1 of the spreadsheet.

9780170442787

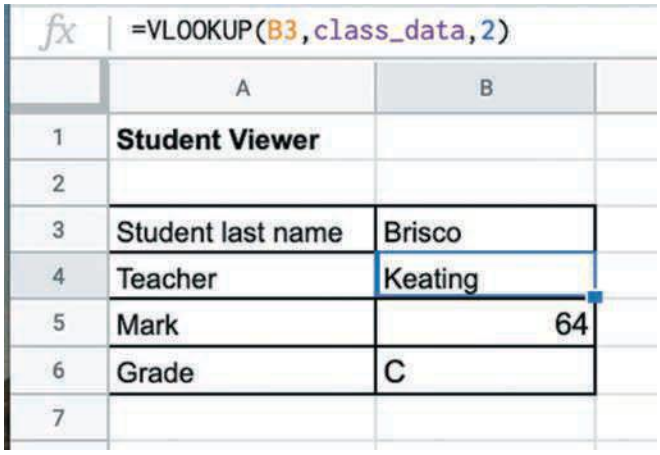


=VLOOKUP(D2,Sheet2!A$2:B$6,2)

	A	B	C	D	E
1	First_name	Last_name	Teacher	Mark	Grade
2	Bealle	Adamides	Miyagi	58	C
3	Dulciana	Aggott	Finn	78	
4	Raquela	Allaker	Miyagi	51	
5	Domeniga	Allnutt	Finn	38	
6	Stefanie	Alwen	Krabappel	77	

	A	B
1	Grade Cut Offs	
2	0	E
3	30	D
4	55	C
5	70	B
6	85	A

Figure 9.19 Grades are calculated using a VLOOKUP function that searches through data of grade cut-offs defined in Sheet 2.



=VLOOKUP(B3,class_data,2)

	A	B
1	Student Viewer	
2		
3	Student last name	Brisco
4	Teacher	Keating
5	Mark	64
6	Grade	C
7		

Figure 9.20 VLOOKUP table used to view a single student record.

SUMIF, COUNTIF and AVERAGEIF

The SUMIF, COUNTIF and AVERAGEIF family of functions work by performing the calculation only on cells that meet certain criteria.

For example, if we wanted to find the average marks for students who were in Krabappel's class, the formula for this would be `=AVERAGEIF(C2:C201, "Krabappel",D2:D201)`. The first parameter is the values being checked (that is, the teacher), the second parameter is the criteria to be met (that is, "Krabappel"), and the third parameter is the values we want to average (that is, the marks). This process could easily be duplicated to find the average mark for all classes (Figure 9.21).

The COUNTIF function could also be used to count the number of students in each class. You will notice it only has two parameters, as the function is just counting the occurrence of the criteria in the range (Figure 9.21).

Manipulating text

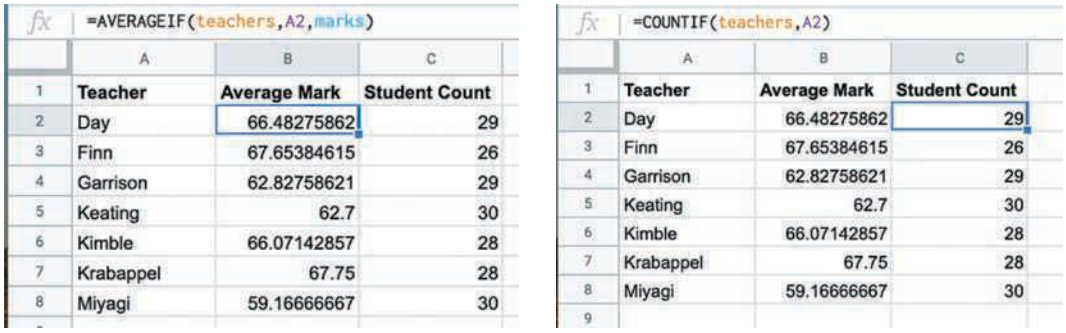
Although most of the processing in spreadsheets is on numerical values, sometimes it is useful to manipulate text. In the example in Figure 9.20, you may have considered the issue of two or more students having the same last name. One option to overcome this is to create a calculated field 'full_name' in the class data sheet by *concatenating* the first and last names together. This field could then be used to search for individual students more accurately (Figure 9.22).

There are many other functions for processing text, such as finding the number of characters in a text string (LEN), extracting parts of a text string (LEFT, RIGHT, MID) or finding text strings within a text field (FIND).

QUERY

In Google Sheets, there is a QUERY function that allows you to perform simple queries like you would with SQL in a database system. It uses the Google Visualization API Query Language across the data set provided as a parameter. This can be useful when you would like to generate a separate summary or filtered data table (e.g. in a dashboard). For example, with the student grade data you could get a table with the top 10 students using `=QUERY(A1:E201, "select A,B,C,D order by D DESC limit 10",0)`.

Microsoft Excel does not have a query function, but you can perform queries using its separate Power Query editor.



=AVERAGEIF(teachers,A2,marks)

	A	B	C
1	Teacher	Average Mark	Student Count
2	Day	66.48275862	29
3	Finn	67.65384615	26
4	Garrison	62.82758621	29
5	Keating	62.7	30
6	Kimble	66.07142857	28
7	Krabappel	67.75	28
8	Miyagi	59.16666667	30

=COUNTIF(teachers,A2)

	A	B	C
1	Teacher	Average Mark	Student Count
2	Day	66.48275862	29
3	Finn	67.65384615	26
4	Garrison	62.82758621	29
5	Keating	62.7	30
6	Kimble	66.07142857	28
7	Krabappel	67.75	28
8	Miyagi	59.16666667	30
9			

Figure 9.21 AVERAGEIF and COUNTIF functions used to produce summary data.

=CONCATENATE(A2," ",B2)

	A	B	C
1	First_name	Last_name	Full_name
2	Bealle	Adamides	Bealle Adamides
3	Dulciana	Aggott	Dulciana Aggott
4	Raquela	Allaker	Raquela Allaker
5	Domeniga	Allnutt	Domeniga Allnutt

=VLOOKUP(B3,class_data,2,FALSE)

	A	B
1	Student Viewer	
2		
3	Student search	Raquela Allaker
4	Teacher	Miyagi
5	Mark	51
6	Grade	D

Figure 9.22 Using the CONCATENATE function on the students' names data allows for the student viewer to search based on full name. Note a fourth parameter of the VLOOKUP function has been used. This Boolean value states that the lookup data is NOT sorted.

INFOBIT

In Google Sheets, users can define their own "named functions". Like named ranges this can improve the efficiency and accuracy of using formula that you may find yourself repeatedly doing and allow you to update all instances of the formula in one go. For example, a school may use a formula to create an ID code for students based on their name. Rather than recreating this formula for each table or file, a named function can be created (Figure 9.23).

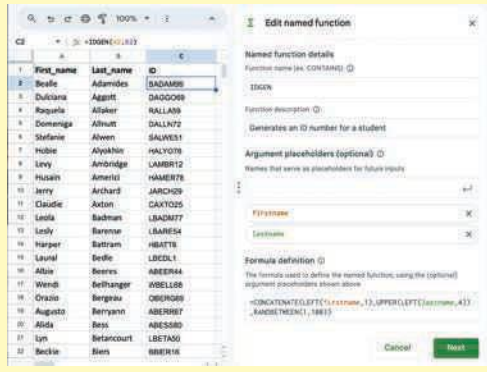


Figure 9.23 The named function takes 2 parameters (firstname and lastname) and uses the existing functions CONCATENATE, LEFT, UPPER and RANDBETWEEN to create an ID composed of the students first initial, the first 4 letters of the last name and a random number between 1 and 100.

SKILL BUILDER

Analysis of video game sales data – Part 4

More advanced functions

We will now perform some of these more advanced IF calculations on the video games data set.

1. Set up a new sheet to store all the results from these calculations. It is good practice to set up a new sheet for more complex functions and results, especially when you have a large data set. By creating a new sheet, you can keep calculations/summaries separate and easy to find.
2. Complete the following IF calculations of the data set. The first one has been done for you as a demonstration.
   - **a** Find the average critic score for Nintendo games (Figure 9.24). Compare this to the average critic score for Microsoft Game Studios.
   - **b** Extend this summary table by including another three publishers.
   - **c** Sum the global sales of PlayStation 4, Xbox One and Nintendo Switch games. You could extend this by looking up the total sales of all PlayStation, Xbox and Nintendo platforms then adding these together.
   - **d** Count the number of games under each of the genres. To find out what all the genres are, go to the filter based on this column and observe all the possible values by which you can filter.

B5 =AVERAGEIF(publisher,A5,critic_score)

	A	B
1	Summary Data	
2		
3	*Average critic score by publisher*	
4	Publisher	Average Critic Score
5	Nintendo	7.788934426
6	Microsoft Game Studios	7.796103896
7		

Figure 9.24 Average critic score by publisher.

ACTIVITIES

Identify

1 Outline the four possible parameters in a VLOOKUP function.

2 What is the difference between a SUMIF function and a COUNTIF function?

3 What does the word 'concatenate' mean?

Analyse

4 Explain how VLOOKUP and CONCATENATION functions could be used to determine the name of a colour given its R, G and B values.

5 Research three of the other text processing functions and explain what they do.

Extend

6 Use the QUERY function to create a summary table of the top 10 video games based on sales.

7 Create a Caesar cipher solver using spreadsheet software. Use Figure 9.25, which shows the set up and functions used in a spreadsheet that encrypts text using a Caesar cipher. Your spreadsheet will need to decrypt.

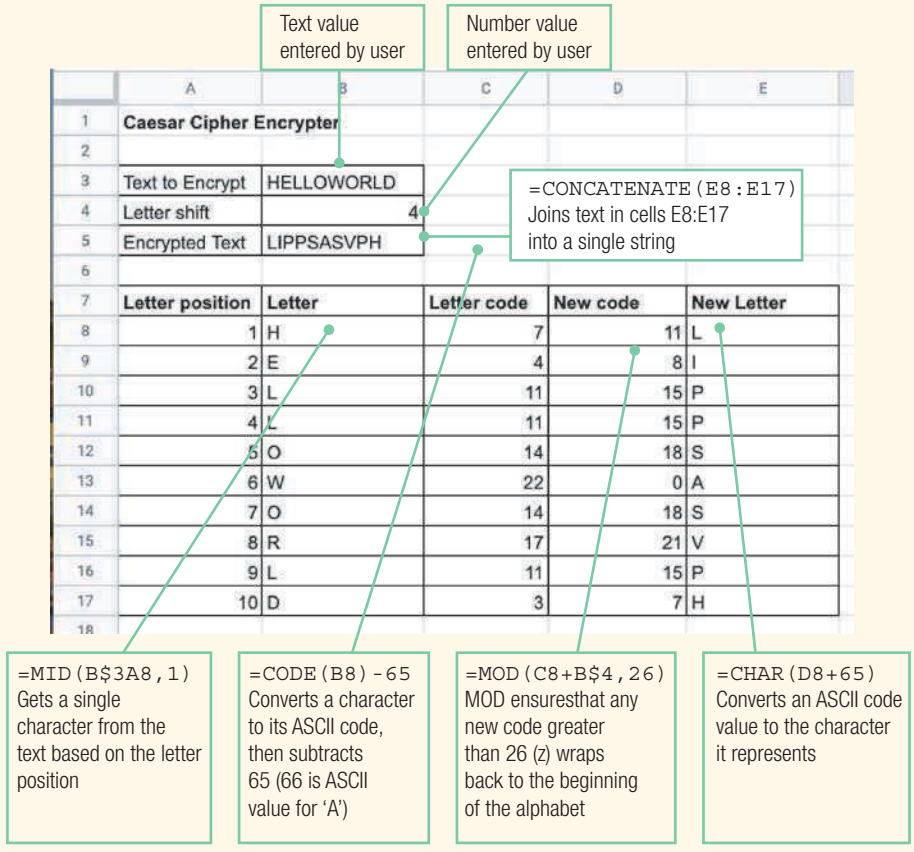


Figure 9.25 A simple Caesar cipher encrypter that can encrypt text without spaces using an input letter shift value.

Computational thinking

9.5 Visual communication and charts

In many instances, data and the information generated from it is only useful when it can be effectively communicated to its intended audience. People can learn very effectively through visual communication.

Visual communication in spreadsheets

You have done some level of visual communication by laying out your spreadsheets in distinct areas and using text formatting such as bolding to highlight headings.

Another method of formatting for visual communication is *conditional formatting*. This automates formatting of cells (e.g. text or fill colour) based on the value in the cell. This can be very useful for highlighting particular calculated values that may be of importance (Figure 9.26). For example, in the home automation budget, component rows could be filled in grey once the product has been purchased or the budget remaining calculated field could turn red if it goes below $0.

The main feature of spreadsheet software that people use to visually communicate is charts.

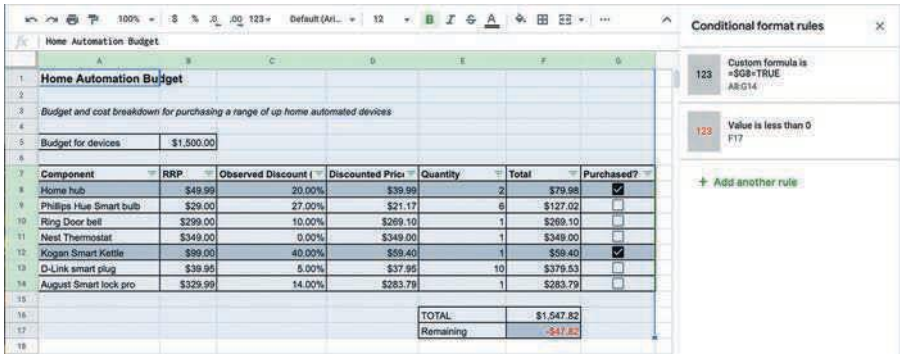


Figure 9.26 Conditional formatting can be used to highlight or communicate a feature of key data in a spreadsheet.

Types of charts

Charts and graphs come in many different forms. Common ones include line graphs, pie charts, column graphs and scatter plots. Selecting the appropriate chart is very important for representing the data accurately and effectively. Figure 9.27 is a flowchart that can be used to select an appropriate chart for your data set.

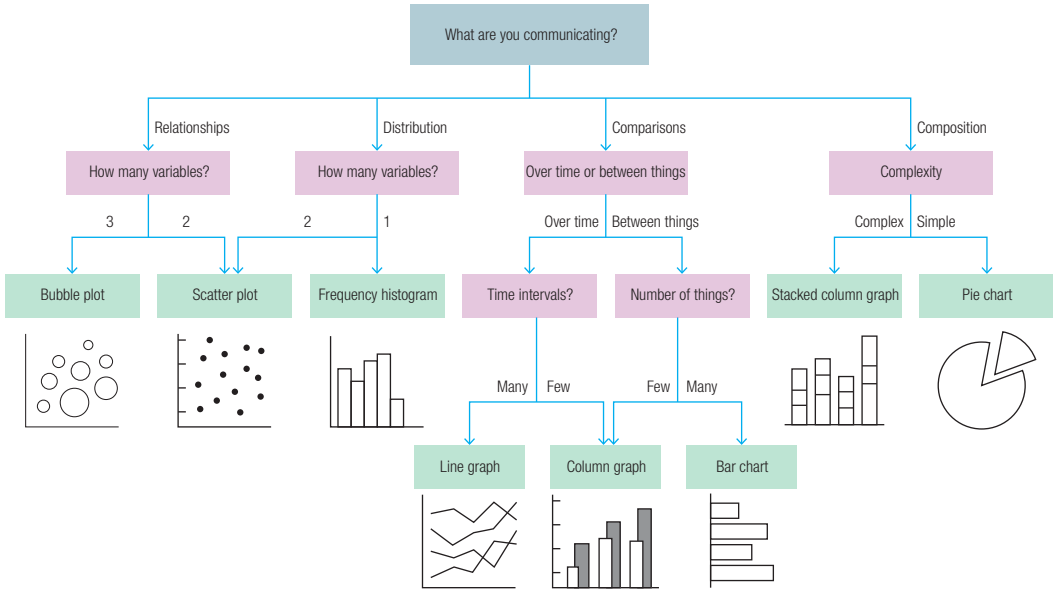


Figure 9.27 Flowchart to help you decide what type of chart you should use for your visual communication of data.

Titles and labelling

The titles and labelling on a chart are very important for clarifying what data and information are being represented (Figure 9.28). The important titles to include are the main title that summarises the whole chart and the titles of the horizontal axis and the vertical axis. The main title should reference the variable(s) used in the chart. The axis titles are the variables and should include any units.

Labelling can occur on individual parts of a chart or a key may be used.

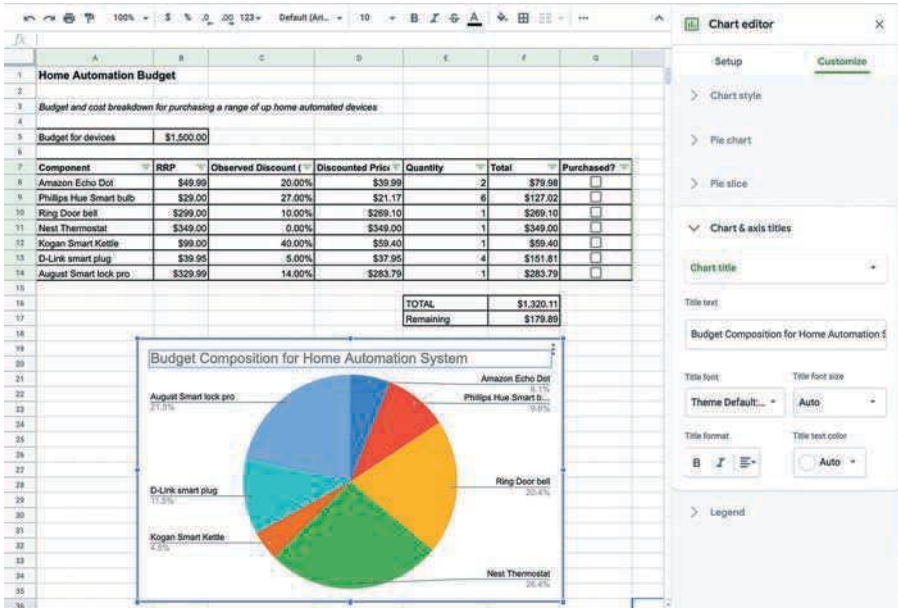
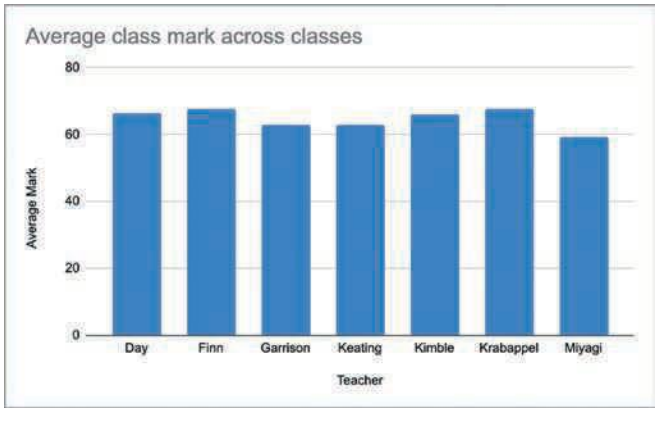


Figure 9.28 Setting up a pie chart to visually communicate the budget composition of components for the home automation system.

Chart adjustments

It may be appropriate to make adjustments to the graph from the default settings, so you are better able to communicate the data. For example, if we were to graph the average class marks for each class using a column graph, the default vertical axis extends from 0 to 80; however, the range of the values is only 59 to 68. It might be more suitable to view only this range of the chart to more clearly show the differences. Figure 9.30 shows the charts before and after adjustment. You can see the differences between the classes in the second chart more clearly, but is this a truthful representation of the data? Is the first chart actually more useful as it shows there is not actually a significant difference between classes? Companies and media reports commonly use this method to exaggerate changes over time.

This example shows that the way we choose to visually communicate data can emphasise certain information and hide other information. In this way, charts can be used to create a *biased* perspective. People looking at the adjusted chart in Figure 9.29 may think that Garrison's average mark is only half that of Finn's, which is not correct. The user creating the chart is responsible for accurately representing the data.



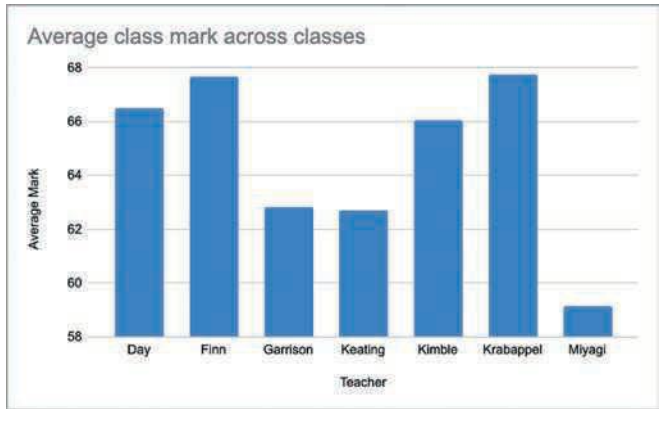


Figure 9.29 Adjusting the range on an axis can present a biased view of data.

SKILL BUILDER

Analysis of video game sales data – Part 5

Creating a pie chart

We will now create a pie chart to communicate the composition of video game sales across North America, Europe, Japan and the rest of the world.

1. Calculate the total sales for each region. Select the four column headings, then, with Ctrl/Cmd held, select the four cells with the total sales value (Figure 9.30).
2. In the chart editor, change the chart type to Pie. You will see an error appear. This is because by default chart tables have variables in different columns, but our summary data is organised across rows.
   To fix this, change 'Combine ranges' to vertical and check the Switch rows/columns box.

Creating a bar chart

Next, we will graph the number of games per genre using a bar chart.

1. First, we need to create the summary data for the graph using the COUNTIF function. In a new sheet, add two column headings: Genre and No of Games. Rather than scanning through the 60 000+ records to see all the different genres, they have been outlined for you in Figure 9.31. Later, we will see how this can be done automatically using pivot tables.

   We will use a bar chart to represent this summary data or information, as we are showing a comparison between many things.

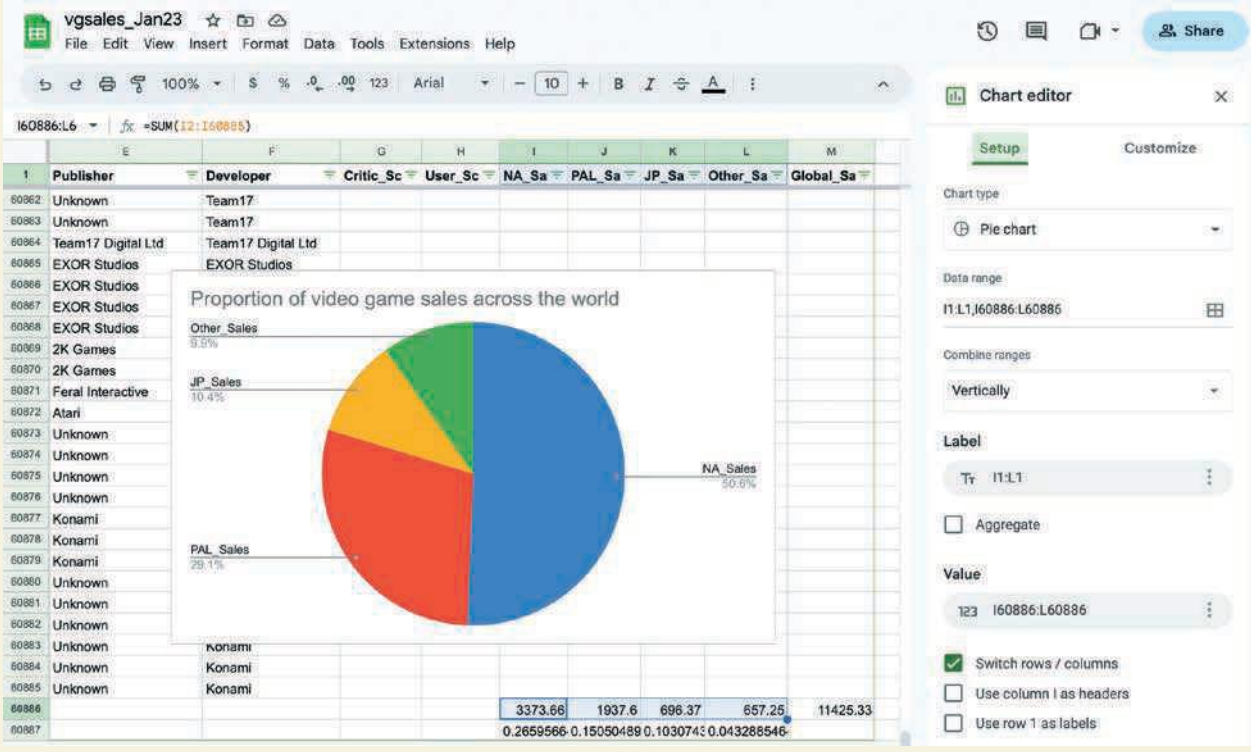


Figure 9.30 This pie chart clearly shows that, North America is responsible for over half (50.6 per cent) of world sales in video games.

B2 =COUNTIF(genre,A2)

	A	B
1	Genre	No. of Games
2	Action	8113
3	Action-Adventure	1430
4	Adventure	5931
5	Board+Game	22
6	Education	31
7	Fighting	2232
8	Misc	9280
9	MMO	108
10	Music	248
11	Party	134
12	Platform	3811
13	Puzzle	3399
14	Racing	3281
15	Role-Playing	5291
16	Sandbox	15
17	Shooter	5123
18	Simulation	2960
19	Sports	5457
20	Strategy	3548
21	Visual+Novel	470
22		

Figure 9.31 Genres of games.

2 Select cells A1:B21 and insert a chart. In the chart editor, change the chart type to Bar and make sure row 1 is being used as headings (not included in the data) (Figure 9.32).

3 Add the main title and titles for the horizontal axis and vertical axis.

4 You may also like to increase the number of horizontal grid lines and sort the data.

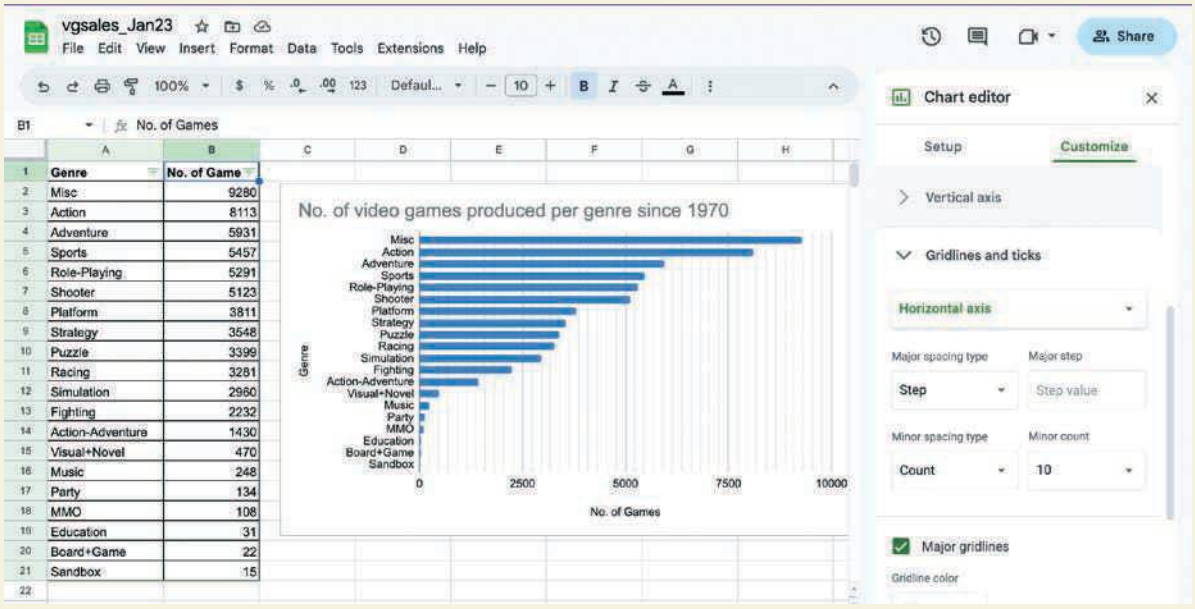


Figure 9.32 This bar chart shows the most common, non-miscellaneous game genre is Action. Sorting the data shows the ranking of game genres based on how common they have been in the last 50 years.

ACTIVITIES

Identify

1 What is conditional formatting?

2 What are the type of graphs that can be used when time is a variable?

3 When are labels or keys useful?

Analyse

4 You have been given the total house points for the latest swimming carnival. You would like to visually communicate this information to the rest of the school using a chart. Which type of chart could you use? Justify your answer.

5 Find an example of a 'biased' graph online. What does it appear to show and what does it actually show?

Extend

6 Create a bar chart of the average critic score per publisher from the summary data you calculated in the Skill builder activity in section 9.4.

7 Find a tutorial online to show you how to create a Gantt chart using a Google Sheets or Microsoft Excel bar chart and use that technique to create your time plans for future projects.

9.6 Modelling, simulating and predicting

A model is a simpler version of a real-world object or idea. A simulation is the process of imitating a real-life occurrence in order to test a model. Spreadsheet software can be used to create models and simulations. They can then be used to help make predictions and gain insights into natural, human and technological systems that would not be possible using traditional non-computerised methods.

Chart trends

A simple model that can be created in spreadsheet software is an equation that represents a trend observed in real-world data. A trendline is created by the software using algorithms to consider all the data points provided and the overall pattern they are suggesting over a continuous variable, such as time. The use of a trendline model for predicting will now be demonstrated.

Data of the annual mean temperature in Australia from 1910 to 2019 was collected from the Bureau of Meteorology (BOM) climate change data web page. Using a line graph to represent this data shows an upward trend (Figure 9.33). By customising the chart to add a trendline and labelling it with its equation, we have created a simple model. This equation shows that the gradient of the trendline is 0.0132, which means that the model is suggesting a 0.0132 increase in temperature each year. This may not seem like much, but looking 100 years into the future this would be a 1.32 degree increase, enough to cause significant environmental impacts such as sea level rising, changes to weather patterns and destruction of habitats.

What is more concerning is that, when you graph the data just for the past 50 years, a time of increasing fossil fuel emission, the gradient of the trendline increases to 0.024, suggesting almost double the rate of temperature increase for our future.

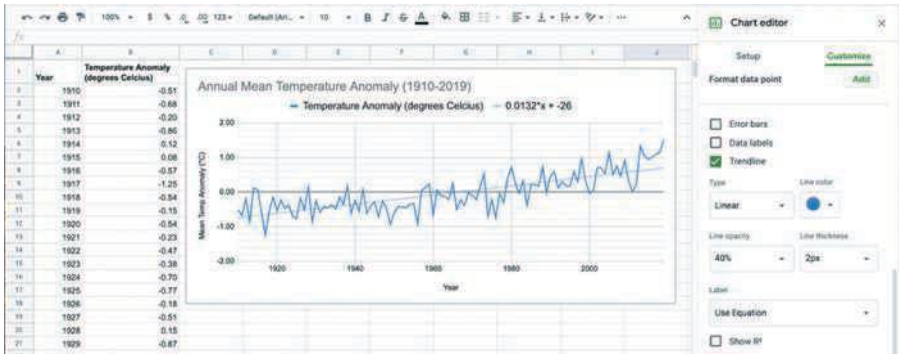


Figure 9.33 Spreadsheet analysis of climate data using charting and trendlines to produce an equation model.

Source: Australian Bureau of Meteorology, http://www.bom.gov.au/climate/change/

Simulating

Spreadsheet software can be used to test a model you have created or to test an existing model.

In the previous climate change example, the equation `Temperature Anomaly = 0.0132*Year - 26` (or `Temperature Anomaly = 0.024*Year - 47.5` if using just the past 50 years of data) could be used to simulate the temperature anomaly for the next 10, 100, 1000 years and so on (Figure 9.34).

You will find more examples of spreadsheet simulations in section 14.3.

What if analysis and goal seek analysis

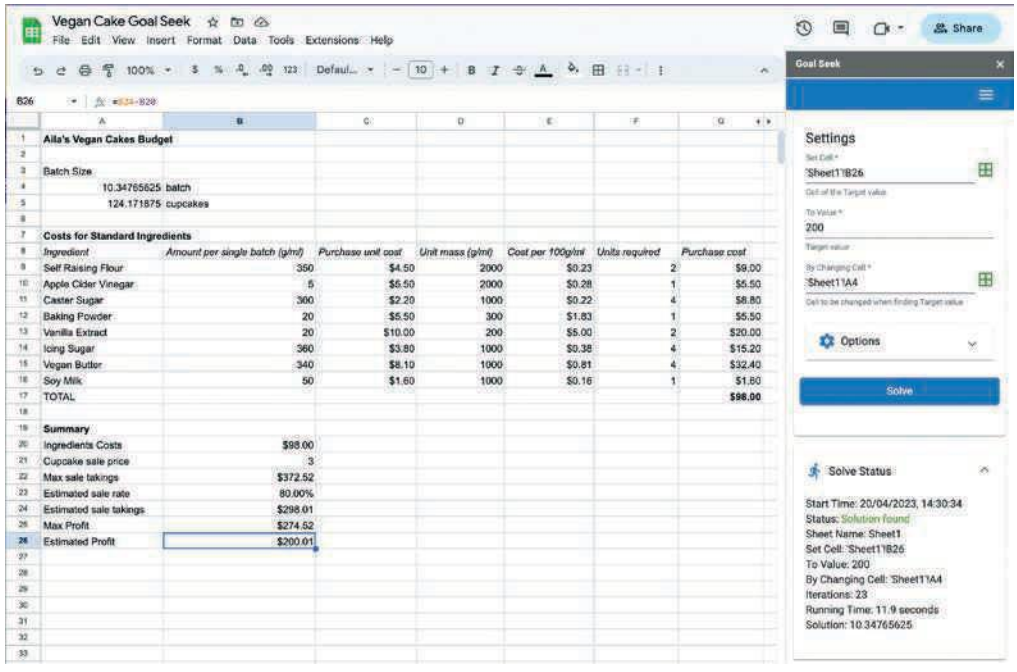
Other ways you can do predictions using spreadsheets are with **what if** and **goal seek** analysis. A what if analysis is simply the process of changing the value of a cell and observing how this impacts any calculation fields that reference this value cell. For example, in the home automation budget, you could change the quantity of one or more of the components and see how this impacts your overall cost.

A goal seek analysis works almost in the opposite way. You define a goal result for a calculated field and the algorithm will modify any referenced, nominated value cells in order to achieve this goal. For example, in the home automation budget, you could prompt the goal seek analysis to change the quantities of Wi-Fi power points, in order to use up the remaining budget (that is, set to zero). Goal seek is a built-in function in Microsoft Excel but not in Google Sheets. If you would like to perform a goal seek analysis in Google Sheets, you will need to link an add-on from the G Suite Marketplace. Figure 9.35 shows an example of using the goal seek add-on in Google Sheets for a cake stall budget.

fx =0.024*A2-47.5

	A	B
1	Year	Temperature Anomaly (*C)
2	2030	1.22
3	2040	1.46
4	2050	1.7
5	2060	1.94
6	2070	2.18
7	2080	2.42
8	2090	2.66
9	2100	2.9
10	2110	3.14
11	2120	3.38
12	2130	3.62
13	2140	3.86
14	2150	4.1
15	2160	4.34
16	2170	4.58
17	2180	4.82
18	2190	5.06
19	2200	5.3
20	2210	5.54
21	2220	5.78
22		

Figure 9.34 Simulating increase in average temperature anomaly over the next 200 years based on a model from 1970–2019 data.



	A	B	C	D	E	F	G
1	Aila's Vegan Cakes Budget						
2							
3	Batch Size						
4		10.34765625 batch					
5		124.171875 cupcakes					
6							
7	Costs for Standard Ingredients						
8	*Ingredient*	*Amount per single batch (g/ml)*	*Purchase unit cost*	*Unit mass (g/ml)*	*Cost per 100g/ml*	*Units required*	*Purchase cost*
9	Self Raising Flour	350	$4.50	2000	$0.23	2	$9.00
10	Apple Cider Vinegar	5	$6.50	2000	$0.28	1	$5.50
11	Caster Sugar	300	$2.20	1000	$0.22	4	$8.80
12	Baking Powder	20	$5.50	300	$1.83	1	$5.50
13	Vanilla Extract	20	$10.00	200	$5.00	2	$20.00
14	Icing Sugar	360	$3.80	1000	$0.38	4	$15.20
15	Vegan Butter	340	$8.10	1000	$0.81	4	$32.40
16	Soy Milk	50	$1.60	1000	$0.16	1	$1.80
17	TOTAL						$98.00
18							
19	Summary						
20	Ingredients Costs	$98.00					
21	Cupcake sale price	3					
22	Max sale takings	$372.52					
23	Estimated sale rate	80.00%					
24	Estimated sale takings	$298.01					
25	Max Profit	$274.52					
26	Estimated Profit	$200.01					
27							
28							
29							
30							
31							
32							
33							

Figure 9.35 Aila made a budget for her cupcake sale at the upcoming markets. Part of the analysis involved her using a goal seek analysis to determine how many cupcakes she needed to make to achieve a profit of $200. She could have also done a similar goal seek analysis for a $200 profit but by changing the cupcake sale price.

Skill builder resource
Simulating Mendelian genetics

Computational thinking

SKILL BUILDER

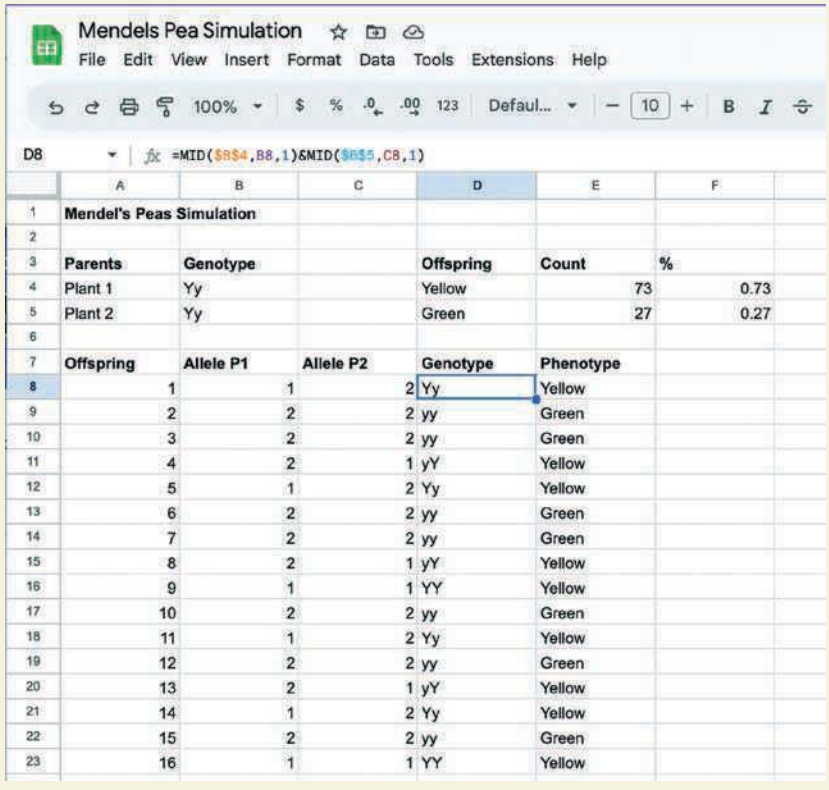
Simulating Mendelian genetics

Gregor Mendel was a 19th century geneticist who studied hereditary features of peas. This led to the laws of Mendelian inheritance, including the law of dominance, which you study in Stage 5 Science. The law of dominance states that offspring will only express one trait of a characteristic defined in a gene (phenotype).

A single allele for a characteristic is provided from each parent (genotype). In peas, there are two traits for the colour of the pea – yellow and green – with yellow being dominant over green. This means that if a plant had yellow peas its genes could contain two yellow alleles (YY) or a yellow and a green allele (Yy) as the dominant yellow allele would ensure only this trait is expressed. A plant with green peas must contain two green alleles (yy). When a plant pollinates with another plant, they will randomly provide one of their two alleles to form the new plant with a 50 per cent chance for each. We can simulate pea colour phenotype of the offspring of two pea plants by applying this law of dominance in a spreadsheet.

Set up a spreadsheet as shown in Figure 9.36.

1. The input area (B4:B5) is where you will set the genotype of each parent, either YY, Yy or yy.
2. The calculation area is where you will simulate 100 plant offspring by randomly selecting an allele from each parent. In cells B8 and C8, put the formula `=RANDBETWEEN(1,2)`, which is identifying whether the first or second allele has been randomly selected.
3. Calculate the genotype using the MID function to select the first or second character from the parent genotype. In cell D8, put the formula `= MID($B$4,B8,1)&MID($B$5,C8,1)`. Note the use of absolute referencing. The symbol '&' is used to concatenate the two text characters together.
4. Calculate the phenotype using an IF statement: `E8 = IF(EXACT(D8,"yy"),"Green","Yellow")`. Note 'EXACT' needs to be used as normally text comparisons are not case sensitive.
5. Select cells B8:E8 and fill down for all 100 offspring.



	A	B	C	D	E	F
1	**Mendel's Peas Simulation**					
2						
3	**Parents**	**Genotype**		**Offspring**	**Count**	**%**
4	Plant 1	Yy		Yellow	73	0.73
5	Plant 2	Yy		Green	27	0.27
6						
7	**Offspring**	**Allele P1**	**Allele P2**	**Genotype**	**Phenotype**	
8	1	1	2	Yy	Yellow	
9	2	2	2	yy	Green	
10	3	2	2	yy	Green	
11	4	2	1	yY	Yellow	
12	5	1	2	Yy	Yellow	
13	6	2	2	yy	Green	
14	7	2	2	yy	Green	
15	8	2	1	yY	Yellow	
16	9	1	1	YY	Yellow	
17	10	2	2	yy	Green	
18	11	1	2	Yy	Yellow	
19	12	2	2	yy	Green	
20	13	2	1	yY	Yellow	
21	14	1	2	Yy	Yellow	
22	15	2	2	yy	Green	
23	16	1	1	YY	Yellow	

Figure 9.36 Simulating the Mendelian genetics in pea plants using a spreadsheet.

6. The output area in cells D3:F5 shows the summary of the offspring. `E4 = COUNTIF(E$8:E$107,D4)`. Fill down for E5.
7. Change the parent genotype to see how this impacts the ratios/percentages of each phenotype in the offspring.

Extension

8. Use conditional formatting to visually represent the phenotypes (green and yellow cells).
9. Extend this spreadsheet to include one or more of the other six pea characteristics studied by Mendel.

ACTIVITIES

Identify

1 What is the difference between a model and a simulation?
2 How can a model or simulation be used to make a prediction?
3 What is the difference between a what if analysis and a goal seek analysis?

Analyse

4 The more data your model is based on, the more confident you can be in how accurately it represents the real-world situation. Why is this the case?
5 In video games, random numbers need to be generated all the time to simulate 'real-world' events. Research the type of random model used in video games. Is there a function for this in the spreadsheet software you are using? If so, try it out.

Extend

6 Visit the BOM climate data web page on the BOM website and download a different data set for analysis, such as average rainfall over time. Graph the data and identify any trends and potential models that can be made using this data.
7 Download and look at Aila's vegan cakes budget in Figure 9.36. Perform other what if and goal seek analyses on the spreadsheet.

Weblink
BOM

Resource
Aila's vegan cakes budget

9.7 Pivot tables

According to many data analysis experts, pivot tables are the spreadsheet's most powerful feature, yet most users have never used one. Pivot tables produce powerful results and are very fast and easy to use.

What is a pivot table?

A pivot table is a summarising tool. It is used to highlight important information from large data sets simply by selecting chosen fields for its rows, columns and values. It allows users to do many of the COUNTIF, AVERAGEIF and SUMIF analyses discussed in previous sections with no formula and only a few clicks. These tables are called pivot tables because they allow us to look at the data in a different way by 'pivoting' the table so it is showing or summarising data based on a different field (Figures 9.37 and 9.38).

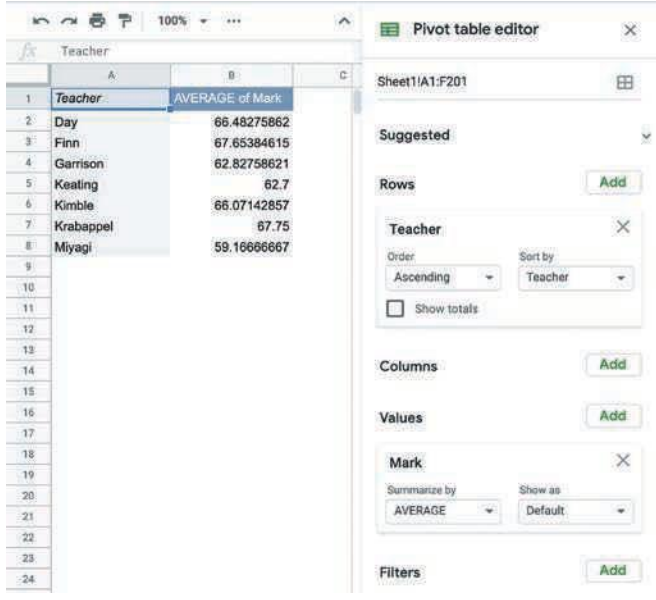


Figure 9.37 A pivot table showing the same summary data from our previous AVERAGEIF analysis in Figure 9.22.

INFOBIT

A pivot table can be a very useful way to analyse a flat-file database if you do not have access to a DBMS, you feel more confident in using spreadsheet software or if you would like to visualise the results from the analysis using charts.

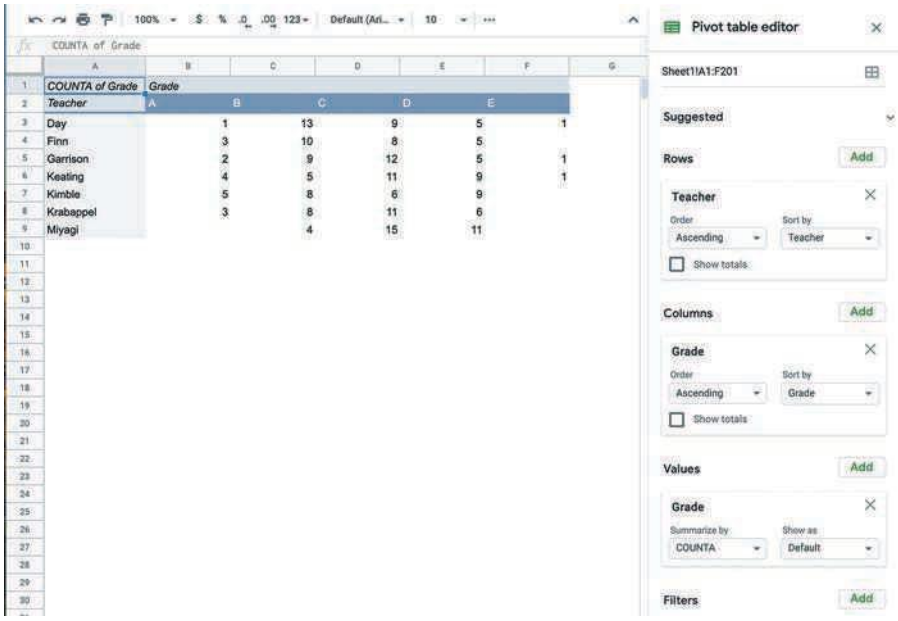


Figure 9.38 A pivot table summarising the Count of each grade for each teacher.

Selecting data

To create a pivot table, you need to select the data you would like summarised across the rows and/or columns and the summary values you would like displayed.

The pivot table editor in the spreadsheet software you are using will list all the options for analysis and may even make some suggestions (Figure 9.39).

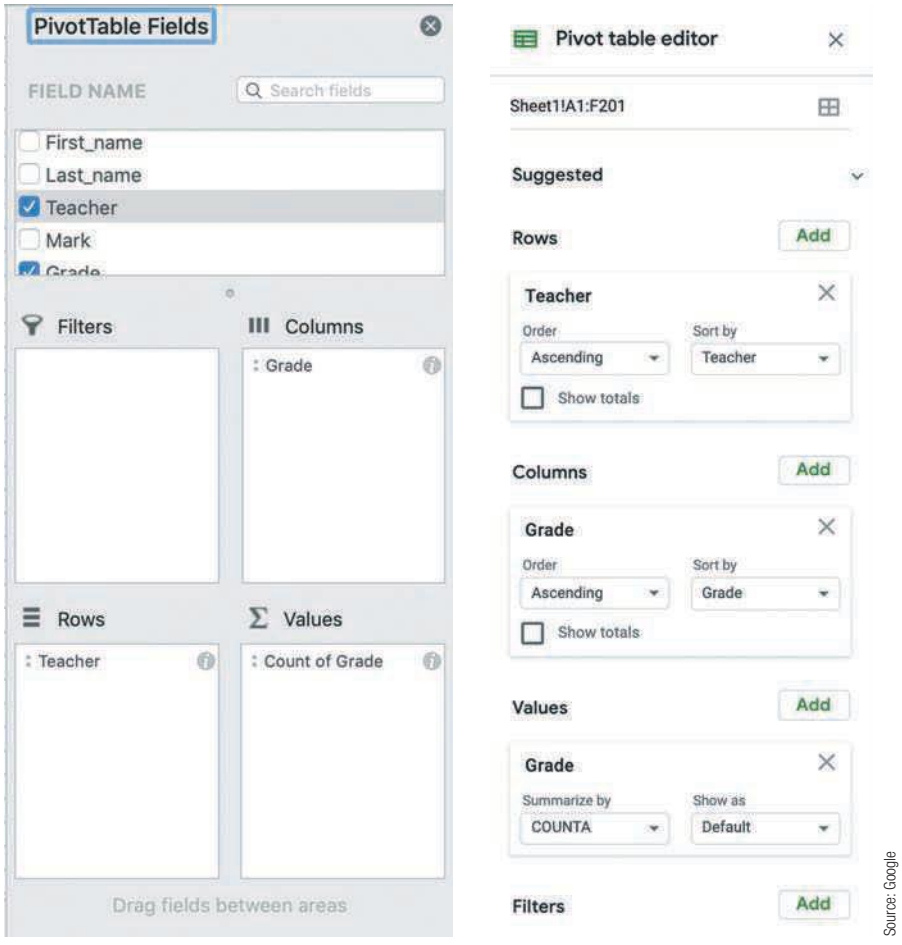


Figure 9.39 Pivot table editor in (a) Google Sheets and (b) Microsoft Excel.

Graphing pivot tables

We may want to graph the summary data in the pivot tables as a more effective way of communicating the generated information. As with other data tables, simply select the cells you would like to graph and insert the appropriate chart (Figure 9.40). With the power of pivot tables combined with charting, hundreds or hundreds of thousands of data values can be represented in a single image.

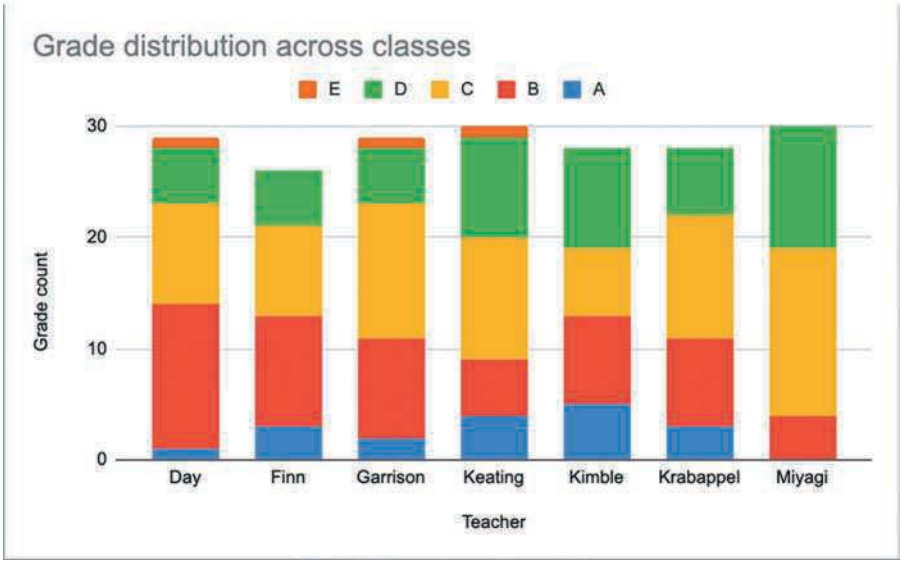


Figure 9.40 Stacked column graph generated from data in the pivot table in Figure 9.38.

SKILL BUILDER

Analysis of video game sales data – Part 6

Pivot table and further graphing

Many of the functions you have used in the video game sales data analysis have involved summarising data. For example, we looked at the count of video games for each genre in section 9.3. Large data sets with lots of fields can only really be analysed by doing this. Although we could keep designing functions for the particular data summaries we need, we have seen that pivot tables are much more powerful and make this process very efficient.

First, create one of the calculations we have already performed (average critic score for each publisher), but this time using a pivot table:

1. Create a new pivot table: Insert > pivot table.
2. In the pivot table editor, add 'Publisher' as the row and 'Critic score' as the value. Make sure you summarise the critic score by AVERAGE (Figure 9.41).

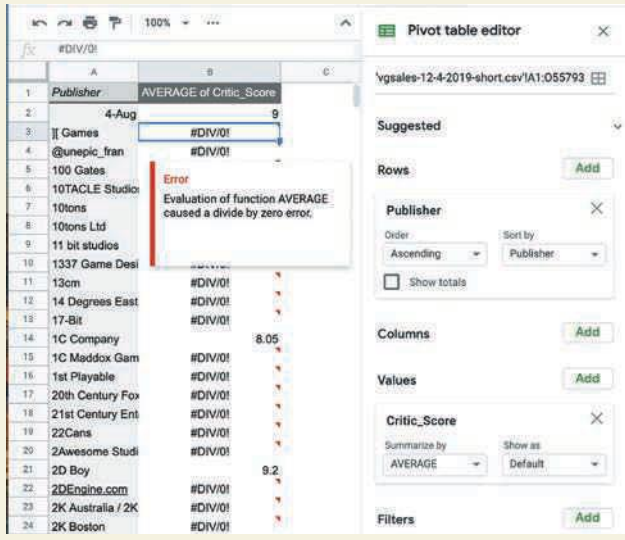


Figure 9.41 Setting up the pivot table by selecting the fields to appear in the rows and values. Error can be dealt with using a filter.

3 You will notice there are a lot of '#DIV/0!' values. This is because many publishers never had a critic score, so the average calculation has a zero in the denominator, causing an error. We can remove these non-meaningful values from our table by filtering based on critic score and unchecking '(Blanks)' (Figure 9.42).

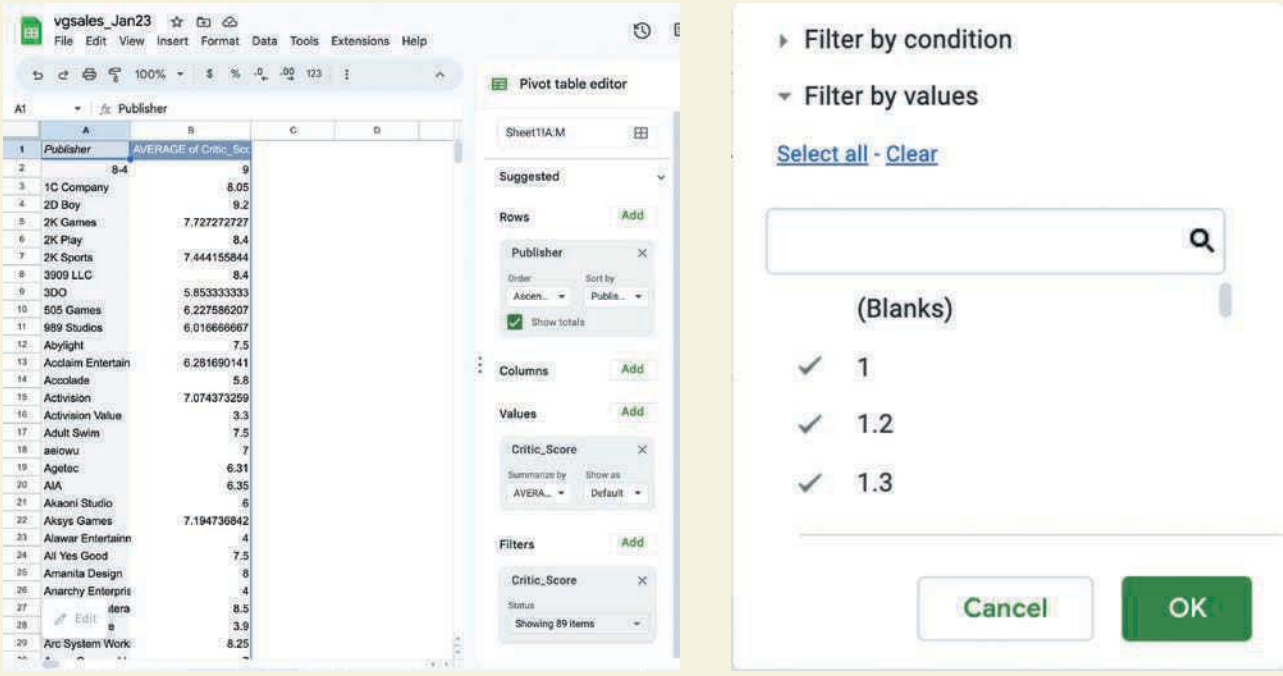


Figure 9.42 Filtering pivot tables.

4 There are still lots of publishers who we would consider insignificant. To determine the more prolific publishers to only include in our analysis, add a new value to the pivot table for publisher that will include a count of games per publisher. Reorder the pivot table based on the publisher count (descending) to see the average critic scores for the most prolific publishers (Figure 9.43).

You will notice the values we gained from our AVERAGEIF calculations for Nintendo (row 2) and Microsoft (row 27), but we can also see values for the many other publishers in the data.

Next, we will summarise the global sales per platform per year and graph this so we can see the trends in platforms over time.

Try this!

Double click on one of the cells in the table to generate a new sheet that includes all the data records that contribute to the calculation in that cell.

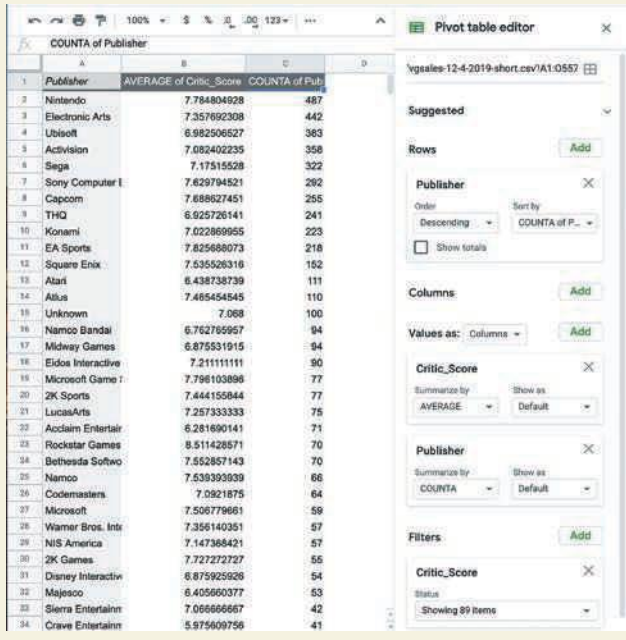


Figure 9.43 Pivot table showing the average critic scores of the most prolific video game publishers.

5 a Create a new pivot table with year as the row, platform as the column and SUM of global sales as the values. Filter to remove data with a blank value for Year.

You will notice that, even as a summary, there is a lot of data to take in visually (Figure 9.44).

b Graph this summary data to highlight some of the key values and trends.

Select cells A2:BW49. Then go to Insert > Chart. In the chart editor:

- change the chart type to Line
- scroll to the bottom and select 'Use row 2 as headers'
- remove the 'Year' series and add a horizontal axis for year.

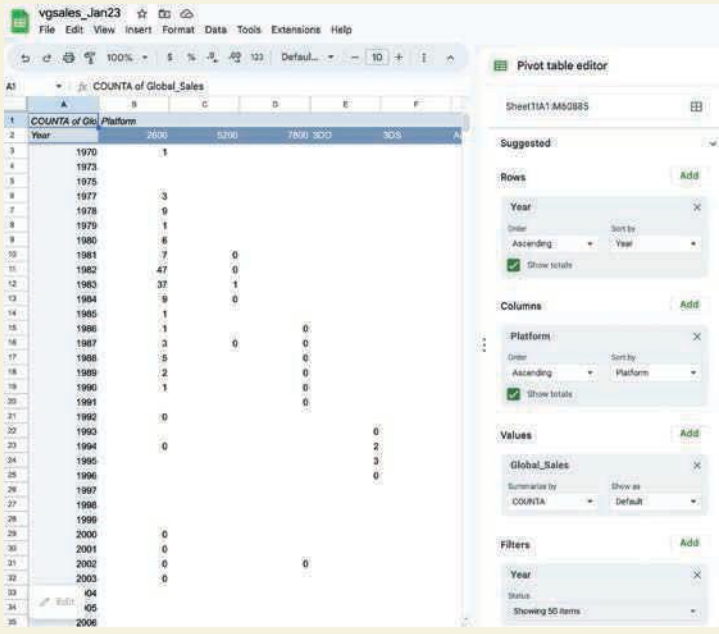


Figure 9.44 Using the pivot table editor to summarise the data on global sales per platform over time.

You should now have a very colourful graph that shows the trend in sales for each platform over time (Figure 9.45). This can easily show us which platforms were the market leaders at particular points in time.

Just hover your mouse over the coloured line at each of the 'peaks' to see which platform had the most global sales at that point in time.

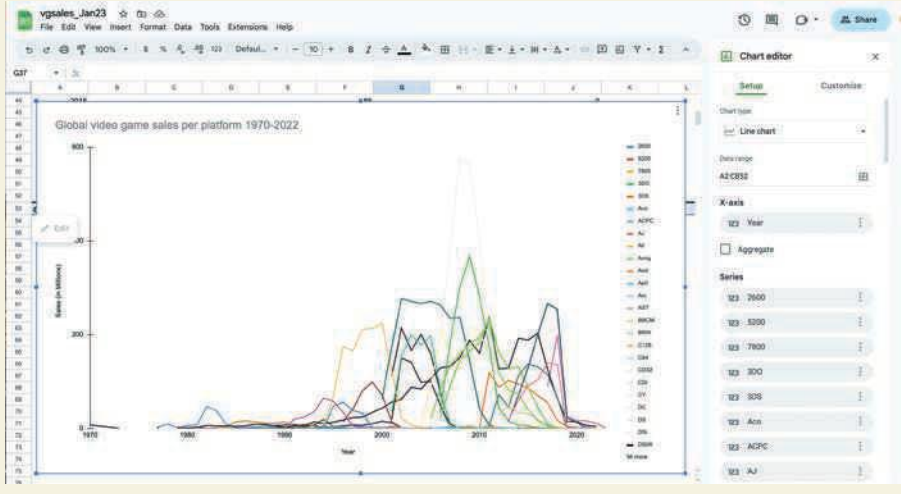


Figure 9.45 Using a line graph with many series to show the global sales per platform over time.

6 With 80 series, this chart is quite busy. Depending on what you are trying to communicate, it might be suitable to filter based on platforms or remove some series from the chart (Figure 9.46).

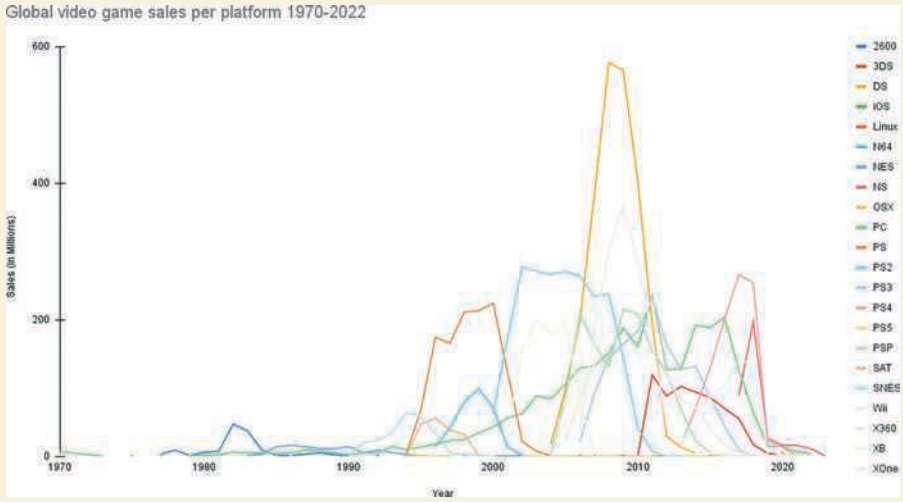


Figure 9.46 Platform leaders at various points in time were identified by hovering over peaks in the graph. Most other platform series were removed from the chart to make it less busy and easier to read.

ACTIVITIES

Identify

1 What is a pivot table?
2 Why are pivot tables considered one of the most powerful tools in spreadsheet software?
3 What are the features you need to define when creating a pivot table?

Analyse

4 Explore some of the pivot tables suggested by the pivot table editor. Was the suggested summary data of any use?
5 What other charts could we use to represent global sales per platform over time? Try one of these out. Do you think it better represents the data?

Extend

6 Explore pivot tables more using the video game sales data set, seeing what other interesting information you can extract and visually display.

9.8 Data dashboards

Spreadsheets do not just have to be where all the number crunching occurs; they can also provide the user interface for users just wanting to view the final results with some control over what they see.

What is a dashboard?

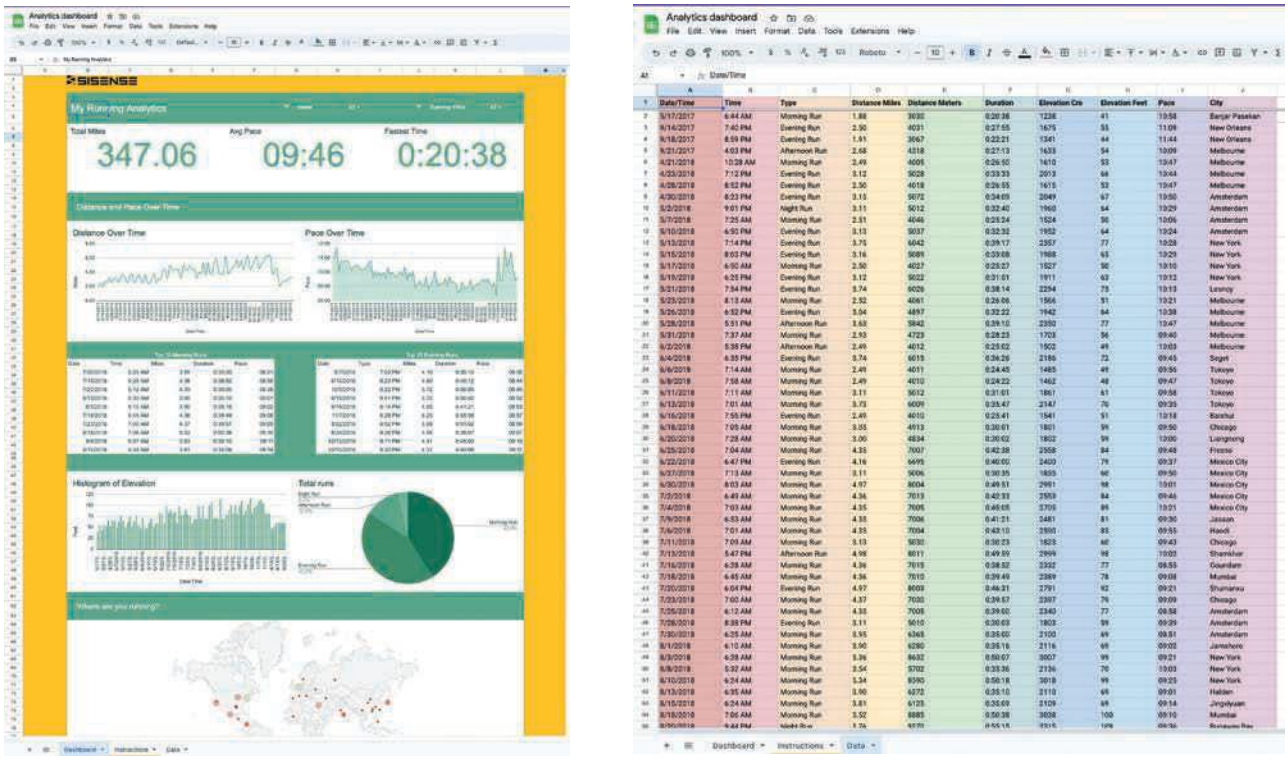
A **data dashboard** is a visual representation of data that allows users to quickly and easily understand complex information. They are typically used to communicate key performance indicators (KPIs) and other important metrics in a user-friendly way.

A spreadsheet dashboard can include various types of charts, graphs, tables and other visual elements that help users understand trends and patterns in the data. A dashboard may also include interactive features, such as drop-down menus or sliders, that allow users to explore the data in more detail, distinguishing them from a static report with the same metrics and charts.

Systems thinking

Spreadsheet dashboards are commonly used in business, finance and other industries to monitor and analyse data related to sales, marketing, operations, etc. They can help users make informed decisions by providing timely and accurate information in an easily digestible format.

Looking at the running analytics spreadsheet in Figure 9.47, the first sheet is the dashboard that visually summarises all the data in a table in the third sheet. In one view we can see key metrics such as average pace, and trends such as a gradual increase in distance over time. This dashboard uses a variety of charts and tables to achieve this. Slicers with drop-down menus in the top bar also allow users to easily filter the data so the dashboard displays a summary of a data subset (e.g. only morning runs).



Source: Google, Sisense

Figure 9.47 A Google Sheets template for creating an analytics dashboard prefilled with sample data.

Dashboard design

Dashboard design needs to carefully consider user interface (UI) and user experience (UX) design. Design principles such as balance, grouping, emphasis and consistency should be applied in order to make the information easy to interpret and to allow users to intuitively know what elements they can interact with.

Consider some of the design features in the dashboard in Figure 9.47. Colour has been used to group sections of the dashboard and to emphasise key numbers. These colours have been used consistently across the whole dashboard. Alignment and space have also been used to group related content and make the user's eyes naturally flow through the dashboard to take in the information. Size has been used to emphasise the key summary metrics and the titles of each visual element.

Appropriate charts have been used for particular attributes and relationships that best represent the information, such as a line graph to show change in numbers over time and a pie chart to show the proportion of value (running type) as a whole.

Microsoft Excel and Google Sheets have a range of templates available, like the one in Figure 9.48, to help you get started with a dashboard design. Other software used to make data dashboards, such as Tableau, include lots of features to make dashboard design easy.

SKILL BUILDER

Analysis of video game sales data – Part 7

Dashboard planning

1 Using online examples as inspiration, sketch a design for your dashboard on a piece of paper (Figure 9.48).

Label the key metrics and charts you will be including as well as any other key design features, such as colour and grouping, and any interactive functionality such as filtering options (slicers).

Design thinking

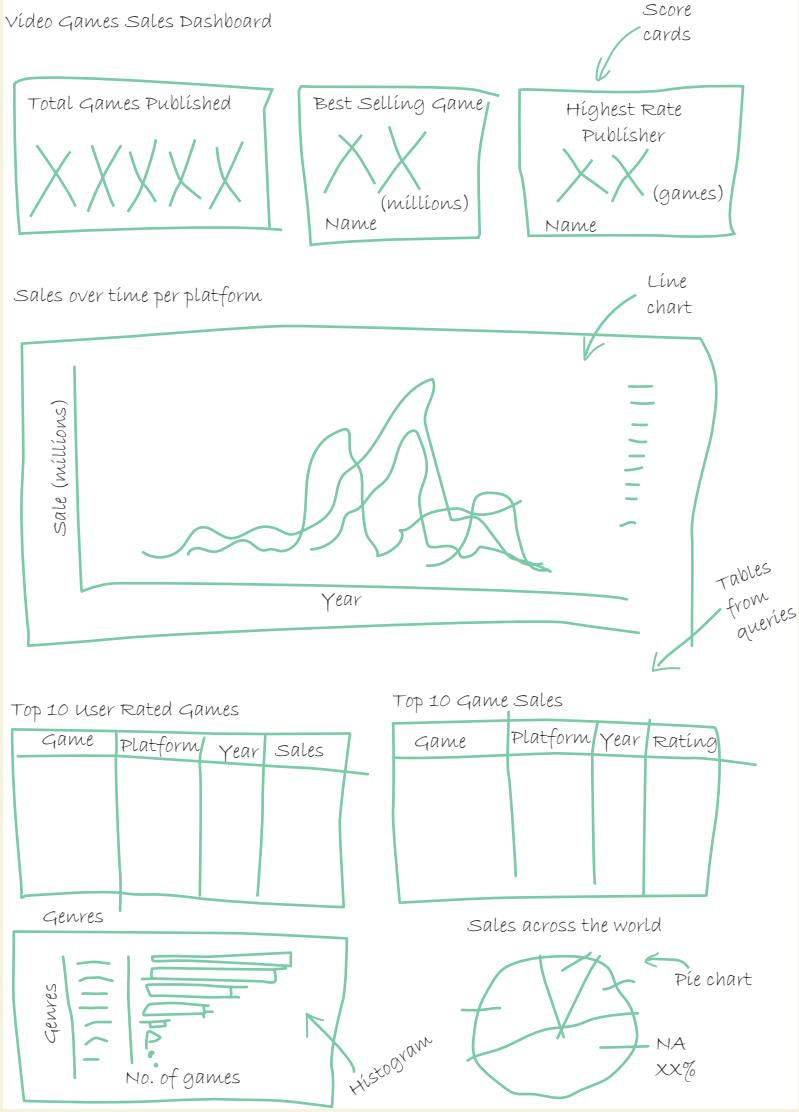


Figure 9.48 Sketch of the design for the video games sales dashboard. How could this plan be improved?

2 Create a new sheet in your spreadsheet file called 'Summary' (Figure 9.49). Here we will calculate, using functions and pivot tables, all the data we want to display in the dashboard.

For each metric and chart you would like to include, generate the summary data.

3 For example, for the biggest selling game, you will need to calculate the B4=MAX(global_sales) then use this value to identify the name of the game with those sales. This can be done using the MATCH and INDEX functions. =INDEX(game_titles,MATCH(B4,global_sales,0)) - MATCH finds the row number of the cell that contains that value and INDEX returns the value of a cell in an array of items based on a row number.

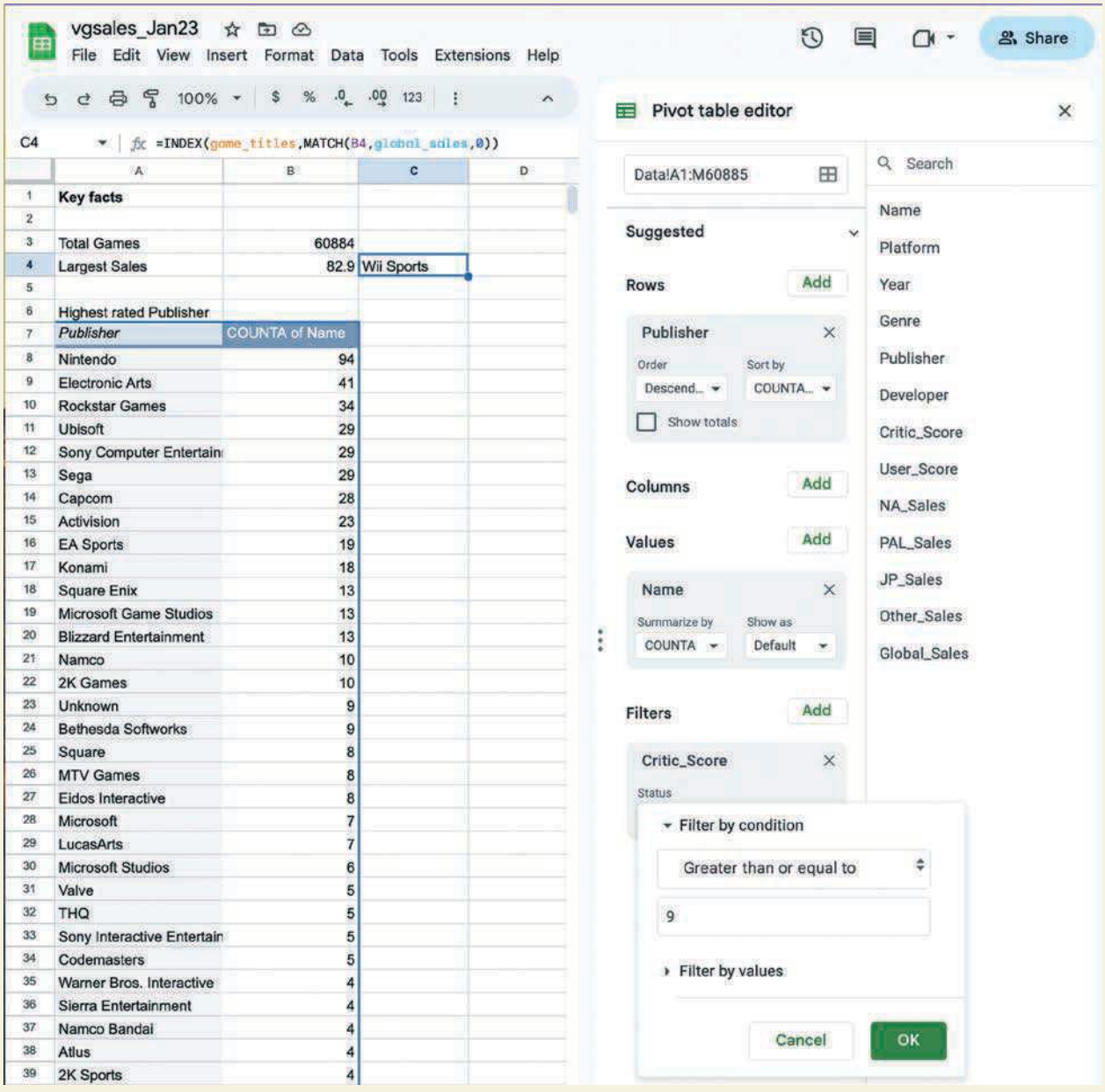


Figure 9.49 In a new 'Summary' sheet, perform calculations to find the key metrics for your dashboard. For the first three scorecards in our dashboard we need to perform a simple COUNT function for total games, a slightly more complex calculation using MAX, MATCH and INDEX functions for the biggest seller and a pivot table to find the highest rated publisher.

Dashboard production

1. Create a new sheet called 'Dashboard'.
2. Follow your design as you insert, modify and arrange the charts using the data from the summary tab (and perhaps from the main data tab) (Figure 9.50).
3. Use the QUERY function to create the filter tables, summarising the top 10 games by sales and user ratings.
4. For example, to create the second table, in the top left cell add `=QUERY(all_data, "select A,B,C,G where (M > 2) order by G DESC limit 10",0)`. This performs a query that returns data from columns A (name), B (platform), C (year) and G (user rating) where M (global sales) is greater than 2 million, ordering the results by G (user rating) in descending order and limiting the results to the top 10.

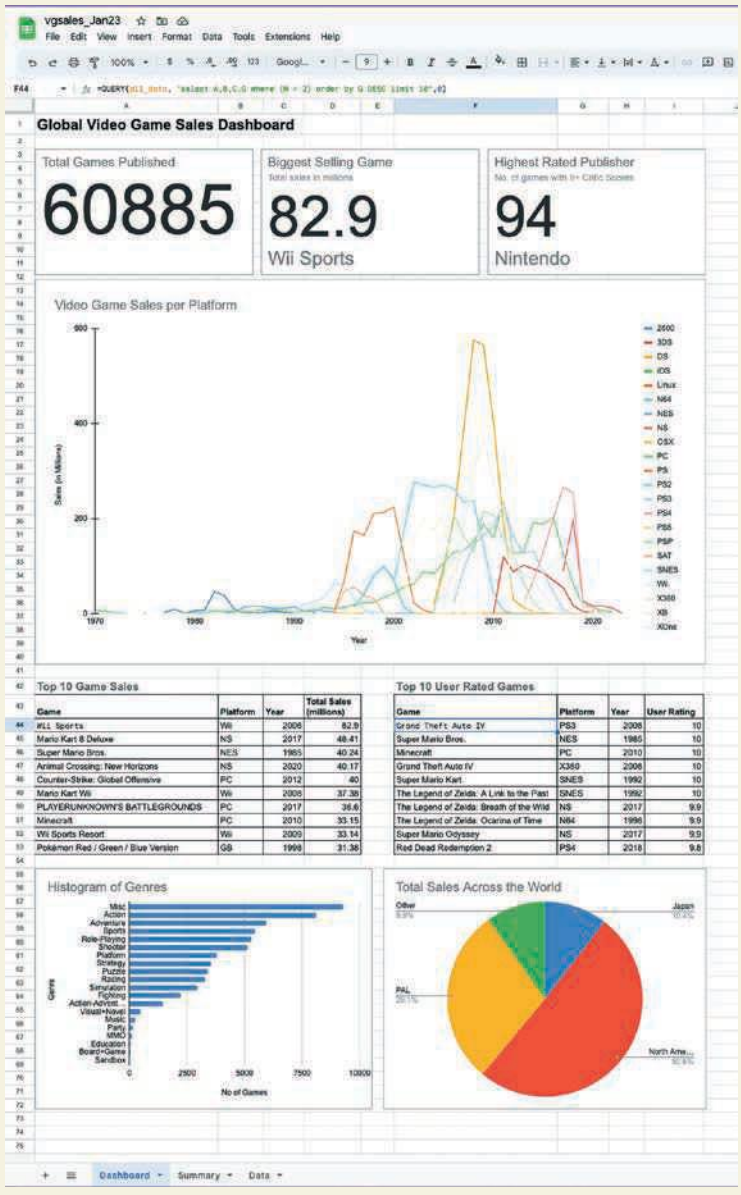


Figure 9.50 The dashboard tab includes scorecard, line, bar and pie charts, which are displaying key metrics and visual summaries calculated from the raw data set. Subset tables show top 10 results found using a QUERY function. The layout is well-balanced and has grouped related content together using alignment. Titles and labels have been included and styled to clearly communicate the information presented.

ACTIVITIES

Identify

1 What is a data dashboard and in what way(s) is it different from a report?

2 Describe three key UI design principles to consider when creating a dashboard.

3 Why is it important to add descriptive titles and labels in the dashboard?

Analyse

4 Evaluate the use of colour in the dashboard in Figure 9.50.

5 Explore the slicer functionality on the data analytics running dashboard from the Google Sheets template. Provide examples of its usefulness in presenting meaningful information.

Extend

6 Suggest two other charts that could be included in the dashboard, considering their placement carefully.

7 Suggest and implement another QUERY that could be included to create a sub-table in the dashboard.

PEOPLE

There are people today who are professional esport athletes. Although the big money and fame relates to competition in popular video games, there are people from all over the world whose esport is problem-solving in Microsoft Excel. In fact, there is an Excel World Championship and an Excel World Cup. Sometimes these are broadcast live on sports channels.

Could you be the next Australian or World Champion in Excel? Visit the MOS Championship website to find out how to enter and prepare for the next competition.

iStock.com/gorodenkoff

Figure 9.51 Esport athletes compete against each other in live or online tournaments.

Weblinks
Microsoft Office Specialist World Championship

Canva

9.9 System integration

Many information systems incorporate spreadsheet functionality and related processes. These systems need a way to integrate the spreadsheet software so data can be easily or automatically imported from or exported to other parts of the system. In this section, we will look at some ways spreadsheets are integrated into larger systems.

Google Forms

Google Forms allows users to create online surveys for collecting data. Although the Google Forms application displays summaries of the data including graphs, users may want to analyse the data further. The application allows this through an one-click **export** of the data into a Google Sheets spreadsheet, with each survey response stored in a new row of the spreadsheet. The file is dynamically linked to the form so future responses are also added to the sheet.

Infographics

An **infographic** is a popular method of visual communication of data. Infographics are usually composed of a collection of numbers, charts and icons, with minimal text to allow people to easily get an overview of a topic. Data and information from spreadsheets can be included into infographics through human processes such as exporting charts as images to be embedded in the graphic or by simply using summary data values to inspire the creation of text and imagery. Some infographic applications also include spreadsheet functionality so data can be entered into a table and automatically graphed in the infographic (Figure 9.52).

Systems thinking

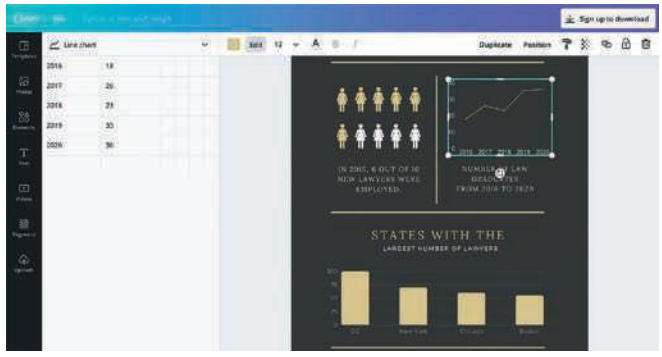


Figure 9.52 Online content creator Canva allows for the creation of infographics with graphs generated using the data entered in the simple embedded spreadsheet functionality.

IoT systems

Internet of Things (IoT) systems are characterised by embedded systems that collect data and share it with other systems via the internet. Many of these systems store data in the cloud using online applications such as Google Sheets. These systems are able to communicate and input data directly into spreadsheets using communication protocols called application programming interfaces (APIs).

Visualisations

Any chart or whole dashboard you make in Google Sheets can be published as a website or embedded into an existing website (Figure 9.53). These charts include the interactive features from the spreadsheet and update as the data from the spreadsheet updates.

APIs also allow for data to be automatically extracted from spreadsheets for analysis in other visualisation tools such as Gapminder.

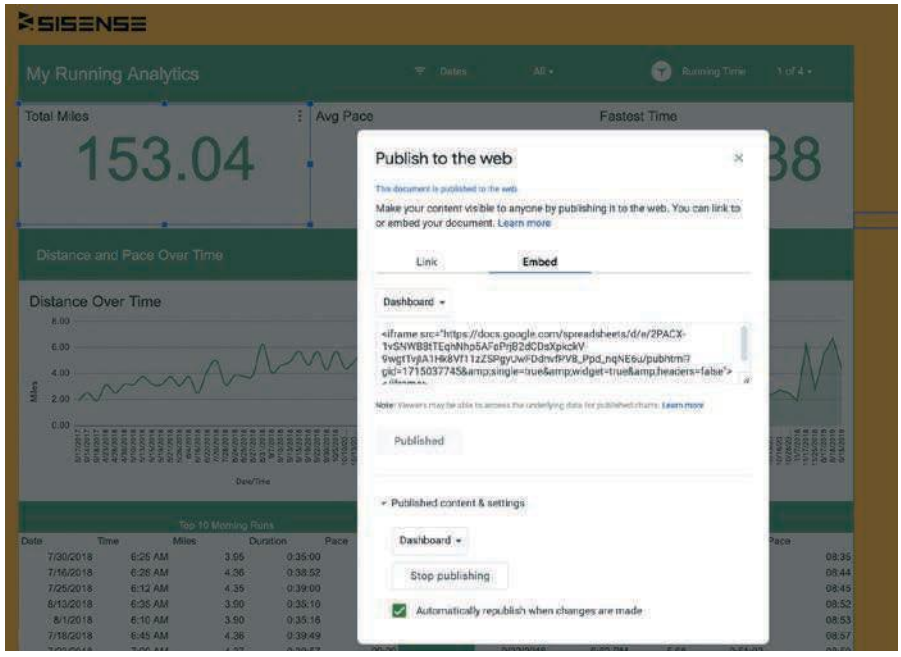
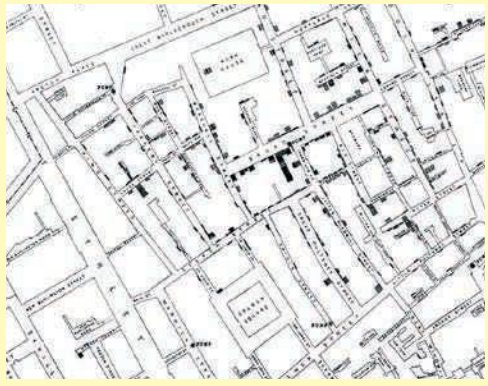


Figure 9.53 Select the three dots in the top-right corner of a chart to access the 'Publish to the web' window. If embedding in an existing website, copy and paste the iframe code within your HTML file.

INFOBIT

In 1854, long before spreadsheet systems, English physician John Snow demonstrated the power of data visualisations. He was in London investigating the deadly outbreak of cholera. Many experts theorised it was being spread through the air. However, when Snow charted the frequency of cholera occurrences on a map, it showed concentrated occurrences at properties near the Broad Street water pump (Figure 9.54). This realisation allowed him to identify the source of the disease spread and ultimately stop the outbreak. This method of disease investigation led to the new study of epidemiology.



By John Snow - Map of the book 'On the Mode of Communication of Cholera'; by John Snow, originally published in 1854 by C.F. Cheffins, Lith, Southhampton Buildings, London, England.The uploaded images is a digitally enhanced version found on the UCLA Department of Epidemiology website., Public Domain, https://commons.wikimedia.org/w/index.php?curid=2278605

Figure 9.54 John Snow's cholera visualisation revealed the source of the disease came from the water supply.

9780170442787

KNOWLEDGE BUILDER

Infographics

1 Examine the infographic that summarises the benefits of using infographics as a communication tool online (Figure 9.55). Doing an audit of the elements included in the infographic, we can see:

- large numbers representing key metrics, such as percentages
- icons coloured, grouped and sized to represent percentages or ratios
- graphs to display comparative data
- icons and imagery related to the fact (e.g. clock = time, brain = memory)
- consistent fonts and colours used throughout.

2 Find an example of an infographic about an environmental issue and do an audit of its content. Summarise your findings in a table.

Number of facts represented	
Icons/Imagery used	
Types of graphs included	
Colour scheme	
How are key words and numbers highlighted?	
Any interesting graphical representation of data?	

3 When creating infographics, it is important to provide the source of your information so the facts can be verified. Where does the statistic '65% of people are visual learners' come from? Is this reliable? Does the environmental infographic that you audited reference its sources?

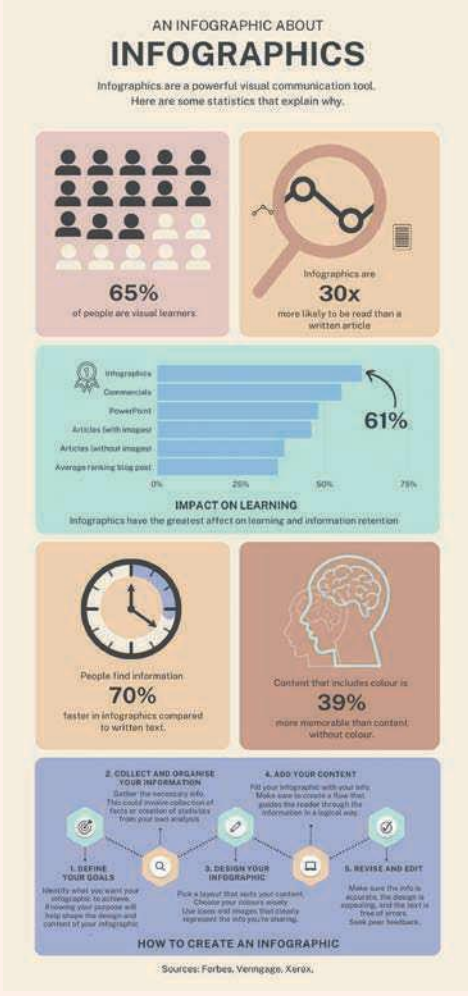


Figure 9.55 An infographic summarising the benefits of infographics as an online communication tool.

SKILL BUILDER

Visualising data using Gapminder

1 Visit the Gapminder website and watch the 'How to use' video.

2 Explore some examples of the various pre-made visualisations, including the *Bubbles* graphs that show five different variables at once, *Maps* that show the change in a location-based variable over time, and *Trends* that graph a variable over time for different regions, including countries (Figure 9.56).

3 Use the Maps tool to graph the yearly carbon dioxide emissions over time for each country. What does this visualisation show? There is a lot of evidence that the global warming trends you observed in Figure 9.33 are due to human action, including carbon dioxide emissions. How does this visualisation support this theory?

Weblink
Gapminder

Source: Gapminder

Figure 9.56 Gapminder is an online tool that allows the visualisation of large data sets and the comparison of many variables.

Chapter review quiz

Weblink
IFTTT

ACTIVITIES

Identify

1 What are some ways that spreadsheet functionality can be integrated into other information systems?
2 What is an infographic?
3 How do APIs allow the integration of spreadsheets into other information systems?

Analyse

4 Explore the IFTTT (If This Then That) website. This software allows the user-friendly integration of many IoT and cloud-based systems with Google Sheets. Find some example 'recipes' that use Google Sheets and explain what they do.

Extend

5 Create a Google form to gain feedback on your infographic from the Skill builder activity. Export the data to a Google Sheets spreadsheet and analyse it using some of the tools and techniques you have learnt in this chapter.

Weblinks:
- Australian Bureau of Statistics (BOM)
- Kaggle
- Bureau of Meteorology (BOM)
- Microsoft Office Specialist World Championship
- Canva
- Gapminder
- IFTTT
- Adobe Color

Skill builder resource:
- Analysis of video game sales data
- Simulating Mendelian genetics

Masterclass:
- Video game sales data

Resource:
- Aila's vegan cakes budget

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

9.10 Project: Large data analysis and visualisation

Harnessing the power of spreadsheet formulas, functions and other tools like pivot tables means you can calculate informative summary statistics from vast quantities of data in a few clicks. Representing those number- or text-based information points in a visual form such as a graph then allows further, sometimes hidden, insights.

The task

In this project you will access a large data set of your choice and perform a range of analyses on the data using spreadsheet software. You will then present key findings (aim for 10–12) in an infographic.

Depending on the time available and permission from your teacher you may be able to continue your analysis of the video game sales data for this project. Instead of using an infographic to display the data you may like to create a dashboard in the spreadsheet or arrange charts that you embed into a website, applying your HTML and CSS knowledge.

Identifying and defining

Tracking your project development

Set up a document to record the key processes during the project development. This document will likely show an iterative process, as you continually plan, produce and evaluate different parts of the whole project over the project's timeline. A logbook is an informal way to easily note down progress. If a more formal documentation report is required at the end of the project, your logbook can be used as a reference when creating a report.

Idea generation and identification of requirements

1 Brainstorm and select areas of interest to you (e.g. water usage, cyberbullying or Olympic Games results). Use the results to source data to create an infographic.

2 Define some of the functional requirements for your spreadsheet and infographic system based on your experience with the video games sales Skill builder, discussions with your teacher and the details in the description of the project (e.g. how many facts do you need in your infographic?).

3 Define some of the non-functional requirements, such as how up-to-date the data should be that is being represented and in what formats the infographic may be distributed and viewed.

Researching and planning

Dataset and information research

4 Explore web repositories (such as Kaggle) to see if there are datasets available for you to download. Download one, inspect the dataset and use the three Vs to evaluate the data. Experiment with a couple of analyses to see whether the data meets your needs.

5 Research similar infographics to get ideas about what data are represented and how they are represented. Summarise your research in your documentation, including screenshots of the analysed infographics.

Design planning

Design thinking

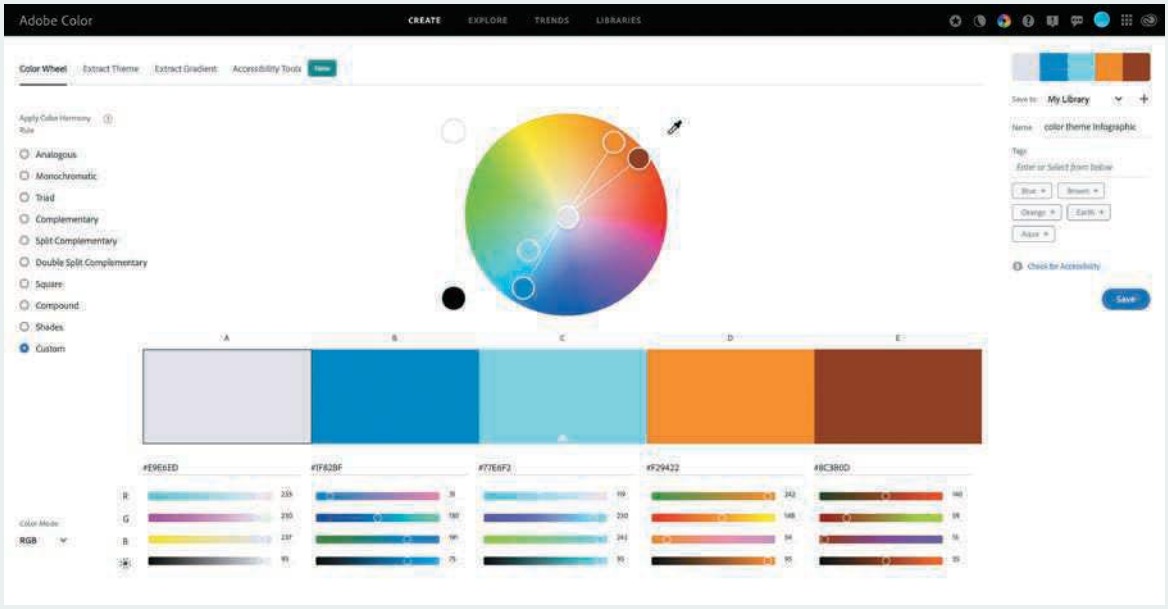
6 Identify tools you could use to make the infographic. You can make it from scratch using graphic design software, such as Adobe Photoshop or Illustrator, or you could use an online tool such as Canva, which automates a lot of design features but can be restrictive if you have a particular idea for the design of your infographic.

7 List ideas for icons, imagery and/or graphs you could use to represent each fact.

8 Select a colour palette for your infographic. Stick with three to five colours and use a website such as Adobe Color to help you pick colours that will work well together (Figure 9.57).

Weblink
Adobe Color

9 Sketch a layout design for your infographic, labelling the key facts, icons needed and colours.



Source: Adobe Color

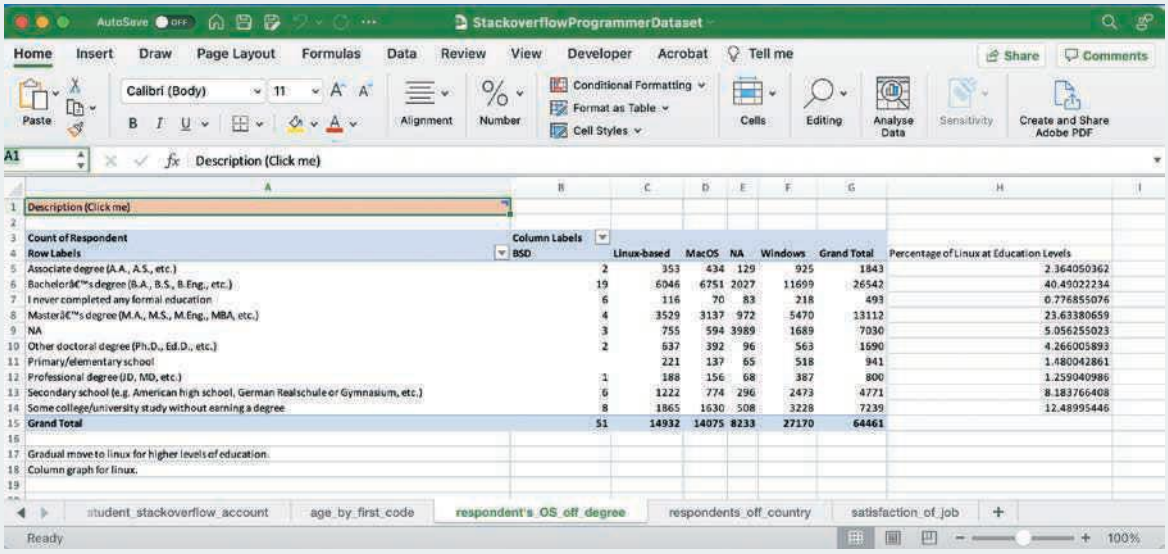
Figure 9.57 Adobe Color can be used to source, create and generate colour palettes for use in your design projects.

Producing and implementing

Data analysis and visualisation

Computational thinking

10 Analyse the data using formulas, functions, pivot tables and charts in order to produce all the identified information for the infographic (Figure 9.58).

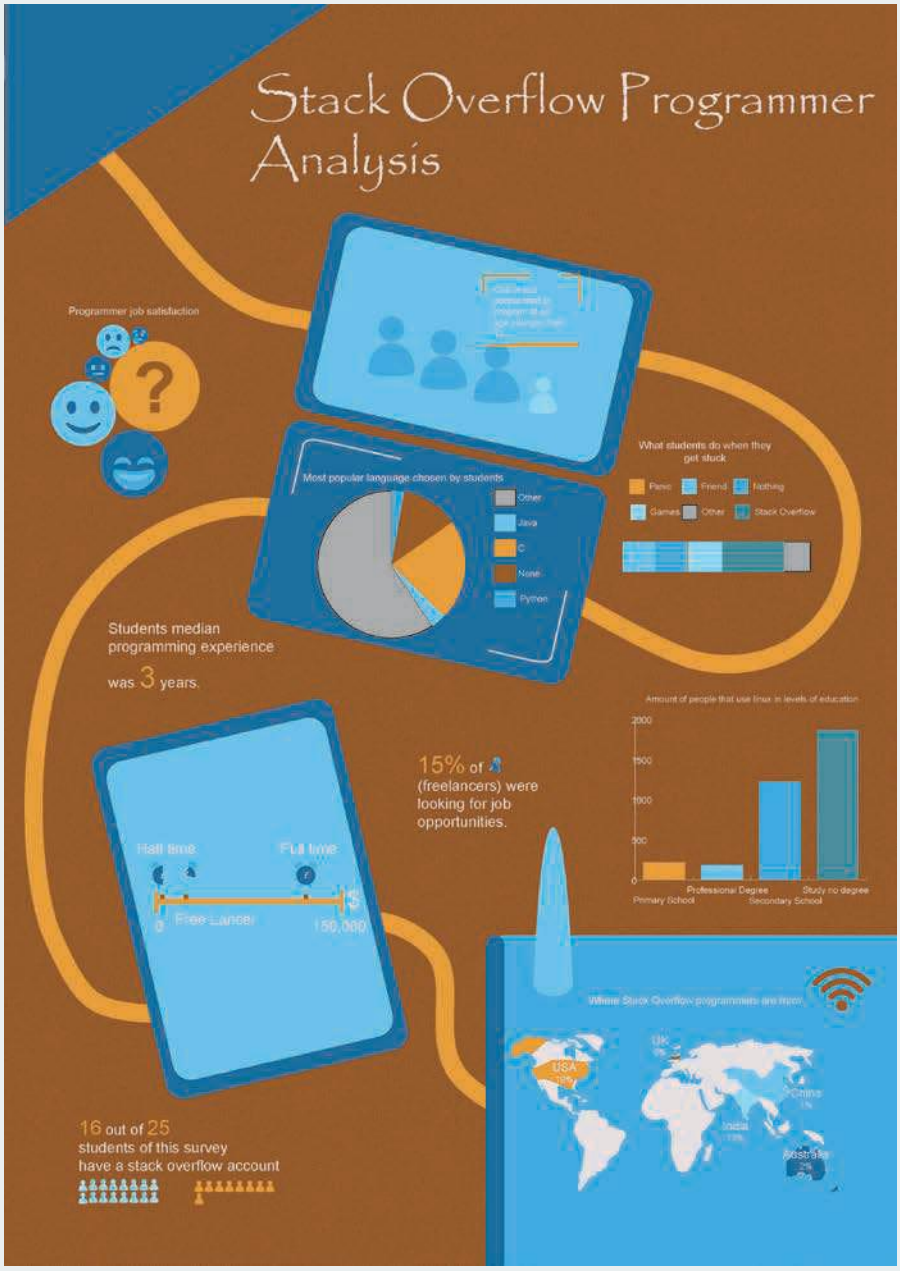


Robert Ng

Figure 9.58 Performing your various analyses over different tabs/sheets helps to organise your spreadsheet.

11 Source or create the imagery that you need for the infographic. Most icons can easily be found in a Google search (filter by usage). You can bring these into Photoshop or Illustrator to modify if needed (e.g. to change the colour). You are also welcome to create your assets from scratch.

12 If you want to include a graph, you can create this in the spreadsheet software, then export the image to include in your infographic. If you would like more control over the look of the graph, you can manually create it as a graphic in Photoshop or Illustrator. However, use the spreadsheet graph as a guide or trace or edit it in the graphic design software to ensure accuracy. Illustrator has a graphing tool that can be used depending on the type of chart.



Robert Ng

Figure 9.59 Example infographic that includes a range of statistics, visually represented using appropriate imagery and charts. What design features have also been effectively applied? How could the design be improved?

Composing the infographic

13 Place all of your elements together into an infographic, trying to maintain balance and consistency in the design (Figure 9.59).

Testing and evaluating

Evaluating and improving the infographic

14 Check that the infographic meets the requirements of accurately presenting key data and information and that it effectively applies the principles and elements of design.

15 Have some peers view your infographic. Use a Google form to gain some feedback data and information. Be sure to include questions that will result in quantitative data you can easily process, as well as qualitative data. This could include whether there is anything they do not understand or ideas on how the infographic could be improved.

16 Make changes based on feedback if necessary.

17 Finalise your documentation, including a personal evaluation of your final product and processes.

Systems thinking

Taking it further

Create a dashboard and embed it into a website, applying your HTML and CSS knowledge.

10 Analysing data using database management systems

FOCUS AREA: Analysing data

Chapter outcomes

In this chapter, you will learn:

- about the purpose and features of database management systems, understanding how they organise and manage data efficiently
- the key components of a database, including entities, attributes, relationships and schemas, and how they contribute to data organisation
- how to design and create a database, including defining tables, fields and data types, and establishing relationships between entities using schemas and data dictionaries
- various data collection techniques, including how to maximise data accuracy through validation and verification and social, legal and ethical issues relating to data collection and access
- to perform data analysis using queries, sorting, filtering and calculations to extract meaningful insights from your database
- the importance of presenting information effectively through forms, reports and visualisations, enhancing communication and decision-making based on database-derived insights
- about the concept and application of integrating data between different systems.

Key terms

data dictionary
data validation
data verification
database
database management system (DBMS)
entity
file
field
foreign key
form
mail merge
primary key
query by example (QBE)
querying
record
report
relationship
schema
structured query language (SQL)
table

IN ACTION

Data at the supermarket

Every day, millions of items of data are processed in supermarkets around the world. We are all familiar with the bar code that is attached to all products in the store, but have you ever thought about the chain of events that is triggered as each bar code is swiped past the scanner at the checkout (Figure 10.1)? In fact, a number of processes are begun to change this seemingly simple item of data into significant and meaningful information. Firstly, the numeric equivalent of the bar code is sent to the store's database to retrieve data about the item – most importantly, the name and price of each item. The display screen at the checkout lists the items that the customer has bought and the final total owing.

If the customer uses the EFTPOS (electronic funds transfer at point of sale) terminal to pay for the shopping, then banks are instructed to transfer money from one account to another. Details of customers and their transactions will be held in the banks' databases. The supermarket's accounting software will need to keep a record of the purchase, including customer details and the total amount.

Loyalty cards are often used to assure the customer that the supermarket is grateful for their patronage. However, in applying for the loyalty card, people have to supply their name and address. This data is then used for market research such as analysing the shopping habits of the individual.

Floor managers also have portable devices that can read bar codes of products and a keypad to check prices and stock levels on the shelves. Goods need to be ordered and delivered from the manufacturers and warehouses, usually in large delivery trucks. Predictions need to be made ahead of time about extra demand for certain products – for example, flowers for Mother's Day and chocolate eggs for Easter.

At the end of the day, the data in the database can be analysed – the floor manager knows how many items need to be replaced on the shelves, the store manager knows how many items need to be reordered from the distribution warehouse and the sales manager receives the feedback on any promotion that the store has been running. As the customer, you have a record of your purchases in the form of the shopping docket.

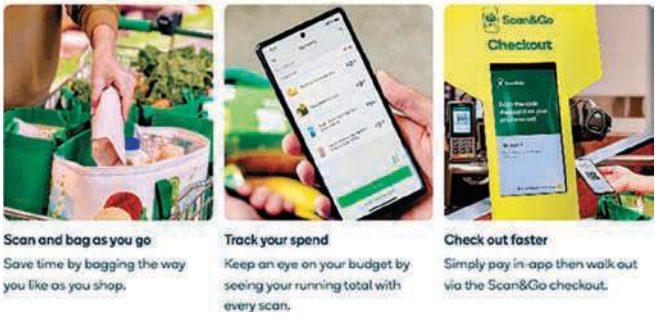
Many consumers are now using online services to do their shopping, often for groceries but also for clothes, holidays, gardening products and music. In each case, data needs to be stored and processed into information for the customer, supplier and manufacturer.

All these operations are possible because the data is organised and stored in databases. In this chapter, you will learn how information and software technology is used to build and maintain databases like these in a wide range of situations.

Shutterstock.com/Matylda Laurence

Systems thinking

Figure 10.1 Data input at the supermarket occurs both at the checkout, to identify products bought, and at the shelves, to check pricing and stock levels.



Source: Woolworths

Figure 10.2 New services are also allowing customers to scan their items as they shop and use a single QR code at the checkout to pay for their goods.

Questions

1. Draw up a table to identify the data about items for sale in a supermarket.
2. When an item's bar code is swiped at the supermarket checkout, what processes must happen for its price and name to be displayed to the customer?
3. There are many stages involved to put products on the supermarket shelf. Draw up a table to identify as many stages as you can. Include a list of people who would be involved in each stage. What data needs to be used at each stage?
4. Why do many stores use loyalty cards? What advantage is there for the business owners?
5. Many supermarkets now encourage customers to use self-checkouts and scan and go services (Figure 10.2). What are the positive and negative impacts of this? Discuss methods that are being used to address the potential negative impacts.
6. Investigate how many people in your class regularly use online shopping. What products do they buy, and from which companies?

10.1 Introduction to database systems

We live in an information age. Our lives are made easier by having volumes of information at our fingertips. We rely on information to save time in the shops, in the library and when travelling long or short distances.

Purpose of a database and database management system

We learn to find the information by looking in books or searching the Internet. Sometimes we find the information we need on a display screen at an airport terminal or on a printed slip from an automatic teller machine (ATM). However, we can only hope to find the information we need if it is where we expect to find it – and that means being organised.

A **database** is an organised collection of data. We rely on databases in many situations to provide us with the information we need in our daily lives. Some databases are simple and easy to build; others can be very complex.

In most cases, when the volume of data becomes very large, it is useful to put the database onto a computer. A school library, for example, would have more books than your personal bookcase library and so would rely heavily on a computerised database.

This has several advantages:

- The data can be organised and therefore accessed more easily.
- Storage space for the data is greatly reduced.
- Human errors in collecting and accessing the data can be avoided.

A **database management system (DBMS)** is a software application that allows users to define, record, edit, analyse and display structured data in a database (Figure 10.3). This allows for entities (e.g. book, person), events (e.g. borrowing) and their attributes (e.g. title, ISBN), as well as the **relationships** between them (e.g. a person borrowing a book), to be modelled.

Uses of databases

Databases are used in nearly all industries and many aspects of our personal lives. Due to the integration of databases into the user-friendly designed information systems, most people have an indirect interaction with them, and consequently do not appreciate their widespread use. The following are some common uses of databases we can relate to in our lives:

- Banking and e-commerce – for example, Amazon manages data about products, its users and their purchases.
- Hospitality – for example, hotels manage data about their rooms, guests and their bookings.
- Health – for example, your doctor manages data about your personal details, details from your visit and results from any medical tests.
- Sport and recreation – for example, a football club manages data about players, teams, games and results.
- Employment – for example, the human resources department in a company is responsible for managing data about employees, their working hours and their pay.
- Education – for example, schools manage data about students, teachers, classes and classrooms.
- Social media – for example, Facebook manages huge amounts of data on people, posts, events and groups.

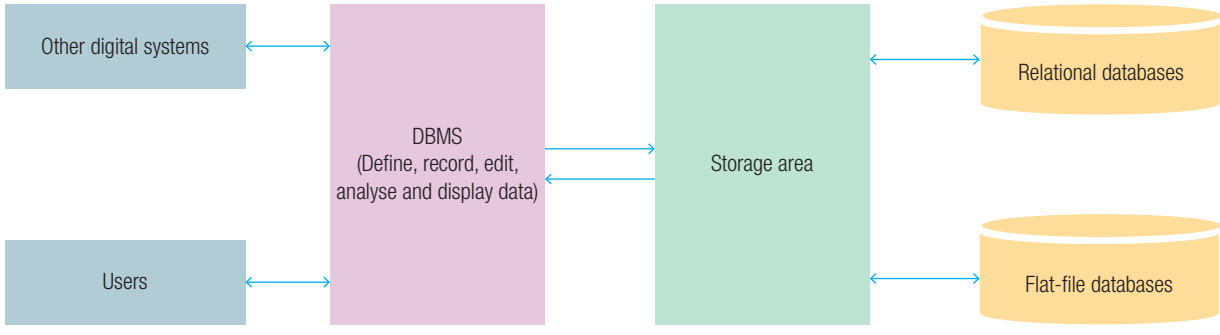


Figure 10.3 A database management system inputs and outputs data and information to and from users and other digital systems. The actual data is on computers, including large servers, organised in a flat-file or relational structure.

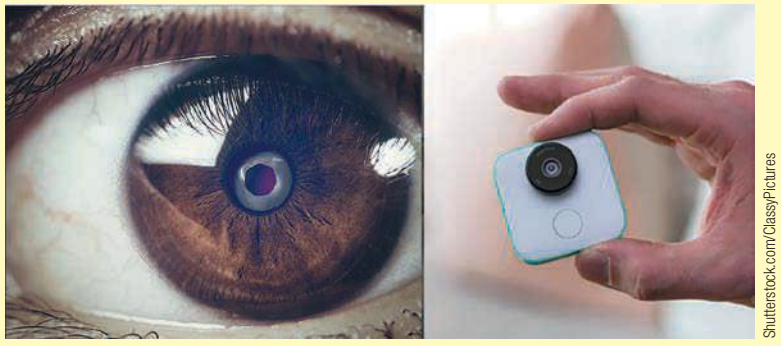
9780170442787

DBMS versus spreadsheet software

In Chapter 9, you learnt how data can be organised into rows, columns and **tables** using spreadsheet software. Generally, when we are working with data about a single thing, a spreadsheet application will provide all the functionality we require to build and analyse a simple, flat-file database. As soon as we have data about multiple related things we should use a DBMS that has the functionality to deal with the complexities of relationships.

INFOBIT

In tomorrow's world, digital storage may be so cheap and sensors so small that we will live 'documented lives', wearing tiny 'life cams' that create video and data streams of our entire life experiences. The information will be stored in databases and auto-archived for instant access at any time (Figure 10.4). A dystopian view of this future has been presented in fictional television, such as *Black Mirror*. Google Clips was a real-life artificial intelligence (AI) product released for this purpose; however, it has been discontinued as users did not find the documentation of their everyday life that useful.



Dreamstime.com/Ricardo Pascual

Shutterstock.com/ClassyPictures

Figure 10.4 Recording and storing our everyday lives in video databases could be useful, but will involve many social and ethical issues that must be considered.

SKILL BUILDER

School cafe DBMS – Part 1

Understanding the data storage and organisation

Strawberry Hill High School recently established a student-run cafe. The cafe was the initiative of the school's careers adviser and some students on the Student Representative Council. They saw the opportunity to provide students with hospitality experience, while also providing staff and senior students with a service that could build positivity about their day at school.

With a minimal budget and limited technology expertise, the cafe was initially set up and run with basic data logging in spreadsheet software for stock and the use of paper-based data storage and organisation, including customer loyalty cards and student rosters. The cafe only sells drinks but does have plans for selling baked goods in the future.

During the first year, the cafe has become increasingly successful, resulting in greater data processing demands and issues with errors due to its paper-based components. The student managers have identified the need for a complete digital system. With the use of a digital system they also see the potential for additional processes that could be used to improve efficiency, customer experience and the further success of the cafe.

1. The main thing (or entity) the cafe keeps information about is drink orders. Identify what data about an order is important.
2. Use spreadsheet software to create a simple flat-file database about the cafe orders. Across the top row in the sheet, add headings for each bit of information you are storing about an order.
3. Add 10 test records to the database, filling in the next 10 rows of the spreadsheet. Include multiple orders from the same customer.
4. Create a filter for the table of records created (Data > Create a filter). Use this to perform some simple analysis on your flat-file database, including sorting and searching (via filtering).
5. How could you use spreadsheet functionality to find customers who had ordered five drinks and were therefore eligible for a free drink? Where could this information be stored? How would a spelling error in a person's name impact this calculation?

This activity should demonstrate the limitations of spreadsheet software when managing a relational database. In the remaining Skill builders you will define, plan, build and test a DBMS for the cafe using Microsoft Access.

Skill builder resource
School Cafe DBMS

ACTIVITIES

Identify

1 What is a database? What is its most important function?

2 Describe some of the advantages of using a database.

3 What is a DBMS? How is it different from the database itself?

4 List some of the databases that you would use in your daily life.

Analyse

5 You would expect a vet practice to have a computerised database. What is the advantage of this for the practice owner and staff? How would it benefit the clients? What entities would be in this database?

Extend

6 Find an example of an online database. What data does it include?

7 Does your school still have a paper-based/physical database of student records? If so, why?

CASE STUDY

Information in an emergency

Doctor Zoë Knights is a specialist in accident and emergency medicine, and works in a variety of hospitals around Sydney. She needs to access and communicate information about patients quickly and accurately (Figure 10.5). When a patient first arrives at the emergency department, the triage nurse allocates a category number to indicate which cases are most urgent. This helps Zoë to prioritise the patients.

The reception staff enter all relevant details into the hospital administration system database. If the patient has not been admitted to the hospital before, they will be allocated a medical record number. This identifies the patient throughout the time they are in the hospital, and will be used again if they are admitted in the future. Alternatively, the patient's record can be found by searching for their name and date of birth.

After treatment, Zoë types her diagnosis into the computer, as well as any instructions for further care. The patient may be sent home, admitted to a medical ward or prepared for surgery.

Some patients might need a computed tomography (CT) scan, a blood test or other tests or treatments. These results can be entered directly into the patient's record by the radiologist or the pathology laboratory. It may be critical to refer back to this information while the patient is undergoing further treatment. This information might be needed for a consultation with a specialist in another hospital or another town. CT images in the database can be viewed remotely by authorised personnel and options for treatment discussed.

In addition to the important medical notes for each individual patient, a database is built for the entire Australian hospital system. This allows analysis of data over many years, such as the number of patient admissions and the types of emergencies that were treated. This data will affect the funding of hospital systems and planning for the future.

Alamy Stock Photo/mjmediabox

Figure 10.5 Emergency vehicles need access to critical data fast!

Questions

1. Why does Zoë need to access the information in the hospital database quickly?
2. Who is responsible for entering the patient's details into the hospital's database?
3. Why is it important for the patients to have a unique identification number in the database?
4. How are specialists in other hospitals able to view the CT images? Why is this an advantage?
5. Visit the Australian Institute of Health and Welfare website and discuss the range of data that is contained in the National Hospital Morbidity Database.
6. Look at the graph in Figure 10.6. How do you think this data was collected? What does it tell you about the number of treatments for older people? What do you think could be done about this?

Weblink
Australian Institute of Health and Welfare

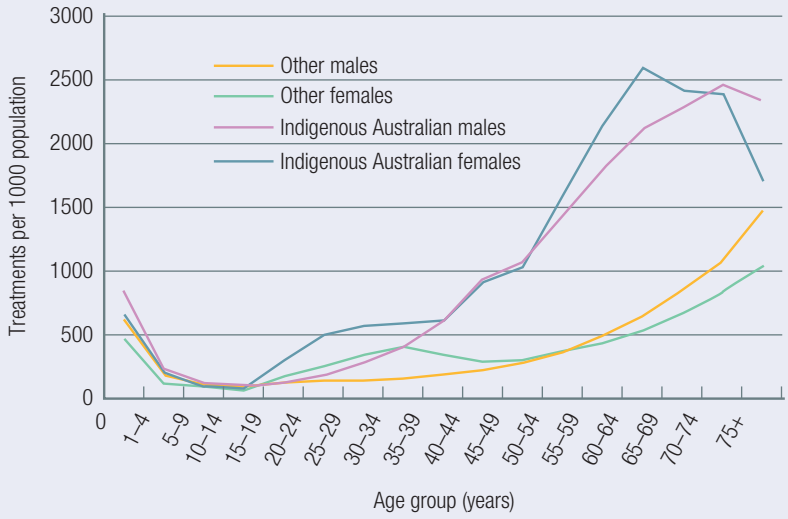


Figure 10.6 Collecting data in the emergency department enables decisions to be made about future needs.

10.2 Building a database

You access and change databases every day – for example, every time you tap your travel card to get on the train or select a song from iTunes or Spotify. However, as a user your interaction is facilitated by a user-friendly interface that hides the complexity of the data structure behind.

Consequently, we are generally not aware of the enormous amount of work involved in collecting and organising the data. We will understand more about this as we work through this chapter.

To start building a database, we need to understand the data structure. We must therefore start to think about categorising our data and understanding the data we need to collect and store within each category. For example, if building a database about songs, such as on iTunes or Spotify, the category of our data is a song and the pieces of data could be title, artist, genre, length and so on.

The things we store data about are called **entities** (a real-world object or concept) and events (an occurrence or happening). For example, a song is an entity and the occurrence of someone streaming the song is an event. The pieces of data about the entity or event are called attributes. For example, a title is an attribute of a song entity and the time played is an attribute of the stream event.

In Chapter 9, you saw how you could organise data in a spreadsheet. You could create a simple database where all data is in one table using spreadsheet software. This is called a flat-file database. In more powerful databases, we use DBMSs to edit, sort and search the data as well as displaying it in different ways.

Components of a database

Think about your school library database. It would have a table of data about the books (entity) plus a table of data about the students (entity). A third table would record the loans of books by students (event). These entities and events obviously have different types of data, so we need separate tables.

We can link the tables so that we can display more details about the students or the books on loan. Organising the data in this way makes the database a relational database.

Tables in a database are generally referred to as **files**: a collection of information about one type of object. One database might contain several files.

A database file is a collection of **records**. A record is the information about one object in the file, such as one book or one student. Records are usually viewed as rows in a table (Figure 10.7). Each record is divided into **fields** that hold a single item of data about the object. A field might hold the book's title or publisher. Fields are usually viewed as columns in a table. Figure 10.8 shows this hierarchy as a chart.

It is important that we also include a field in each file to hold a unique identification code for each record. This field is called a **primary key**. A library may have several copies of the same title, or two books may have similar names. If each physical book has a unique number code or ID, it cannot be confused with any other book.

Entities in a database can be related in one of three ways: one-to-one, one-to-many and many-to-many. Figure 10.9 demonstrates these relationships. Many-to-many relationships are organised in DBMS by creating a junction table between the two entities. Relationships are created by linking the primary key (PK) in one table to the usually identically named foreign key (FK) in the other table. For example, Student_ID (PK) in the Student table is linked to the Student_ID (FK) in the Loan table.

In order for the DBMS to accurately store and process the data, we must define the data type in each field. Most data that we enter into the tables is text or numbers. For example, a book's author (text) and its price (number). Numeric data can be formatted as currency (such as price) or dates if required. Another simple data type is the Boolean or logic data type that indicates a true/false type of value. This might be used to indicate whether a book in the database is on loan or whether a return is overdue.

Other data types that could be stored in a database include graphics or other digital media files. However, it is important to remember that all the data is stored in memory and secondary storage in digital form, so large graphics or audio files will use more storage and processing power.

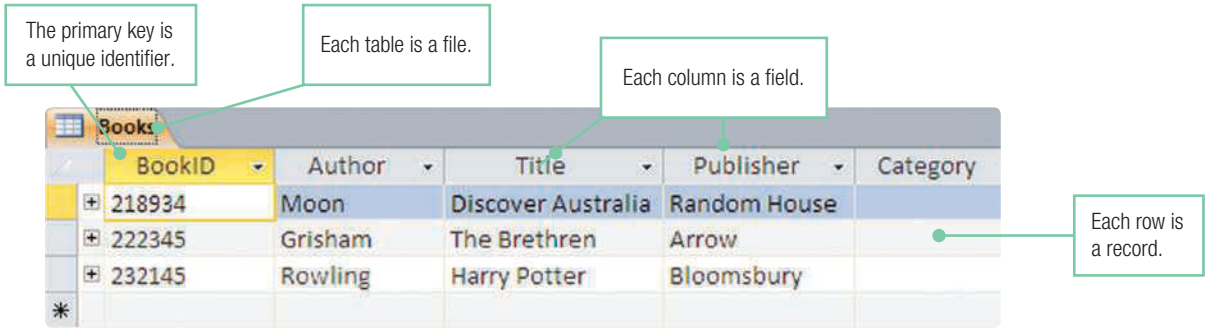


Figure 10.7 A books file in a library database displayed as a table.

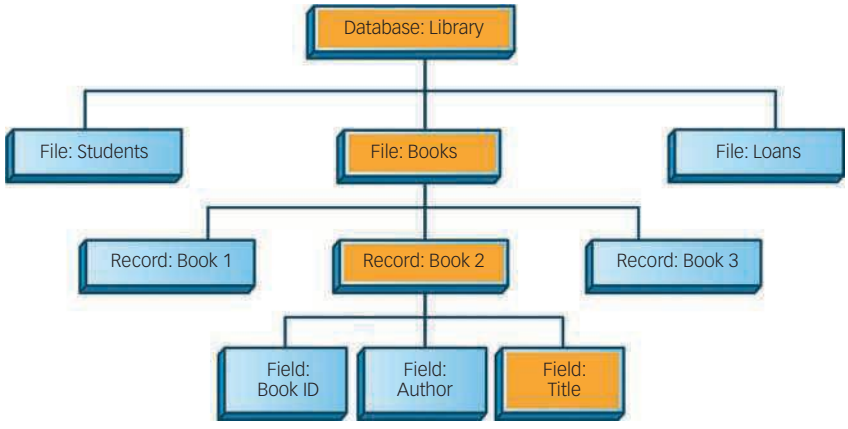


Figure 10.8 The hierarchy of components in a database structure.

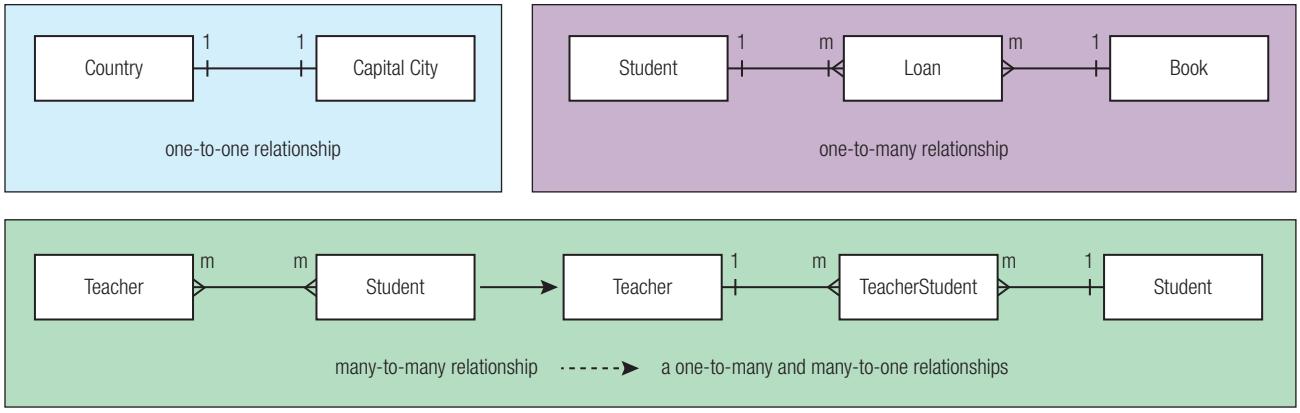


Figure 10.9 Although there are three types of relationships, relational databases generally use one-to-many relationships. One-to-one relationships are not common as most information that is related in this way would be in one table. Many-to-many relationships will be converted to two one-to-many relationships using a junction table (e.g. TeacherStudent).

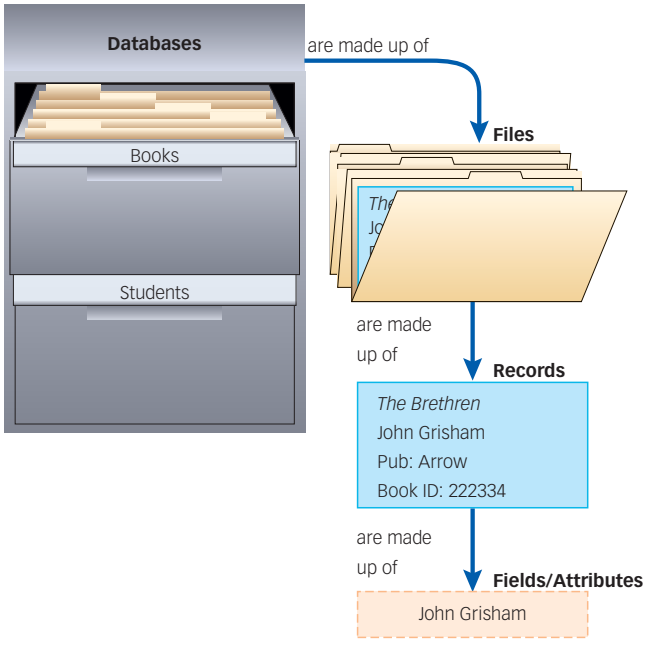


Figure 10.10 A database is like an electronic filing cabinet.

Tools for planning a database

Diagrams and other documentation are effective at communicating the planned components of a database. It is essential at this point that the data be organised correctly in the different tables and that data types are appropriately assigned. As the project progresses, it is more difficult to implement major changes in the design. Two important documentation tools are **schemas** and **data dictionaries**.

A schema or schematic diagram summarises the tables, fields (including keys) and relationships in a database. A rectangle represents each table and includes the object name and fields, including keys. Rectangles are linked with a labelled line showing the relationship between the tables. Schemas can be sketched by hand or mocked up in drawing software, such as diagrams.net.

A data dictionary illustrates and describes the fields and data types that will be required in the database. It also includes field size to show the specific amounts of memory that will be allocated by the DBMS.

Many DBMSs create schemas and dictionaries for you automatically. These can support in the development of the database, such as the linking of tables in the schema, or allow you to add some functionality to your database, such as defining formatting and validation rules for input data in the data dictionary. Figure 10.11 shows a data dictionary created for planning and in a DBMS.

INFOBIT

Some databases are designed to be closed or 'read-only'. This means that it is not intended that the data will be altered by the user. For example, the database that is issued by the Australian Bureau of Statistics after each census would not require updating.

Field Name	Data Type	Field Size	Example
Student ID	Number	7	2001335
Family Name	Text	30	Smith
First Name	Text	20	Mary
Date of Birth	Date	8	25/04/87
Year Group	Text/Year	2	10

Students

Field Name	Data Type	Description
Student_ID	AutoNumber	Code allocated by School Administrator e.g. 890243
Family Name	Text	
First Name	Text	
Date of Birth	Date/Time	Format dd/mm/yyyy
Year Group	Text	E.g. Year 9

Figure 10.11 Data dictionary for a student's file, as a table and as seen in Microsoft Access.

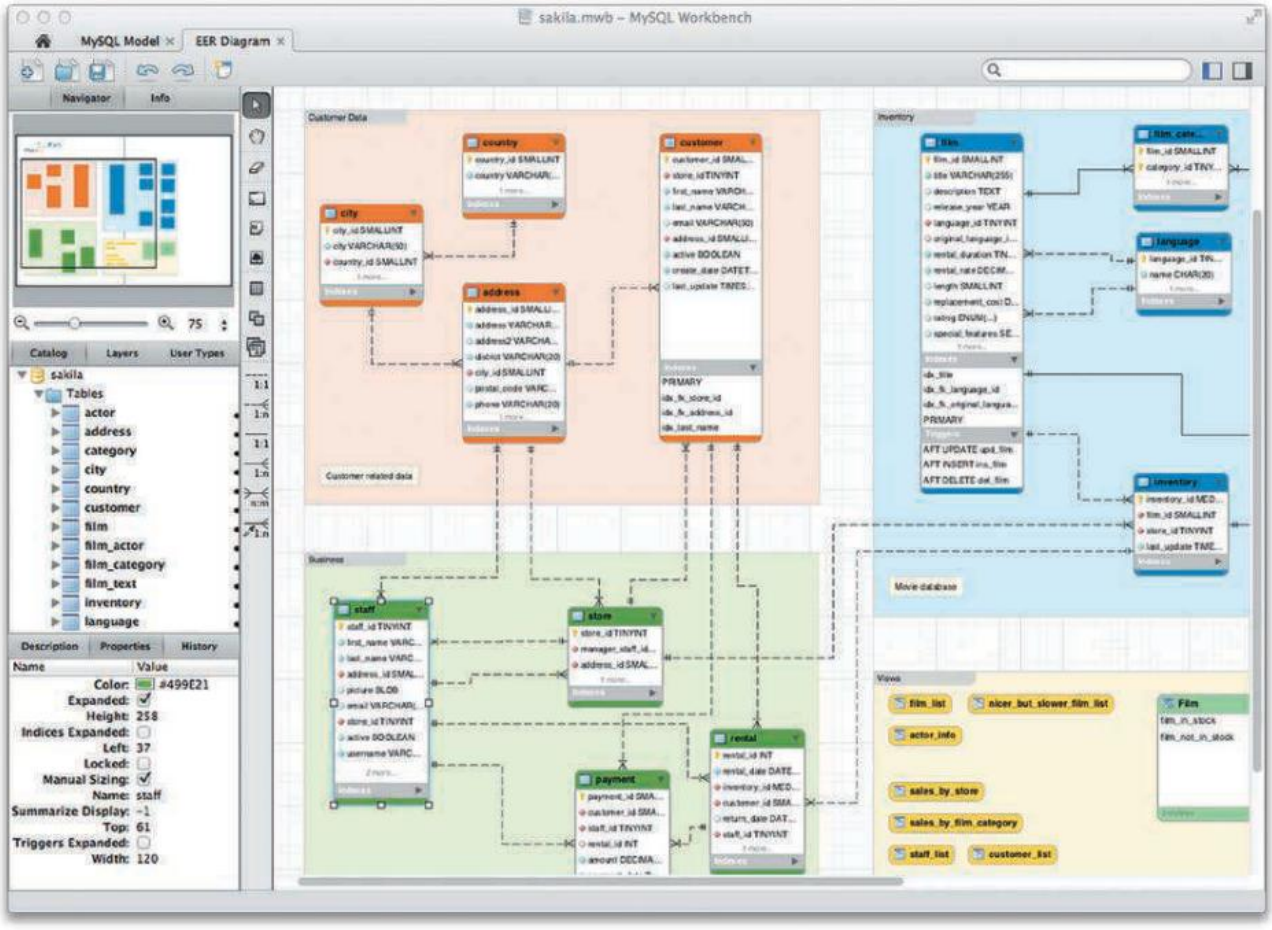


Figure 10.12 The interface for the MySQL workbench has similar functionality to Microsoft Access, such as allowing the user to view and edit the tables and relationships.

DBMSs for building a database

There are many DBMSs that allow the building, editing, **querying** and reporting of a database. The platform on which you intend to use the database and the other systems you need to integrate with will influence your choice of software.

Microsoft Access

Microsoft Access is a user-friendly DBMS that many people are familiar with. It is commonly the database application selected for teaching database design due to the simple interface and controls, and, because the files are stored directly on a user's computer, it does not require setting up or linking to a server. However, Access does have limitations. As it is a desktop-run application, it does not support multiple users or integration with web-based systems. Even as a desktop application, it only runs on Windows. Any individual or company requiring an application to support a database behind a website or other online digital system without setting up their own server will need a different DBMS solution.

MySQL

MySQL is a relational database management system including a server. It is open-source, very powerful and widely integrated with many other systems, making it the most popular DBMS in the world. Big database-driven web applications, including Facebook, Netflix, Airbnb, Uber and Amazon, use MySQL. Although MySQL does not have a graphical user interface (GUI), a separate application can be installed on a desktop computer (Windows, Mac or Linux) to allow user-friendly interaction (Figure 10.12). The relatively complex processes associated with setting up and accessing a database on a server make it less accessible for people first learning databases. However, the functionality benefits over Microsoft Access mean most people in the industry invest in learning how to use it.

In this chapter, we will be focusing on learning the principles and skills of database design and management using the easier-to-learn Microsoft Access. However, your teacher may choose to extend these activities to learn another DBMS or you may like to do personal research to develop your own skills. There are many online tutorial services taking users through the steps to develop a MySQL database.

SKILL BUILDER

School cafe DBMS – Part 2

Plan the database

This database is similar to the library database and many databases that store data about a transaction. We will follow a process to guide the planning of the database tables, fields, keys and relationships, and data types.

1. Identify the tables in the database: For the cafe database we will start by assuming data is being stored about the entities *Customers* and *Drinks*, and the event *Orders* the customers make to get a drink.
2. Determine the relationship between the tables. Similar to the library system, an Order connects the Customer and Drinks table. A customer can make many orders and a drink can be associated with many orders. In this context, we will keep orders simple; they can only include one drink and be associated with one customer. Figure 10.13 shows these relationship links.

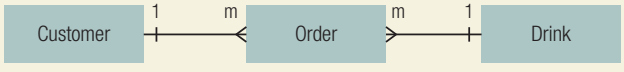


Figure 10.13 Relationship between tables.

3. For each table, identify the fields – what collection of data needs to be stored about each object?
   - **a** All tables need a primary key. A good convention is to name the primary key after the table, with 'ID' (short for identification) at the end. For each record, this will be a unique value.
   - **b** In order to link tables, we need **foreign keys**. In this case, as in the library database, the order will contain foreign keys for the Customer and Drink associated with the Order. In general, in a one-to-many relationship, the foreign key in a table can only be linked to one record from the other table (that is, the Order table can only be linked to one Customer and one Drink). Use the primary key name from the other table (CustomerID, DrinkID) as the name of the foreign key in the Order table.
   - **c** Customer: They could be a teacher or student. We would need to know their name, a way to contact them (email and homeroom makes most sense in this context), and any other attributes that may be useful for this system.

d Drinks: The type of drink (for example, flat white, cappuccino, hot chocolate or tea), description, base price.

e Order: The order table already has its primary key and the two foreign keys. Common fields to include about transactions like this include the date they occurred, details about the order (milk, size, sugars), how the order will be delivered (delivery, drink in or take away), any discounts/vouchers, other instructions, total cost and method of payment.

4 We will summarise the first three steps in a *schema diagram*. Copy the template below (Figure 10.14) into your book or a suitable software application. Complete the schema diagram using the information above.

Finally, construct data dictionaries to describe the fields in each table, including their data type and memory size. The Drink table data dictionary has been done for you (Table 10.1). Use the same template to construct the Customer and Order dictionaries.

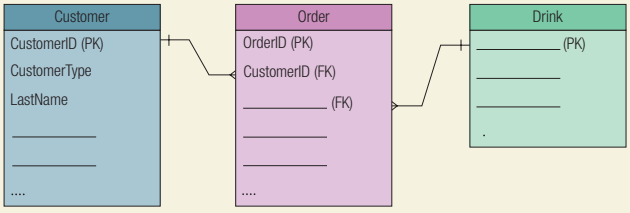


Figure 10.14 Schema diagram for the first three steps.

Setting up the database in Microsoft Access

If you have planned your database carefully, building it in Microsoft Access should be straightforward.

1 Open Microsoft Access and create a new database. An empty table will have been created for you. Select *View > Design View*. You will be prompted to name the empty table. Call it 'Customers'.

2 Use this data dictionary view to add and describe the fields for your customer table. The primary key's data type is set as AutoNumber by default. This means it will generate a unique number for each new record.

3 Repeat for the Drink and Order tables.

4 You should now have a database with three tables outlined in the Object panel and tabs for each open table in the main window (Figure 10.15).

5 Link the tables: Go to Database Tools and select Relationships. Select all three tables and click 'Add'. Drag and drop to draw the relationship links between Customers:CustomerID and Orders:CustomerID, and Drinks:DrinkID and Orders:DrinkID. Your relationships should look like the set-up in Figure 10.16.

Table 10.1 Drink table data dictionary.

Field name	Data type	Data format	Field size	Description / Example(s)
DrinkID	Number	NN	2	The unique two-digit ID number of the drink (e.g. 11)
Type	Text	XX…XX	20	The common name of the drink (e.g. Espresso)
Description	Text	XX…XX	100	A brief description of the drink (e.g. A shot of coffee in a small cup)
Base Price	Currency	$#.##	5	The cost of the drink (e.g. $2.50)

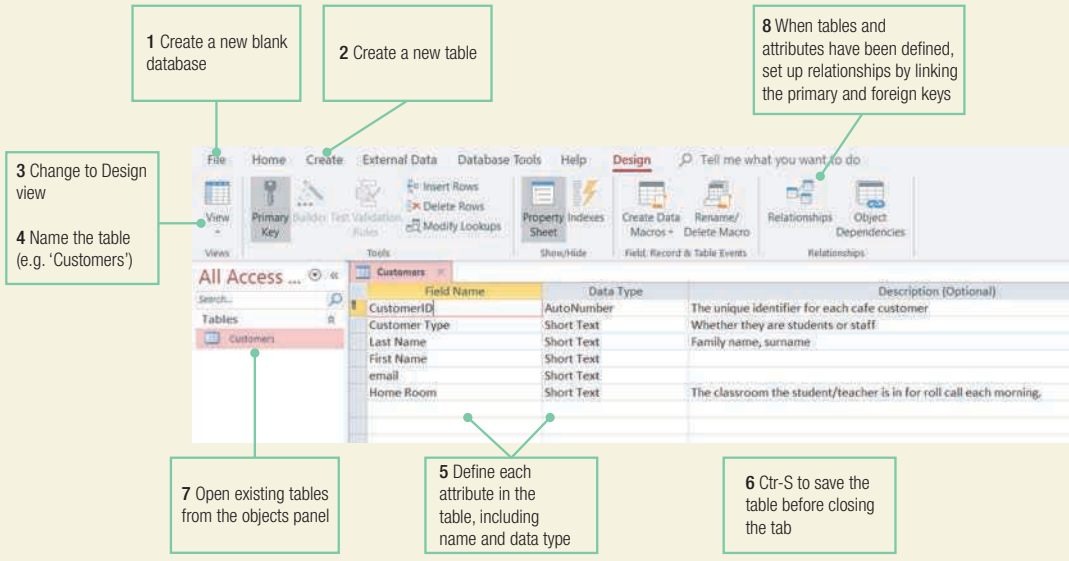


Figure 10.15 Setting up a relational database in Microsoft Access.

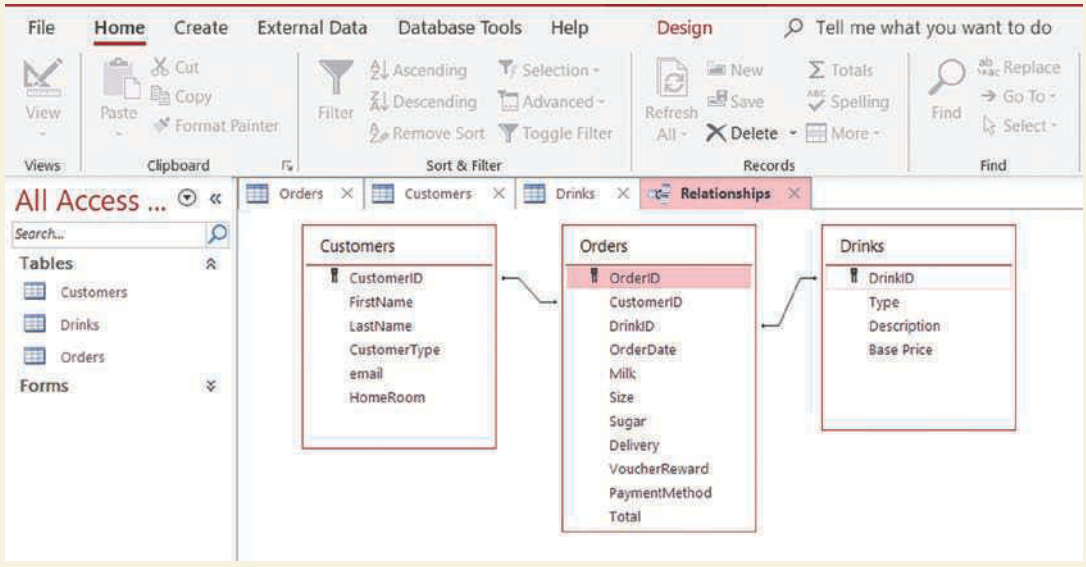


Figure 10.16 Defining the relationships in the cafe database.

ACTIVITIES

Identify

1. How is a field different from a record?
2. List the common data types. Which of these can be stored in a database?
3. Draw up a table to represent a file of data about products in a supermarket.
4. Why is it important to plan a database using schemas and data dictionaries before building it in a DBMS?

Analyse

5. What is the difference between a flat-file database and a relational database?
6. What is the difference between an entity and an event? How are they mostly linked in relational databases?
7. Why do we need to separate data into the different tables in a relational database?
8. What is a primary key used for in a database? Find out how you are uniquely identified in your school's administration system.

Extend

9 Look at the data types available in your database software application. List them and give an example of each. Can your database software store integers and decimal numbers? How are they different?

10 Find out if you can use lookup lists for data entry in your database application. What are they and why are they so useful for users?

INFOBIT

Data warehouses are used to deposit large volumes of data that are not needed immediately by a company, or that they intend to share with others.

PEOPLE

Building and maintaining databases

Depending on the size of the project, the end user may build themselves a database, or a project manager may need to be employed to lead a team of experienced professionals to design, build and maintain the system (Figure 10.17). Consider what needs to be done to create large and complex databases such as those used by supermarkets and banks. The whole system needs to be carefully planned and designed. Data needs to be entered and security issues considered. As people begin to use the database, there will be an ongoing need to maintain and update the system.

Systems analysts are the people who plan and design databases for large commercial institutions. They must understand the purpose of the system and how it will be used. Systems analysts need to communicate effectively with management and with end users so that everyone's needs are met. They will also need to be well-qualified in technology and have a wide perspective on current trends and costs.

Database administrators are responsible for the maintenance of the database. They will need to ensure the data entered is checked or verified and that the data is secure. Backup copies may be made daily or even hourly and access to data is restricted to authorised people.

In some organisations, there needs to be continual input of data. A market research company may have conducted a survey in a shopping mall and then needs the data to be entered into its database. Data entry personnel need to have fast and accurate keyboard skills and a keen eye for detail. If data is delivered by telephone from customers, they need a polite and patient manner.

Shutterstock.com/bbernard

Figure 10.17 There are many people involved in maintaining databases.

Questions

1 Identify the personal qualities needed for each of the jobs described.

2 Look at some newspaper advertisements or online career searches to determine the qualifications required for each of these jobs.

3 Search again the advertisements you found for personnel in the information technology (IT) industry. Enter the details into a database. Use fields such as job title, qualifications, experience, salary offered, location, employer, contact address and phone number. Draw up a second table to record names of likely clients. How will you link the tables?

4 Imagine you are a systems analyst who has been employed by a firm of solicitors to design a database application. Draw up and explain a list of tasks that you will need to perform before you can design their database. What data will need to go into the database? Discuss what security measures you would need to implement in the system.

10.3 Collecting and checking data

Once you have set up your database structure in a DBMS, you can begin adding data. Data records may be added one at a time as they are created. For example, a new book record will be created each time the library purchases a new book. Or records can be imported from previous data stores, such as a spreadsheet. This process is commonly used when users are migrating a database from one DBMS to another. In this section, you will look at some sources of data and some methods for collecting and checking it. There are many social and ethical issues concerned with these processes. Data needs to be accurate and current, while people's privacy must be protected.

Data sources

Data is gathered from a wide range of sources and in various ways. Sometimes it might seem a fairly simple task to collect data. To collect the data about books in your school library, you might walk around the bookshelves writing down the titles and authors' names on a piece of paper. You could then type the data into a computer using a keyboard. However, in larger projects, the system becomes more complicated. Figure 10.18 shows several situations in which data is being input to a database. Think about the types of data and how they are collected in these and similar situations.

Science Photo Library/ANDREW BROOKES, NATIONAL PHYSICAL LABORATORY

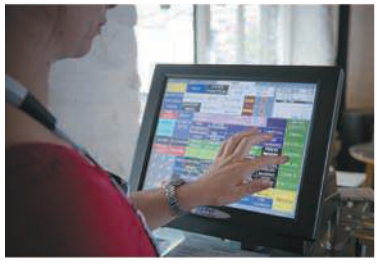
Biometric security devices use biological characteristics to identify people. How does the computer know which person the data belongs to?

Thinkstock/Jupiterimages

Computers use specialised input devices to read information stored as optical marks, bar codes and special characters. Where would you see examples of this type of data?

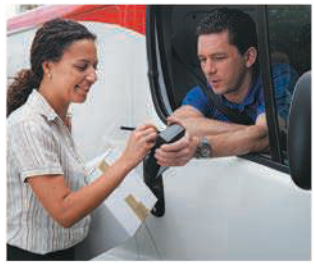
Alamy Stock Photo/Moodboard Stock Photography

Data collected in the workshop is transferred to the main database that keeps track of parts used.



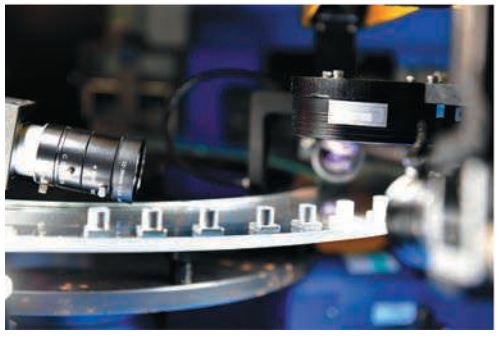
Alamy Stock Photo/Keith morris

Specialised terminals like this one make it easy to log restaurant orders. How has the nature of this person's work changed as a result of technology?



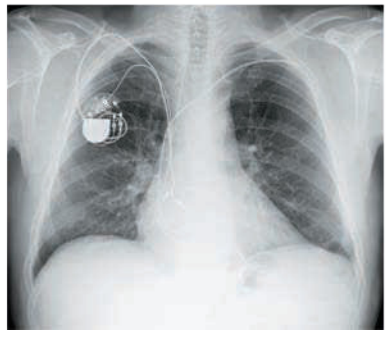
Alamy Stock Photo/Image Source

Couriers use pen-based computers to log deliveries and record signatures.



Shutterstock.com/i viewfinder

Large scale manufacturing often relies on computer vision for quality control. Cameras collect image data and AI technology is used to identify key features within the image that indicate an issue with the product's quality.



iStock.com/froud55

Medical devices such as pacemakers continually collect data on a patient's health status. This data is transmitted back to the health provider's system where it is stored in databases for real-time and ongoing analysis.

Figure 10.18 Data is collected from a wide variety of sources for input to a database.

Some of the data sources that are commonly used are:

- transactions, such as at the ATM or EFTPOS terminals
- application forms, when applying for a job or joining a club or platform
- questionnaires, in the government census or market research.

In some cases, data input will be automated, while in others, data entry clerks will be needed.

The examples given above are primary sources of data. The information is collected directly from the people (or objects) involved in the generation of the data. Also, consider all the different hardware devices that are used to input the data in the examples above and in our daily lives. Scanners collect data from bar codes, Quick Response (QR) codes and even images with the assistance of AI technology. Radio-frequency identification (RFID) technology is also used everywhere, to collect a toll on a road, to pay for a product with a tap and to use public transport.

Due to the developments in this technology, data input has become fast and reliable. By avoiding human intervention as much as possible, time is saved and errors are reduced.

There are times, however, when data is gathered from secondary sources, such as newspapers or research documents. It may be a quicker and more convenient method of collecting data, but the data may not be totally reliable. In all cases, sources should be documented and acknowledged. If anyone needs to go back and check the data later, it should be easy to find, and copyright issues must be respected.

Data validation

When data is added, it is important that the accuracy and validity of the data is considered. One way to do this automatically is to use **data validation**. Data validation checks that the data makes sense. For example, the date of birth for a student should make their age within a certain range. A 34-year-old is unlikely to still be at school! A DBMS can include data validation that rejects data that does not make sense or generates a warning message (Figure 10.19).

Another method of data validation is to restrict the input for a particular field using a defined list. For example, a student's year group can be one of only six. Providing this list of year groups for the user to select from removes the chance of an invalid year being added, such as Year 14, thereby reducing the chance of inaccurate data being added.

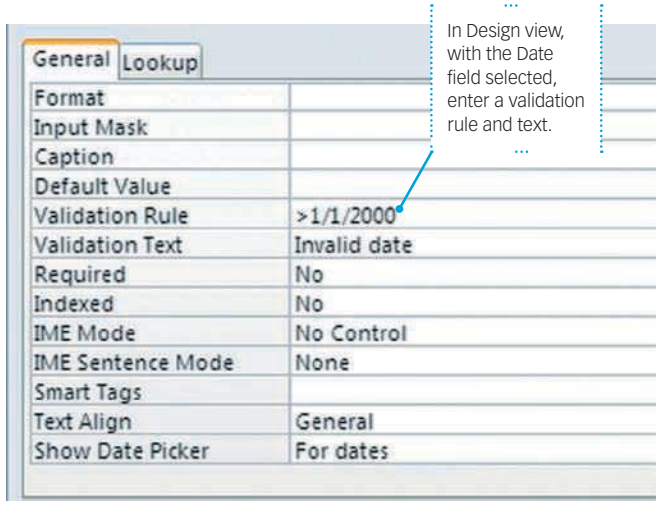


Figure 10.19 Creating a validation rule for the date of birth field in Microsoft Access.

Data verification

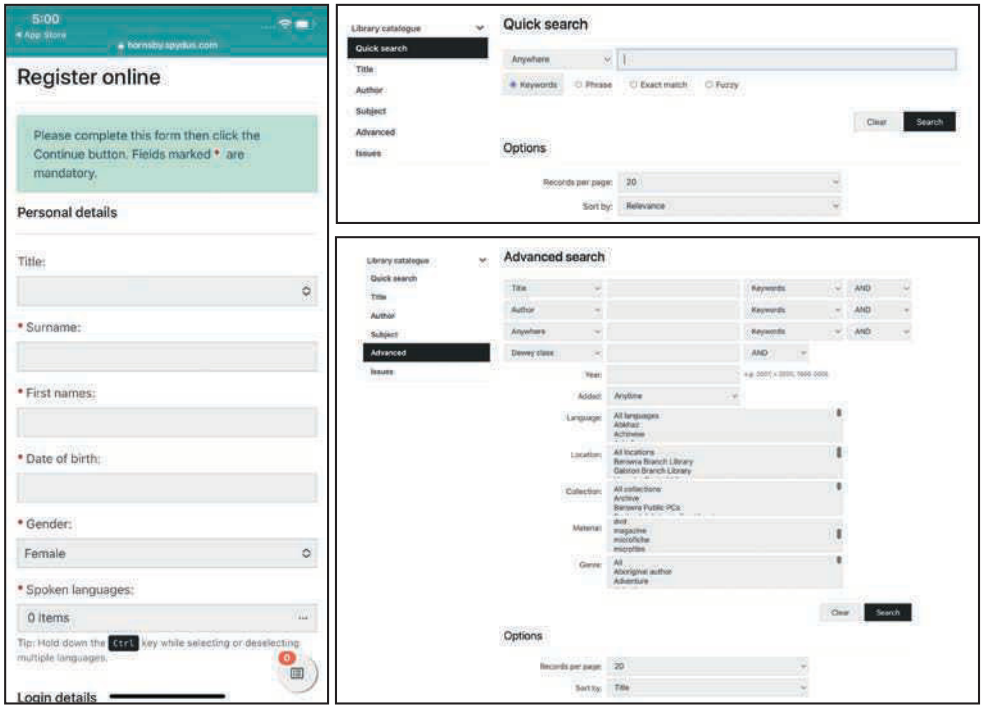
Although data validation can reduce data inaccuracy, the only way to prevent it is through **data verification**. These are processes that check that the data is true or correct. To verify the date, the data entry clerk would need to check back on the source document, the application form or birth certificate.

When you log on to a network or website, you need to enter your username and password to prove that you are the authorised person. Your password will be verified by comparing it with the original one stored in the database. When it comes to finance and transactions, verification is used to prevent user error and malicious events from occurring. A bank will often send a verification code or request a confirmation of transaction via SMS or email to minimise the chance of an unwanted transaction occurring.

Bar codes and customer codes on invoices often include check digits to help with verification. The check digit is calculated by applying a formula. The input device can then apply the same formula to the input code and compare the answer with the check digit.

Forms

A well-designed data entry **form** is key to minimising manual data entry errors or when viewing records for editing or analysis. The graphic design features are important in presenting the correct interface to the user (Figure 10.20). For example, the wording and size of text would need to be chosen carefully. Database designers must put careful thought into what the users will see on the screen. This plays a major part in making the database user-friendly and ergonomically sound.



Source: Hornsby Shire Council

Figure 10.20 Data entry form for a library loans desk. Data entry forms for registering for a library account on an app and searching for an item in the library on a website using a quick search with a few data entry fields and an advanced search with many data entry fields. What form design features have been used to minimise data entry errors?

Importing

Importing allows bulk amounts of data to be added to the database. This may be useful if there is already a data file containing records for an object that are to be included in the new system. For example, in a library system all the student records could be imported from another school system. We can also test a new system by generating test data and importing this. If a business needed to upgrade its database by having to re-enter all its data from the keyboard, it would be an enormous task. Fortunately, many software programs can understand each other's data, as long as it is in a common format – for example, delimited text. The delimited format can take various forms. One of the most common is the tab-delimited format, which uses the American Standard Code for Information Interchange (ASCII) tab character to separate each field and a carriage return to separate each record in a database. Files can be interchanged between spreadsheets and databases using this technique.

Later in this chapter, you will learn more about how data importing and exporting is used when databases are integrated with other systems.

Transaction processing upload

In reality, new records in databases are created automatically when a transaction occurs. When you use a debit card to purchase a drink at a shop, a transaction record is created in your bank's database that includes data such as time, location and amount. This record is linked to your customer record through the debit card number, which serves as a primary key. Can you think of some transaction records you initiated this week that would have created a database record? Remember, a transaction is really anything that is done; it does not need to involve money.

Updating

Updating existing records in a database involves making changes to the data already stored within specific records. This can include modifying attributes like names, addresses or quantities. It's a crucial process for ensuring data accuracy, especially when dealing with changing or evolving information, and helps to maintain the reliability and relevancy of the database's content over time.

Social, ethical and legal considerations of data collection

Data collection can have various social, ethical, and legal considerations that people and organisations developing and using the database, as well as individuals whose data is being collected, should be aware of.

Social considerations include:

- Privacy: People have the right to keep their personal information private, and data collection practices that infringe on this right may be viewed as socially unacceptable.
- Bias: Data collection methods that are biased or discriminatory can perpetuate inequality and affect vulnerable groups negatively.
- Transparency: People should have the right to know how their data is being collected, used and shared, and organisations should be transparent about their data practices.

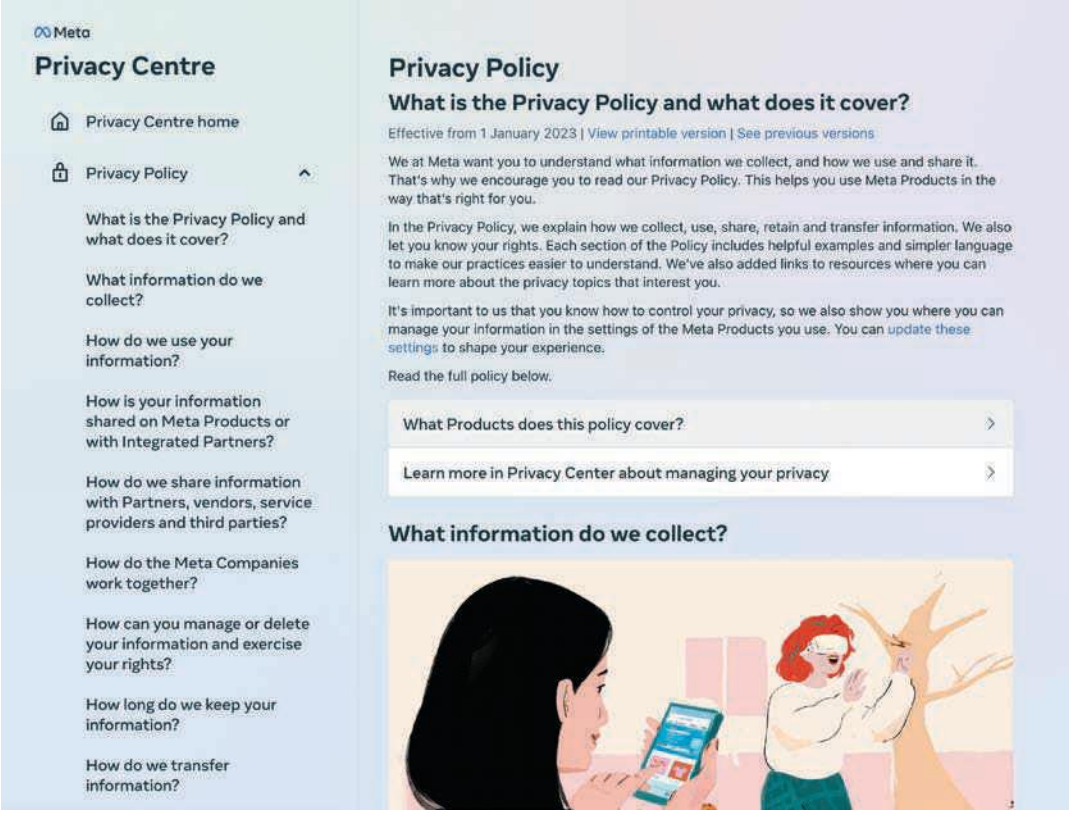
Ethical considerations include:

- Consent: Individuals need to provide consent before their data can be collected.
- Purpose limitation: Only necessary data should be collected and that data should only be used for the purposes that are clearly communicated to individuals.
- Data quality: Collected data should be accurate, current and relevant.

Legal considerations include:

- Compliance: Organisations must comply with relevant laws and regulations governing data collection, such as the Australian Privacy Law.
- Data protection: Organisations must take appropriate measures to protect the data they collect from unauthorised access, use or disclosure. With most data stored and access online via the cloud. Up-to-date cyber security measures need to be in place to protect personal data.
- Data retention: Organisations must ensure that they do not retain personal data for longer than necessary and they must have policies in place for the secure disposal of data when it is no longer needed.

When producing and implementing a database system, it is important to specify what data is being collected, who owns that data and how the data will be protected. These key points, along with other features addressing the above issues are usually outlined in a system's data or privacy policy for users to read and acknowledge before using the system (Figure 10.21).

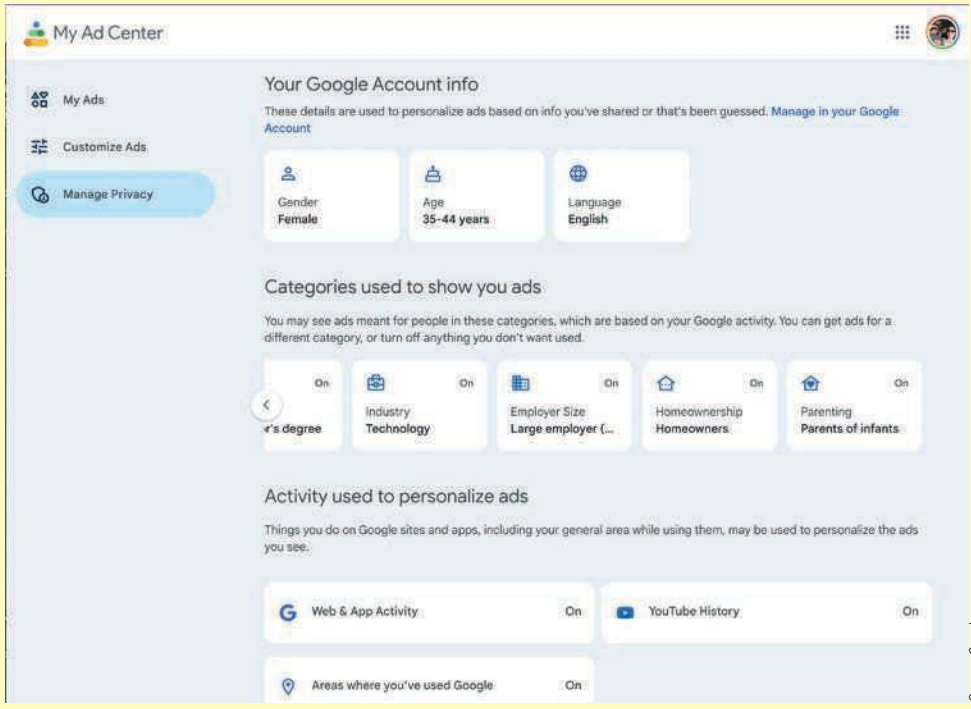


Source: Meta

Figure 10.21 Meta's privacy policy. Users of Facebook and Instagram should read this policy before creating an account so they are fully aware of what data is being collected, who owns the data and how the data is being protected.

INFOBIT

Data trails are the digital footprints left behind by individuals or organisations when they interact with digital technologies. These trails can include data such as search history, location data and personal information often without us being fully aware. There are many potential issues associated with data trails including privacy, security, discrimination and misuse of data.



Source: Google

Figure 10.22 In your Google account's data and privacy section you can see what Google has predicted about you using your online activity data trail for customised advertising. How accurate is yours?

SKILL BUILDER

School cafe DBMS – Part 3

Manually add data

Sometimes it is suitable to add data to your database directly in the database table. You will do this for the Drink data.

1. Add a record for each of the drink types: Cappuccino, Flat White, Long Black, Espresso, Double Espresso, Latte, Macchiato, Mocha, Hot Chocolate, English Breakfast Tea and Green Tea.
2. Save the table (Figure 10.23).

For the two tables where data will be added to regularly by users (Customer and Order), we will include data validation and use a data entry form to improve user-friendliness and minimise user error.

Validation

1. For all of the fields, consider what would be a valid value.
   - a. Is there a range of values it should be within? For example, number of sugars should be within 0 to 4.
   - b. Are there a limited number of options it can be? For example, size can only be Regular or Large.
   - c. Is there a format it should be in? For example, email should be of the format <?*>@<?*>.<?*>, where <?*> is a placeholder for one or many characters.
2. Add this validation using the data dictionary view. Figure 10.24 shows the implementation of the above.
3. Test out the validation by adding some records with fields that meet the validation checks and some that do not meet the validation checks (Figure 10.25).

Drinks

DrinkID	Type	Description	Base Price	Click to Add
244	Cappuccino	Shot of coffee with milk, froth	$2.50	
248	Double Espresso	2 shots of coffee	$3.50	
246	English Breakfast	Regular tea	$2.00	
247	Espresso	Shot of coffee	$2.50	
245	Flat White	Shot of coffee with milk	$2.50	
252	Green Tea	Green tea	$2.00	
250	Hot chocolate	Chocolate powder with milk	$2.50	
251	Latte	Small amount of froth	$2.50	
253	Long Black	Shot of Coffee with hot water	$2.50	
254	Macchiato	Shot of coffee with dash of mi	$2.50	
249	Mocha	Shot of coffee with chocolate,	$2.50	
(New)			$0.00	

Figure 10.23 Drinks records added to the database using the table/datasheet view.

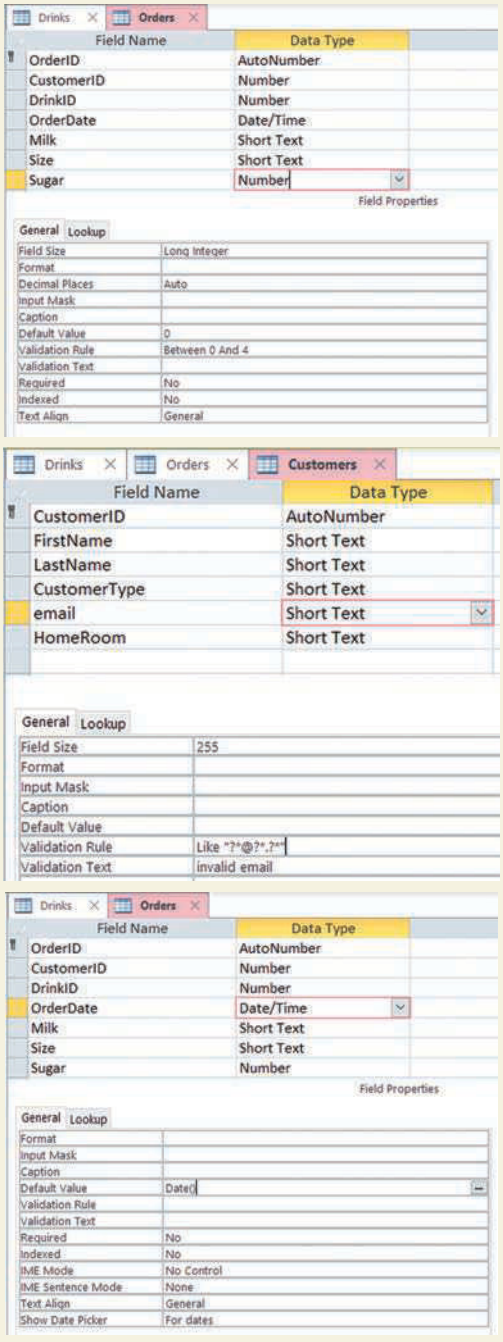


Figure 10.24 Defining validation rules and default values.

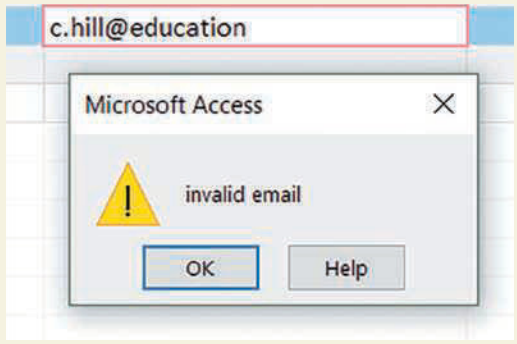


Figure 10.25 Validation text can be defined to give meaningful feedback to users when they enter invalid data.

Form design

We will first create the new customer form and in the next section we will create the new order form from a query (Figure 10.26).

1. Go to Create and select Form Design.
2. Click 'Title' in the Design menu ribbon to add the cafe name and form name to the *Form Header*. Click on the Format tab to adjust the formatting of the text, such as size, font and colour.
3. In the *Detail* section, from the field list, drag across and position all of the fields you have identified for this form. Adjust any formatting to suit.
4. Design a logo for the cafe and include this in the form header. Adjust other formatting to suit the style and theme of the cafe.
5. Ask another student in your class to use this form to add new data to the database as a customer. Observe them and ask for feedback.

Importing data

We will generate some data and import it into our database to allow further testing of the system before implementation. Realistic mock data can be generated using online tools such as Mockaroo. It is important to define the field name the same as in your database so Access can easily recognise it when importing.

1 Go to the Mockaroo website (or a similar service) and generate data for the customer table.
   - **a** First name, last name and email can be selected from the predefined categories.
   - **b** For the HomeRoom field, use the custom list option with random distribution. (This means each option is equally likely.)
   - **c** Also use custom list for the CustomerType with values of 'student' and 'staff'. Change the random distribution to custom and set the likelihood of student to 3 and staff to 1. This means there will be approximately three student records for every one staff record.
   - **d** Generate 200 records in Microsoft Excel format.
2 In Microsoft Access, go to External Data > New Data Source > From File > Excel File. Follow the Import window prompts (Figure 10.27).
3 Inspect your imported data in table view. Try sorting records by LastName and Filtering based on CustomerType (Figure 10.28).

Optional:

4 Repeat the test data generation and import for some order records. Think about the distribution models to use for fields such as DrinkID and Milk, as some are going to be more common than others.

Weblink
Mockaroo

Systems thinking

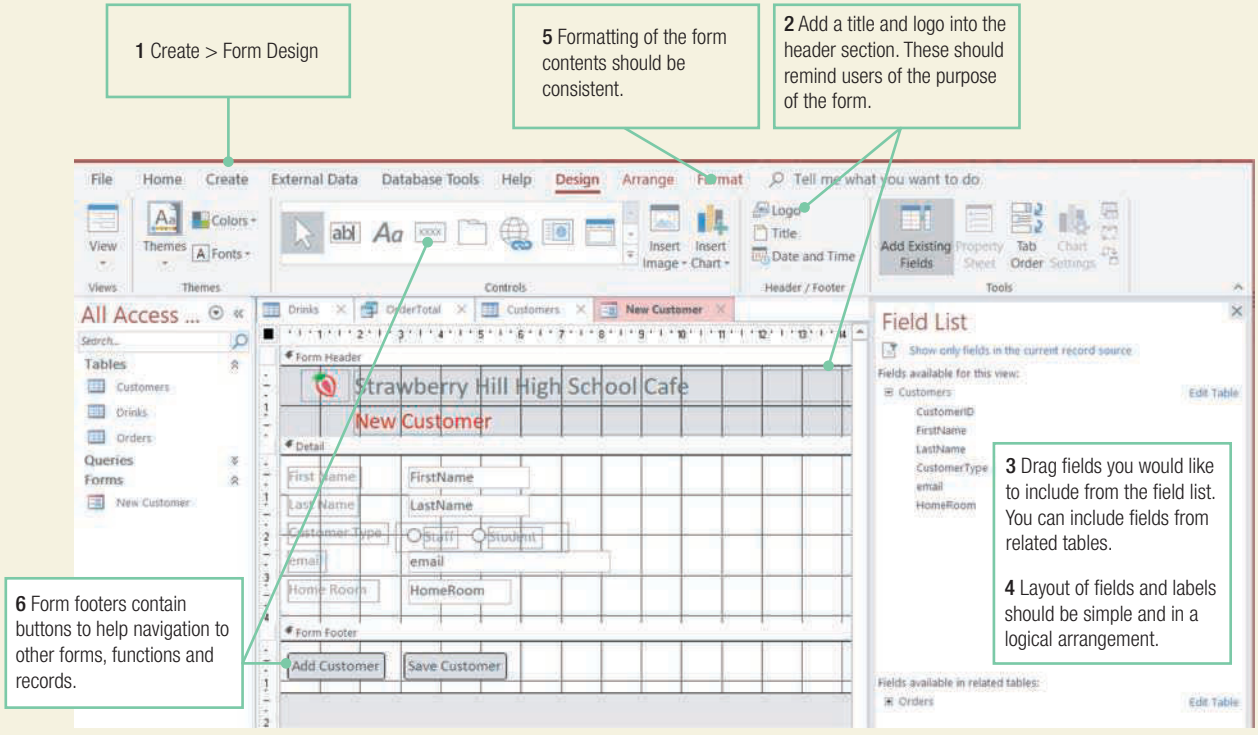


Figure 10.26 Form design for adding new customers.

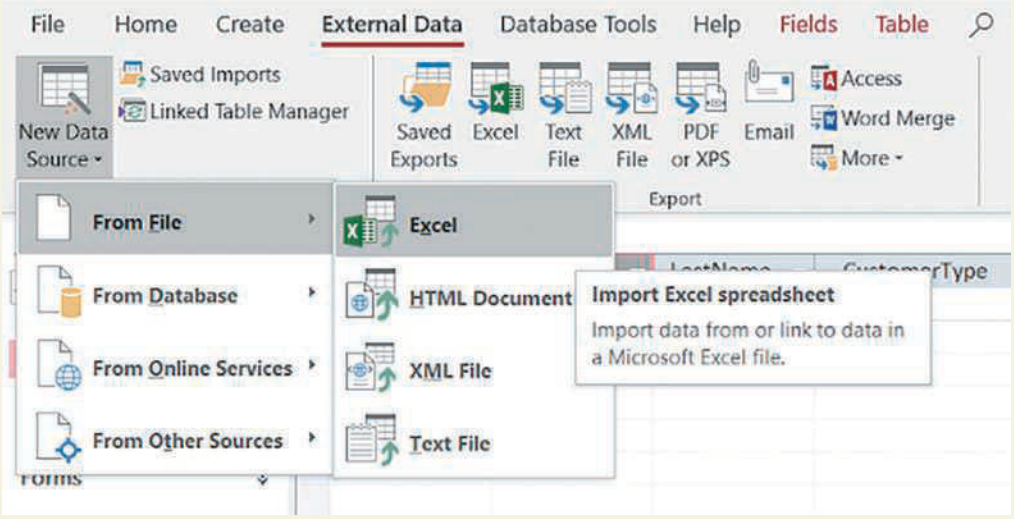


Figure 10.27 Microsoft Access can import data from a variety of sources, including spreadsheets.

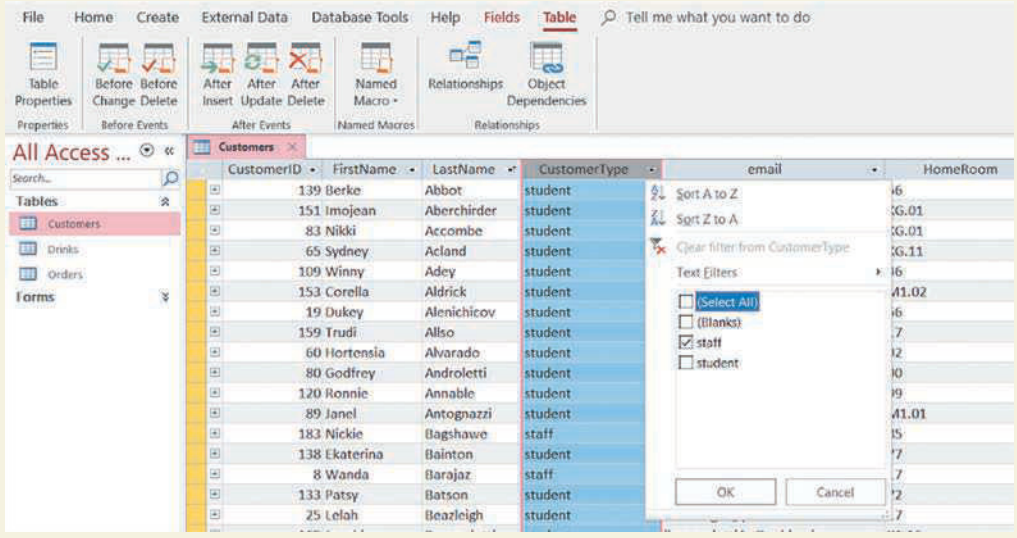


Figure 10.28 Simple exploration of data, including sorting and filtering, can be done in the table view.

ACTIVITIES

Identify

1 What is the advantage of using automated data collection devices?

2 What is the difference between validation and verification? Give some examples of each.

3 What is the purpose of a form?

Analyse

4 If you applied for a student concession card, what data would the authorities need to verify?

5 Find an application form that is used to apply for membership to a club or platform. Evaluate the effectiveness of the validation features.

6 In 2022, Optus experienced a data breach that impacted millions of customers. Find an article about the breach and identify which social, ethical and legal considerations are associated with this event.

Extend

7 Research how check digits work in bar codes. Outline the algorithm for calculating and checking the check digit.

8 Use a website form to search through an online database. Evaluate the form design in terms of use of design features and user-friendliness.

9 Find three current cybersecurity practices that are commonly used by online organisations to keep their recorded data secure.

10.4 Analysing data

Analysing is the process of turning our organised data into information. Useful information is the ultimate goal of all information systems and a DBMS includes a range of functionality to support its generation.

Sorting a database

Sometimes it is necessary to display records in a different order. In a library database, it may be convenient to have book records arranged in alphabetical order by author's name. However, if a student is looking for a particular topic, it might be better to have titles sorted by category. In the previous Skill builder, you did a quick exploration of your imported customer data by sorting the records based on last name (Figure 10.28).

Figure 10.29 shows a grocery supplier's list, with products sorted first by category and then by product. This is known as secondary sorting.

Product ID	Product Name	Cost	Category
9	Chocolate Biscuits	$6.90	Baked Goods
11	Scones	$7.50	Baked Goods
12	Beer	$10.50	Beverages
15	Coffee	$34.50	Beverages
41	Cheddar	$20.55	Dairy Products
23	Mozzarella	$26.10	Dairy Products
24	Almonds	$7.50	Dried Fruit & Nuts
17	Dried Apples	$39.75	Dried Fruit & Nuts
5	Dried Pears	$22.50	Dried Fruit & Nuts
7	Walnuts	$17.44	Dried Fruit & Nuts
10	Marmalade	$60.75	Jams, Preserves
4	Rasberry Jam	$18.75	Jams, Preserves
19	Gnocchi	$28.50	Pasta
20	Ravioli	$14.63	Pasta
6	Curry Sauce	$30.00	Sauces
21	Pepper Sauce	$15.79	Sauces
22	Tomato Sauce	$12.75	Sauces

Figure 10.29 A grocery supplier's list sorted first on category and then on product name. You will notice that items have been alphabetically sorted within each category.

In many database applications, sorting can be done on the original data table and also on the results of a search. It might be worth noting, though, that in sorting the data, we are only arranging how it should appear on the screen or paper. The way the data is arranged in memory or secondary storage is quite different from how we see it displayed (Figure 10.30).

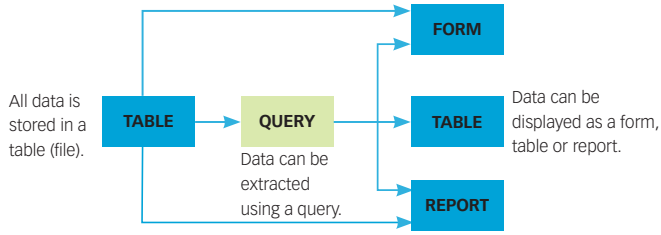


Figure 10.30 Data flow in a database. Data is stored only once – in the table or file. It can then be displayed in different ways using tables, forms, reports and queries.

Querying a database

Querying is the process of extracting then displaying certain data for a particular purpose, usually in a form, table or **report** (Figure 10.29). For example, finding all the loan records that are overdue so a reminder report can be sent to students. Querying a database can be done in two main ways:

- **query by example (QBE)**, where the user fills in a query table with conditions for selecting records and instructions on ordering the results
- coding the query using **structured query language (SQL)**. This code also includes instructions for record selection under certain conditions and how to order the results.

Nearly all DBMSs integrate SQL for analysis and some of the more user-friendly GUI applications, such as Microsoft Access, include QBE functionality.

QBE

When performing a QBE in Microsoft Access, you first select the tables that will be involved in the query. You then complete the query table, identifying each field you would like to display and defining the criteria

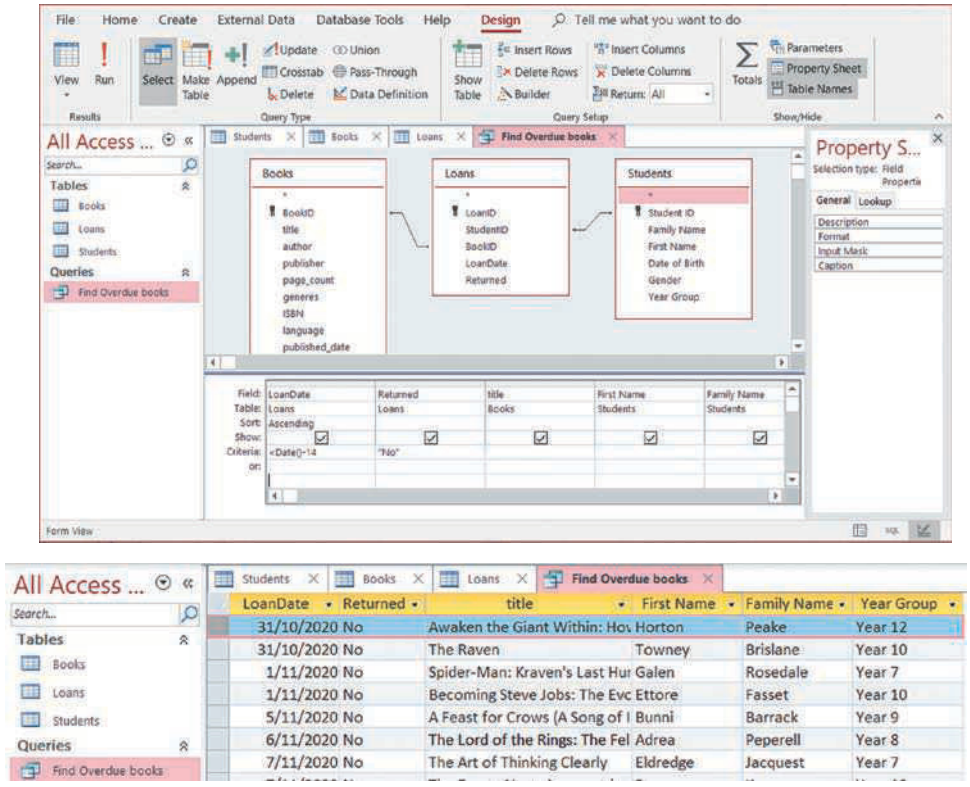


Figure 10.31 QBE to find all overdue books and the results of the query.

by which to select the records based on that field. In the example in Figure 10.31, a query was created in Access to find all the overdue books. Data about the loan, book and student was relevant to the query, so all tables were included. The results of the query are displayed in a datasheet view.

SQL

SQL is the standard language for relational database management. One of the most common SQL commands is a SELECT query. This query instructs the DBMS to find records that meet defined criteria. The identified attributes (including attributes from linked tables) of these records are then displayed in a defined order. The format for a SELECT query is:

```
SELECT <Attributes to display>
FROM <Tables>
WHERE <Criteria from which to select records>
ORDER BY <The attribute by which the
  selected records should be ordered> <ASC/
  DESC>
```

When querying a single table, SQL is quite straightforward. For example, to find the Title and Author of thriller fiction books, the SQL would be:

```
SELECT Title, Author
FROM Books
WHERE Genre = Like "*Thriller*" AND "*Fiction*"
ORDER BY Author ASC
```

When querying across multiple tables, the relationships between the tables need to be defined using *joins*, which is beyond the scope of this chapter but have been briefly summarised for you in Figure 10.32. We can theoretically represent this kind of query in a SELECT statement by including in the WHERE criteria the primary keys that match the foreign keys of the loan records that meet the rest of the criteria. For example, the query to find overdue books would be:

```
SELECT Loans.LoanDate, Books.Title,
Students.FirstName, Students.FamilyName,
Students.YearGroup
FROM Loans, Books, Students
WHERE Loans.LoanDate < Date()-14 AND
    Loans.Returned = "No" AND Loan.Student_ID =
    Student.StudentID AND Loan.BookID =
    Book.BookID.
ORDERBY Loans.LoanDate ASC
```

Figure 10.33 shows the actual SQL generated for this query.

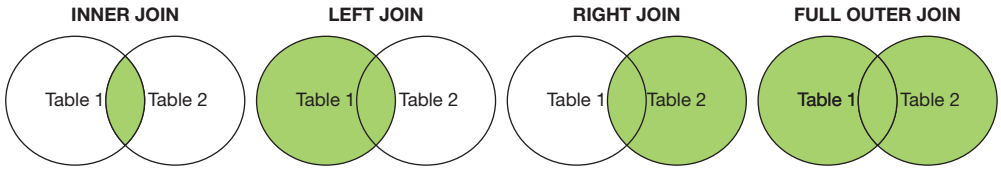


Figure 10.32 The four possible joins that can be used in SQL databases. The inner join means only find the records which match both the table keys. A full outer join means find any record which has a match to one or both of either of the table keys.

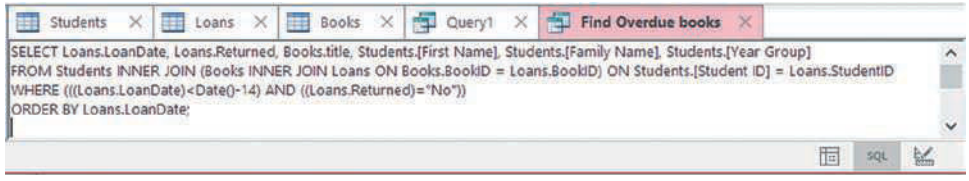


Figure 10.33 Generated SQL from a Query design. Table relationships are defined using *INNER JOINs*. Attribute names that contain a space have been grouped with square brackets so the DBMS knows it is a single identifier.

Mathematical calculations

Databases can perform mathematical calculations just like in a spreadsheet. These are particularly useful when one field is based off one or more other fields in the database. We may wish to have the age of our students stored in the database. Obviously, the value of this data will change over time, and so it would make more sense to store the students' dates of birth rather than their ages. We can set up a calculated field to show the current age of the student by calculating the difference between the date of birth and the current date. Figure 10.34 shows how this is done in Microsoft Access.

Standard functions may also be used for groups of data, such as sum, average, minimum and maximum values, or a count of the number of items. We will look at some of these in the next section.

Macros

The term 'macro' means the same in databases as in spreadsheets, as discussed in Chapter 9. A macro is a series of tasks that we want to group together so that they can be run one after the other literally at the press of a button.

A macro is an object like other objects available in Microsoft Access (tables, forms, queries and so on), except that you create the macro to automate a task or series of tasks. Macros become particularly useful when the tasks need to be constantly repeated. In designing our database, we may want to click a button on a form so that it displays a message before it moves to another form. This involves three separate tasks: display a message, close first form, open second form. If we group these tasks together in a macro, they can be run one after the other at the click of a button.

The tasks to be carried out in the macro can be defined in a dialogue box, or they may be recorded directly as the user performs each step. Figure 10.35 shows a typical dialogue box for defining the tasks in a macro. The macro is then commonly attached to a labelled button on the user interface (UI) or assigned a keyboard shortcut. You might also get a macro to run automatically on opening a database. For example, you may want a user menu form to open automatically when the database is opened.

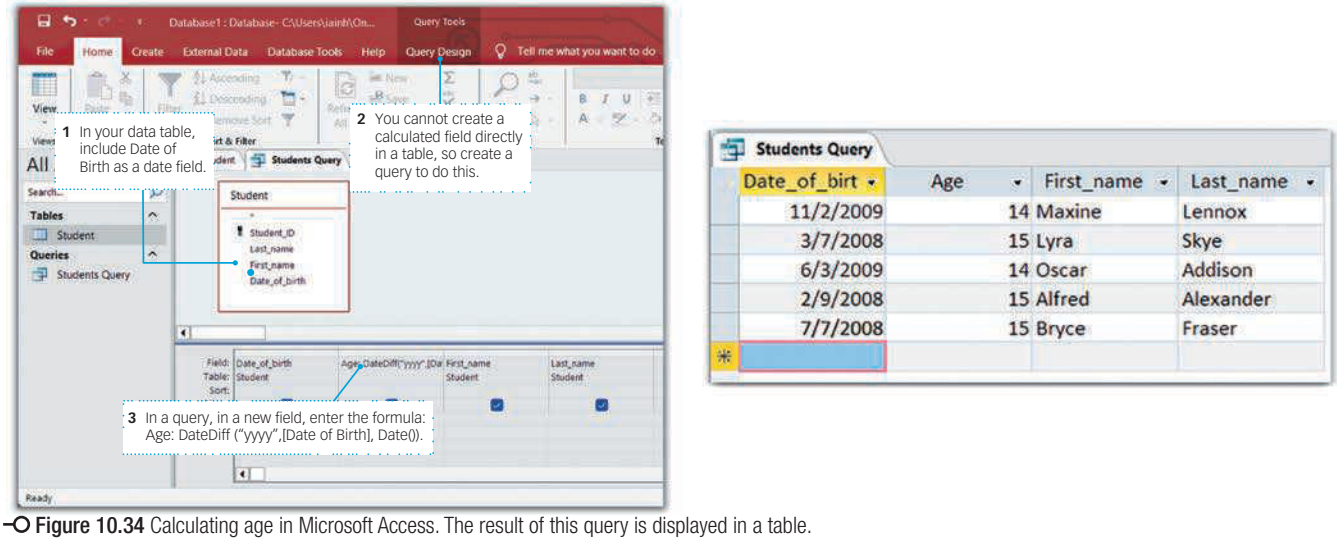


Figure 10.34 Calculating age in Microsoft Access. The result of this query is displayed in a table.

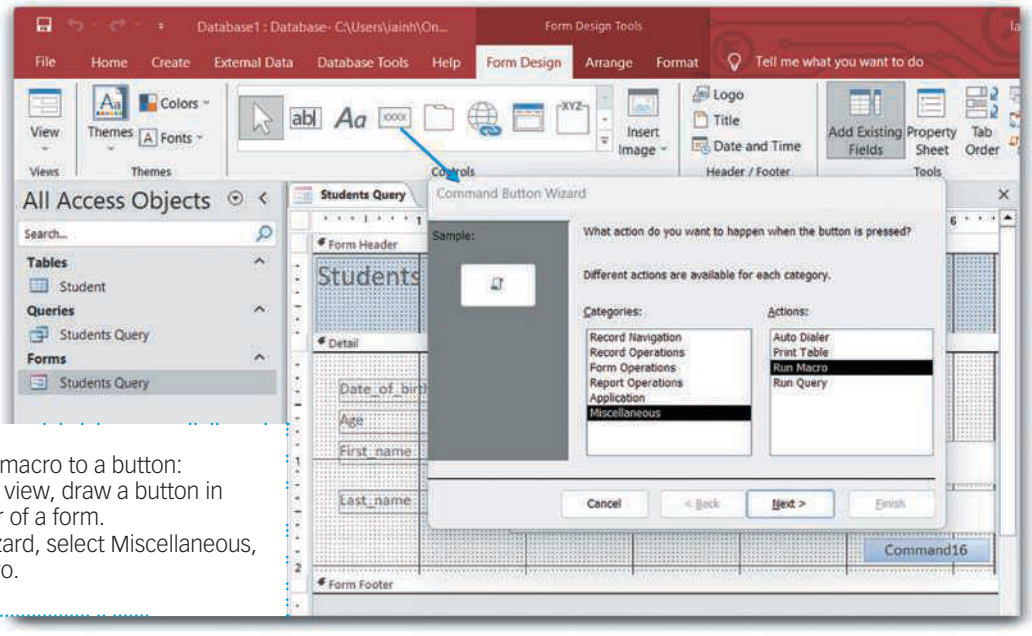


Figure 10.35 Creating a macro in Microsoft Access. The macro is then attached to a button on a form.

CASE STUDY

Australian War Memorial databases

Weblink
Australian War Memorial

The Australian War Memorial (Figure 10.36) combines a shrine, a museum and an extensive archive. The purpose of the Australian War Memorial is to commemorate the sacrifice of those Australians who have died in war. Its mission is to assist Australians to remember, interpret and understand the Australian experience of war and its enduring impact on Australian society.

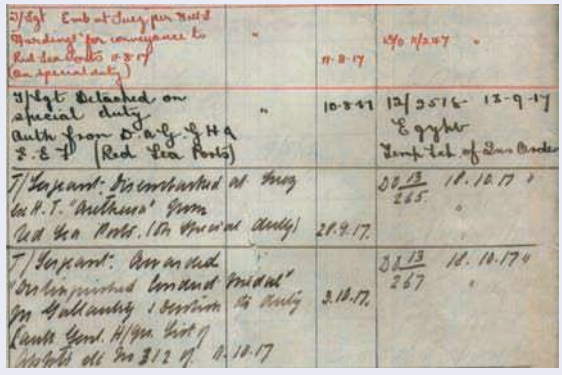
Everyone can use the site to search for collections of books, photographs and articles, as well as biographical databases that store data about individuals, awards and their fates. For example, you can find a list of all Australian soldiers who served and/or received honours in various wars, including the First World War, the Second World War and the Vietnam War.

Dreamstime.com/Clearviewstock

Figure 10.36 The Australian War Memorial.

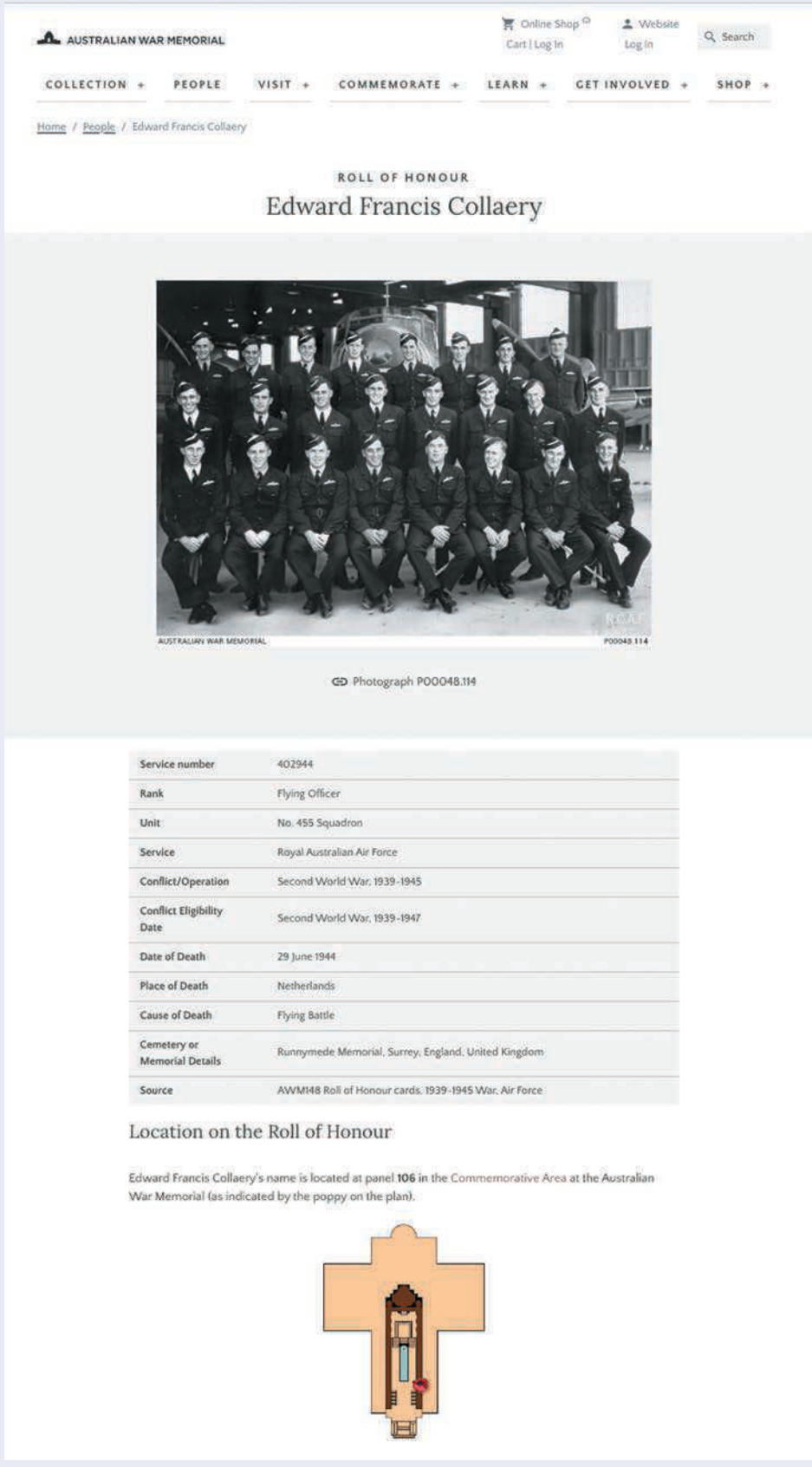
Questions

1. Visit the Australian War Memorial website's database collection. Research information about a family member, family friend or any name on the list. See if you can create a significant biography about the person. Don't forget to acknowledge the sources of your data.
2. Discuss the privacy issues that might be involved with researching information about people online.
3. Figure 10.37 shows a section from the service record of a particular soldier. How do you think the data would be stored in the electronic database application that you have used?
4. Use the information in Figure 10.38 and your exploration of the online database to create a schema of the portion of the database storing Person, Unit and Conflict records. Create a data dictionary for the Person table.



Source: Australian War Memorial

Figure 10.37 Data from the service record of a sergeant in the First World War.



Source: Australian War Memorial

Figure 10.38 The service record of Edward Collaery from the Australian War Memorial database.

SKILL BUILDER

School cafe DBMS – Part 4

Sort and search data for analysis

1 Perform some secondary sorting on the database in each table using field headings in the datasheet view. What information can you gain from this simple analysis?

2 Create a new query using Query Design. This query will find all order records from today (returned when you call the Date() function) where the customer used their own cup. Order the results by CustomerType and then LastName (Figure 10.39).

3 Plan and implement three more queries that could reveal useful information. Summarise your findings.

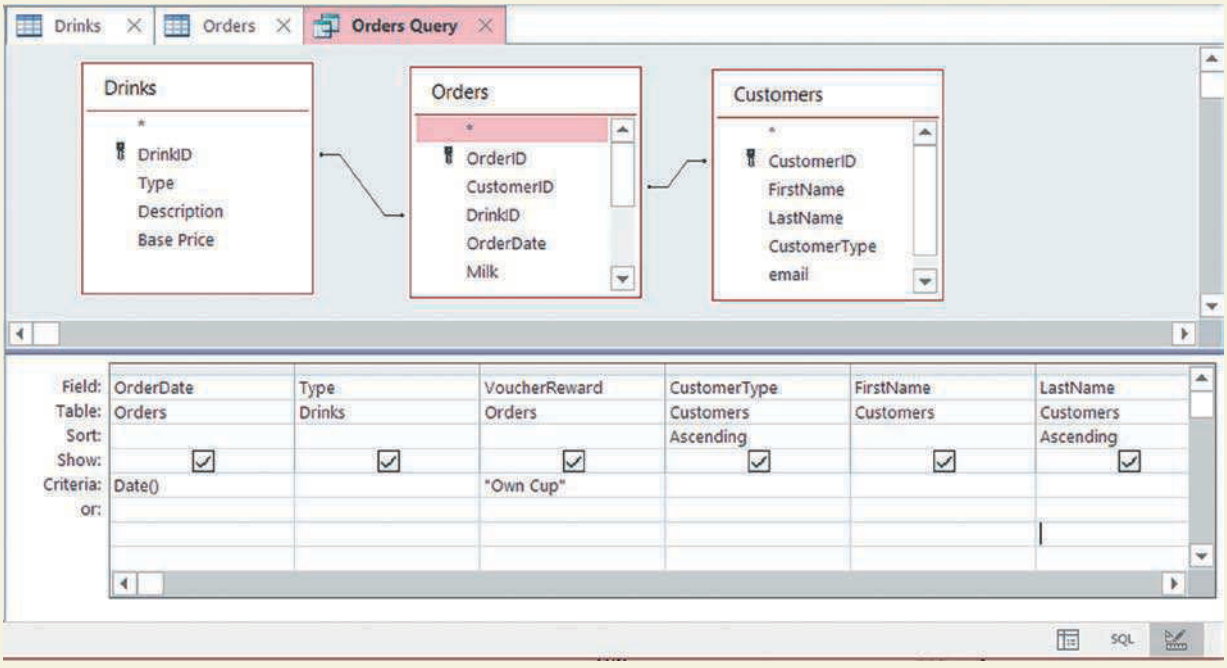


Figure 10.39 Query Design allows for easy selection of fields and definition of criteria.

Calculations

Computational thinking

The users of the system would like an automatic calculation of cost for the drink order (Figure 10.40). This will be based on the drink price, the size of the drink, the type of milk and any vouchers/rewards applied. Calculations are best performed in a query.

1 Consider the rules for calculating the order total:
- It costs $1 to upgrade to a LRG from a REG drink size.
- Soy and almond milk cost 50c extra.
- If a customer brings their own cup, they receive a 50c discount.
- Customers using their Free Drink or New Teacher voucher get the drink for free, regardless of the total cost.

2 Create a new query. Add all of the tables.

3 In the first query column for the field, add 'Orders.*'. This will include all the values in the order table.

4 In the next three columns, add the calculation formulas shown in Table 10.2.

Table 10.2 Calculation formulas.

Formula	Explanation
`DrinkTotal: [Base Price] + IIf([Size]="LRG",1,0) + IIf([Milk]="Soy" Or "Almond",0.5,0)`	The drink total is the base price plus $1 for a large and plus 50c if the milk is soy or almond.
`Discount: IIf([VoucherReward] Is Null, 0, IIf([VoucherReward]="Own Cup",0.5,[DrinkTotal]))`	If there is no voucher or reward used, there is no discount; otherwise, if the customer uses their own cup, there is a 50c discount. The final VoucherReward conditions of Free Drink or New Teacher mean the discount is the value of the DrinkTotal.
`OrderTotal: [DrinkTotal]-[Discount]`	The order total is the drink total minus the discount.

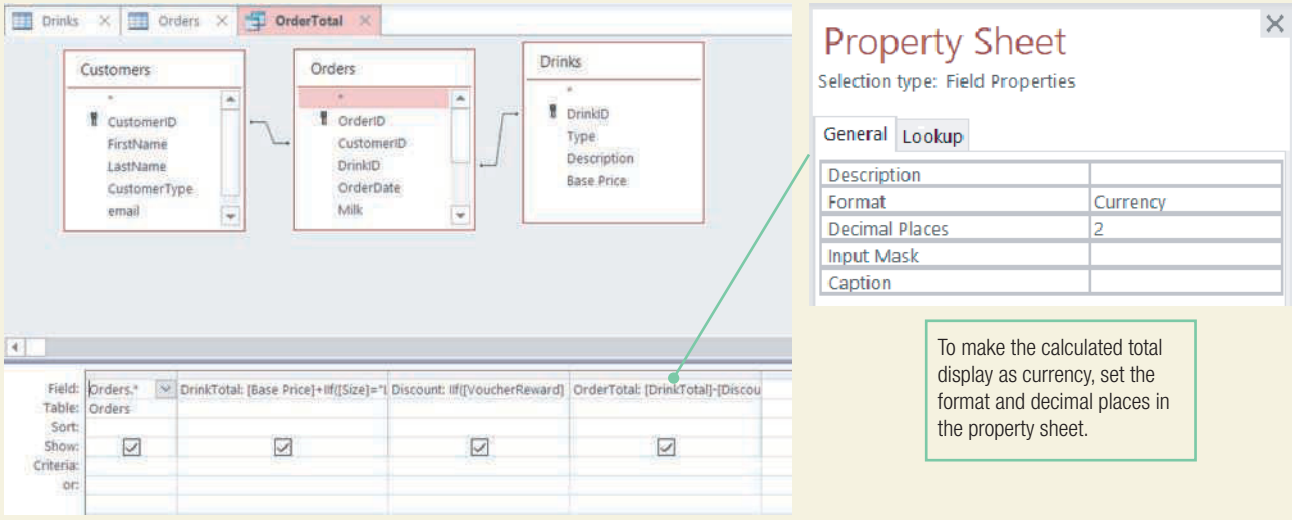


Figure 10.40 Calculated fields should be done through a query. Calculated fields can also rely on the results of other calculated fields, as in this example. Define the format and other field properties of the calculated field in the property sheet.

When entering the formulas, be careful to follow the syntax precisely. Access will not understand the query if you make even a tiny error, such as missing a bracket or using an uppercase instead of a lowercase character.

5 Run the Query.

6 Inspect the results (Figure 10.41), performing some manual checks to see if the calculations are working as expected.

Create a form from a query

We will now create our new order form from the Order Total query function.

1 With the OrderTotal query selected in the objects panel, go to *Create* > *Form*. A new form will be autogenerated for you that includes all the fields from the query.

2 This form layout and styling will need to be changed to make better use of space, present the fields in more logical grouping and to better suit the purpose of the form. A sketch, as shown in Figure 10.42, can be a good way to plan the layout of a new form.

3 Go to Design View and modify the form design as desired (Figure 10.43).

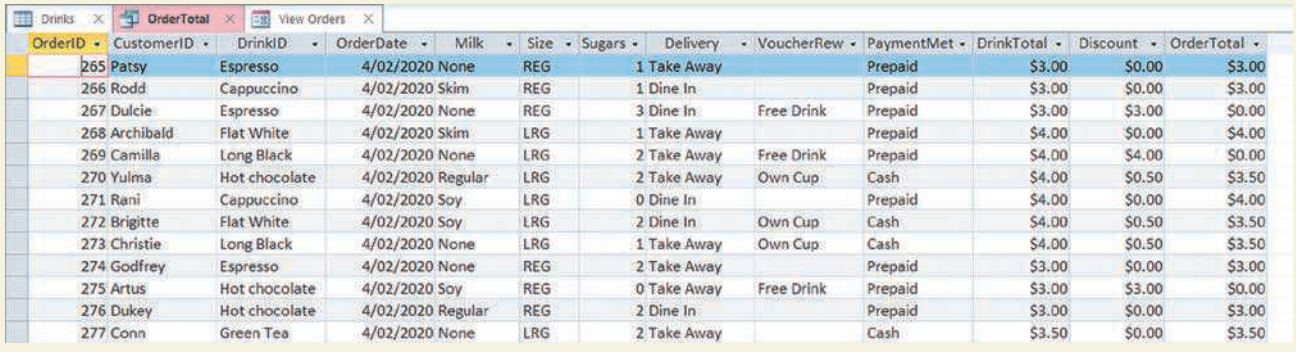
Add buttons to run built-in macros

Finally, we will include some buttons in the footer to perform some simple database record processing, including adding, editing and deleting.

1 In Design view, click on the button tool and draw in the footer.

2 Work through the Command Button Wizard to build the button for creating a new order record, including selecting the macro action (Record operations: Add New Record) and defining the look of the button (text or image).

3 Format the button to match the rest of the form design.



OrderID	CustomerID	DrinkID	OrderDate	Milk	Size	Sugars	Delivery	VoucherRew	PaymentMet	DrinkTotal	Discount	OrderTotal
265	Patsy	Espresso	4/02/2020	None	REG	1	Take Away		Prepaid	$3.00	$0.00	$3.00
266	Rodd	Cappuccino	4/02/2020	Skim	REG	1	Dine In		Prepaid	$3.00	$0.00	$3.00
267	Dulcie	Espresso	4/02/2020	None	REG	3	Dine In	Free Drink	Prepaid	$3.00	$3.00	$0.00
268	Archibald	Flat White	4/02/2020	Skim	LRG	1	Take Away		Prepaid	$4.00	$0.00	$4.00
269	Camilla	Long Black	4/02/2020	None	LRG	2	Take Away	Free Drink	Prepaid	$4.00	$4.00	$0.00
270	Yulma	Hot chocolate	4/02/2020	Regular	LRG	2	Take Away	Own Cup	Cash	$4.00	$0.50	$3.50
271	Rani	Cappuccino	4/02/2020	Soy	LRG	0	Dine In		Prepaid	$4.00	$0.00	$4.00
272	Brigitte	Flat White	4/02/2020	Soy	LRG	2	Dine In	Own Cup	Cash	$4.00	$0.50	$3.50
273	Christie	Long Black	4/02/2020	None	LRG	1	Take Away	Own Cup	Cash	$4.00	$0.50	$3.50
274	Godfrey	Espresso	4/02/2020	None	REG	2	Take Away		Prepaid	$3.00	$0.00	$3.00
275	Artus	Hot chocolate	4/02/2020	Soy	REG	0	Take Away	Free Drink	Prepaid	$3.00	$3.00	$0.00
276	Dukey	Hot chocolate	4/02/2020	Regular	REG	2	Dine In		Prepaid	$3.00	$0.00	$3.00
277	Conn	Green Tea	4/02/2020	None	LRG	2	Take Away		Cash	$3.50	$0.00	$3.50

Figure 10.41 Results of the query that created a field to hold the calculation of the order total for each record. Manual checks are done to test the correctness of the processing. For example, with OrderID 268 we expect the order total to be $4: Base price of a flat white ($3) plus $1 to upgrade to LRG size and with no VoucherReward to create a discount. This check should be performed on a number of records with a variety of field combinations.

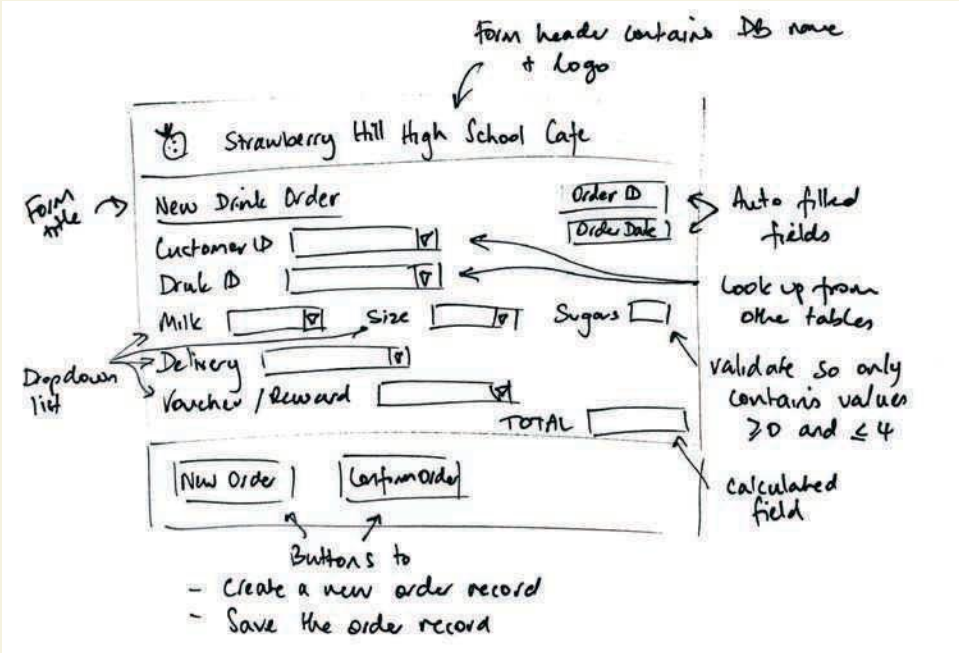


Figure 10.42 A sketch to plan the data entry form for a new drink order.

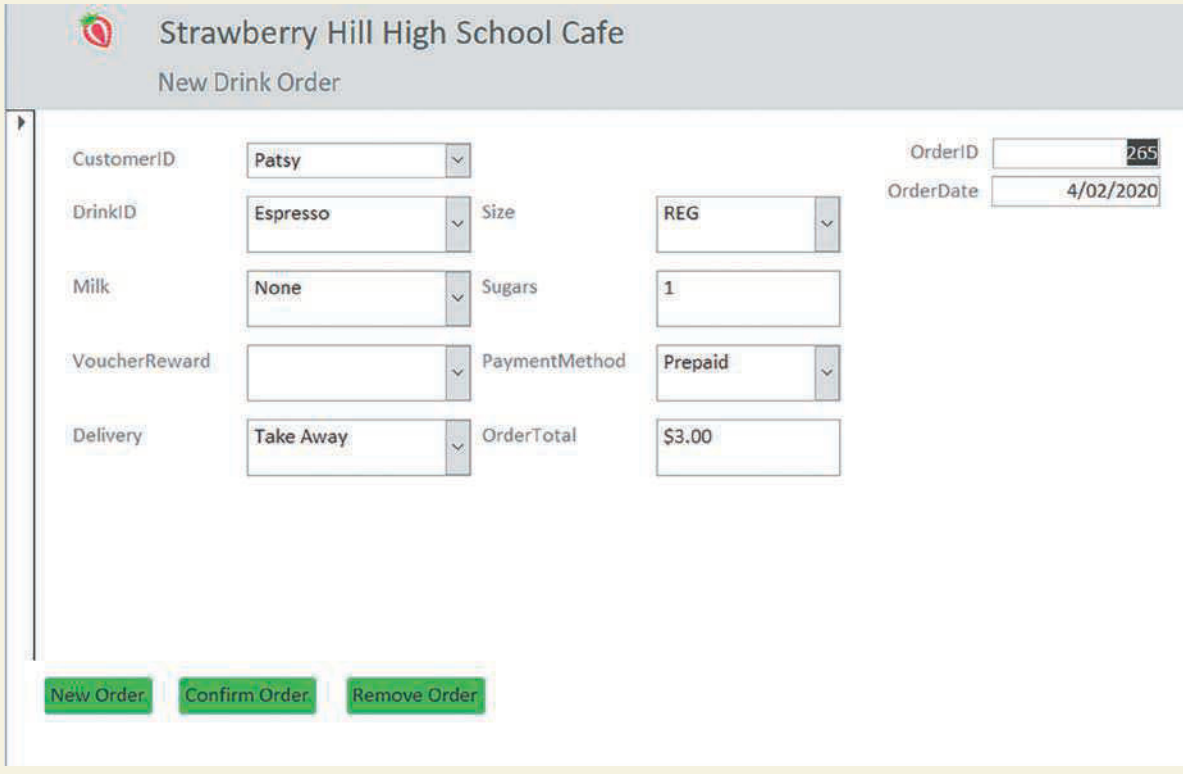


Figure 10.43 New order form generated from a query and redesigned in Design view. Buttons have been added that call simple macros to create a new record, save a record and delete a record.

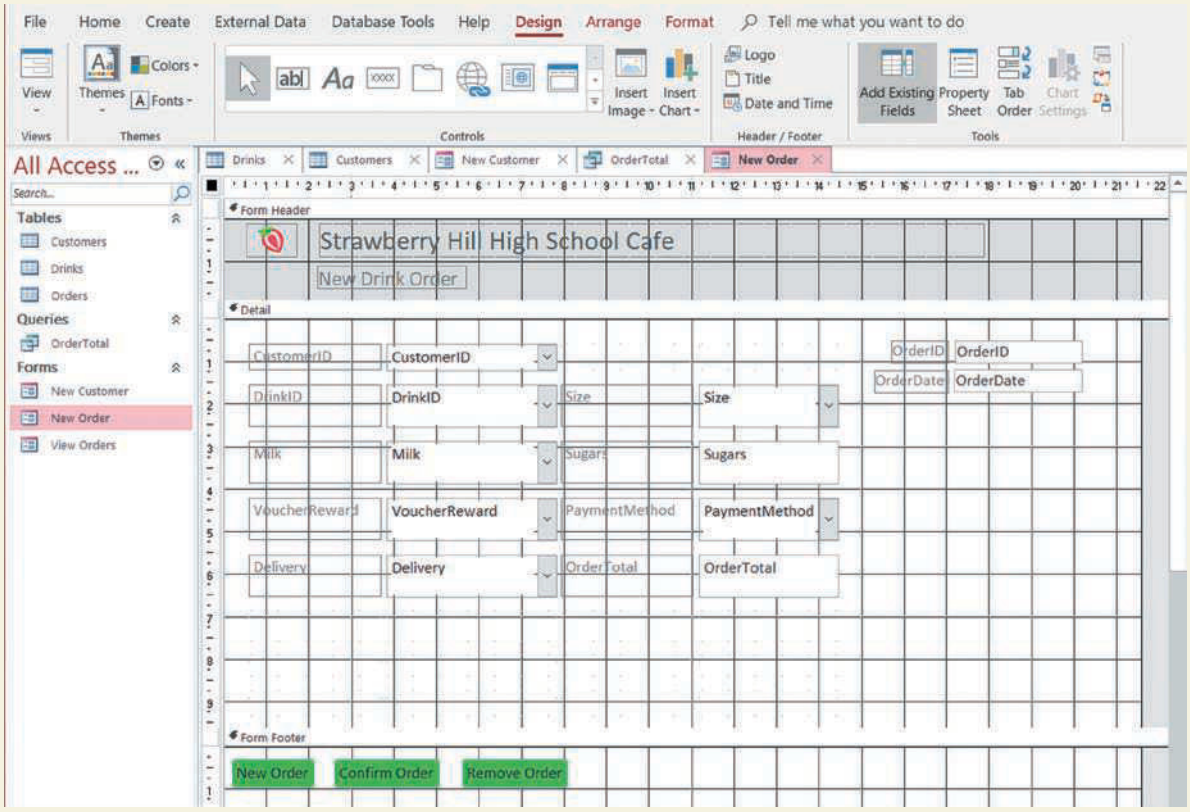


Figure 10.43 (*continued*)

ACTIVITIES

Identify

1. Does performing a secondary sort edit the data in the database? Explain.
2. How should the data in the library database be sorted if looking for the most recent loan records?
3. What is the difference between QBE and SQL?
4. What are the four key words used in an SQL statement?
5. Where are macros often used in a form?

Analyse

6. How can the days overdue figures be calculated in the library database?
7. Explain how queries can be used to create forms.

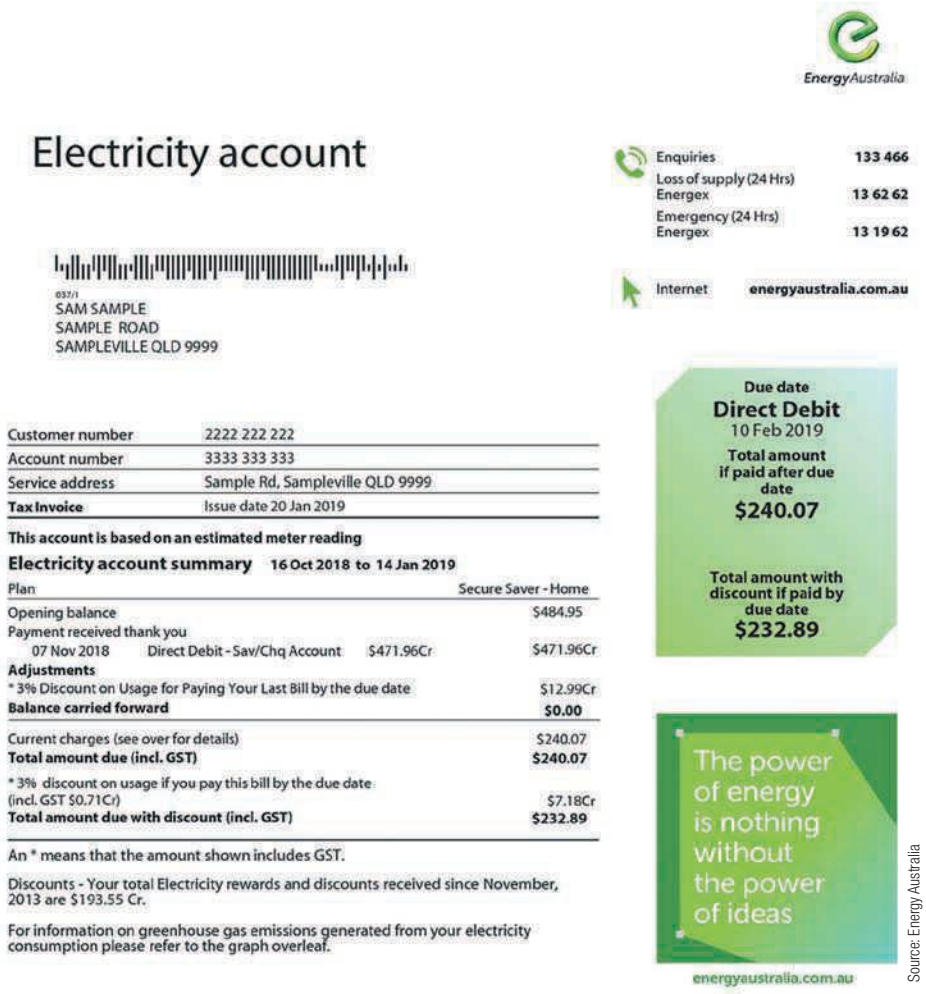
Extend

8. Think of another form that would be useful in the cafe database and create it.
9. Visit the SQLZOO website or complete the Grok 'Introduction to Databases (SQL)' course to learn how to do more SQL commands.

Weblink
SQLZOO

10.5 Presenting information

Once the data has been organised in a database and analysed through querying, the resulting information should be presented in a format that is easy to interpret by the intended audience. Database management systems can use forms and reports to present information. Data visualisation, such as charts, can be included in these displays to help clearly and simply communicate the information. An energy bill (Figure 10.44) is an example of a database report that uses text formatting, tables and graphs to help communicate a range of information.



EnergyAustralia

Electricity account

Enquiries 133 466
Loss of supply (24 Hrs) Energex 13 62 62
Emergency (24 Hrs) Energex 13 19 62

Internet energyaustralia.com.au

037/1
SAM SAMPLE
SAMPLE ROAD
SAMPLEVILLE QLD 9999

Customer number	2222 222 222
Account number	3333 333 333
Service address	Sample Rd, Sampleville QLD 9999
Tax Invoice	Issue date 20 Jan 2019

This account is based on an estimated meter reading

Electricity account summary 16 Oct 2018 to 14 Jan 2019

Plan	Secure Saver - Home
Opening balance	$484.95
Payment received thank you 07 Nov 2018 Direct Debit - Sav/Chq Account $471.96Cr	$471.96Cr
Adjustments	
* 3% Discount on Usage for Paying Your Last Bill by the due date	$12.99Cr
Balance carried forward	**$0.00**
Current charges (see over for details)	$240.07
Total amount due (incl. GST)	**$240.07**
* 3% discount on usage if you pay this bill by the due date (incl. GST $0.71Cr)	$7.18Cr
Total amount due with discount (incl. GST)	**$232.89**

An * means that the amount shown includes GST.

Discounts - Your total Electricity rewards and discounts received since November, 2013 are $193.55 Cr.

For information on greenhouse gas emissions generated from your electricity consumption please refer to the graph overleaf.

Due date
Direct Debit
10 Feb 2019
Total amount if paid after due date
$240.07
Total amount with discount if paid by due date
$232.89

The power of energy is nothing without the power of ideas
energyaustralia.com.au

Source: Energy Australia

Figure 10.44 An energy bill presents a variety of information from the database of Energy Australia customer accounts and usage. It is organised and formatted to group related information together, such as payment information, and to highlight key information such as amount due and due date. A combined stacked column and line graph is used to visually represent the peak and off-peak energy usage for the past year, and greenhouse gas emissions. A table is also included to show the customer's daily consumption compared with the average.

EnergyAustralia

Your electricity usage and service calculation

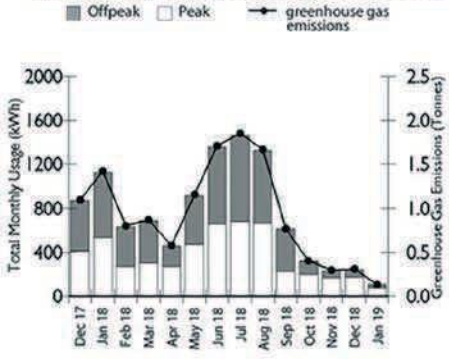
Account number	3333 333 333	Billing period	16 Oct 2018 to 14 Jan 2019
NMI	444444444444	Service address	Sample Rd, Sampleville, QLD 9999

Details Secure Saver - Home	Meter number	Bill days	Current reading	Previous reading	Total Usage	Charge/ Rate	$
Final Bill -16/10/2018 - 14/01/2019 - 91 Days	12345	91					
Energy Charges							
16/10/2018 - 14/01/2019 - 91 Days							
* Secure Saver - Home Peak Consumption			5860	5348	511.236 kWh	$0.3927000 per kWh	$200.76
* Secure Saver - Home Off Peak Consumption			4953	4749	204.495 kWh	$0.1884300 per kWh	$38.53
* Secure Saver - Home Supply Charge					91 days	$1.276000 per day	$116.12
Retailer Solar Buy Back Rate (3.37078 kWh/day)^					-306.741 kWh	$0.099 per kWh	$30.37Cr
* QLD Government Electricity Rebate					91 days	$0.93379 per day	$84.97Cr
Total Current Charges (incl. GST of $29.14)							**$240.07**

An * means that the amount shown includes GST.
^ This figure is your usage shown as an average per day over the number of days that apply to this rate.

Next meter read
Your next meter reading is planned to occur during 9 - 19 Apr 2019.
Please ensure safe and clear access to the meter. If your meter is not accessible an estimate is taken.

Electricity usage and greenhouse gas emissions



Average daily use (kWh)
This account: **7.87**
Same time last year: **27.68**
Average Peak cost per day (incl. GST): **$2.21**
Average Offpeak cost per day (incl. GST): **$0.42**
Total greenhouse gas emissions (Tonnes) this account: **0.90**
For more information visit www.climatechange.gov.au
To reduce your greenhouse gas emissions, call us on 133 466 to find out how you can support green energy.

72% decrease in usage since the same time last year

Your household daily consumption (kWh)		7.87
Household size without swimming pool	**Average daily consumption benchmark (kWh)**	**Rating**
1 person	14.96	✓
2 people	20.21	✓
3 people	25.79	✓
4 people	28.69	✓
5 people	28.69	✓

Compare your electricity usage with similar households in your area.

How it works:

- Select the household size that represents the number of people in your home,
- Compare the 'Average daily consumption benchmark' to 'Your household daily consumption'.

To find out more about how average household energy usage is calculated and get some energy efficiency tips, visit www.energymadeeasy.gov.au

This benchmark applies to residential electricity consumption. It is calculated based on figures provided by the Australian Energy Regulator (AER) and is indicative only. Consumption benchmarks exclude some large appliances, such as pool pumps, and generation systems (e.g. PV). For more information or useful energy efficiency tips, visit: www.energymadeeasy.gov.au

Source: Energy Australia

Figure 10.44 (*continued*)

Forms

You have already seen how forms can be used to collect data and display the results of a query. Forms are also used for displaying the final results for the end user. In a transaction processing system, an invoice is a customer form that also contains a subform with all the customers' items for purchase and summary calculations.

Reports

A report is a published document that contains the results of a query visually arranged and formatted to allow clear display on a screen or on paper. Previously, reports were typically designed to be printed on standard paper, such as A4. Reports are now commonly designed for display on digital platforms such as websites. A report differs from a form in that it is only used for output, and may contain a large volume of data about many records for use by management.

Various design elements, such as text, pictures, lines and boxes, can be used to enhance the appearance of a report and improve communication of the information.

Database software makes it possible to organise the way the data is laid out on the printed page. In design mode, headers and footers can be inserted for the whole report and for individual pages. A report header might contain the report name and the company logo. The footer on each page might contain the page number and perhaps the date. If the number of columns is large, landscape orientation could be used. Grouping the data can also make it easier to understand the information presented. Such presentations of data are generally referred to as summary reports. They can be quite useful for users who do not want to wade through pages of detail. For example, at the library, the librarian might need a report that summarises the overdue loans so the students who borrowed the books can be quickly identified and contacted (Figure 10.45).

Overdue books

Monday, 14 December 2020
11:51:48 AM

BookID	title	LoanDate	First Name	Family Name	Year Group
321	Battle Sight Zero	18/11/2020	Boyce	Woodings	Year 9
690	Batman and Robin Vol. 1: Batman Reborn	30/11/2020	Sharia	Vearncombe	Year 9
591	Indistractable: How to Control Your Attention and Choose Your Life	9/11/2020	Nelson	Grigorescu	Year 10
297	Think Like a Freak: The Authors of Freakonomics Offer to Retrain Your Brain	24/10/2020	Lavina	Attoe	Year 10
183	The Mysterious Affair at Styles (Poirot)	28/07/2020	Constantine	Bachs	Year 11
52	The Raven	31/10/2020	Towney	Brislane	Year 10
742	A Forge of Valor (Kings and Sorcerers--Book 4)	28/09/2020	Sybilla	Miko	Year 10
454	The Weight of Honor (Kings and Sorcerers--Book 3)	27/09/2020	Dorolice	Please	Year 8
241	The Painted Man (The Demon Cycle, Book 1)	24/09/2020	Guilbert	Laurent	Year 12
1261	The Secrets of Saving and Investing with Alvin Hall: Simple Strategies to Make Your Money Go Further	6/10/2020	Laird	Blowin	Year 9
907	Platform Revolution: How Networked Markets Are Transforming the Economyâ€"and How to Make Them Work for You	13/11/2020	Arline	Wornum	Year 9
1019	The Neatest Little Guide to Stock Market Investing: Fifth Edition	16/10/2020	Danice	Stackbridge	Year 11
792	No Mercy: The brand new novel from the Queen of Crime	12/11/2020	Sergeant	Jessett	Year 7
1075	The Hitchhiker's Guide to the Galaxy: 42nd Anniversary Edition	8/11/2020	Jessie	Stelfox	Year 9
206	The Infinite Game	21/08/2020	Lionello	Crambie	Year 7
1178	Captivate: The Science of Succeeding with People	26/10/2020	Godart	Burns	Year 11
858	Eeny Meeny: DI Helen Grace 1	19/09/2020	Godart	Burns	Year 11
871	Batman: Arkham Unhinged Vol. 1	3/10/2020	Tatiana	Hubbart	Year 9
91	How To Win Friends and Influence People	20/11/2020	Angeli	Satterfitt	Year 10
433	Find Me: A TOP TEN SUNDAY TIMES BESTSELLER	15/10/2020	Hubie	Heiton	Year 12
934	One Small Step Can Change Your Life: The Kaizen Way	24/09/2020	Jacinta	Monkton	Year 9
1104	Death in an Elegant City: An English Village Murder Mystery	11/10/2020	Gerrilee	Sully	Year 10

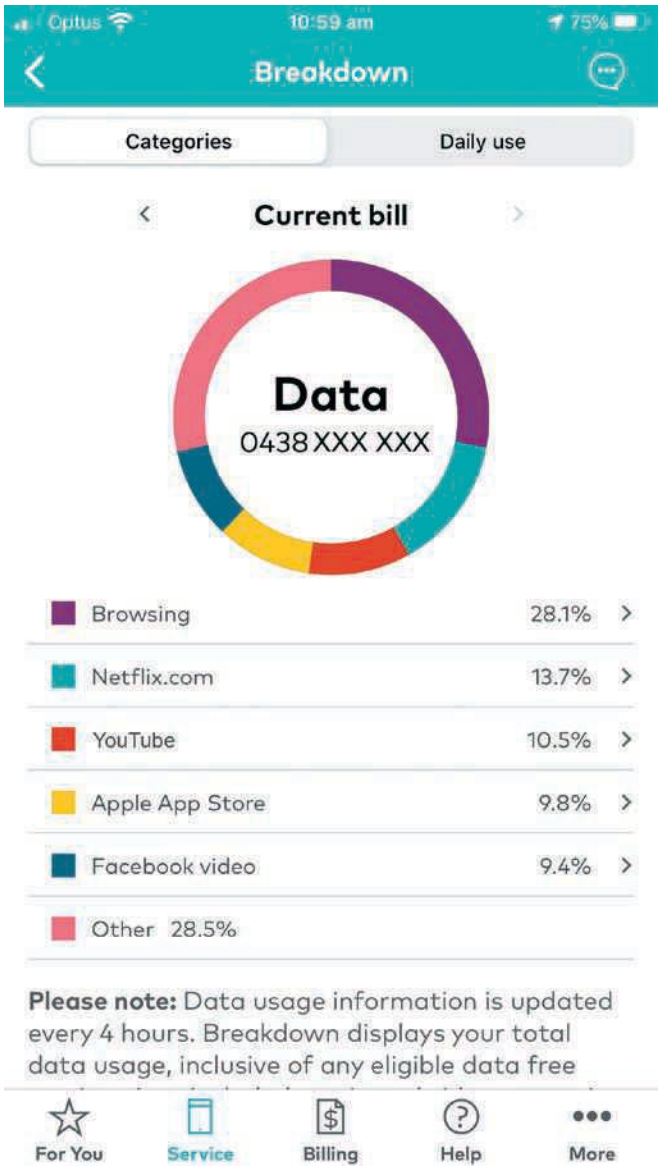
Page 1 of 5

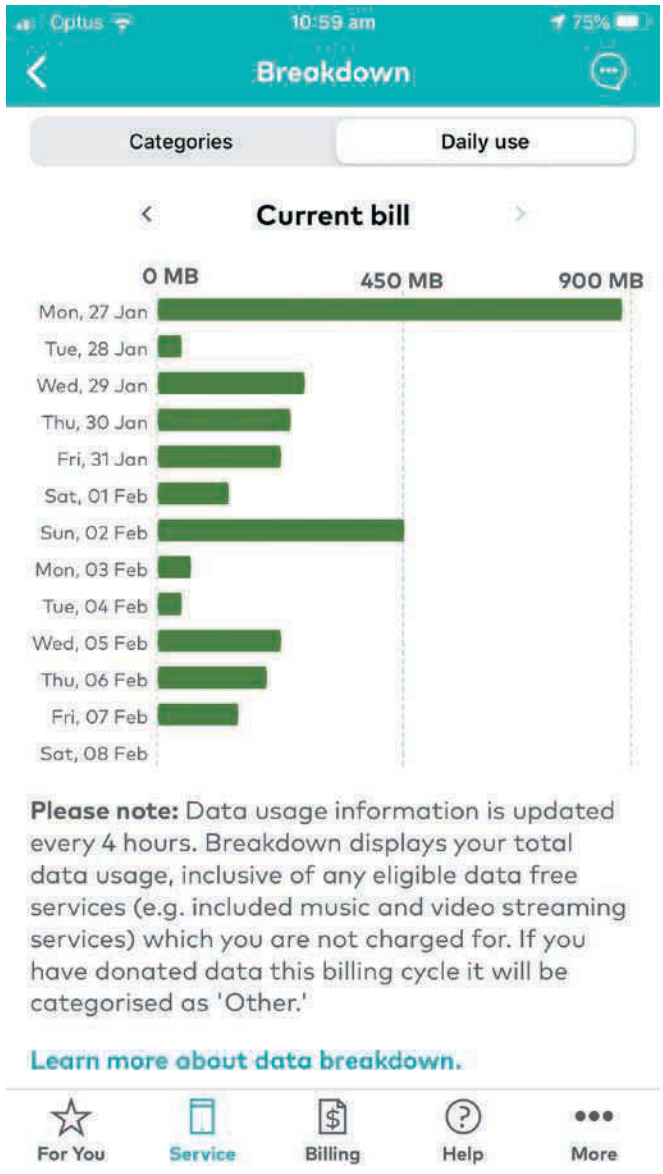
Figure 10.45 First page of a report of the overdue books from the Library database management system.

Visualisation

Often results from a database analysis are best presented graphically. Quantitative data is best displayed as a chart. For example, a pie chart is a clear way to visualise the distribution of parts to a whole, while a bar chart or column graph allows the comparison of different times or things.

Figure 10.46 shows the data breakdown visualisation in the Optus app that summarises the proportion of data usage across the main services used and the data usage over time. Database management systems, such as Microsoft Access, can generate charts that summarise key information, which are then included in reports (Figure 10.47).





Source: Optus

Figure 10.46 Data usage breakdown for a mobile phone plan.

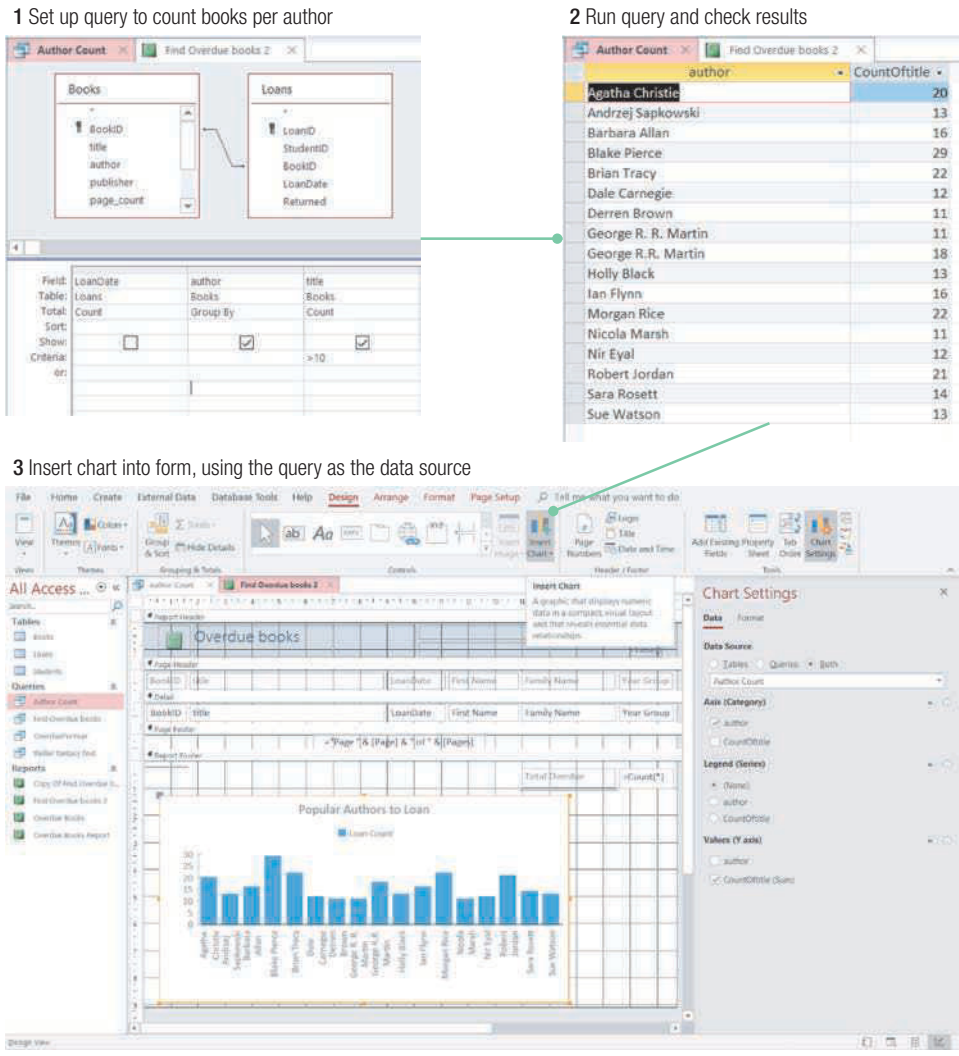


Figure 10.47 A chart can be added to a report or form to clearly display numerical information such as summary calculations. This column graph shows the most popular authors to loan.

SKILL BUILDER

School cafe DBMS – Part 5

Create a report

We will generate a report called 'Enviro Heroes' that will include the names of all the customers who used their own cup this week (Figure 10.48).

1. Create a new query. Only add the customer and order tables to the query, as we are only using data from these.
2. Complete the query table to find the first and last name of all the customers who used the VoucherReward 'Own Cup' in the past week.
3. Run the query and check the results.
4. Create a new basic report that will generate the contents and layout for you.
5. Modify the layout to fit nicely on a page and include the cafe logo.
6. View in print preview mode to check the design.
7. Print as a PDF document.

Chart visualisation

It would be interesting and useful for the staff and students running the cafe to know what the most popular drink, milk, delivery method and payment method are. We will generate a report that includes charts that visualise this numerical information.

1. Create four queries that count the number of each drink type, milk, delivery method and payment method from the orders table. Figure 10.49 shows the query to count the drink type.
2. Create a new blank report.
3. Add an appropriate title and the cafe logo to the header.

4 Create a bar chart for the drink type and place in the details section of the report. Complete the chart settings and properties to make the chart display as you like.

5 Create pie charts to illustrate the milk, delivery and payment query results.

6 Arrange all the charts in the detail section of the report so the layout is balanced (Figure 10.50).

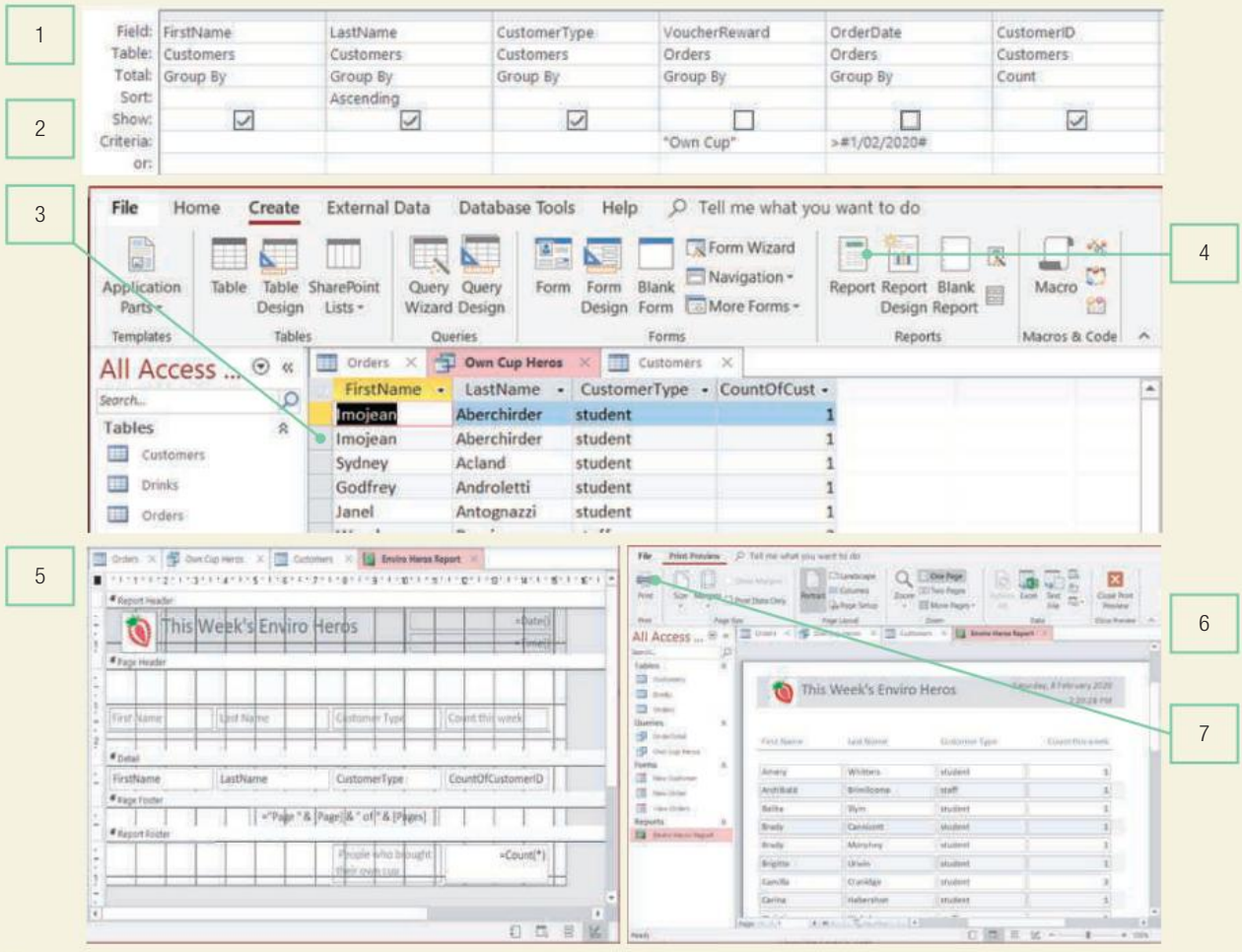


Figure 10.48 Steps to create the 'Enviro Hero' report from a query.

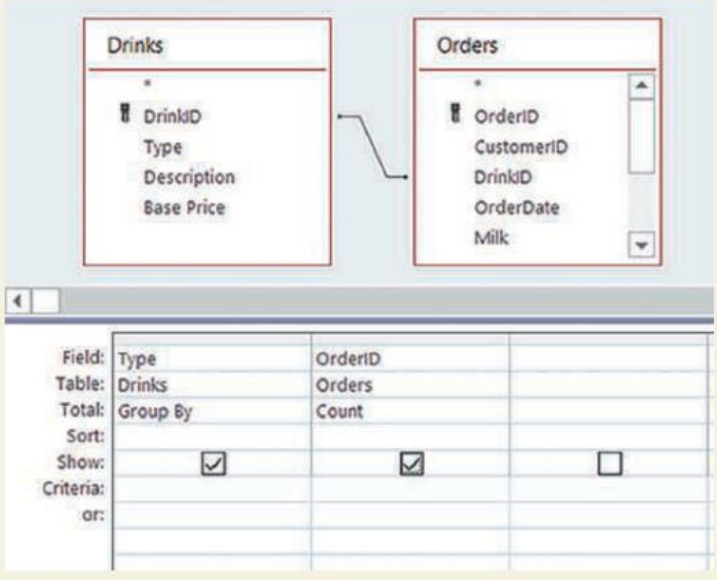


Figure 10.49 Query to count the number of each drink type ordered.

Figure 10.50 Chart visualisation of order summary information.

ACTIVITIES

Identify

1. What is the main purpose of building a report?
2. Where does the data come from that appears on a report?
3. In what ways can reports be formatted to improve communication of information?
4. Why are reports divided into header, data detail and footer? What information is displayed in each section?
5. What is the main purpose of a summary report? How are they produced?

Analyse

6. Why is a report classed as having an output function? Is a form input or output or both? Justify your answer using a supporting example.
7. A supermarket docket lists store information as well as items bought by the customer. In what way can this docket be classed as a report from a database? Which parts of the docket represent the header, the body and the footer of the report?

Extend

8. Create some more queries and charts to add to the order summary visualisation report.
9. Collect an energy bill from your house. Analyse the report sections, identifying what information is displayed in each and if any visualisation is used. Share and compare your findings with a peer.

10.6 Integration

Current software allows us to transfer data between different applications and types of computers. This means greater flexibility for using data in different contexts. We have already seen how organised data from another system can be imported into a database management system. In this section, we will look at this further as well as how data can be exported from a database management system for further analysis or display.

Importing existing electronic data

Most database applications are able to store graphics, sound or video clips in their data tables. Even though these files have been created in a different application, the database software is able to work with it by importing the data. It is also possible to establish links between the database and these media files, rather than import the files themselves. This has the advantage of keeping the database file sizes smaller, but it may restrict the portability of the application.

Many software packages now have options to integrate personal information, such as appointments, phone numbers and to-do lists, with email functions. Mobile phones and tablets may allow you to share information with your office-based computer and between software applications.

If a business needed to upgrade its database by having to re-enter all its data from the keyboard, it would be an enormous task. Fortunately, many software programs can understand each other's data, as long as it is in a common format – for example, delimited text. The delimited format can take various forms. One of the most common is the tab-delimited format, which uses the ASCII tab character to separate each field and a carriage return to separate each record in a database. Files can be interchanged between spreadsheets and databases using this technique.

Delimited text means that a text file contains words or fields that are separated by the 'tab' key press or by a comma, while records may be separated by the 'Enter' key press.

Data in many applications can be converted into plain text by removing all of its formatting. This option is commonly found in one of the menus of the application window, either as a 'save as' or an 'import/export' command.

Mail merge

Figure 10.51 shows how a **mail merge** function is useful for customising letters and producing mailing lists from a database. A chain of retail stores is planning a special promotion on a particular item. From their database, they can identify all customers who have spent a significant amount of money in their stores. A personalised letter including an offer of a 15 per cent discount might encourage the customer to visit the store once again. The mail merge option may also be used to print the labels for the mailouts.

A mail merge wizard in a word processor will help you through the practical steps.

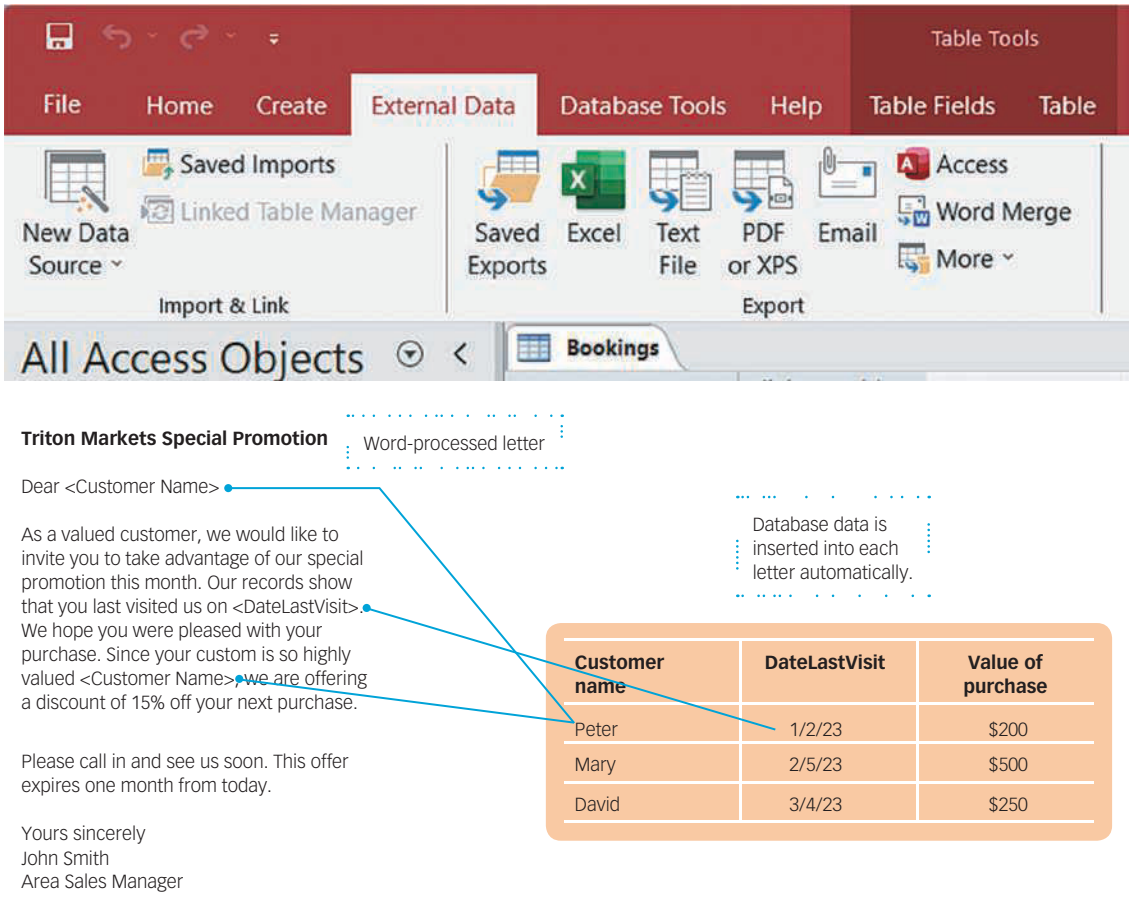


Figure 10.51 Mail merge allows us to personalise letters to customers and friends.

Exporting

Saving data in the delimited text file format is a useful way of transferring or exporting data between different database applications. However, some databases also allow you to transfer or link data directly to another type of application, such as a spreadsheet or presentation software. Although charting functionality is available in many database management systems, other applications such as spreadsheet software give users greater control and features, so being able to export the data is useful.

Web-based databases

Most databases exist online, stored in servers managed by cloud services such as Amazon, Google or Oracle. MySQL is the most common database management system used online. When a user would like to import data from a device or form into a database, or export the data from a database for display in another system, such as a website or mobile application, an application programming interface (API) is used.

In many database-driven websites, a server-side scripting language called PHP is used to create, access and edit databases. Figure 10.52 shows a webpage document (HTML) that includes PHP code to connect to a MySQL database and display some data from the database using an SQL SELECT command.

An advantage of storing databases on a network server is that it can be accessed by many people from different workstations or even in different countries. The data on the server needs to be secured by passwords and firewalls to prevent access by unauthorised people. If the data is sensitive or confidential, it may be encrypted, which means it is unreadable except by people with a special coded key.

Chapter review quiz

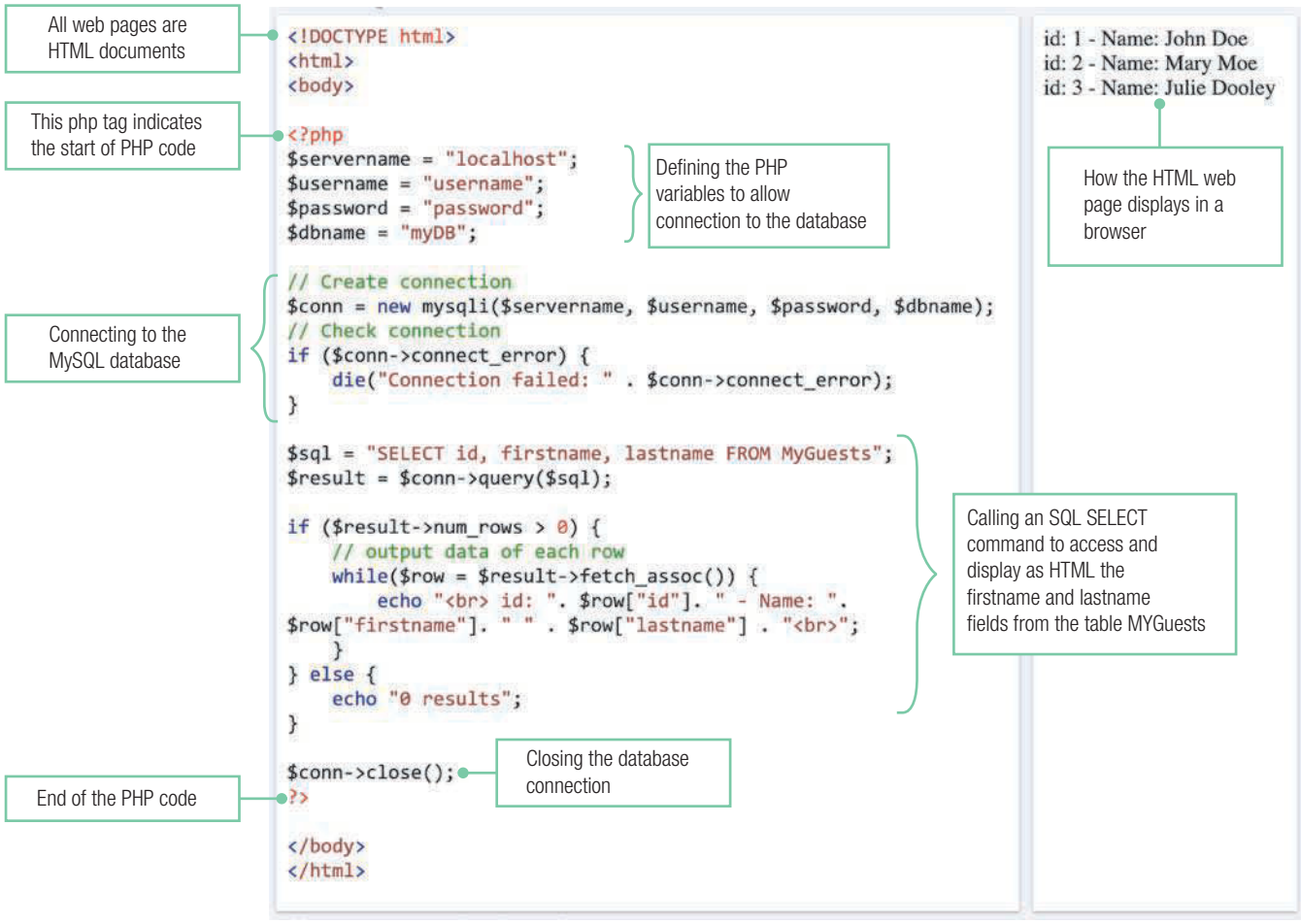


Figure 10.52 PHP code is integrated into an HTML webpage document to allow connection and access to a MySQL database. Data from the database can then be formatted using hypertext markup language (HTML) or processed using other web-based programming languages, such as JavaScript. Here the data is simply displayed as lines of text.

SKILL BUILDER

School cafe DBMS continued – Part 6

Online ordering

The goal of this task is to create an (almost) live data upload into Microsoft Access through a web-based form. To do this, we will:

- create a Google form to allow customers to order their drinks online (Figure 10.53). The responses from the Google form will be added to a Google Sheets spreadsheet that we will publish to the web
- use Microsoft Excel to create a connection to the web document to import data into a sheet every three minutes
- link the Excel sheet to a new table in Access that we can refresh at the click of a button.

1 Go to Google Forms and create a new form. Include all the fields from the database order table, plus a field for pick-up time. Use suitable control structures, such as multiple-choice and drop-down menus, for data validation and to make the form user-friendly. Style the form to be consistent with the rest of your information system.

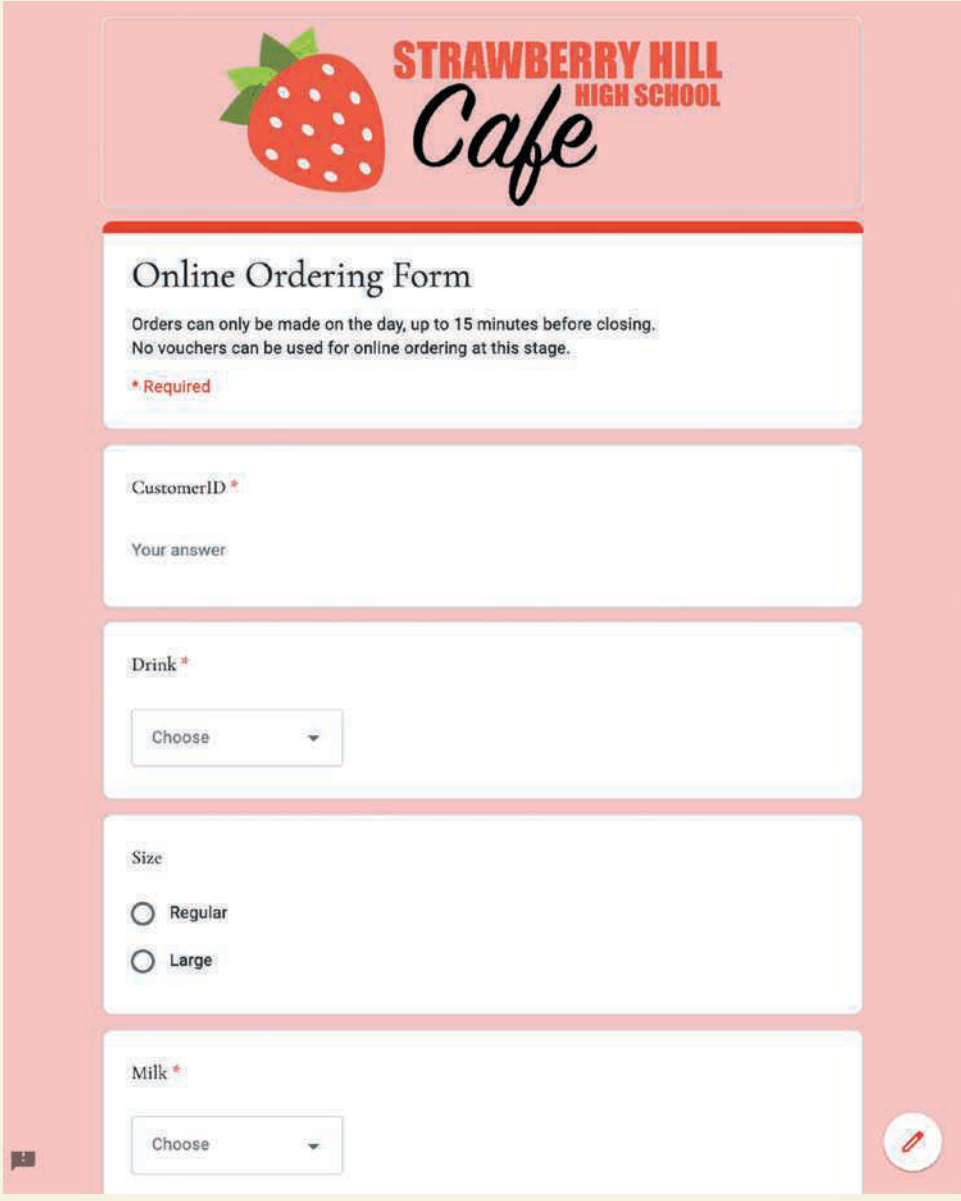


Figure 10.53 Published Google form for online drink orders.

2 Preview the form and add 3–6 test orders.

3 In the edit view of the form, select the Responses tab then click on the Google Sheets icon to create a new spreadsheet to store the responses (Figure 10.54).

4 Open the spreadsheet to view the response data.

5 Publish the spreadsheet to the web: *File > Publish to the web*. Complete the settings like in Figure 10.55. Copy the generated link that should end in '=csv'.

6 Open a new Microsoft Excel file.

7 Under the Data tab, click on the import data from web icon. Paste in the csv weblink and click OK. In the preview window, you should now see the data from your Google Sheets spreadsheets.

8 Change the refresh settings to occur whenever Excel is opened and every three minutes.

9 Save the Excel file.

10 In Microsoft Access, go to the External Data tab. Select Linked Table Manager then add a new link to an Excel data source. Browse to find the Excel file.

11 Work through the link spreadsheet wizard to check that all the data is correct. Name the new Linked table 'Online Orders'.

12 Add a new order through the Google form and watch it appear in your Access database.

	A	B	C	D	E	F	G	H	I
1	Timestamp	CustomerID	Drink	Size	Milk	Sugars	Delivery	Payment	Requested Time
2	2/9/2020 7:35:13	156	Espresso	REG	None	1	Take Away	Prepaid	8:00
3	2/9/2020 7:35:39	66	Cappuccino	LRG	Regular	0	Take Away	Cash	7:50
4	2/9/2020 7:36:00	37	Latte	REG	Soy	2	Dine in	Prepaid	7:45
5	2/9/2020 7:55:10	81	Hot Chocolate	REG	Regular	0	Dine in	Cash	8:10
6	2/9/2020 7:56:47	142	Cappuccino	LRG	Almond	1	Take Away	Prepaid	8:05
7	2/9/2020 8:11:56	122	Flat White	LRG	Soy	0	Take Away	Cash	8:25
8									

Figure 10.54 Responses from a Google form can be automatically imported into a Google Sheets spreadsheet.

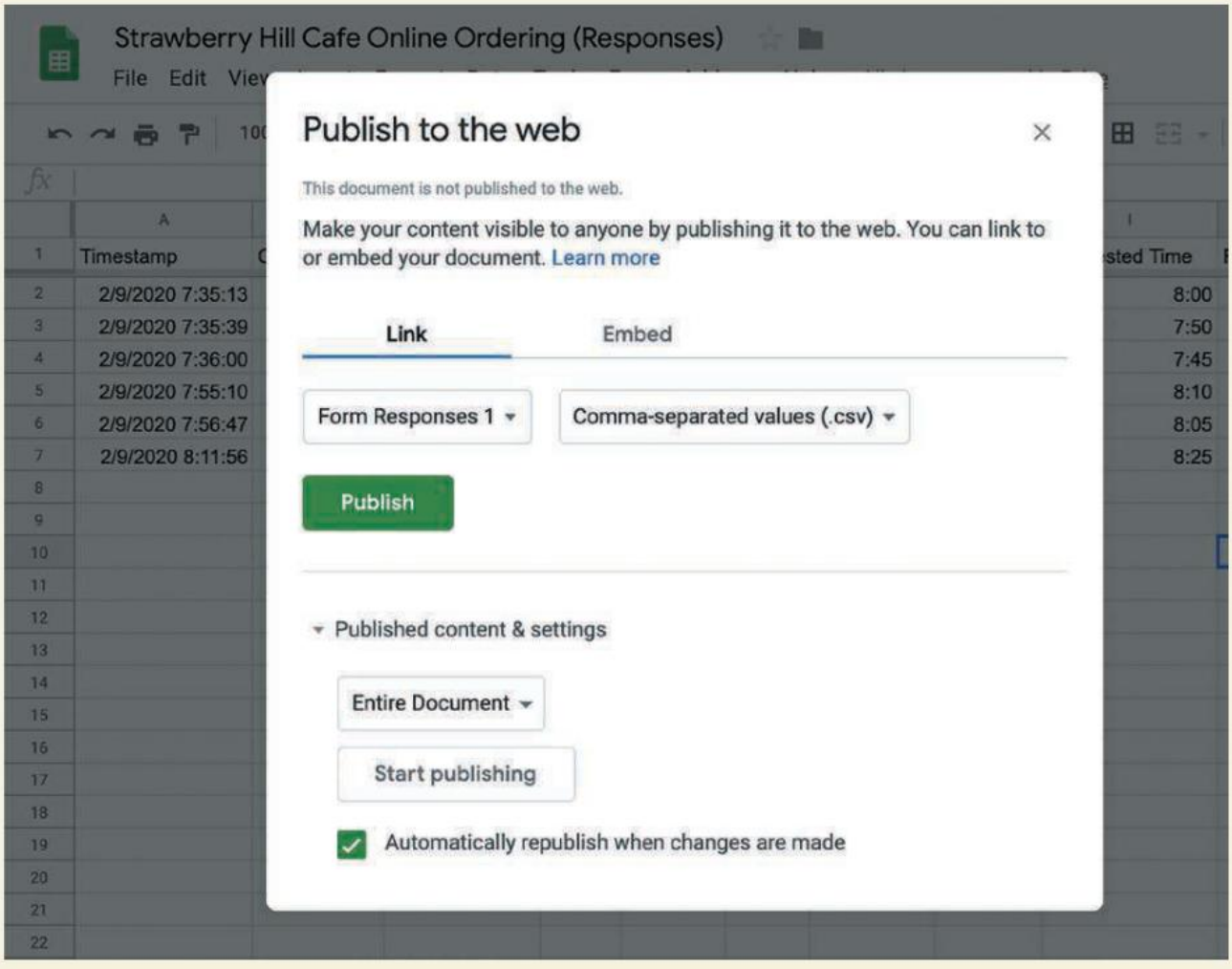


Figure 10.55 Publishing a spreadsheet to the web allows it to be viewed by anyone with the link.

ACTIVITIES

Identify

1 What is the advantage of linking media files to a database rather than importing media files? What is the disadvantage?
2 What is 'delimited text'? How is it commonly used?
3 What is the advantage of saving a document as a text-only file?
4 How does a mail merge function work? List the steps that you would need to create a mail merge document.
5 What are the advantages for a marketing manager in using a mail merge option?

Analyse

6 Some schools or clubs might use a mail merge option to print mailing labels. When would this be useful?
7 When would publishing spreadsheets to the web like in the 'Skill builder' activity be a security or privacy issue?

Extend

8 Export the order table from the cafe database to a Microsoft Excel file. Perform some data analysis, including calculations and charting on the data.
9 Visit the W3Schools website to explore how PHP works to create database-driven web pages.
10 Complete Grok Learning's 'Introduction to Cloud Computing' course to learn how Python and JSON is used with APIs to access structured data from online sources.

Weblink
W3Schools

Weblinks:
- Australian Institute of Health and Welfare
- Mockaroo
- Australian War Memorial
- SQLZOO
- W3Schools

Skill builder resource:
- School Cafe DBMS

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

9780170442787

10.7 Project: HPE event management

It's National Health and Physical Education (HPE) Day coming up and you have volunteered to design and manage a physical challenge event to encourage physical exercise and to build student spirit (Figure 10.56). The challenge event will allow students from all grades to compete in house teams (or individually) in a selected physical activity or activities (e.g. basketball free throws, push-ups, scavenger hunt) with prizes awarded to the teams with the highest score and to all students in the house with the greatest participation.

Shutterstock.com/Drazen Zigic

Figure 10.56 HPE Day is an initiative that encourages Australian school students to get active. Schools can host a wide range of HPE activities.

Your task

For this project you will need to design a database to store all the data associated with the event including the Houses, Students, Teams, Events and Results. You will use a suitable DBMS, such as Microsoft Access, to allow for user-friendly collection of data via a form, efficient access to and analysis of data via queries and informative displaying of information via reports.

Identifying and defining

Tracking your project development

Set up a document to record the key processes during the project development. This document will likely show an iterative process, as you continually plan, produce and evaluate different parts of the whole project over the project's timeline. A logbook is an informal way to easily note down progress. If a more formal documentation report is required at the end of the project, your logbook can be used as a reference when creating a report.

Challenge idea generation and definition

1. Brainstorm experiences where you competed against others either individually or in a team. Select one or two of these experiences and identify how points were allocated, who collected and stored the points, and how the winner (or winners) were determined.
2. Select a challenge event for the fundraiser, justifying your selection. Then identify the following factors/issues:
   - **a** The users (including who collects and analyses the data and whose data is being collected).
   - **b** The data that will be used and collected and its data type, and the method of data creation or collection (Table 10.3).
   - **c** Format and content for the final presentation of the project.

Systems thinking

Table 10.3 Data types and methods of collection.

Data	Data type	Method of creation/collection
Event No Event date	Number Date/Time	Each plank event is defined by the year group and gender of the students participating. These are both at the top of a printed form table used to record student results. All event records are added to the DBMS via the table view.
Student first name Student last name Gender Year House	Text Text Text Number Text	Students write down their name and house in the printed table form as they register to participate in the current event. They also need to note down their gender and year group as a form of validation that they are in the correct event.
Time	Time	Timekeeper starts timer at beginning of the event. As each student ends their plank, the time is recorded in the printed table next to their name. Once all students have finished planking, the completed printed forms are handed to the recorders who use the competitor form to add the student details then the new result form to add the plank time.

Identification of requirements and considerations

3 Define and document some of the functional requirements for your DBMS based on your experience with the cafe Skill builder, discussions with your teacher and the details in the description of the project (e.g. the entities you need to store data about).

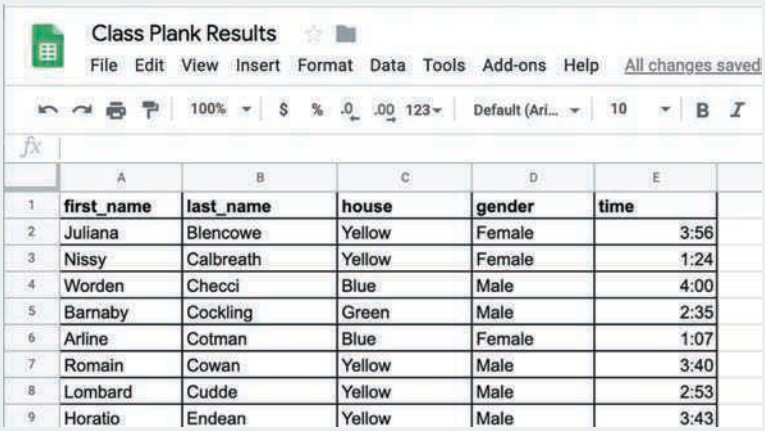
4 Define and document some of the non-functional requirements, such as how the personal details of the participants will be kept secure and facilitating the access to the database from different devices at the same time.

5 Discuss factors/issues that could impact the development of your solution (e.g. time, resources, money). Summarise this in your documentation.

Researching and planning

Researching through experimentation

6 Run a small, in-class challenge and collect the data manually. You could get everyone to do a plank (or other suitable challenge) and record the time results (Figure 10.57).



	A	B	C	D	E
1	first_name	last_name	house	gender	time
2	Juliana	Blencowe	Yellow	Female	3:56
3	Nissy	Calbreath	Yellow	Female	1:24
4	Worden	Checci	Blue	Male	4:00
5	Barnaby	Cockling	Green	Male	2:35
6	Arline	Cotman	Blue	Female	1:07
7	Romain	Cowan	Yellow	Male	3:40
8	Lombard	Cudde	Yellow	Male	2:53
9	Horatio	Endean	Yellow	Male	3:43

Figure 10.57 Data recording of sample mini-challenge event as part of research.

7 Reflect on the process of collecting and organising the data for the mini challenge. Consider how data was entered into the database system and whether there were any errors during collection. Identify whether a flat file or relational database is needed to organise the data then define the entities and events, recording this in your documentation.

Planning and modelling the DBMS

8 Plan how you will organise your data for the real challenge event. Use a schema to document what entities and attributes will you include and how these entities will be linked. At minimum you will need a table for Competitors, Results and Events (Figure 10.58). In the mini challenge the data could be split into Competitor (student_id, first_name, last_name, house, gender), Result (student_id, event_id, result) and Event (event_id, event_name, gender).

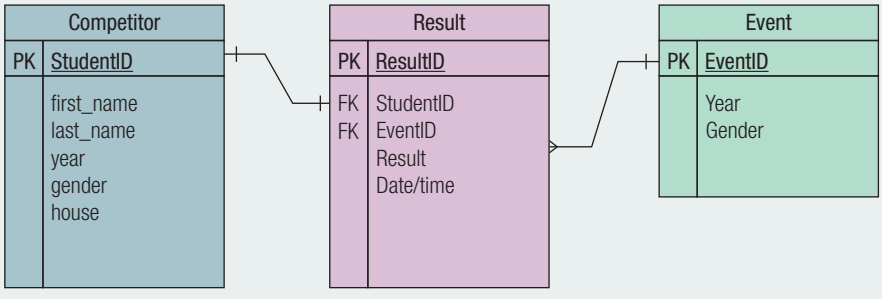


Figure 10.58 A database schema for a simple challenge event. An event has many results (one per student participating). There is a one-to-one relationship between the competitor and the result as a competitor can only enter the event once to get a single result. If you chose to run different events (e.g. plank and pushups) then this relationship will become a one-to-many relationship.

9 Prepare a data dictionary that identifies all the fields (including keys), their data types and any data validation that will be used.

10 Sketch a UI form that will be used for data entry and viewing records. Make sure you include suitable screen elements that will make data entry efficient and minimise user input error. Also include suitable data validation or verification steps/functionality.

11 Build a prototype of the form using Google Forms and have it evaluated by a peer and teacher. You may like to investigate the use of the pre-fill feature of Google Forms, which can make data collection more efficient and reduce errors.

Producing and implementing

Building and testing the DBMS

Note: Before you build your database, you must know exactly what data to include in your design and how it will be organised. Each of the required forms, queries and reports must have been documented, along with UIs and controls.

12 Set up the database in accordance with your models and designs.

13 Populate the database with appropriate test data in the tables. This may be entered using the form or generated using a simulated system and imported into Access. Create some sample queries and generate forms or reports from these to view the results in a user-friendly form.

14 Build any additional functionality you think may be useful to the system, such as online event registration or a mail merge feature.

15 When testing the system using test data or having a peer use the system, record issues that were identified and detail how they were addressed in your documentation.

Implementing the DBMS

16 Produce a user manual. Be sure that you explain clearly with screenshots what your application can do and how to do it.

17 Implement the system by running the challenge event, first on a small scale in your class.

18 Analyse the collected data using your queries and generate the reports to display the final results.

Testing and evaluating

User testing and feedback

19 Reflect on where in the data-collection process errors could be introduced. Evaluate what data validation and verification was included in your system to minimise inaccurate data. Record a summary of any issues in your documentation, including suggestions on how these could be resolved.

20 After implementing the system in your class, have users provide feedback using a survey.

Final evaluation and reflection

21 Reflect on whether the functional and non-functional requirements have been met. Display as a checklist in your documentation, with annotations as required. For example, explain how your solution considered and appropriately addressed social and ethical responsibilities, such as privacy of data.

22 Suggest and document three ways you could improve the system to be more user friendly or efficient and/or meet additional requirements.

Taking it further

This project can be modified to also address content from the spreadsheet chapter. The challenge event will be raising funds for a charity. To manage the funds for the day and to ensure you are going to make a profit that can be donated to the selected charity, you will be creating a budget using spreadsheet software.

11 Building mechatronic systems

FOCUS AREA: Building mechatronic and automated systems

Chapter outcomes

In this chapter, you will learn:

- about mechatronics
- the components of mechatronic systems
- about microcontrollers, actuators and sensors
- real-world applications of mechatronics
- to understand and use the sensors on a mobile device
- about autonomous vehicles
- to data log values from an accelerometer
- to design and build a range of mechatronic devices
- to code mechatronic devices using a general purpose language
- to use sensors to address accessibility issues
- to use analogue and digital data and convert between them
- about PWM
- to design and build a range of automated control devices.

Key terms

accelerometer
actuator
control loop
control system
data logging
end effector
gyroscope
infrared
LiDAR
magnetometer
mechatronics
microcontroller
microprocessor
PIR sensor
pitch
pulse wave modulation (PWM)
RADAR
roll
sensor
servo
solenoid
stepper motor
ultrasonic
yaw

9780170442787

IN ACTION

Life-changing mechatronics

From artificial joints to cochlear implants and insulin pumps, millions of people have medical devices implanted in their bodies that owe much to technologies commonly found in mechatronic systems. On the horizon are bionic eyes for the vision impaired, and muscle implants that allow paraplegics to stand and even walk. In Australia, researchers are exploring electrically conducting plastics that could stimulate and guide nerve fibres to repair spinal cords.

Artificial limbs, or prostheses, are used to replace a missing body part which may have been lost due to trauma, disease or congenital defect (Figure 11.1).

Research in robotics and digital technology has already resulted in substantial improvements in artificial limbs. Amputees may now have lighter, smaller, better-controlled, more lifelike and affordable options.

A current frontier for research in prostheses is to develop bionic legs which can be integrated with body tissues, including the nervous system. Bionic legs that are currently in development have been using the Raspberry Pi 'microcontroller' to coordinate muscle contraction signals and 'sensor' data from within the bionic leg to guess what a user is going to do next and then respond accordingly.

The next significant development in amputee technology is likely to be in the area of thought-controlled bionic limbs. Brain–machine interfaces are in the development phase and may eventually enable quadriplegic patients to control devices with their mind. Researchers believe they will be able to instantly respond to central nervous system instructions and, as a result, more closely replicate normal movement and functionality.

Questions

1. What is a prosthesis?
2. Identify five advantages of mechatronic over traditional prosthetics.
3. What function does a microcontroller perform in a prosthetic?
4. What potential improvements in prosthetics may be seen in future?

Flickr.com/Thomas Quine/CC BY 2.0

Figure 11.1 Basic artificial limbs have been used for hundreds of years. This example was designed during the First World War.

11.1 What is mechatronics?

Mechatronic devices can be described as 'smart machines'. A mechatronic engineer combines mechanical, electronic and software engineering to design and create computer-controlled machines that respond to their specific environment. Robots, drones and washing machines are familiar examples of mechatronic devices, however, **mechatronics** is set to impact our lives in far greater ways in the coming decade.

In this chapter, you will have opportunities to code, create and test your own mechatronic devices. In the projects you will build a remotely monitored irrigation system and an all-knowing automaton wizard. Although you will have plenty of experiences creating and using mechatronic devices, you will also learn about how mechatronics works. In the numerous Skill builders and bonus Nelson MindTap projects you create many interesting mechatronic devices, including a chatterbox puppet or a fascinating 'moody useless box' toy.

The importance of mechatronics

Environmental pressures, resource availability and climate change mean we need to be far more efficient in our consumption of energy and resources (Figure 11.2). Simple examples are around us already: such as escalators which only speed up when shoppers tread on a first step. Efficient lighting in many carparks and stairwells switch on only when a person enters each sector. Front doorbells with proximity sensors transmit to your phone video and audio of deliveries and visitors. Dishwashers which display how much water they are using, air-conditioners that sense when someone leaves the room, cars that monitor their maintenance and smart street lighting will become commonplace.

INFOBIT

The term 'mechatronics' was invented in 1969 by Tetsuro Mori, a senior engineer for the Japanese company, Yaskawa, by combining the words *mechanics* and *electronics*.

Mechatronics everywhere!

Look around school or home and you will see many devices which integrate mechanical, electronic and computing technologies. Whether you want to print a school assessment task, fly a drone, or blend your favourite smoothie for breakfast, mechatronics makes it possible. Modern aircraft use many mechatronic systems comprising thousands of mechanical and electrical components, each with sophisticated computer hardware and software.

INFOBIT

Hygienic handwashing: Apple Watch can automatically detect when its owner starts handwashing using its motion sensors and microphone by listening for water running and sensing relevant wrist movements. It then starts a countdown animation.

Mechatronics is making important contributions to the Internet of Things (IoT). It is a popular activity in the 'maker movement' and is the basis for many projects in STEM education. It is essential in the various fields of engineering.

In this chapter – as you encounter exciting current examples of mechatronic applications – use your imagination to conjure future possibilities for these technologies.

Alamy Stock Photo/0ZSH0TZ

Figure 11.2 The Return and Earn scheme in New South Wales uses mechatronic reverse vending machines to reduce litter and encourage recycling.

In the industrial world, mechatronic systems are now responsible for a wide variety of tasks, from controlling assembly lines to placing frozen dinners into trays. Mechatronic systems provide the basis for modern factory automation and manufacturing (Figure 11.3) where they perform a huge range of functions, including:

- welding and spray-painting
- assembly operations
- palletising and material handling
- dispensing operations
- laboratory applications
- 3D printing.

Systems thinking

Mechatronics introduces you to the components of mechatronic systems: **sensors**, **actuators**, **microcontrollers** and the circuits that connect them. You will design and build exciting mechatronic devices in both the Skill builders and the Project.

In the following chapter you will study in more depth two specialised examples of mechatronic systems – automated **control systems** and robotics.

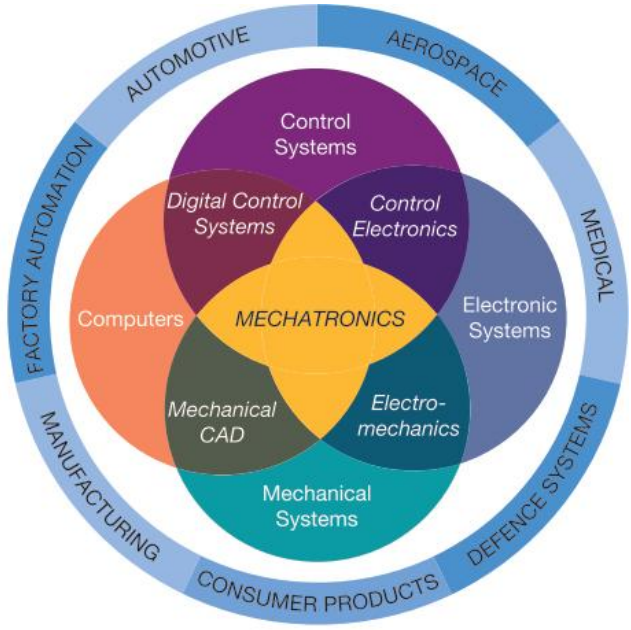


Figure 11.3 A mechatronic engineer combines mechanical, electronic and software engineering in designing and creating the wide variety of computer-controlled devices in our world and require the expertise of many specialties within these fields.

Source: Wikimedia Commons/Ahm2307/CC BY 2.5

Our approach to resources

This chapter has been developed so that you can learn about mechatronics and to program the micro:bit using a general purpose language such as Python without having to purchase a great many other components.

Not all schools or students have access to expensive kits of mechatronic sensors and actuators.

The micro:bit controller already has an impressive number of sensors built in. To make this unit as accessible as possible we have designed activities using just the micro:bit and a few other sensors and actuators. You can do most activities and projects with just a micro:bit and a **servo**.

However, in the Nelson MindTap resources we list some recommended micro:bit kits and provide more complex projects for those who would like to try them. Then in the next chapter we recommend some robots for that section of this focus topic.

11.2 Basic components of a mechatronic system

Sensors and actuators

The three main components which together form a mechatronic system are: *sensors*, *actuators* and a *microcontroller*.

Sensors measure their environment and send this data to the device's microcontroller, manipulate it in some way depending on its programming and instruct the actuators to perform some mechanical action.

Mechatronic systems often also include feedback loops to tell the microcontroller when, say, a bottle is full or an action is complete or needs adjustment (Figure 11.4).

The parts of a mechatronic device controlled by actuators are known as *effectors*. A robot arm that welds car bodies on a production line is an example of an 'effector' and the motors that move it are its actuators.

Microcontrollers

Microcontrollers are the brains of mechatronic systems. They are small dedicated computers on a single board (Figure 11.5). They receive and process data and interact with a range of system components including **accelerometers**, **infrared** sensors, motors, pumps and valves.

Although they are computers, microcontrollers are slow and have very little memory. However, they are cheap and very easy to interface with real world devices. You will find them in most electronics these days, from microwave ovens to televisions.

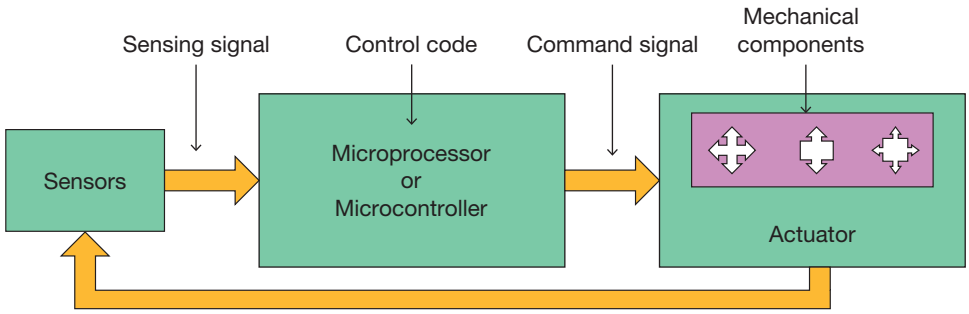
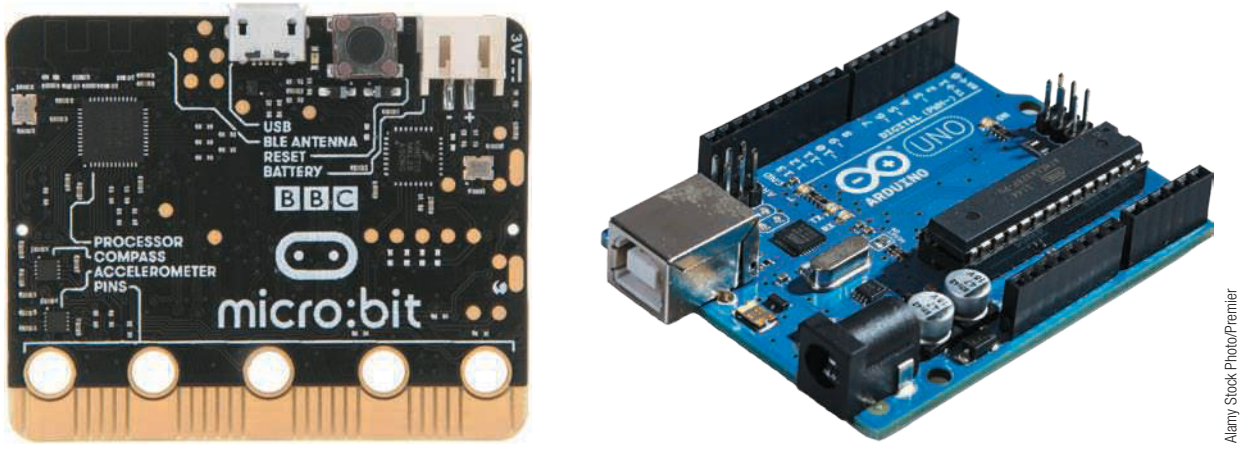


Figure 11.4 Mechatronic systems use sensors to send data to microprocessors which contain programs which instruct actuators.



Alamy Stock Photo/urbanbuzz

Alamy Stock Photo/Premier

Figure 11.5 Popular microcontrollers: a micro:bit (left) and Arduino (right). Microcontrollers can be described as miniature computer motherboards and are integral components in contemporary mechatronic systems.

ACTIVITIES

Identify

1. What two words form the word mechatronics?
2. Identify and describe an application of mechatronics found in most homes.
3. What is the function of a microcontroller in mechatronic devices?
4. Name a microcontroller used in prototyping mechatronic devices.

Analyse

5. Outline the difference between an effector and an actuator and provide an example of each.
6. Is there a difference between robotics and mechatronics? Explain your answer.

Extend

7. Research the operation of a typical desktop 3D printer, draw a diagram, identify its sensors and actuators and describe their functions.

11.3 Sensors

As human beings our senses are an essential part of our lives. They provide us with the ability to interpret our world. Our eyes give us sight, our ears interpret sound, our nose senses smell, our skin responds to touch and temperature and our mouth experiences taste. Although mechatronic sensors are modelled on nature, they typically cannot match the amazing range and precision of our human senses.

Sensors are input devices. Mechatronic devices use them to collect real world data such as temperature, light, noise, acceleration and movement. Sensors send this data to a central processing unit ('**microprocessor**') as input for programming functions which produce an appropriate response, perhaps by moving an output device, such as an actuator, or switching something on.

Types of sensors

Sensors obtain information from their environment. The type of sensor used will depend on the type of data required by the system. Sensors can be analogue or digital and may be classified as either active or passive.

Active sensors require physical interaction between the sensor and another device. These sensors use sound waves rather than light or visual data. For example, active sensors may be used to track people in a building or they may be used to trace the location of ships at sea.

Passive sensors require a change in the environment in order to operate. Most security systems use sensors called passive infrared detectors (PIRs) to detect intruders and turn on an alarm. PIRs are electronic devices that detect body heat when intruders enter or move around a protected area.

Table 11.1 Various forms of sensors are used to measure different types of data.

Type of sensor	Function
Temperature settings	Measure current temperature and may include control units, such as those used to control temperature in air conditioning units
Motion and pressure sensors	Detect sudden changes of movement
Optical/light sensors	Detect changes in light or changes in colour
Chemical sensors	Detect the presence and quantity of various chemicals

INFOBIT

Touchless thermometers are able to detect body temperature from a distance, of critical importance in detecting potential COVID infection. A person stands in front of a small lens that focuses the infra red radiation from our body and the temperature sensor displays the reading within seconds.

Common applications of sensors

Security alarms

There are many types of security alarms used in homes and commercial buildings. You may have one in your home. A common design is an alarm that uses a detector to pick up unexpected movements in its path. These types of systems have two main components:

1. a source of light, such as a laser beam
2. a photoelectric sensor.

The system may use reflection where the beam bounces back unless interrupted, or an uninterrupted beam directed at a receiving sensor.

In a home security system, a light beam (visible or infrared) may be aimed across a passageway. When somebody walks between the light source and the sensor, the path of the beam is momentarily blocked. The sensor registers a drop in light levels and sends a signal to the control box. This activates a timer and allows the homeowner to enter a security code. If the security code is not entered, an alarm sounds.

iStock.com/vchal

Figure 11.6 Passive infrared motion detectors (PIR) are often connected to security alarms.

Automatic doors

Automatic doors use a sensor to open doors at the required time. The sensor is often placed at the head of the door (Figure 11.7). It is passive and may be an **ultrasonic** device or an infrared detector. When the presence of a person is detected, a message is sent to the controller to open the door.

Shutterstock.com/Hatsaniuk

Figure 11.7 Automatic doors use optical or motion detection sensors to activate their motorised opening and closing mechanisms.

Vehicles

Late-model cars contain an astonishing array of sensors to allow engines and electronic systems to run at peak efficiency. Sensors are used to measure speed, revs, lane and obstacle proximity, fuel level, rain, tyre pressure, engine temperature, battery condition and to respond to collisions by inflating airbags.

iStock.com/teddyleung

Figure 11.8 Engine control units accept inputs from a range of sensors and display a variety of information about a vehicle's performance.

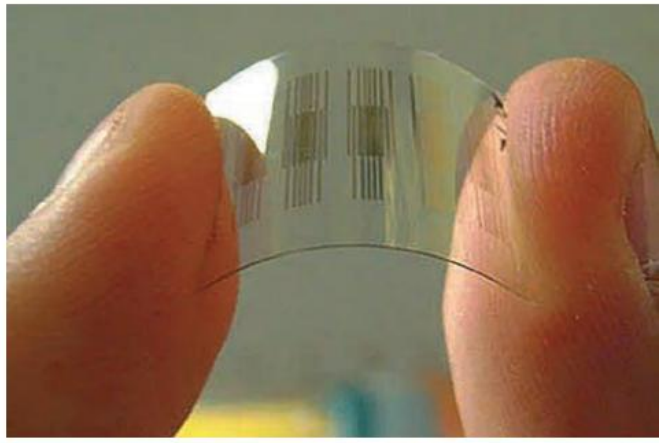
INFOBIT

Developments in chemical sensors now enable doctors to perform a blood test and obtain immediate results. Hand-held sensors allow physicians to get an immediate on-site chemical analysis, without having to wait for laboratory reports.

Nanosensors

Tiny sensors called 'nanosensors' offer great potential, particularly in medicine (Figure 11.9).

Development of these nano-enabled sensors is in the early stages, but the technology is expected to be a strong force in the treatment of a range of illnesses from the common cold to cancer. The present era of nanotechnology has reached a stage where scientists are able to develop tiny programmable machines that can work inside a patient's body.



Reprinted from Nanotechnology and Functional Materials for Engineers, Yaser Dahman, Chapter 6 - Nanopolymers, Copyright 2017, with permission from Elsevier.

Figure 11.9 A nanosensor embedded on a piece of plastic. Sensors such as these can detect explosives and toxins in water.

Nanosensors can measure chemicals and biological agents present in the air and were used during the Beijing Olympics to measure air pollution. One great strength of nanosensors is they can detect incredibly small amounts of viruses, bacteria and fungi. Cantilever (bending) nanosensors can even detect changes at the genetic level as shown in Figure 11.10.

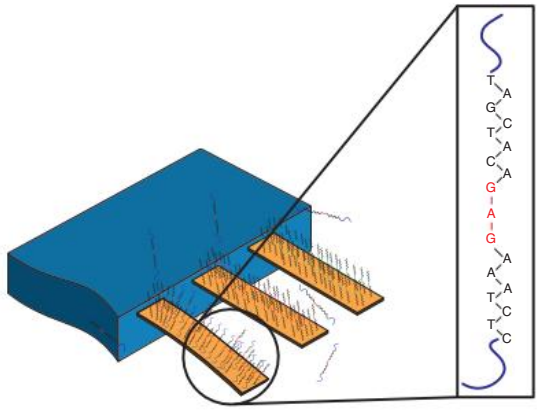


Figure 11.10 In this nanosensor used for DNA testing, the cantilever 'arm' on the left is encoded with a particular sequence for a mutation. If this same sequence is present in a sample being tested, the corresponding segment of RNA binds to it, causing it to bend and change its conductivity. This provides clear evidence that the genetic mutation is present.

Source: University of Basel, Department of Physics

In the future, nanotechnology may enable engineers to construct sophisticated nanorobots that can navigate the human body, transport important molecules, manipulate microscopic objects and communicate with physicians by way of miniature sensors, motors, manipulators, power generators and molecular-scale computers.

KNOWLEDGE BUILDER

Exploring your phone's sensors

Systems thinking

Although not a mechatronic device, your phone is an excellent way to familiarise yourself with the sensors used by mechatronic devices.

Table 11.2 lists some of the input sensors in a computer most of us use every day – a mobile phone. Phones have more sensors than a desktop computer! Mobile phones are not mechatronic devices but they are a great way to become familiar with various sensors we need to use later.

Individually, or as part of a group, copy and complete this table. If in a group, use a collaborative tool such as Google sheets to record responses.

Table 11.2 Some of the input sensors in a mobile phone.

Sensor	What does it measure?	What is it used for?
Accelerometer		
Ambient light		
Front camera		
Rear camera		
Proximity		
Gyroscope		
Magnetometer		
Barometer		
NFC		
Touch fingerprint		
Temperature		
Moisture		
Face ID		

SKILL BUILDER

Phone and tablet sensors in action

There are exciting and easy-to-use free apps for Apple and Android phones and tablets such as iPads which access and display data from the phone's many sensors.

1 Download *Arduino Science Journal* (Figure 11.11) (available for both Apple and Arduino). Press the plus symbol to create an untitled experiment. Press the pencil edit icon then select Sensors. Add multiple sensors to examine data for acceleration in each of x, y and z directions, barometric pressure, brightness, compass ('**magnetometer**' + 'accelerometer'), linear acceleration (which ignores gravity), magnetometer and sound intensity. Click the black 'i' symbol to read background explanations for each of these.

2 Make a data recording using *Arduino Science Journal* by clicking the Record button. After a few seconds moving your phone around press Stop. Click the back arrow at the top to see the actual recording. Select it. Click 'Review' at the top to see the actual recording. Click the three dots and export the data by selecting Save to files (the file will be in .csv format for spreadsheet analysis). It is also possible to send the whole experiment using Archive. This will be a file that can be imported by any device that has Science Journal installed.

3 Download the app *Sensors Toolbox* (Figure 11.11). Examine the accelerometer, 'gyroscope' and magnetometer data and graphs. Use the column graph icons to view data in the form of graphs and use the top right drop-down icon to export it in CSV format by clicking 'Share'.

4 Compare the UI and UX of these two apps. See Chapter 7.

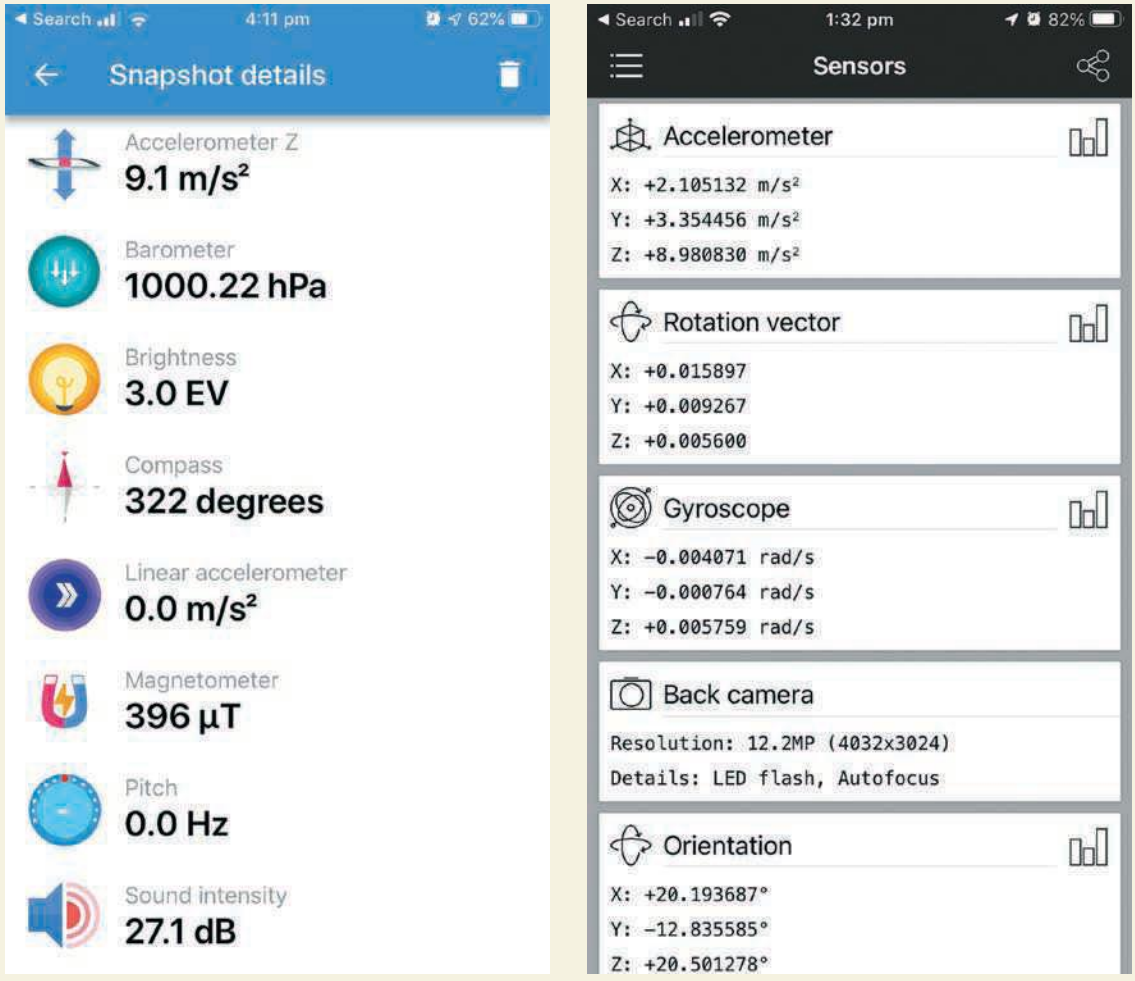


Figure 11.11 Screenshots showing the apps *Arduino Science Journal* and *Sensors Toolbox*.

ACTIVITIES

Identify

1 Identify five sensors found in a typical mobile phone.

2 What is the function of the magnetometer sensor in a phone?

3 What is one advantage of a nanosensor over a regular sensor?

Analyse

4 Describe the difference between an active and a passive sensor and give an example of each.

5 Describe the mechanism of airbag operation in a car accident, identifying the sensors and actuators involved.

Extend

6 Research nanosensors and describe one medical application for them.

11.4 Accelerometers

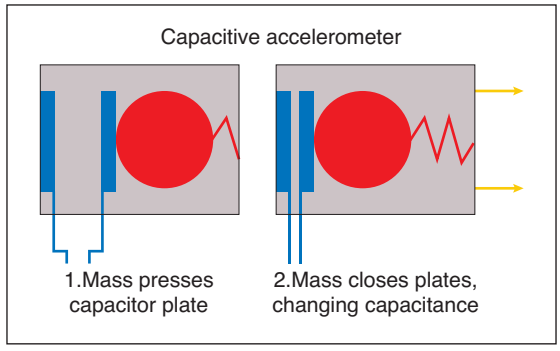
One sensor is used so widely we will examine it here in more detail. This is the accelerometer. Accelerometers are used to measure acceleration typically as three separate measurements: along each of the x, y and z axes.

Accelerometers have a great many applications in mechatronics:

- trigger safety airbags in car accidents
- in combination with **gyroscope** sensors, they keep robots upright and drones horizontal
- undo typing errors on phones by recognising the shake gesture
- count steps
- detect earthquakes
- measure acceleration during rocket lift-offs
- Nike running shoes use them to calculate the amount of time the shoe is on the ground versus the time it is in the air and mobile devices then convert the data into an accurate measure of the runner's speed
- in video game controllers, accelerometers sense the angle a controller is held when simulating swordplay
- mobile phones use accelerometers more than any other sensor – among many functions performed, they recognise a phone's tilt, adjust its display from portrait to landscape and eliminate unsteadiness and shake when taking photos.

How do accelerometers work?

Accelerometers are examples of miniature components called microelectromechanical systems (MEMS). Common accelerometers work by using a weight on a spring which presses against capacitive plates (Figure 11.12). Capacitors are electronic components which are able to store electrical charge for a short time. As two plates move nearer each other, the capacitance between them changes. From these changes, acceleration can be measured.



But the plates physically look more like this:

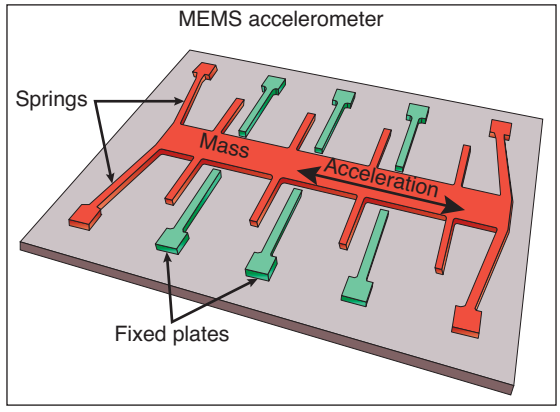


Figure 11.12 An accelerometer senses acceleration by measuring changes in the capacitance of its charged plates as they move.

KNOWLEDGE BUILDER

Understanding acceleration sensors

In the next section, you will explore accelerometers using a micro:bit microcontroller board – you can do this even if you do not own one as a simulated one is provided on the website! First you need to understand how acceleration is measured.

The tiny accelerometer on this micro:bit can be seen on its reverse side, see Figure 11.13.

From physics, you know that acceleration is produced when an object is acted on by an unbalanced force. In an accelerometer, these forces move tiny accelerometer weights in the x, y and z directions.

In Figure 11.13:

- x represents acceleration left and right along the x axis
- y represents acceleration forward and backward along the y axis
- z represents acceleration up and down along the z axis.

Even when a micro:bit is lying flat and not moving it will show the (unbalanced) force of gravity acting on it along its vertical z axis, but will show no acceleration in either the x or y directions.

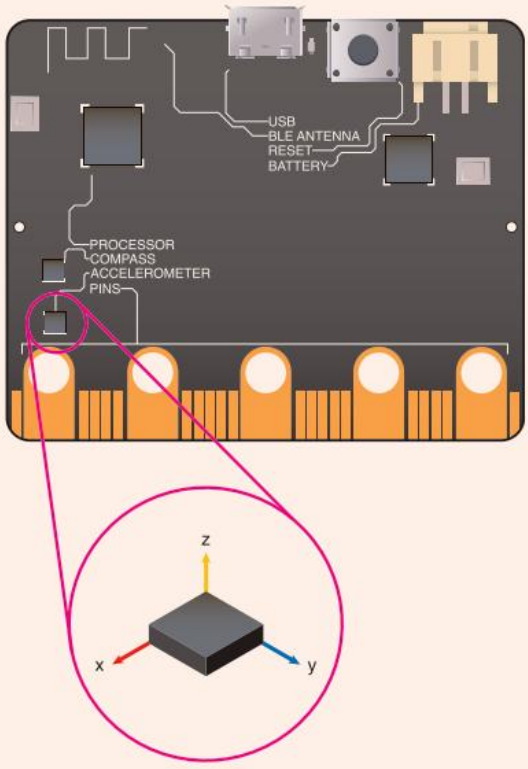


Figure 11.13 Direction of forces acting on the accelerometer acting along each of the axes of a micro:bit.

Calculating acceleration

Using Python and mathematics with which you should be familiar, we can find a value for overall acceleration if we know it for each separate axis. Acceleration is a *vector* quantity, which means it has a magnitude (size) and a direction. Imagine you are only interested in horizontal acceleration (only along the x and y axes if the device is lying flat).

To calculate the magnitude or strength of the acceleration (ignoring direction), you could draw the x and y accelerations (they are at right angles) using line lengths to represent their size, as shown in Figure 11.14.

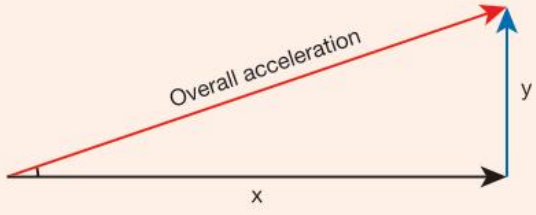


Figure 11.14 Overall acceleration can be calculated from those acting along each axis.

If the black arrow represents the acceleration in the x direction, and the shorter blue arrow the acceleration in the y direction, then the red arrow shows the overall effect. Imagine the black and blue arrows are attached to an object and are strings pulling in the directions shown, then the object would move in the direction of the red arrow.

The length of the red side of the triangle in Figure 11.14 can be calculated using Pythagoras' theorem (the triangle is right angled) to give the overall strength of the acceleration:

$$acceleration = \sqrt{x^2 + y^2}$$

The method is the same in 3D. We simply add the accelerometer's z axis into the formula. The total overall strength or magnitude of acceleration in 3D is given by:

$$acceleration = \sqrt{x^2 + y^2 + z^2}$$

Measuring tilt using an accelerometer

As well as straight line movement, an accelerometer can be used to detect tilt. This is one of its most useful mechatronic applications. It does this by sensing changes in the force due to gravity along each of the axes.

On the micro:bit:

- rotation about the x axis is called **pitch**
- rotation about the y axis is called **roll**
- rotation about the z axis is called **yaw**.

If you imagine the micro:bit to be an aeroplane flying forward, as in Figure 11.15, you will get the idea. In fact, aeroplanes use accelerometers in combination with other sensors for this purpose.

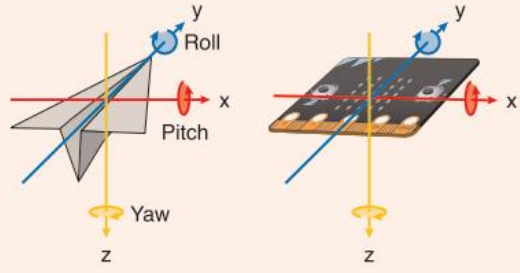


Figure 11.15 Using an accelerometer to measure rotation around each axis.

It is worth noting that if a micro:bit board is dropped its accelerometer in 'free fall' will measure zero as it would not be experiencing any forces. One of the best features of the micro:bit is that it uses its accelerometer to sense a number of gestures useful for different types of input: "up" , "down", "left", "right", "face up", "face down", "freefall", "3g", "6g", "8g" (where g is the acceleration due to gravity of 9.8 m/s2) and "shake".

SKILL BUILDER

A note about coding

In this course you are required to use a general-purpose language such as Python to code (introduced in Chapter 4). In this chapter we use visual Blockly code as a springboard for Python coding and to illustrate mechatronic concepts.

Capturing and graphing acceleration data

Here, we use MakeCode's ability to capture a live data stream from its sensors. We will investigate 'data logging' in more depth later in this chapter.

Coding

1. Using MakeCode's Block code editor, create and run the following program. You will find the blocks in the matching colour-coded 'Libraries'. Delete the 'on start' block, which appears, by dragging it off to the right.
2. Accelerations on each axis are shown as live data in the bottom right corner of the MakeCode's micro:bit simulator (see Figure 11.16). Experiment by 'tilting' the on-screen simulator board in different directions and observe changes in values. Notice that when the board is 'flat', only the z axis shows any acceleration (due to a downwards gravitational force of 1g = 1 000 milligravities) and displayed as -1023. If you flip an actual board over this will change to +1023.
3. The 'serial write numbers' instruction block (open the Advanced library) saves live data to an array. Replace your code with the serial write numbers array block shown in Figure 11.17. Move acceleration blocks shown into the array's white spaces. Press the 'Show data Simulator' blue button and Makecode plots values for x, y and z accelerations while you tilt the simulator board. Observe the effect on acceleration along each axis when tilting the board in various directions.
4. Note the live data being captured at regularly timed intervals in the lower window. The line graphs of the data can be viewed (Figure 11.18). Pause your simulation and export this data in .csv (comma separated value) format (using the download button top right). Open the exported file in a spreadsheet app such as Excel or Google Sheets and use the charting tool to create your own graphs.
5. Makecode also has the ability to record the data live from a real micro:bit – much more realistic than using its simulator! Connect your micro:bit via USB cable to your computer. Click the three dots next to Download (see Figure 11.19) to Connect your device. Make sure you have the latest firmware update on the micro:bit.

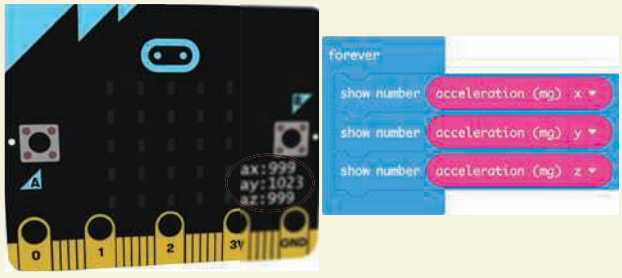


Figure 11.16 This simple MakeCode block program allows us to 'tilt' the on-screen simulated micro:bit in different directions and observe accelerations on each axis. Here the Makecode simulator is displaying these values on its board.

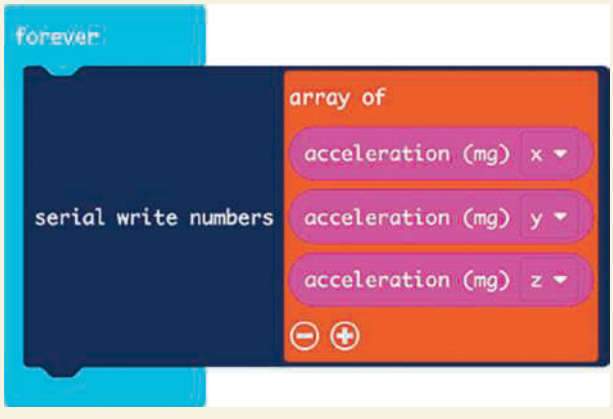


Figure 11.17 Observe the effect on acceleration along each axis when tilting the board by observing the values using 'Show Data Simulator'.

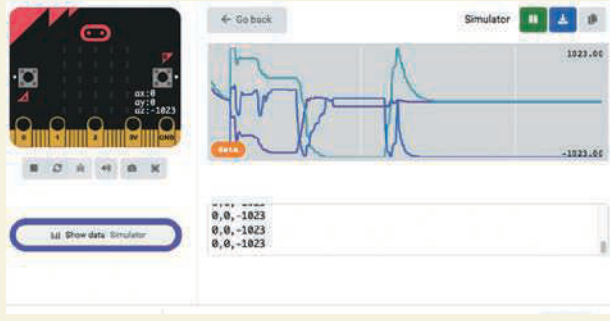


Figure 11.18 Line graphs (shown in green, blue and purple) showing live acceleration data for each axis.

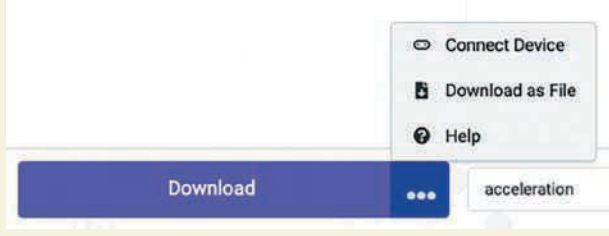


Figure 11.19 Connect an actual micro:bit and click the three dots to connect your board then click Download to download your code to your micro:bit.

Weblink
MakeCode's Block code editor

6 Now download your code to the micro:bit. A second button for data logging from your device ('Show data Device') will now appear below the one for the simulator as shown in Figure 11.20.

7 View your Simulator or Device data on the console. Click the blue download button above the graphs to download a .csv file of your data (see Figure 11.21). Open this in a spreadsheet such as Google Sheets or Excel. Delete rows containing unrepresentative data and use your spreadsheet's tools to create a suitable line graph.

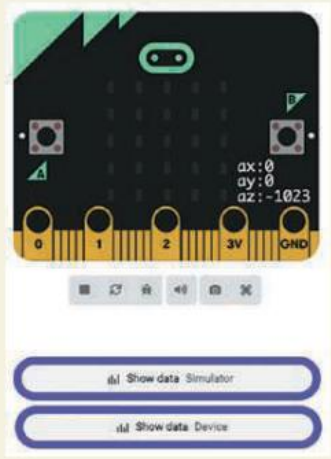


Figure 11.20 Capturing a live stream from a physical micro:bit rather than the onscreen simulated option.

20	4.076	10[illegible]	1023	102[illegible]
21	4.301	1023	1023	1023
22	4.532	1023	1023	1023
23	4.756	1023	1023	1023
24	4.988	1023	1023	1023
25	5.212	1023	1023	1023
26	5.44	1023	1023	1023
27	5.661	1023	1023	1023
28	5.888	583	641	-544
29	6.111	-156	-128	-1003
30	6.341	-277	-129	-977
31	6.562	-657	-239	-747
32	6.783	-838	-337	-481

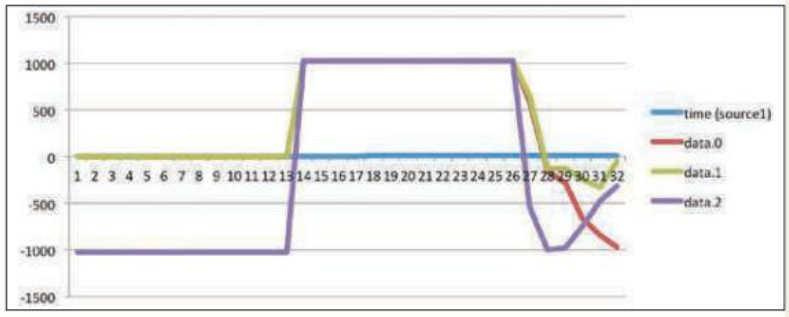


Figure 11.21 Section of live data stream of acceleration along each axis opened in a spreadsheet and graphed.

CASE STUDY

When sensors go wrong

In 2019, following a second catastrophic crash in five months of a Boeing 737 Max, this time belonging to Ethiopian Airlines, aviation authorities grounded the entire global fleet of 737 Max aircraft. A preliminary report stated the pilots were unable to prevent the plane repeatedly nosediving. The captain and first officer followed safety procedures recommended by Boeing but couldn't stop their aircraft going into a fatal dive shortly after take-off. All 157 passengers were killed.

A stall can happen when an aeroplane flies at too steep an angle (Figure 11.22). This reduces the lift generated by the wings, potentially making the plane drop. To recover from a stall, a pilot would normally push the aeroplane's nose down but in the 737 Max, an automated system was designed to lift it back to 'normal' flight position and repeat the process if it detects the plane is still tilted at too great an angle.

The report stated that a short time after take-off, two sensors that measure the tilt angle of the aeroplane's flight began to record different readings (see Figure 11.23).

If two sensors gave conflicting data, the system's algorithms were programmed to select the worst scenario. The system interpreted this as a dangerously steep climb and repeatedly pushed the nose of the aeroplane down.

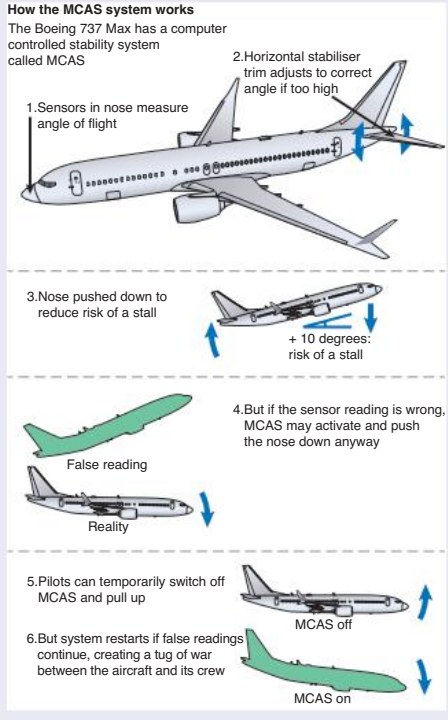


Figure 11.22 How sensors in the MCAS stability system work.
Source: Boeing, The Air Current and BBC

Systems thinking

The report suggested the pilots followed correct steps to disengage the system, but even after manually steadying the aeroplane, the system was designed to reset itself and pushed the nose down until the aircraft eventually crashed.

Adapted from 'Boeing 737 Max: What went wrong?', BBC News, 5 April 2019.

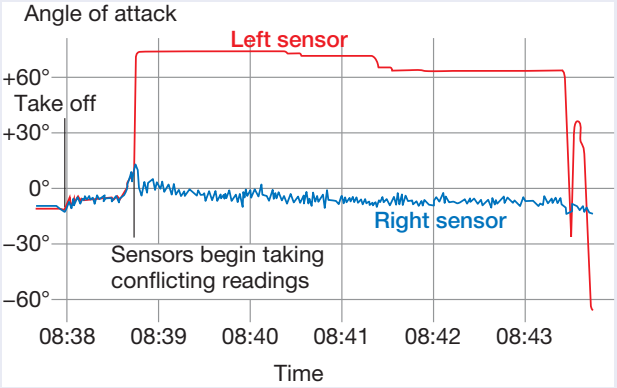


Figure 11.23 Different readings register shortly after the aeroplane's take-off.

Source: Ethiopian Aircraft Accident Investigation Bureau and BBC

You will now simulate this accident in the following Skill builder!

Questions

1. Explain the preliminary investigation's findings in your own words.
2. Who/what was the cause of the tragedy?
3. Assuming the initial findings are correct, identify three safeguards that might prevent such an accident.
4. It would be possible to disguise the cause of the accident by altering data collected during the flight. Although not an issue in this case, what precautions could be taken to guard data security in such a critical control system?
5. Similar challenges exist in building a completely autonomous (self-driving) car. These may involve mechanical, electronic or computer programming issues. The greatest challenge according to the researchers at the University of Sydney is anticipating unpredictable human behaviour. Identify five scenarios that might result in an accident and suggest a suitable response algorithm for each.

SKILL BUILDER

Building a pitch and movement alarm

A pitch tester

In this Skill builder, you will first simulate the accelerometers on the Boeing 737 Max (see previous Case study) then code a movement alarm.

The micro:bit's onboard accelerometer can measure rotation around each axis (called *pitch* in the case of tilting up and down around the x axis – see Figure 11.15) and identify when the board is horizontal (flat). A mathematical formula calculates the board's angle using the acceleration due to gravity on each of the three axes.

Before starting, interpret what the code shown in Figure 11.24 will do.

Systems thinking

Coding

Using MakeCode's Block editor, code and run the following program.

Note: You can always search for the correct block by entering a keyword in the Search field above MakeCode's libraries.

1. The accelerometer must measure repeatedly, so we place the code inside a forever 'loop'.
2. From the Variables library, create a variable and name it `pitch`. From the Variables library, drag out the `set…to` block.
3. From Input library under '…more', drag out the rotation block, selecting `pitch` from its dropdown menu. Complete the remaining blocks. Note that 'absolute' is a maths function which removes any negative sign in front of a number, and treats all numbers as positive (or zero).

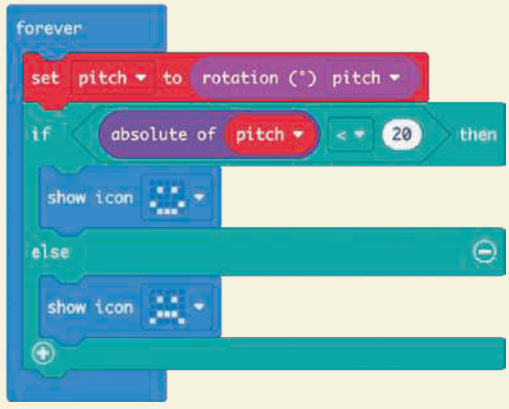


Figure 11.24 Simulating the Boeing 747 tilt sensor.

4. Test your code using MakeCode's micro:bit simulator. You can tilt the simulated micro:bit by moving over it with your computer mouse. Note that only tilting around the x axis (pitch) will trigger the frown face.
5. Load the code onto a real micro:bit and test it! Write a description of what is happening.
6. Add code to provide a readout of the pitch when button A is pressed and take a measure of actual pitch every 10 degrees by using a protractor. Enter data manually in a spreadsheet for the two sets of results and create and compare the two line graphs. Comment on the accuracy of the accelerometer's measurement of pitch.

Weblink
MakeCode's Block code editor

Skill builder resource
Building a pitch and Movement alarm

7 If you have a v2 micro:bit with onboard speaker, modify your code to create an alarm sound as well.

Questions

1 Why do we use inequality to test (if pitch < 20) rather than equality (if pitch = 20)?

2 What angle would you suggest to trigger the actual plane pitch warning?

3 Why do we take the 'absolute value' of the pitch?

Weblink
micro:bit Python editor

A movement alarm in Python

It's time to take your coding skills to the next level (Figure 11.25)! This time you use general purpose Python code rather than block coding to create a sophisticated alarm using an accelerometer. After triggering, your alarm will report time elapsed since it was armed, allowing you to determine the exact moment it was triggered. Imagine the many uses such a device might have!

In this course, it is important you code using a general purpose programming language. First, complete Chapter 4.

In this program:

- pressing button A arms the system. A triangle is then displayed for 2 seconds – enough time for you to remove your hands!
- a slight movement of the board will set off the alarm by detecting changes from a normal z axis acceleration.
- Turning the board over will disarm the system

Figure 11.25 Create your own movement alarm!

- Optional extras include radio transmission to a remote micro:bit and speech (v2 micro:bit)!

1 Read the code carefully and understand where each of the events described above occurs then draw a flowchart to interpret its logic. Why is the cross displayed on loading the code?

2 Using a micro:bit Python editor, type this code. We suggest using the micro:bit Python editor using the weblink provided. Lines beginning with # are comments, not code, and may be omitted. Do NOT copy and paste code if using the PDF version of this text, as tabbing and spacing errors will occur. Instead, copy each line of code then download it to your micro:bit. You will also learn more about coding by doing this.

```
from microbit import *
alarm_armed = False
# set static value
strength = accelerometer.get_strength()
while True:
    if button_a.was_pressed():
# arm alarm
        alarm_armed = True
# display triangle for 2 secs
        display.show(Image.TRIANGLE)
        sleep(2000)
        display.clear()
# reset static value
        strength = accelerometer.get_strength()
    if (alarm_armed is True) and accelerometer.get_strength() < strength - 100:
# display until reset
        display.show(Image.NO)
    if accelerometer.was_gesture('face down'):
        alarm_armed = False
        display.clear()
```

3 Add a feature to record the exact time the alarm is triggered! In MicroPython (the version of the Python language used for the micro:bit), the time elapsed since the board is powered on is given by:

```
running_time()
```

If this time is recorded at the moment the board is armed and then again when the alarm is triggered, you can calculate the number of milliseconds that have elapsed. Do this by inserting the additional lines shown in red.

```
from microbit import *
start = 0
end = 0
alarm_armed = False
# sets static value
strength = accelerometer.get_strength()
while True:
    if button_a.was_pressed():
        alarm_armed = True
        start = running_time()
        display.show(Image.TRIANGLE)
        sleep(2000)
        display.clear()
        strength = accelerometer.get_strength()
    if (alarm_armed is True) and (accelerometer.get_strength() < strength - 100):
        end = running_time()
        display.show(Image.NO)
    if button_b.was_pressed():
# changes millisecs to secs
        display.scroll(str((end - start)/1000))
    if accelerometer.was_gesture('face down'):
        alarm_armed = False
        display.clear()
```

- Test by arming with button A, wait a while, then shake the micro:bit and press button B. Time elapsed will display.
- Now that the alarm has triggered, keep shaking the micro:bit and press button B a few times.
- There is a *logic error* in the code, as the micro:bit keeps responding to movement and the elapsed time keeps increasing.

4 The code keeps updating the value of `end` whenever the accelerometer if statement is triggered. We need to change it to trigger once only until a reset. Change the accelerometer `if` statement to eliminate this problem. Also add a line to display 'NO' image in the button_b if statement to show the alarm has been permanently triggered. The solution is provided on Nelson MindTap.

5 Upload the program to your micro:bit, detach the USB cable, and plug in the micro:bit's battery pack. Arm your board by pressing button A and note the exact time on a clock. Wait a short time. Move the board a little to trigger the alarm (LEDs will display a cross). Press button B and read the elapsed time in seconds. Add this to the clock time you recorded earlier and you will have the exact moment the board was moved. The solution is provided on Nelson MindTap. This Skill builder will be handy!

6 Make improvements to this device beginning with the following (solution available on Nelson MindTap):

- transmit an alert to a second micro:bit
- add an alarm sound (v2 micro:bit)
- add speech (v2 micro:bit).

CASE STUDY

Accelerometers save lives

Detecting falls

iStock.com/sturti

Figure 11.26 Many older people wear a button-activated assistance alarm around their necks; however, these are little help if they are immobilised by a fall.

Preventing falls for older people is critical. Nearly half of those people over 70 who have a fall are admitted to hospital and if this involves a hip fracture, one-third will die within the year. Two hundred and fifty thousand people are hospitalised annually due to falls.

Not only senior citizens suffer falls (Figure 11.26) – mountaineers, construction workers, window washers, painters and roofers can experience life-threatening falls. It would be useful if an accelerometer could distinguish a fall from other movements, such as sitting or lying down.

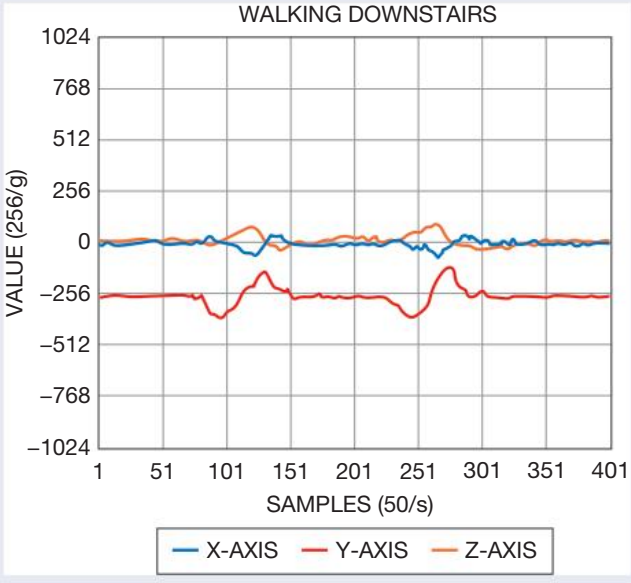
Examine the graphs in Figure 11.27 showing data from a fall detector attached to a person's belt. Measurements of acceleration for the x, y and z axes are shown, as well as the

overall (combined) strength. In this device, the vertical y axis is the one that experiences the force of gravity. This shows here as -256 with the x and z axes measuring 0.

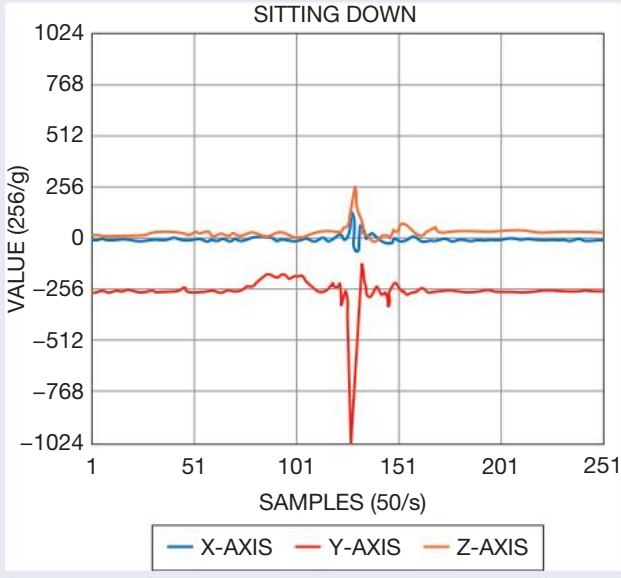
- *Start of the fall:* the person briefly experiences weightlessness and the green line (overall acceleration) tends towards zero (see 1 in graph (c)).
- *Impact:* when the human body hits the ground the green graph shows a large very brief spike (see 2 in graph (c)).
- *After fall:* The graph shows the time the person remains motionless (see 3 in graph (c)).
- *Comparing before and after:* after a fall, the person's body is in a different position. The red y axis shows a zero value but the orange z axis now shows +g. (Compare 4 and 3.) This is because the person is lying on their side with the z axis of their accelerometer pointing downwards.

Using this data, software engineers were able to develop and code algorithms to raise an appropriate alarm via the Internet. The device can send further critical alerts if the person remains motionless.

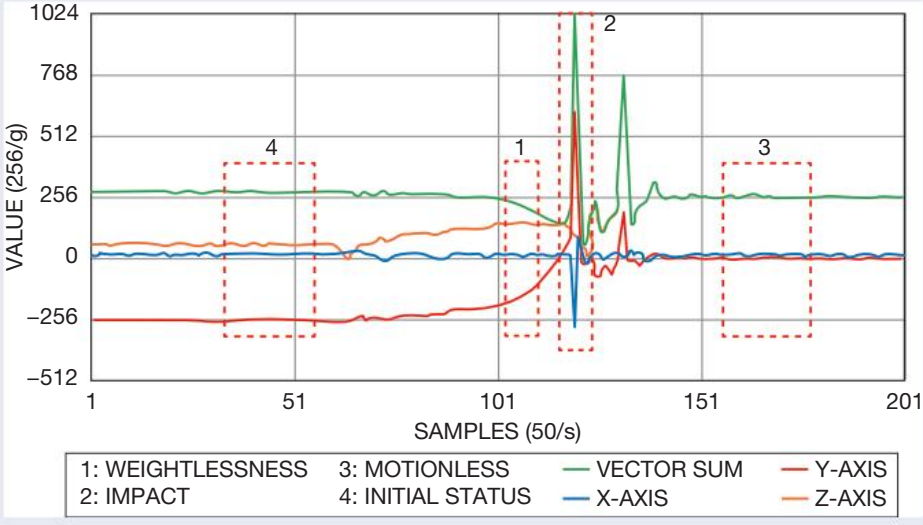
In the next Skill builder, you will recreate this case study experiment.



(a) Accelerometer data for walking downstairs.



(b) Accelerometer data for sitting down.



(c) Accelerometer during a fall.

(d) Fall detector.

Figure 11.27 Accelerometer data for different movements. (a) Accelerometer data during person accidentally falling. (b) A fall detector worn on a belt. The z axis will be vertical if the wearer is lying down. (c) The graph shows four features of a person having a fall. (d) Fall detector.

ACTIVITIES

Identify

1 Identify three different applications of accelerometers in mechatronic devices.
2 What common electronic component is used by an accelerometer?
3 Sketch the three axes for an accelerometer and indicate the gravitational force.
4 'RADAR' and 'LiDAR' are both used as sensors in autonomous vehicles. What do these acronyms stand for?

Analyse

5 Describe how an accelerometer sensor works.
6 Describe how x, y and z measures of acceleration taken by an accelerometer are combined to create one overall measure.

Extend

7 Describe how pitch, roll and yaw can be calculated using an accelerometer.

11.5 Data logging

Data loggers are electronic devices using built-in sensors to record data over time. These mechatronic devices can either store the data internally or transmit it to another device, usually a personal computer. By graphing the data, trends can be observed over time that would otherwise not be observed.

Data logging is used in many applications. Examples include weather stations, road traffic, wildlife counts, soil moisture, dam water levels, solar panel electricity production and pollution measurements.

In the following Skill builder, you will use a micro:bit as a data logger to recreate the previous case study.

SKILL BUILDER

Building a data logger to detect falls

Using the micro:bit as a data logger

You have learned earlier that the micro:bit can be used as a data logger. It can use automated logging over time to record temperature, light intensity, movement, magnetism, sound level and button presses for tally counts.

You can then import the data into a spreadsheet and view or present it as graphs, where you can examine and interpret it. It can also be exported to a spreadsheet where it can be analysed further.

Using visual code

1 Open the MakeCode website and create a new project called Fall data logger.
2 Using the Serial library create the program shown in Figure 11.28. We use an array here so all three graphs appear together. Next run it using the built-in micro:bit simulator. Click on *Show data Simulator* and examine the live data feed and graph while rolling your mouse pointer over the simulated micro:bit board.

Figure 11.28 Code for logging data for accelerations as a single graph.

3 Connect a real micro:bit and download the code. The *Show data device* button will appear. Click it and observe the characteristics of each of the x, y and z direction acceleration graphs by reproducing ('faking'!) the three situations from the case study: *steps, sitting* and *falling*. What do you observe?

4 Repeat the *falling* experiment while watching the live data graph then pause it. Export this in .csv (comma separated values) format using the blue download button (top right). Open the file in a spreadsheet. Select a relevant section of the data and create a line graph for this using the charting/graphing tool in your spreadsheet.

Analyse the three graphs and compare your observations to the conclusions of the previous case study.

Using Python

Weblink
micro:bit Python editor

1 Repeat the experiment this time coding in Python using your editor.

If you have a v1 micro:bit see 'Data logging with v1 micro:bit in Python' in the Nelson MindTap resources.

If you have a v2 micro:bit a data logging tool is included and the data file MY_DATA.HTM is produced right on our micro:bit!

2 (v2 micro:bit only) Download the following code and run it. Note if you previously used MakeCode, this connection may still be open and you need to close that window first before downloading Python code. Holding down button A will data log the x, y and z accelerations.

```
from microbit import *
import log
while True:
    if button_a.is_pressed():
        display.show(Image.YES)
        log.set_labels('x', 'y', 'z')
        log.add(
            x=accelerometer.get_x(),
            y=accelerometer.get_y(),
            z=accelerometer.get_z(),
        )
    else:
        display.show(Image.NO)
```

3 Open the MY_DATA file which will appear on the micro:bit to view your data. See its graph by selecting Visual Preview.

4 While holding button A, repeat the experiment from the previous falling experiment. Note to see the latest data that changed after you first opened the data file you must unplug your micro:bit and plug it back in again. To completely clear the log, first unplug the board then download the code again.

5 Download your data using the Download button and open it in a spreadsheet. Create your own graphs by selecting data in the spreadsheet and inserting a Scatter graph in Excel or a Line graph in Google Sheets.

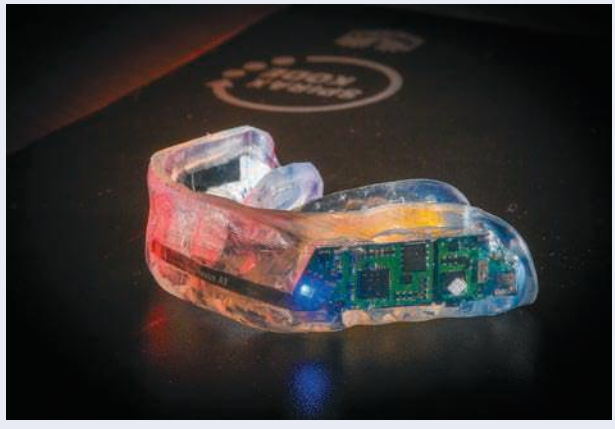
CASE STUDY

Accelerometers fight sports brain injury

Fairfax Syndication/Craig Golding

Figure 11.29 Collisions between two players can cause serious concussion injuries.

Two footballers converge on a tumbling ball. Heads clash yet one player is left unhurt and the other is knocked out cold. What causes one player to be so badly concussed while the other can play on? Why can seemingly identical collisions leave one player with brain injury and another with not even a headache (Figure 11.29)?



Fairfax Syndication/JUSTIN MCMANUS

Figure 11.30 Sports high-tech mouthguard with accelerometers.

HitIQ developed mouthguards with three accelerometers embedded. Data from the mouthguard's accelerometers helps team doctors diagnose and rehabilitate athletes after brain injury.

Accelerometers collect data describing forces along each of the x, y and z axes. By fitting the mouthguards to the maxilla, the fixed upper jaw, it is as if the sensors are attached to the skull (Figures 11.30 and 11.31).

Fairfax Syndication/JUSTIN MCMANUS

Figure 11.31 The HitIQ mouthguard being tested on a crash dummy.

HitIQ believes the data provides evidence that rotational accelerations are far more dangerous than direct linear ones. Rotational force is like a right hook or uppercut to the jaw, knocking a fighter's head off its axis and causing the brain to twist inside the skull, damaging connective tissues and disrupting essential functions.

Adapted from Chip Le Grand, 'Hi-tech mouthguards in fight to tackle concussion and its consequences', *Sydney Morning Herald*, 30 August 2019.

Questions

1. Research advertisements for fall monitors, find their cost and describe how they operate.
2. Why are falls treated so seriously in the case of older people?
3. What are the characteristics on an accelerometer graph which indicate a fall?
4. What are the characteristics on an accelerometer graph which might indicate brain injury in sports players?

SKILL BUILDER

Testing rotational impact

Using the data logging techniques of the previous Skill builder, perform an experiment similar to the above case study either using Python coding or visual coding using MakeCode's rotational acceleration blocks from the Input library (these appear by clicking *...more* under *Input*).

Write down your conclusions for the effect of a simulated twisting impact for *rotational* acceleration. As only pitch and roll are available, the axes you need to test may need the micro:bit to be held vertically. (See Figure 11.15.)

11.6 Distance sensors

Another important category of mechatronics sensor measures distance.

Your phone has a proximity sensor near the top of the screen which turns off the display when you are holding it near your ear to eliminate the annoyance of a bright screen light and your cheek touching onscreen buttons. This operates only when the phone's accelerometer detects the phone is upright (another use for the accelerometer!) otherwise it would trigger constantly. Your phone's proximity sensor emits a beam of infrared light and uses the reflected light to register if an object is within range.

Proximity sensors are common on production lines where they are used to sense when a manufactured object is aligned at a certain position for performing some operation (such as filling a bottle with soft drink or placing components in a device). These work by detecting changes in capacitance between the sensor and an object.

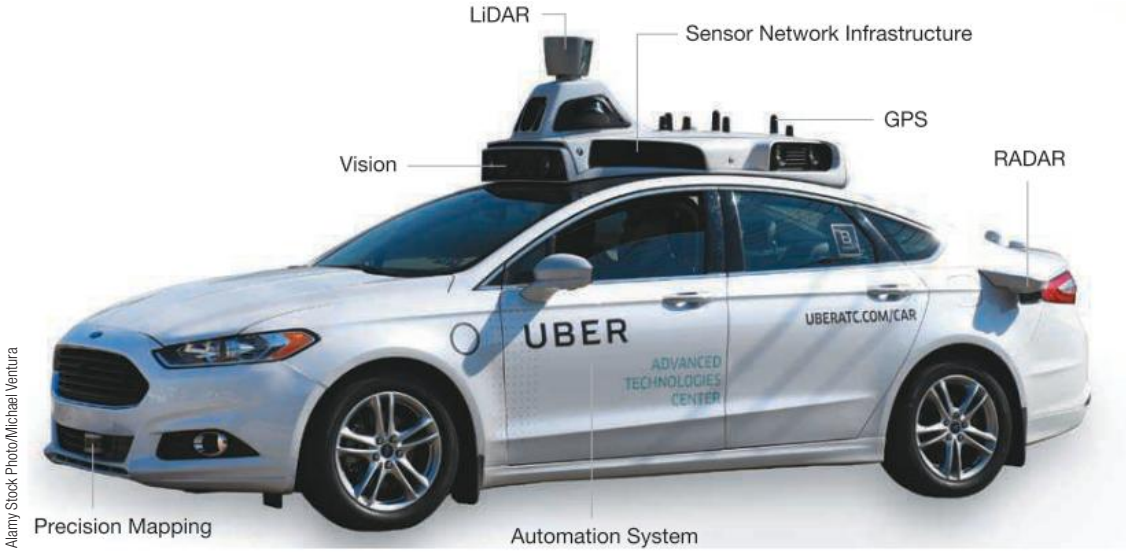
There are four main types of distance sensors (Figure 11.33):

Infrared sensors measure the distance to objects. As the light cannot be seen by humans, these can be used discreetly around our homes to trigger alarms or turn on lights.

Ultrasonic sensors measure distance using sound waves. They send out a sound wave at a known frequency and measure time taken for the wave to bounce back, converting this to a distance.

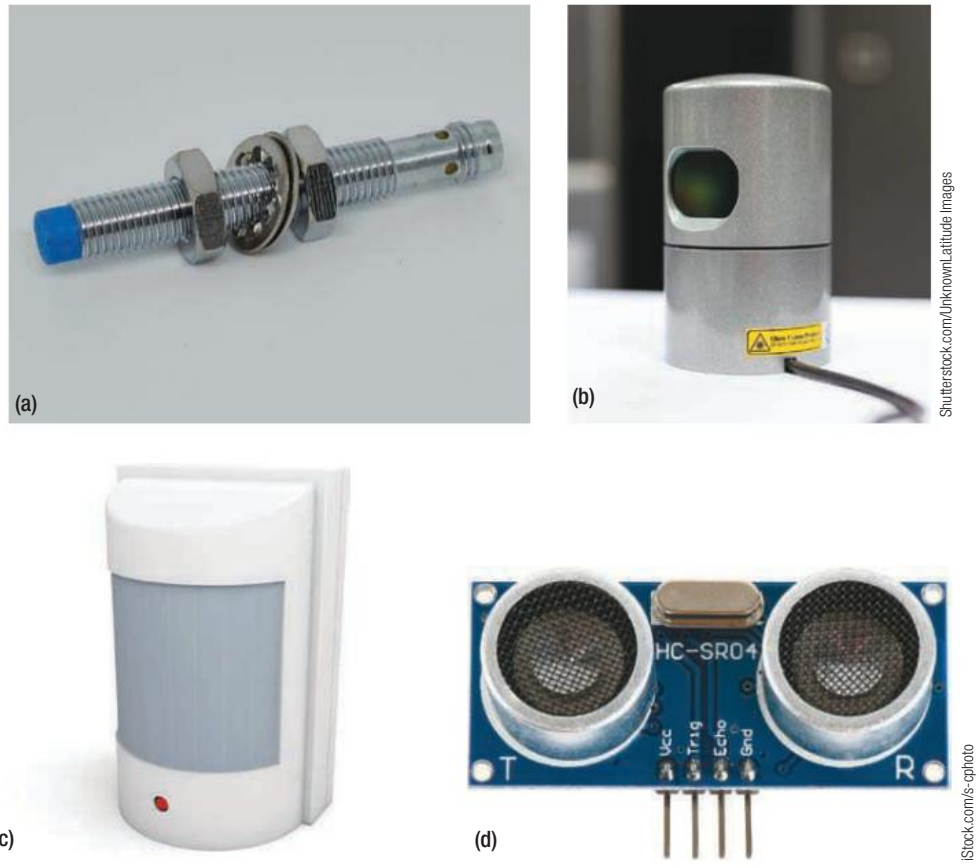
RADAR (RAdio Detection And Ranging) sensors use radio waves to evaluate distance or velocity by measuring the time for the reflected wave to bounce back.

LiDAR (Light Detection And Ranging) sensors illuminate a target with laser light and measure the reflected light to evaluate distance. Algorithms can use this to build 3D representations of distant objects (Figure 11.32).



Alamy Stock Photo/Michael Ventura

Figure 11.32 Proximity sensors are vitally important for autonomous vehicles. Image shows an Uber autonomous vehicle, using LiDAR and other proximity and distance sensors.



Shutterstock.com/Orest Malanchuk

Shutterstock.com/UnknownLatitude Images

iStock.com/Olivier Verriest

iStock.com/s-cphoto

Figure 11.33 Examples of distance sensors: (a) proximity, (b) LiDAR, (c) infrared, (d) ultrasonic.

Using ultrasonic sensors with micro:bits

The standard ultrasonic sensor for hobbyists and prototyping is the HC-SR04. Ultrasonic sensors need higher voltages and do not work well on the 3.3V supplied by the micro:bit. To build projects using this sensor you may need a micro:bit expansion board with a separate 5V power supply. There are many suitable expansion boards available. The Nelson MindTap resources offer some suggestions for suitable kits. However, we suggest a simple way to deliver additional voltage in the following Skill builder.

SKILL BUILDER

Using an ultrasonic distance sensor

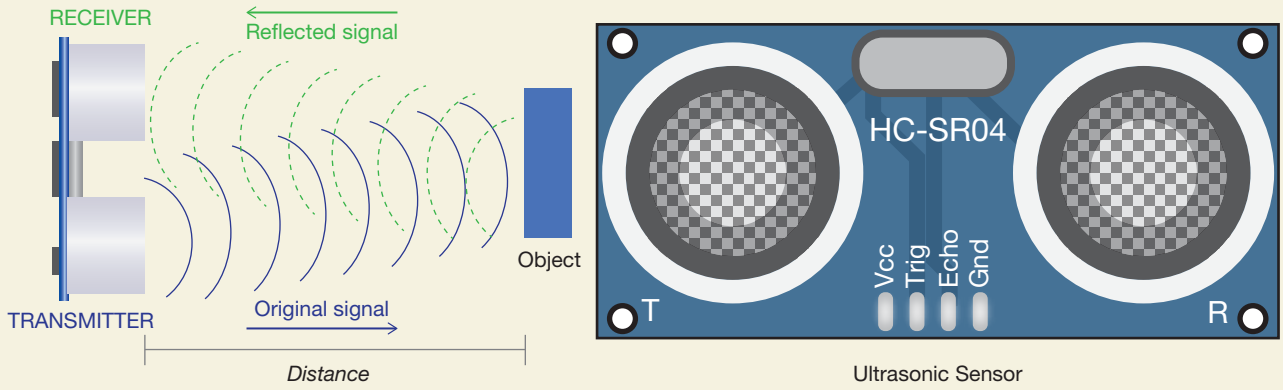


Figure 11.34 How an ultrasonic sensor works.

An ultrasonic sensor (US sensor) is a device that uses sound waves to detect the presence and proximity of objects. An ultrasonic sensor is cheap and can be extremely useful in mechatronics projects.

It sends out high-frequency sound waves that bounce off objects and then measures the time it takes for the sound waves to return.

This sensor (Figure 11.34) uses 40 kHz, which is twice the 20 kHz frequency of normal human hearing. It is also the frequency bats use for echolocation in caves.

Note if the target is positioned in such a way that the ultrasonic signal is deflected, rather than reflected back to the ultrasonic sensor, the calculator distance may be incorrect.

It is also important that the reflected surface is suitable. Objects such as fabrics and carpets can absorb acoustic signals which may then not be reflected back to the sensor or may be too small to detect enough reflected ultrasound.

Trigger and echo

The standard ultrasonic sensor has 4 pins. To start measuring distance it sends a pulse on the **TRIG (trigger)** pin which signals the **ECHO** pin to send a pulse and measures its duration until it is received again. That duration is the time elapsed between an ultrasonic burst being transmitted and the **echo** being received.

At room temperature, the speed of sound is 343 metres per second (34 300 cm).

So to calculate distance we multiply time 't' by 34300 to give the distance in centimetres travelled by the sound wave.

However it must travel both there and back, so we then divide this by 2:

Distance = (t/2) x 34 300 cm (t is measured in seconds)

In our coding time is measured in microseconds (which is usual for ultrasonic sensor code in mechatronics) so:

Distance = (t/2) x 34 300/1 000 000
(for t measured in microseconds)
= (t/2) x 0.0343 cm

Note, if it travels only 1 cm this equation becomes:

1 = (t/2) x 0.0343 cm, which gives t as 58.3 microseconds.

As these sensors can measure down to a distance of 3 cm, they are able to measure accurately as tiny a time interval as 175 microseconds or 0.000175 seconds!

Very impressive!

Coding the ultrasonic sensor

Materials required

- micro:bit
- 4.5V (3 x AAA) battery pack
- SR04 ultrasonic sensor (electronic shops or online)
- (optional) mini breadboard

How to wire your sensor

The HC-SR04 is powered by the two pins labelled VCC (+) and GND (–). Left to right, its four pins are: 5V(VCC), Trig, Echo and Gnd.

Although you can connect these pins to your micro:bit's 3V and GND pins as shown in Figure 11.35, it is likely not to work! Your readings might all be '0'. The reason is that they are designed for receiving 5V to operate properly.

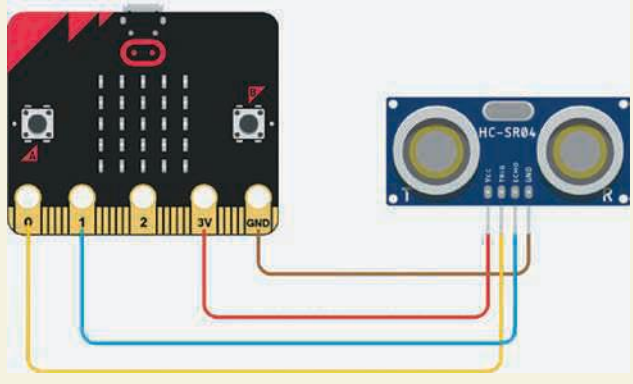


Figure 11.35 Simple wiring for US sensor, supplying only 3V from the micro:bit may not be enough.

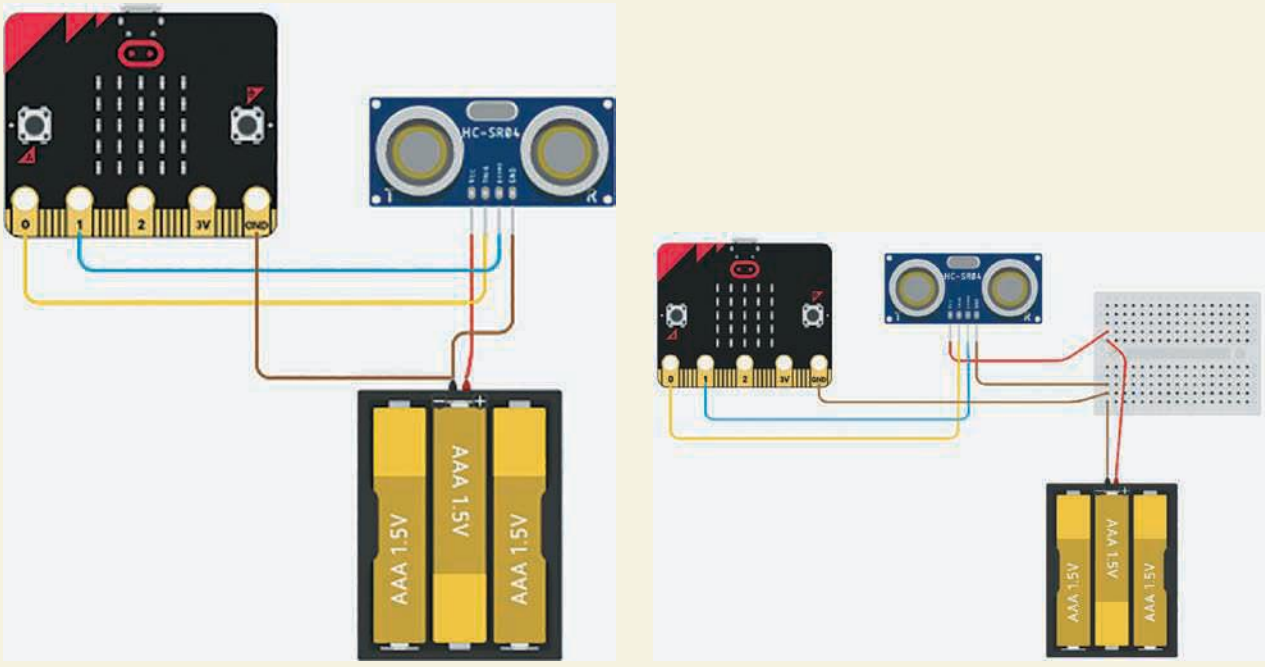


Figure 11.36 Wiring an ultrasonic sensor to supply 4.5V. (left) without a breadboard (right) using a breadboard. This sensor's pins fit into breadboard holes on the long side. Note that vertical lines of pins on breadboards are internally connected.

A better way

1 You can solve the power problem above by adding a 4.5V battery pack to power the US sensor.

All you need to do is move the + red wire from the micro:bit 3V pin to the battery's positive terminal and add an additional wire to connect the micro:bit's GND pin to the battery's negative terminal (see Figure 11.36).

Note that now both the sensor and the negative battery terminal will be joined to the same GND pin. This is because circuits need a common ground voltage level.

Coding the micro:bit to detect distance

2 Read the following code and understand what each section of code does.

```
from microbit import *

# special function to time pulse
from machine import time_pulse_us

# define pins for TRIG and ECHO
trig = pin0
echo = pin1
# setup I/O pins for TRIG and ECHO
trig.write_digital(0)
echo.read_digital()
while True:
# output a pulse to trigger sound
    trig.write_digital(1)
    trig.write_digital(0)
# time ECHO signal
    micros = time_pulse_us(echo, 1)
# convert time to seconds
    echo_time = micros / 1000000
# calculate distance in cm
    dist = (echo_time / 2) * 34300
    display.scroll(str(int(dist)))
    sleep(100)
```

Copy and download the above code to the micro:bit to enable your US sensor.

3 Test your sensor by placing your hand in front of it at various distances.

Save this code as we will use it in a later Skill builder for coding an automatic opening door.

11.7 Actuators

What is an actuator?

Actuators are mechanical devices used to move things. In mechatronic devices, they act as output devices.

In simple terms, actuators represent the muscles of automated systems. They convert commands into actions and perform the physical work of a control system.

Actuators are not always electrically driven. They may also be hydraulic (operated by liquid pressure) or pneumatic (operated by air pressure) and most are driven by software. An actuator connects a control system to its environment.

Many actions that were mechanical in the past have been replaced by electronic actuators. Some examples are automatic car window winders, windscreen washers, automatic boot opening and the operation of closing and opening valves, such as in automatic washing machines.

The most common action that actuators perform is rotation, although this can be changed into linear motion using a slider and a crank. In the case of steam locomotives, the action is performed in reverse, or rack and pinion (see Figure 11.37).

Weblink
How Slider Crank Mechanism Works

Shutterstock.com/Stokkete

Shutterstock.com/AndrejsProkt

Figure 11.37 Steam engine wheels turn linear into rotational movement using a crank and rack and pinion gears perform the reverse function.

Weblink
Gripper based on dielectric minimum energy structures

INFOBIT

Researchers strive to imitate characteristics of the actuators our bodies use – our muscles. Soft actuators are being developed to harvest fruit or manipulating organs when operating on the body. This will open up a wide range of new applications for mechatronics. Soft actuators move flexibly through changes at a microscopic level. Shape memory polymer (SMP) actuators are similar to our muscles, providing a response to a range of stimuli such as light, electrical, magnetic, heat, pH and moisture changes.

An SMP can be used as a gripper actuator.

Motors

Motors are easily the most common actuator. There are three very different types of motors used widely in mechatronics: DC motors (Figure 11.38), servos and **stepper motors**.

DC motors

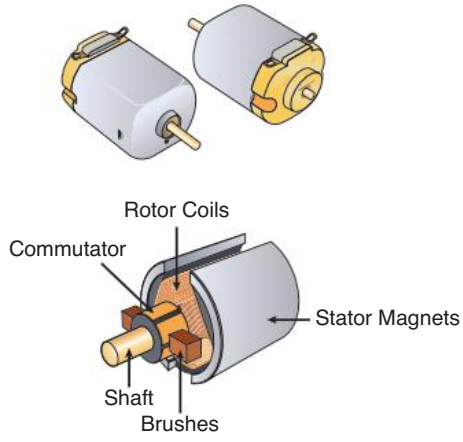


Figure 11.38 DC motors provide constant rotation.

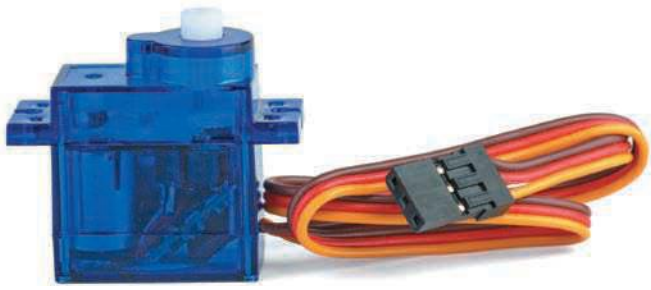
Weblink
How DC Motors Work

Most battery powered moving toys use DC motors. DC motors are constructed of many windings of copper wire around a rotor. When this rotor experiences the opposing magnetic field of the two permanent magnets around it, it is forced to rotate.

This also works in reverse! If a DC motor of this type is spun in reverse (such as by wind by attached fan blades) it produces a small current. (We will use this characteristic in a later Skill builder.)

This animation provides a detailed explanation of how DC motors work.

Servos



iStock.com/digitalr

Figure 11.39 A popular hobby servo motor used for prototype mechatronics systems.

Hobby servo motors (Figure 11.39) like the ones you will be using in many of your projects are different from standard DC electric motors as a voltage controls the *position* of the motors axis rather than the *speed* of rotation. A servo motor does not rotate constantly in the way a DC motor does (although it does have a DC motor inside it). Instead, it rotates to a set angle and holds that angle (Figure 11.40).

Weblink
How does a Stepper Motor work?

This behaviour is needed in a wide range of mechatronic applications. Examples include focusing for cameras, window and door openers, robotic arms, steering for radio controlled vehicles, drones and keeping cameras level.

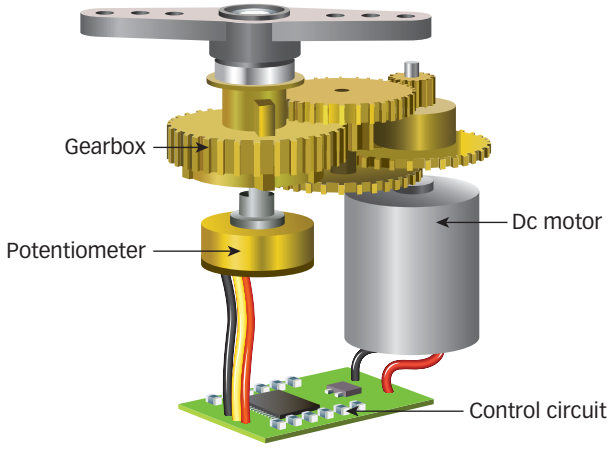


Figure 11.40 A servo motor holds its rotational position by using feedback from a potentiometer.

To achieve this behaviour, a servo motor has a 'closed-loop' control system that uses feedback to control its final position.

One feedback technique uses a potentiometer (a variable resistor) attached to the output axle of a regular DC motor. As the axle turns it changes the voltage delivered to a control circuit by turning this potentiometer. This voltage will be proportional to the rotation of the large gear and can be measured against the voltage originally sent to the servo, thus causing the motor to rotate until the two are the same. This explains why hobby servos sometimes 'tremble' or 'stutter' if they do not have sufficient voltage and why their motion is typically limited to 180 degrees (some servos are capable of 360-degree rotation).

You will be using servos soon in a Skill builder and in the Project at the end of this chapter.

Stepper motors

Stepper motors also provide precise movement to a specified location but without the need for a feedback loop. The rotational movement can be controlled to an accurate number of degrees. Stepper motors are useful in automated systems when very precise movements are required (as in 3D printing).

The motor's shaft can be commanded to move and hold any one of its many step positions without requiring feedback as does the servo. The operation is simple when seen, but difficult to explain in words. The best way to understand how it does this is by viewing an animation.

Stepper motors are used in 3D printers and other devices that need to move in a precise and controlled manner. Servo motors are used in devices that need to move smoothly. See Table 11.3 and Figure 11.41.

Table 11.3 An overview of common actuators.

Type of actuator	Function
Solenoids	Used to move an arm through a small movement
Electric motors	Provide smooth, continuous rotational motion in a clockwise and anticlockwise direction
Servos	Provide continuous angular movement to a specified location, often from 0 to 180 degrees using closed loop control and high torque at low speeds. Lower power needed
Stepper motors	Provide precise stepped angular movement without feedback control and high torque at high speeds. Higher power needed
Relays	Act as a switch to control mechanical movement
Pumps	Used to force liquid or gas into piston chambers to cause mechanical movement

Shutterstock.com/monkey business images

Figure 11.41 Stepper motors control the precise movements required by 3D printers.

Solenoids

A **solenoid** consists of a magnetised iron rod surrounded by a tightly bound coil of wire. When an electric current passes through the wire, a strong magnetic force is produced that is strong enough to move the rod. The principle is the same as that used in a motor. A push movement may be used to connect the starter motor of a car. A pull movement may be used to open the valve of an automatic watering system.

Relays

Relays are sometimes called 'switches'. They consist of a solenoid connected to a mechanical switch. Relays use a small electric current to turn on a large electric current. For example, a 5-volt signal from a computer could be used to turn on a 240-volt power supply to an electric motor. Relays often serve as a safety mechanism for systems using large currents.

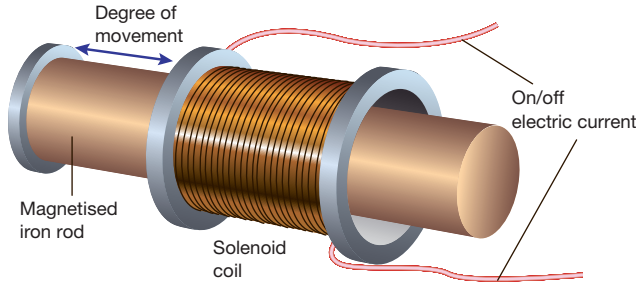


Figure 11.42 A solenoid uses magnetic currents to move a rod.

Pumps

Pumps can be hydraulic (oil) or pneumatic (air). The forces created by a pump make the pistons move, causing a mechanical action (Figure 11.43). Pumps are often used in robotic arms and can achieve rapid and precise movements.

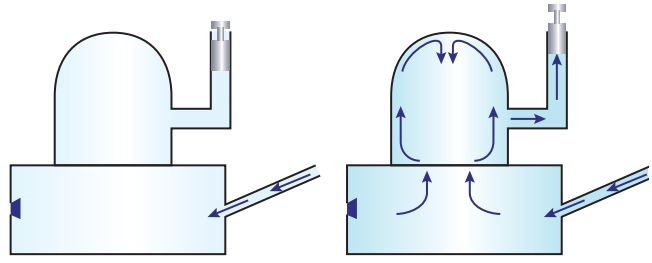


Figure 11.43 A pump pushes gas into a chamber until it is so full that it forces pistons to move.

KNOWLEDGE BUILDER

Understanding pulse wave modulation

Microcontroller boards use **pulse wave modulation (PWM)** to control servos. They need to do this because they produce only digital (on or off) voltage pulses rather than continuous analogue voltage. This works well for regular DC motors, which keep turning while voltage is switched rapidly on and off and also works well with LEDs, which blink too fast to see the flicker, but less well with audio devices. The audio they produce is like the tinny sound of old computer games. The pins of a microcontroller board (such as a micro:bit) only deliver either 3.3V or 0V so a solution is to convert digital to analogue voltage – but how?

How servos use PWM

The solution is a clever one: by switching full voltage on and off very quickly. This technique is called PWM. Hobby servos respond to PWM signals. The length of the signal pulse during one cycle controls the output voltage. Typically, a pulse of 1 ms rotates a servo axle to one extreme, 2 ms to the other extreme, and 1.5 ms sends it to the middle.

In our projects, a pulse is sent every 20 ms and its width is only 1–2 ms. The width of pulse is called its *duty cycle* (expressed as a percentage). This determines the rotational position of the servo shaft as shown in Figure 11.44.

Computational thinking

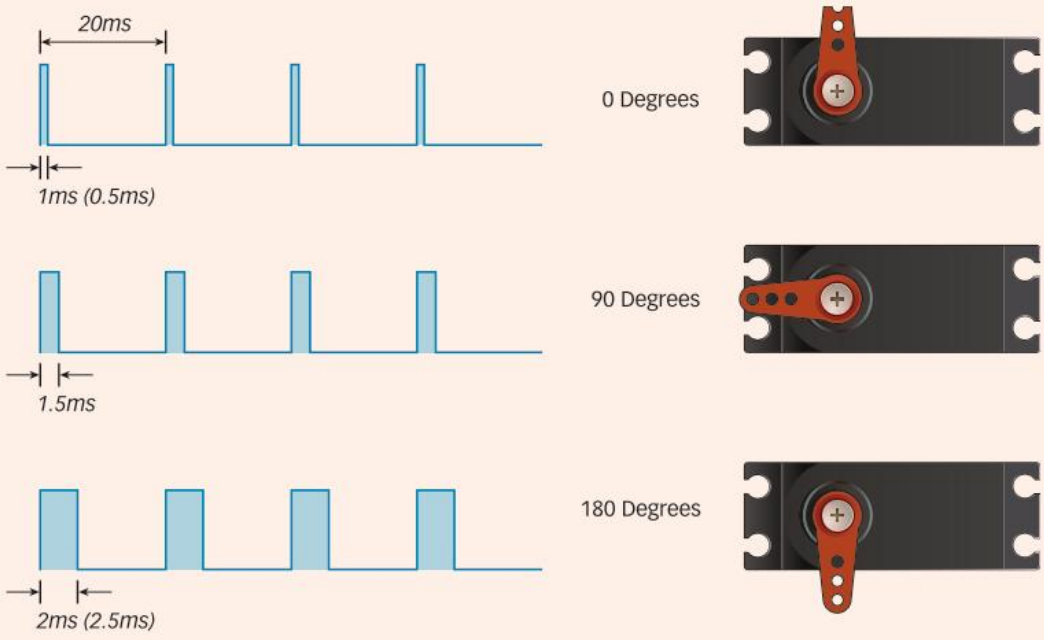


Figure 11.44 Explaining pulse wave modulation (PWM).

CASE STUDY

3D printers and the girl who asked for a frog

Some of the greatest technological mechatronic advances (Figure 11.45) began as toys. In 1988, American Scott Crump's daughter asked him for a toy frog. At first he tried making a frog shape using a glue gun loaded with polyethylene and candle wax by building it up layer by layer. His kitchen ended up with lots of plastic burned onto saucepans. His wife convinced him to move to the garage, where he spent many weekends trying to perfect his methods.

Getty Images/South China Morning Post

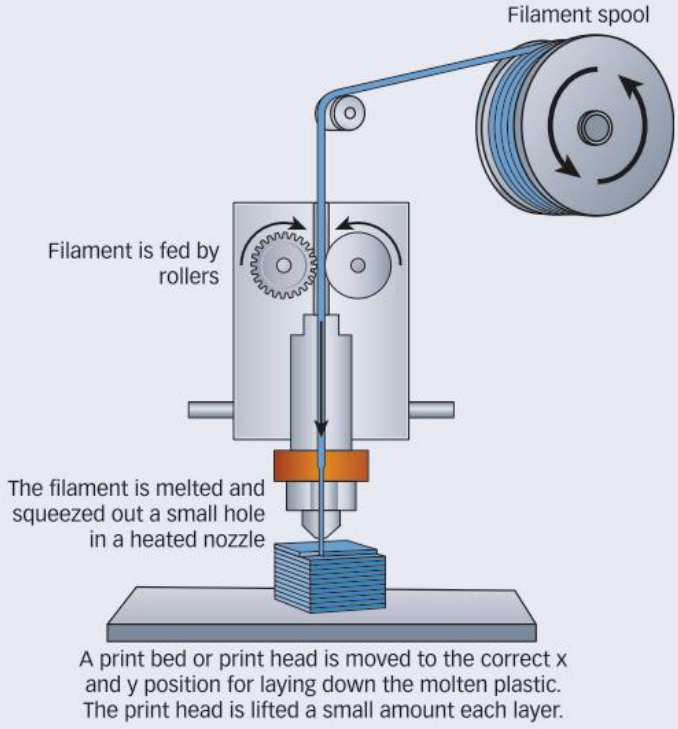


Figure 11.45 (a) Inventor, Scott Crump and (b) an illustration of his FDM 3D printer.

Design thinking

He saw his process could be automated by squeezing molten plastic through a tiny nozzle forming objects one layer at a time. In 1989, he patented this idea and started the (now) billion dollar 3D printer company Stratasys. He called his process Fused Deposition Modelling (FDM). FDM melts thermoplastic filament and layers it to create an object directly from files modelled in 3D computer applications.

Scott Crump's success was the result of continuing on when he wasn't certain he would succeed. His belief in his vision provided him with confidence to press on, even though he had only sold five systems over the first three years. He was willing to mortgage his house. He reached a point where he was willing to ask others to help him create a commercial version of his printer and hired a team of plastic, mechanical, computing and electronic engineers.

Ideas that seem silly or trivial can change the world. Steve Jobs, co-founder of Apple computers once said: 'Here's to … the round pegs in the square holes … the ones who see things differently…They change things.' Now the world has 3D printers and it all started with a girl who asked her dad for a frog!

ACTIVITIES

Identify

1 Identify three common actuators.

2 What is the inspiration for most actuators?

Analyse

3 Outline the difference between a servo and a DC motor.

4 Briefly outline the difference between the internal mechanisms of a servo and a stepper motor.

5 Briefly describe how PWM uses digital pulses to drive analogue devices.

Extend

6 The majority of desktop 3D printers use plastic filament and FDM technology; however, a second method uses epoxy resin and stereolithography. Research and describe this second process.

SKILL BUILDER

A transmitting mini weather station

In this Skill builder, you create a mini weather station to transmit data and graph wind speed and temperature over time. The micro:bit has a great easy-to-use radio communication feature. You will code using both visual code and Python.

How it works

An interesting feature of DC motors is that if spun in reverse (such as by wind on attached fan blades) it produces a small current. Although the micro:bit does not have a wind speed tester by connecting the output wires of a DC motor to the GND pin and an analogue read pin, you can measure the voltage produced. The micro:bit can then transmit this data to another micro:bit which can capture it. The output file can be opened in a spreadsheet and graphed.

Materials required

- micro:bit board (including battery holder, batteries, USB cable)
- 6V DC hobby motor (as illustrated in Figure 11.38. Make sure it is *not* a long base geared version)
- double-ended crocodile leads (x 2)
- (optional) plastic half-egg shapes (as in Kinder Surprise chocolate holders) (x 4)
- (optional) fan blades to fit DC motor (see Nelson MindTap for 3D print files)
- craft materials: dowel rod or cheap craft paintbrush wooden handle, A4 plastic document holder (the kind that opens 2 sides only), small soft rubber grommets or rubber bands, paddle-pop sticks (x 2).

Building the device

Build either or both a bladed fan (Figure 11.46) or the more versatile anemometer (Figure 11.47) to measure wind strength.

Option 1: Building a fan

1 Construct a fan using a standard A4 plastic document wallet (see Figure 11.46). Tear along its welded seam, open flat and form a square. Make 4 diagonal cuts stopping 30 mm from the centre.

2 Punch a hole in the corner of each blade and one hole in the centre. Push a rubber grommet or wrap an elastic band onto a short piece of dowel. Fold the holes in the tips of each blade towards the centre and push the dowel through all the holes, securing them with a second grommet or band.

3 Drill a 2 mm hole in one end and fix the axle of the DC motor into this hole.

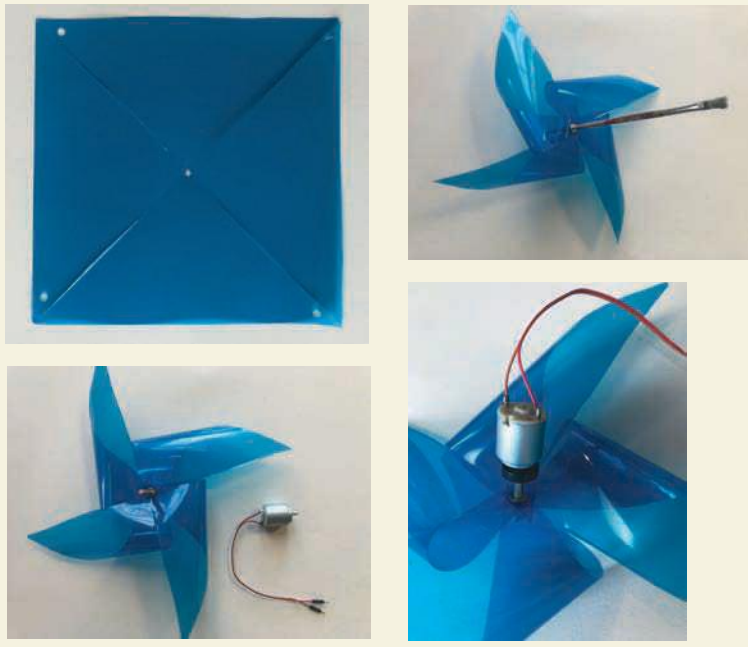


Figure 11.46 Constructing fan blades and attaching DC motor.

Skill builder resource
A transmitting mini weather station

Option 2: Building an anemometer

Wikimedia Commons/Public Domain

Figure 11.47 An anemometer is used for measuring wind speed and is common in meteorological weather stations. The word comes from the Greek word for wind, *anemos*. It has cups which catch wind in any direction.

Insert 4 egg-shaped plastic shapes (as used for Kinder Surprise chocolates) to the ends of 4 paddle-pop sticks, using tape or elastic bands (see Figure 11.48).

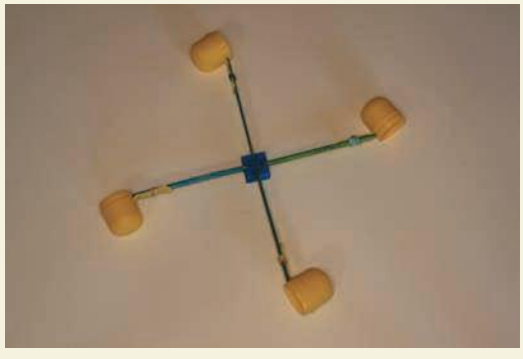


Figure 11.48 Create an anemometer (Figure 11.47) from paddle-pop sticks attached to 4 x Kinder Surprise containers. A 3D printing file is in resources for printing a plastic hub. Other solutions are possible.

Weblink
MakeCode's Block code editor

To make your anemometer, either design a holder and 3D print an object to hold paddle-pop sticks OR 3D print one using the file supplied on Nelson MindTap: *anemometer_hub* (see Figure 11.49). Ensure the DC motor spindle fits tightly.

Design thinking

Visual coding

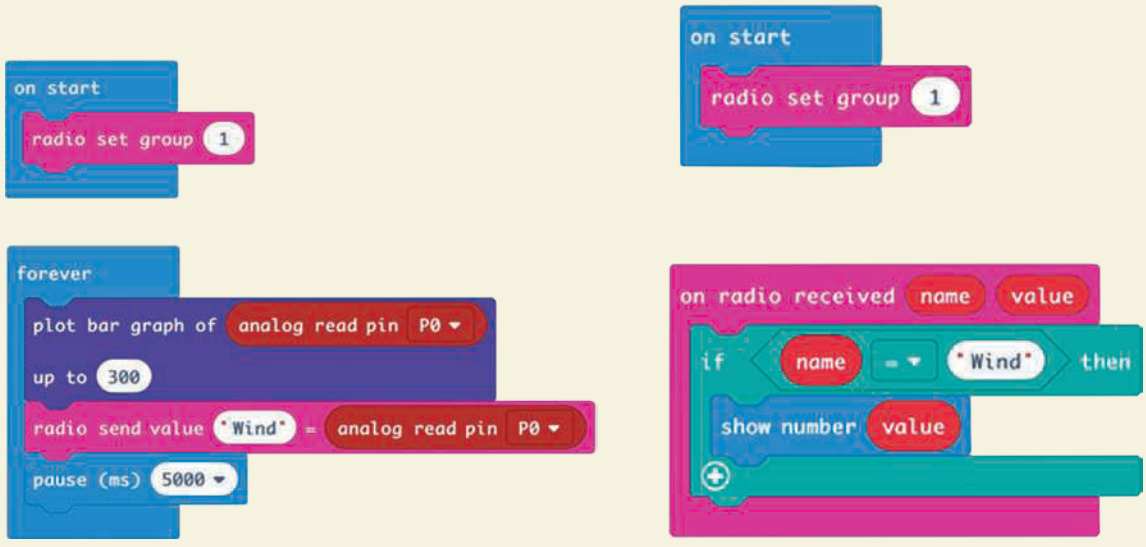
1. Using MakeCode's Block code editor, first code, then download the two programs in Figure 11.50 to two micro:bits via USB. One will be the sender and the other the receiver of the data.

Build and connect the device

2. The sender micro:bit will have a fan or anemometer attached. Use crocodile clips to join one terminal of the DC motor to pin 0 (P0) of the micro:bit and the other to ground (GND). Your readings will be incorrect if these connections are not made.
3. Manually spin the fan. Do you see the bar graph 'grow' on the *sender* micro:bit? If not, try spinning in the other direction. Do you see readings changing on the receiver micro:bit?

   If there is sufficient wind, test it outside.
4. Add temperature transmission to your code.

Figure 11.49 This 3D print file is available in the Nelson MindTap resources: *anemometer_hub*.



Send

Receive

Figure 11.50 The sender and receiver programs.

Coding in Python

We now use general-purpose code rather than visual code to send and receive data using the micro:bit.

Code for sending

First, we code the sending micro:bit, with motor and fan attached. Recall that the motor is connected to pins 0 and GND on the micro:bit and that each student group must use a different numbered channel 1-255.

Use Microsoft's MakeCode Python editor to prepare this code and flash it on the *sender* micro:bit to test data collection.

```
from microbit import *
import radio
# radio must be turned on
radio.on()
# different channels each group
radio.config(channel=1)
# set signal to full strength
radio.config(power=7)
while True:
    value = pin0.read_analog()
# send only strings
    radio.send(str(value))
# send data at regular intervals
    sleep(100)
    display.scroll(str(value))
```

Watch readings change on the LEDs as you catch the breeze or spin it manually (both ways).

Code for receiving

Next we code the receiving micro:bit.

5 Remove the 'display' statement from the *sender* board as it slows down data transmission and flash this updated version to your board.

Load the following code on the second *receiver* micro:bit.

```
from microbit import *
import radio
# radio must be on
radio.on()
# channel same as sender
radio.config(channel=1)
radio.config(power=7)
while True:
    incoming = radio.receive()
    display.scroll(str(incoming))
    if incoming is not None:
# must include this
        print(int(incoming))
```

Power up both sender and receiver boards and even without a wind sensor connected, you will see it working.

6 Connect the DC motor to blades and attach it to the *sender* board. Make sure the *receiver* micro:bit is plugged in to your computer. Spin the motor by hand, or use a wind source to spin the fan blades. Click the Show serial button (bottom right) and watch the data values being received (Figure 11.51).

7 Modify the code to include transmission of temperature readings.

Logging the data onto the micro:bit

8 If you have a version 2 micro:bit or later, you can capture a log of the received data in a file stored on the micro:bit itself.

Weblink
micro:bit
Python editor

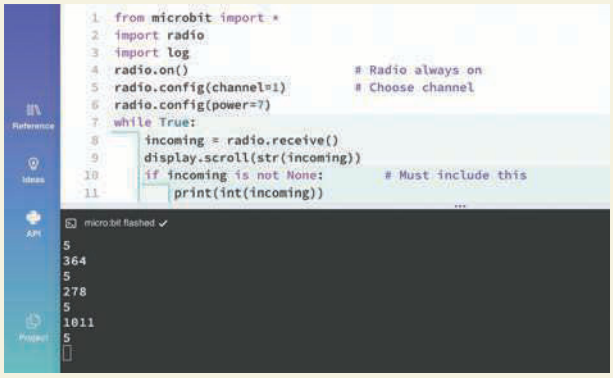


Figure 11.51 Data being received by the second micro:bit appears in the serial window.

9 Add the two lines shown in red in the following to your code. Make sure you type brackets and parentheses accurately! Note: you must start off your code with the usual import statement and add another: `import log`

```
# NOTE: channel must be
# same as sender
radio.config(channel=1)
radio.config(power=7)
while True:
    incoming = radio.receive()
    display.scroll(str(incoming))
    if incoming is not None:
            print(int(incoming))
            log.add({'wind':int(incoming)})
```

10 Detach the *receiver* micro:bit from your computer and attach a battery. Collect some data by spinning the motor. The data will be saved automatically in a file called MY_DATA.HTM on the micro:bit. Reconnect your micro:bit by USB cable. Open this file on the micro:bit and a webpage will offer you various choices (as shown in Figure 11.52).

This is a data log. Select 'Visual preview' to see a graph of the results.

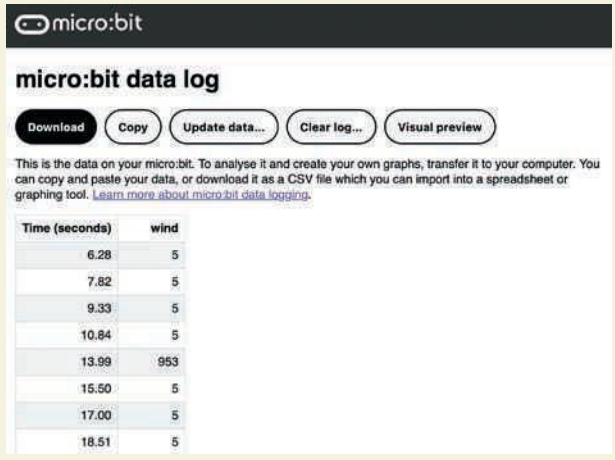


Figure 11.52 Data logging to a v2 micro:bit.

11 Select *Download* to create a .csv file of the data. This format can be opened by any spreadsheet. Open this .csv file in Excel or Google Sheets. Use the spreadsheet tools to create a chart to represent the wind data. Use a scatter graph in Excel and a line graph in Google Sheets.

Part of a graph of actual data created in a spreadsheet is illustrated in Figure 11.53.

Figure 11.53 Actual wind data graphed over time.

CASE STUDY

Crowdsourcing 3D printed prosthetic limbs

AAP Image/EPA/BALAZS MOHAI

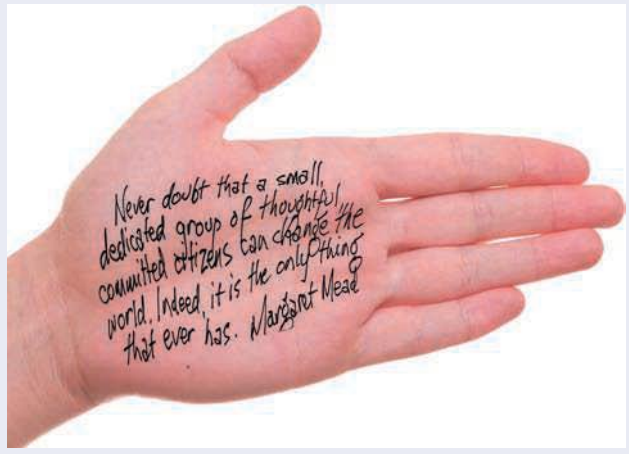
Figure 11.54 e-NABLE uses crowdsourced 3D printers to build prosthetic limbs.

One of the earliest uses of mechatronics to produce a prosthetic is told by the founder of the organisation e-NABLE.

In 2011, Ivan Owen created a crazy metal puppet hand to wear to a steampunk convention. He posted a short video on YouTube. This resulted in an email from a carpenter in South Africa, who had lost his fingers in a woodworking accident. For a year they worked online together on assembling bits and pieces to create a device. A year later, the mother of a 5-year-old, Liam, who had been born without fingers on one hand, asked for the device. Ivan realised his design would need to change size many times as Liam got bigger.

In trying to solve this problem, Ivan came across the story of a Corporal Coles, whose prosthetic hand was created in the early 1800s by an Australian dentist, Dr Robert Norman. This design used whalebone, cables and pulleys. Ivan realised he could create a similar device using the mechatronics of a 3D printer! He taught himself 3D design software and, with two donated 3D printers, made the world's first 3D printed hand. Watch the video using the weblink supplied.

Ivan decided to publish the files as open-source and public domain, so that not only could Liam have a hand, but anyone in the world could download and print his devices. People owning 3D printers could now log in, choose their city, select a file, print part of the hand, then post it. After two years, he reached 7 000 members and 2 000 devices in 45 countries. Consider the quote in Figure 11.55.



Pixabay/Niek Verlaan

Figure 11. 55 Margaret Mead quote.

Weblink
Liam and his hand

ACTIVITIES

Identify

1 What is a solenoid used for?
2 What device is used to measure wind strength?
3 Identify a software application that can be used to collect data when data logging.

Analyse

4 Describe one application of data logging.
5 Describe how a solenoid might be made at home.

Extend

6 The MU editor application (see weblink) is a Python editor for micro:bit like the one you have been using.

It includes a REPL tool which can be used with the micro:bit. Research what a REPL is and describe what it is used for.

Identify the equivalent REPL tool in the micro:bit Python editor.

Try downloading and using the MU editor with the last code activity you completed. Which editor do you prefer?

Weblink
MU editor

11.8 Microprocessors

The microprocessor is the heart of a mechatronic system. Its circuits are controlled by a chip which receives input from sensors and sends appropriate signals to actuators. Many of the activities in this chapter use a micro:bit as the microcontroller.

Often the controller of an automatic control system is a computer. Like other systems, it is set up to accept data from input devices, process it and send the data to output devices.

Analogue and digital output pins

Many microcontrollers, including the micro:bit, have connectors that act as either input or output.

On a micro:bit, five large 'pins' (really tabs that connect to 'pins' on the CPU) are labelled 0, 1, 2, 3V and GND. Pins 0, 1 and 2 are known as GPIO, or general-purpose input/output pins, and operate in four different modes

Systems thinking

depending on how they are coded in the program. These modes are:

- *Digital output* which delivers either an on ('HIGH') or off ('LOW') (3V or 0V).
- *Digital input* which detects an on or off signal such as from a switch connected to 3V.
- *Analogue output* which uses PWM, a way of using digital pulses to deliver a variable voltage. (PWM was explained in detail in an earlier Knowledge builder.)
- *Analogue input* which is able to measure changing voltages connected to it.

11.9 Control systems

There are two main types of control systems: open loops and closed loop systems (Figures 11.56 and 11.57). An open-loop control system does not use feedback. An example of an open-loop system is a griller, where you choose the temperature and settings to initiate cooking. While the food is being cooked, there is no check made by the system on the state of the food. In the manufacturing industry, open-loop automation often involves devices that pick up and place objects.

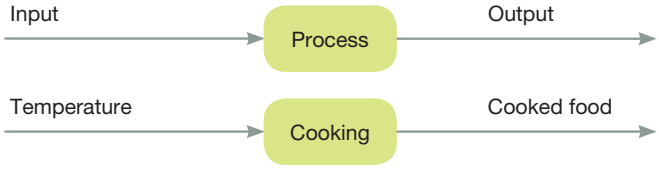


Figure 11.56 An example of how an open-loop system, such as a griller, works.

In contrast, a closed-loop system uses feedback and the output of the system is connected in some way to the input. The data from the output is used to adjust the input. An example is a temperature control system, or thermostat, used to control an air-conditioning unit in a building or to control oven temperature.

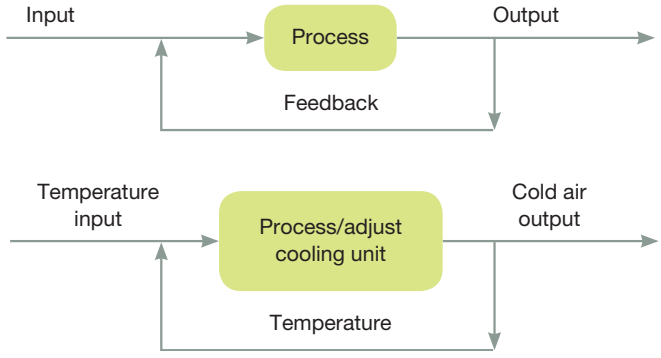


Figure 11.57 An example of how a closed-loop system, such as an oven with a thermostat, works.

In the previous section, you studied two types of motors. A DC motor is an example of an open-loop system. Its input current is not affected by its output rotation. Servos and stepper motors are examples of closed-loop systems. In a servo, the output controls its input as its output shaft is linked through a potentiometer.

SKILL BUILDER

On the level: A control system with feedback

In this Skill builder, using only a micro:bit and a servo, build a control system to keep an object perfectly level even when the hand holding it is tilted. It is surprising how simple this task is! You will first use visual code and then complete the Skill builder by coding using Python.

You may have seen such *gimbals* in action (look up the interesting origins of this word). A gimbal uses motors and intelligent sensors to stabilise cameras and other objects. Check out online videos by looking for 'great gimbal shots' and be amazed!

The micro:bit has an onboard accelerometer sensor. This is how a mobile phone knows when you turn it from portrait to landscape orientation. As we can measure the roll of a micro:bit in degrees (see previous Skill builder), we can use this data as feedback and rotate a servo by the same amount but in the opposite direction. Any object attached to this axle will therefore stay level (Figure 11.58).

Shutterstock.com/Natalie magic

iStock.com/windjunkie

Figure 11.58 (a) Gimbals keep cameras pointing in the same orientation while the operator is free to move. (b) A drone mounted with a camera gimbal underneath.

Coding visually

1 Inspect Figure 11.59 and interpret it. Illustrate it by drawing a very simple flowchart. Confirm your interpretation using MakeCode's simulator.

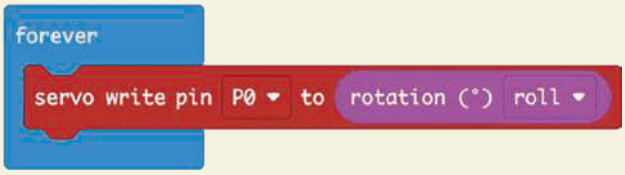


Figure 11.59 MakeCode 'rotation roll' instruction.

2 Build the code shown in Figure 11.60 using MakeCode. You will find the servo write block under Advanced in the Pins library. To locate any block in MakeCode you can use the search function at the top of the library panel. First define a variable called 'roll'.

3 Test this using MakeCode's onscreen simulator. Then test this code by downloading to a physical micro:bit. The rotation of the servo might be in the same direction as the roll of the board. If so, reverse this by changing the + to a – in your code. You need your servo rotation to be in the *opposite* direction to the roll of the board.

4 If we set the servo axle to its halfway point of 90 degrees then add (or subtract) the amount of roll (depending on orientation) as the micro:bit rotates, the object level will remain level. Then we access the tilt value by adding *on button A pressed* code as shown in Figure 11.60. Modify the code as shown, view its JavaScript equivalent code using the JavaScript tab and test it using MakeCode's onscreen simulator by tilting the on-screen micro:bit and watching the servo spindle move to compensate for the tilt (see Figure 11.60).

Building the gimbal

Materials required

- micro:bit standard kit (micro:bit, USB lead, battery pack)
- small hobby servo (e.g. SG90)
- crocodile leads (x 3)
- male to male pin leads (x 3)
- paper cup, standard ruler, paddle-pop stick, elastic bands

Skill builder resource
On the level: A control system with feedback

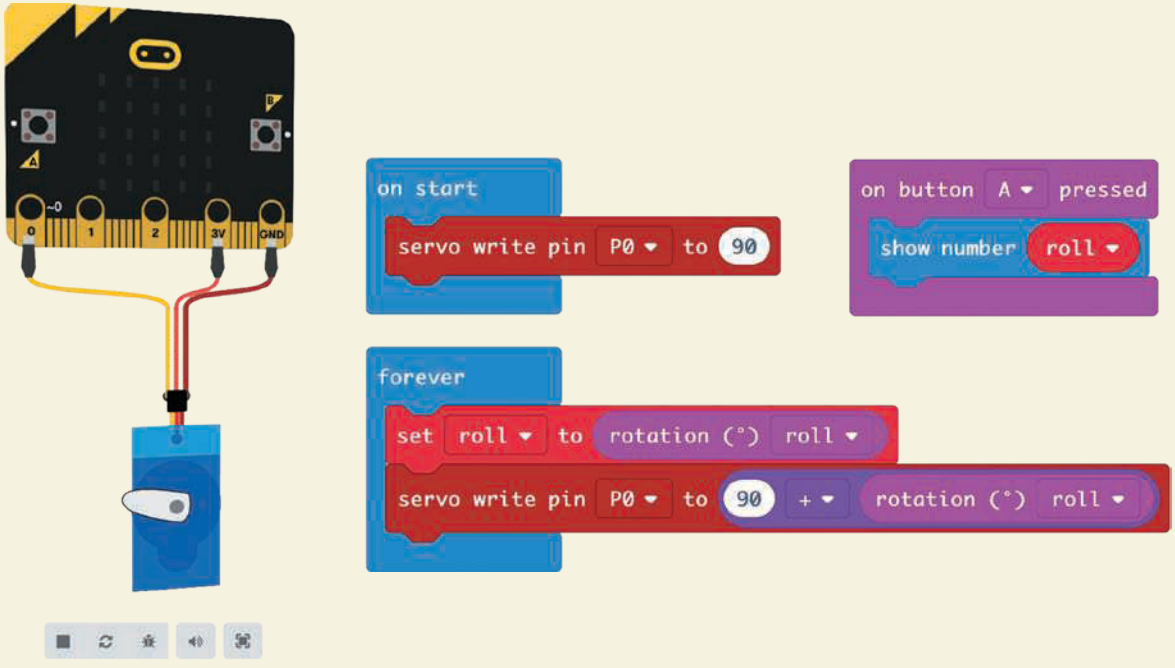


Figure 11.60 Simulation of gimbal using micro:bit and visual code.

5 Attach the servo's yellow control wire to pin 0 (P0) on the micro:bit, connect the red wire to its 3V pin and the brown wire to the GND pin as shown in Figure 11.60.

6 Attach the double ended 'horn' (its correct name) to the servo spindle. Using the small screws that came with the servo, attach the wooden paddle-pop stick to the white plastic servo horn. Assemble and connect items as shown in Figure 11.61.

7 Load the program onto the micro:bit. Leave both the micro:bit battery pack and USB connected to supply sufficient voltage and hold the ruler horizontally. Adjust the servo horn and paddle-pop stick as required. Press button A and test the device works.

8 Tilt the ruler back and forward and watch the stick maintain its position! If the axle rotation turns in the opposite direction to what you need, what would you change? Use your prototype to keep a paper cup upright.

Coding in Python

Now we code in Python. In the following, we introduce each important feature for your code, and leave it to you to get your code working. (There is a solution provided in the Nelson MindTap resources: *gimbal_python.py*.)

First, open the micro:bit Python editor to edit the placeholder program.

Weblink micro:bit Python editor

9 As there is no simple equivalent to the rotation block in Python, you will need to use the following simple function to generate the servo 'duty cycle'. The earlier Knowledge builder on PWM explains how duty cycles work.

```
# PWM duty cycle for given angle
def set_servo_angle(angle):
    duty = 40 + (angle * 4) / 9
    pin0.write_analog(duty)
```

10 You will need the MicroPython function

```
accelerometer.get_x()
```

to obtain a measure of the acceleration about the x axis when the board is tilted and convert this to an angle. This function delivers values in the range -1023 to +1023 corresponding to angles 0 to 180 degrees. Thus each degree corresponds to 2026 ÷ 180 or 11.4 degrees.

Explain why the corresponding angle in degrees will be:

```
(reading + 1023)/11.4
```

The servo must rotate in the opposite direction to create a gimbal so we need (this may need to be modified for the orientation of your servo):

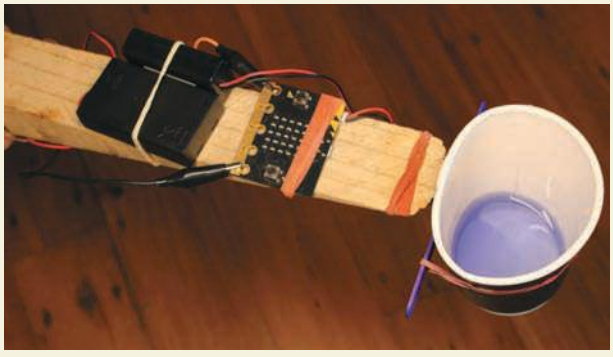
```
set_servo_angle(180 - (reading + 1023)/11.4)
```

11 Download your code to your prototype. Test, then add code so button A shows tilt in whole degrees.

Your gimbal's design is a rough prototype. Sketch an improved design that could be 3D printed or laser cut and attached to the servo horn, micro:bit to attach a smartphone/camera for steady video filming.

Computational thinking

Design thinking



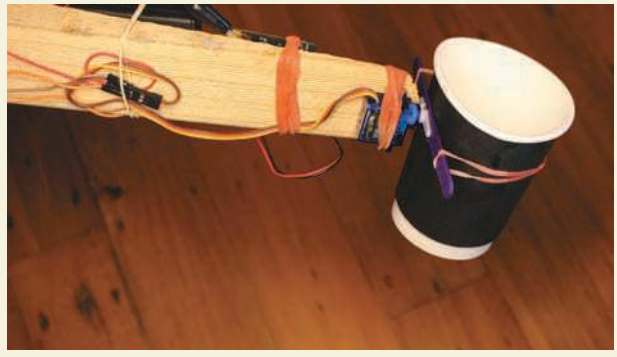


Figure 11.61 Setup for micro:bit-powered gimbal.

ACTIVITIES

Identify

1 What type of mechatronic device is the micro:bit?

2 Name two microprocessors.

3 What is a gimbal and what is it used for?

4 Draw diagrams to represent an open-loop and a closed-loop system.

Analyse

5 Compare the behaviours of digital and analogue pins on a microprocessor.

6 Examine the Javascript in your code for the gimbal Skill builder (use the tab at top) and identify in it each coloured block in your code.

7 Describe one application of a closed loop system identifying sensors and actuators.

Investigate

8 Research and describe vehicle cruise control as an example of a closed loop system.

11.10 Automated control systems

In our everyday lives, we use devices that are operated by manual control; that is, they need to be operated by humans. Examples include cars, sewing machines, vacuum cleaners and electric drills. Automated systems, on the other hand, are designed to operate under automatic control. 'Automatic control' means they will operate independently or without human intervention. These devices are given appropriate instructions through special programs. Robots can be classified as computer systems that use automated control.

The purpose of an automated control system is to maintain a desired result. It achieves this result by manipulating one or more variables of another system (Figure 11.62 and Table 11.4). A good example of this is an air-conditioning unit that uses a thermostat to maintain a constant programmed air temperature. This involves:

- input – the air-conditioned reads the room temperature using a temperature sensor
- process – this reading is compared to the set temperature entered by the user
- output – signals are sent to turn the thermostat on to adjust the temperature.

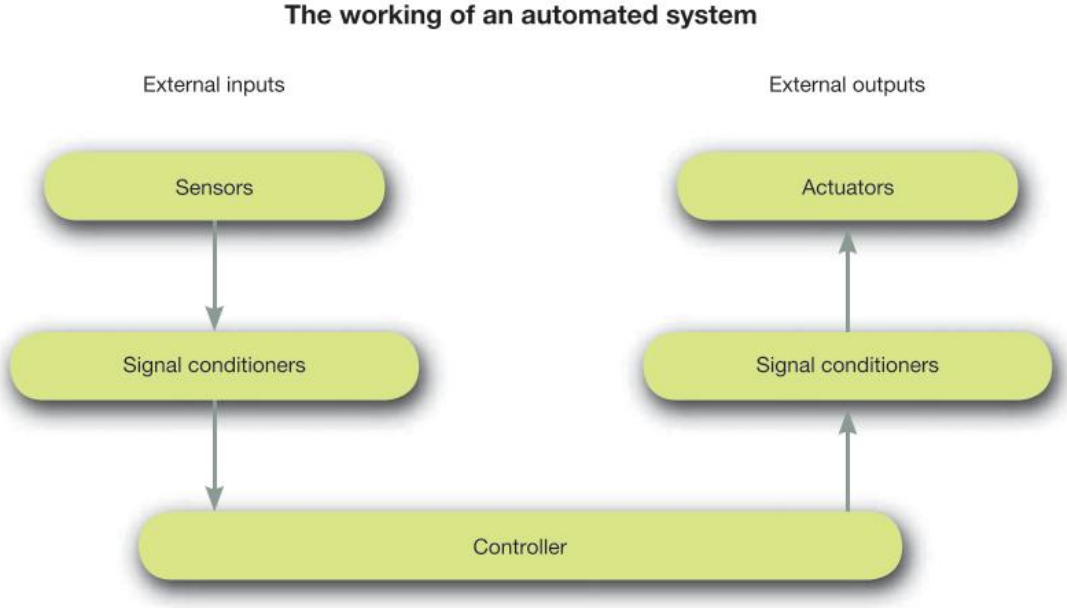


Figure 11.62 A diagrammatic representation of an automated control system.

Table 11.4 The components of an automated control system.

Components	Functions
Sensors	Input devices that detect conditions in the environment then send the results as signals to the controller
Signal conditioners	Devices that help keep the signals passed to and from the controller 'clean'; i.e. reduce the amount of interference in the signals
Controllers	Devices that process the input signals from the sensors to produce the output signals required to drive the actuators
Actuators	Output devices that perform the physical actions that have an effect on the environment

Types of automated control

In the manufacturing industry, a common method of categorising automated control systems is according to the type of operation they perform and the type of products they produce. The three main types of automated control systems are continuous, batch and discrete. The features of each are outlined in Table 11.5.

Table 11.5 Features and examples of different types of control systems.

Control systems	Features
Continuous	Designed and built to perform one manufacturing task, 24 hours a day, 7 days a week, with some time required for system maintenance. Not flexible; for example, a new model of car may require a new assembly plant to be built.
Batch	Produces large quantities of a particular product, but can be quickly altered to produce new or alternative products. The control system is capable of being quickly reprogrammed.
Discrete	Entire system is involved in producing a single item. Produces items only in small quantities. Least suited to large-scale production. Expensive.

Household control

Computer control systems are already a common feature in many households. These include automated:

- security systems
- climate control systems
- integrated home automation systems
- vacuuming
- sprinkler systems
- audio-visual sound systems.

The second project at the end of this chapter offers you the opportunity to create your own automated system.

SKILL BUILDER

Skill builder resource
An accessible automatic door

An accessible automatic door

Required

- the device you created in *Skill builder: Using an ultrasonic distance sensor*
- servo
- breadboard
- crocodile to male pin wires
- male to male pin wires

In this Skill builder you create a prototype of an automatic door opener and closer which a person with a disability could operate.

When a person or object gets closer than 10 centimetres from the sensor, the door will open, otherwise it closes.

To do this activity you will need to have first completed *Skill builder: Using an ultrasonic distance sensor* in section 11.6. We now add a servo to the previous project in order to open and close a door.

1 Consider this: How could an ultrasonic (US) sensor be used to assist opening and closing a door for someone in a wheelchair?

2 Open your previous code from *Skill builder: Using an ultrasonic distance sensor* and download it to a micro:bit.

Connect the US sensor as before by recreating the wiring shown in that activity. Test the device to make sure it is working correctly.

3 Add a servo to the unused pin 2 and separately power both it and the US sensor using a breadboard by wiring as shown in Figure 11.63. The US sensor can also be plugged straight into the breadboard if you wish to save on some of the wiring.

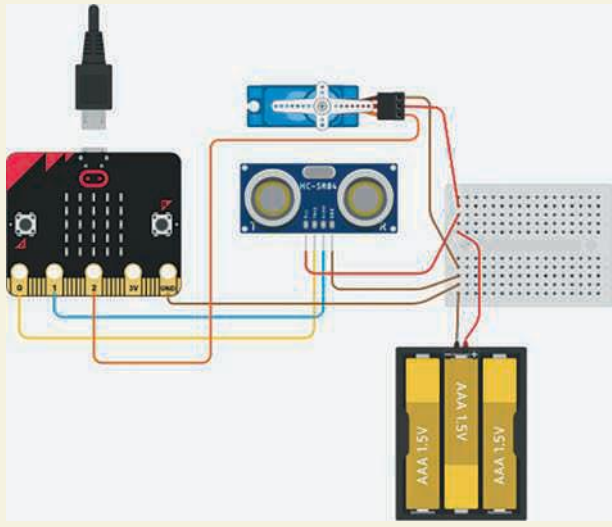


Figure 11.63 Wiring for the automatic door using a servo and US sensor.

4 As in a previous Skill builder, the following Python function will move a servo a certain number of degrees.

```
# moves servo through angle 0 - 180
def set_servo_angle(angle):
    duty = 40 + (angle * 4) / 9
    pin0.write_analog(duty)
```

Add this function to your existing code before the `while True` loop.

5 You can now turn the servo 180 degrees simply by calling this function using: `set_servo_angle(180)`

Modify your code by adding an `if…then…else` loop to get the servo to move a suitable number of degrees when the ultrasonic sensor reports a distance is less than 10 centimetres and to turn back to 0 degrees when the distance is greater than or equal to 10 centimetres. Test this loosely on a table.

The completed solution is in the Nelson MindTap resources.

Design thinking

6 Build your hinged closing door using a paper clip to connect the servo to a hinged door using a small cardboard box. Attach the ultrasonic sensor to detect when an object is close and open the door. One possible design is shown in Figure 11.64.

Take it further

7 Modify the code so the door stays open for a fixed time. If you have a kit or access to other sensors (such as a sound sensor, light sensor or **PIR sensor**), select one and replace the US sensor with it.

Identify how this might be used to address the needs of a person with accessibility issues, such as operating the door and also operating lights automatically.

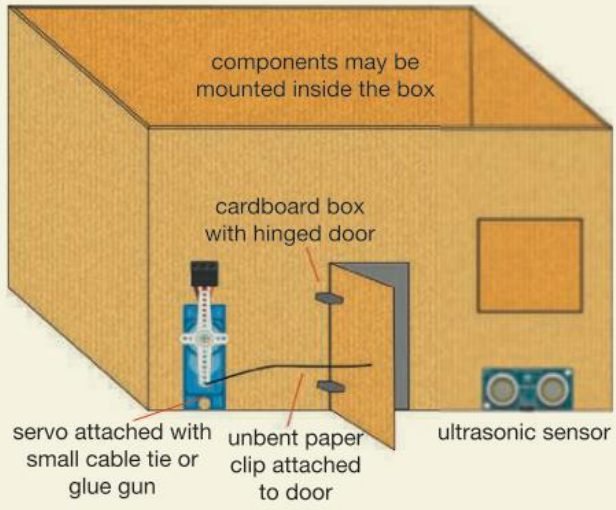


Figure 11.64 A possible design for an automatic door opener.

CASE STUDY

Mars mechatronic 'mole' faces problems

The Insight Mars mission

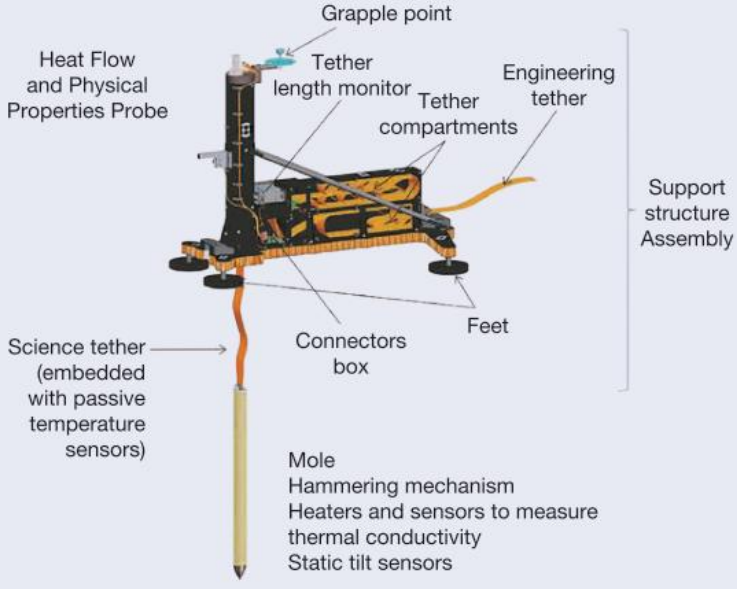
Before reading on, we suggest you download NASA's free AR application *Spacecraft AR* from your device's app store to your mobile device and select the *InSight* lander under *Mars*. You will be able to explore the landing craft in amazing detail (Figure 11.65), including the subject of this case study: its HP3 hammer probe.

Many earlier Mars missions have scraped soil from the surface of the planet and analysed it, sending the data back to Earth. However, before this InSight Mars mission, no soil has been sampled beneath the surface.

A mechatronic probe, a 'self-hammering nail' (Figure 11.66) called the HP3 and referred to as 'the mole', is designed to burrow five metres below the Martian surface using embedded heat sensors to study thermal properties of Mars' interior, revealing unique information about the planet's geological history.

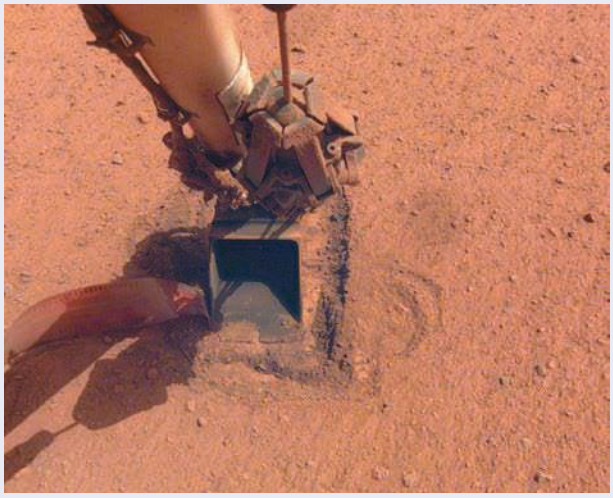
NASA/JPL-Caltech

Figure 11.65 View the Mars InSight landing craft in amazing detail using augmented reality.



NASA/JPL-Caltech/DLR

Figure 11.66 The parts of the HP3 instrument carried to Mars by the InSight lander.



NASA/JPL-Caltech

Figure 11.67 InSight's robotic arm attempting to get the hammer probe digging again by scooping then pressing soil tightly around it.

Other sensors on the device detect echoed vibrations from the hammering, providing data that can be used to determine whether the Mars core is molten or solid.

The data may also reveal if life might have existed on Mars.

Probing problems

After the craft landed, a robotic arm lifted the probe out onto the surface. The principle of operation of the hammer is a cylindrical tube inside which a plunger or hammer is lifted repeatedly then accelerated downwards inside the tube, hitting the tube's pointed end hard, driving it into the soil (see Figure 11.67).

Unfortunately, the probe drilled only 30 centimetres and then refused to go any deeper, tilting at an angle of 15 degrees. Since getting stuck, the mole has hammered 220 times, making slow but steady progress.

Engineers were brought in to problem-solve and over many months they devised a plan.

The plan

Design thinking

They decided they would try to use the robot arm on the InSight lander to first scoop soil around the drill then press against it to create enough friction for it to keep digging. By positioning the edge of the robotic arm on the top rim of the probe's outer shell, they hope to prevent it jumping upwards and instead, this will force it to go deeper and eventually engage properly.

The video on the Space.com website explains what is planned.

Weblink
Space.com

Questions

1. Mechatronic engineers often need to solve problems such as this. Can you suggest a design modification for the probe that could help it engage with the surrounding soil?
2. What additional features for the robotic arm might have helped with this problem?

Chapter review quiz

Weblinks:
- MakeCode's Block code editor
- micro:bit Python editor
- How Slider Crank Mechanism Works
- Gripper based on dielectric minimum energy structures
- How DC Motors Work
- How does a Stepper Motor work?
- Liam and his hand
- MU editor
- Space.com

Skill builder resources
- Building a pitch and movement alarm
- A transmitting mini weather station
- On the level: A control system with feedback
- An accessible automatic door
- MakeCode Python editor

Skill builder
- Automatic door

Bonus projects
- Chatterbox
- Arduino: the moody useless box

Project resources
- The amazing all-knowing wizard

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

11.11 Project 1: An autonomous remote-sensing irrigation system

In this chapter, we provide two mechatronic projects, which build on skills you have acquired in this chapter's Skill builders.

In this first project, you build a remote controlled irrigator.

In the following project, you build and program an all-knowing automaton mechatronic wizard.

In addition, two bonus projects are provided on Nelson MindTap. For those who are ready for an advanced challenge, one online project uses an Arduino microprocessor board to build the famous 'useless box' but with a variation: ours is moody! (See the folder named *Moody Box*.) Those who wish a gentler introduction to mechatronics or who have limited resources can create an animatronic chatterbox made from a Kinder Surprise. (See the folder named *Chatterbox*.)

By completing projects for this topic you will learn a great deal, not only about mechatronics, but also about design thinking, systems thinking, computational thinking and experience coding in four computing languages (Javascript block code, Javascript, Python and C++). These projects will build skills to enable you to create great inventions of your own!

Your task

Build a working automated watering control system using two micro:bit boards communicating over a radio link. This is a great example of mechatronics being used in an automated system. This project may be completed individually, in pairs or in small groups.

The first micro:bit is stationary and will sense soil moisture and control a servo to water the soil. The second micro:bit will remotely monitor and display data from the system via a radio network (radio communication is very easy on a micro:bit!).

We will guide you through Python coding. If you wish to try out the visual code as well, the complete visual code is provided in Figure 11.71.

Identifying and defining

Begin by listing your own set of outcomes against which you can measure the success of your project.

Materials required

- micro:bit standard kit (micro:bit, USB lead, battery pack)
- small hobby servo (e.g. SG90)
- crocodile to crocodile leads (x 3)
- crocodile to pin leads (x 3) or male to male jumper leads (x 3)
- large common nails (x 2)
- large paper straw, paddle-pop stick, paper cup, Blu-Tack®, scissors, elastic bands, tape
- tray of soil or small plant in pot
- (optional) AAA 1.5V cells and holder to supply additional servo voltage (x 3)

Bonus project
Arduino: the moody useless box

Researching and planning

Design thinking

If you only had a paper cup, straws, servo (controlled by the micro:bit), elastic bands and a paddle-pop stick, could you design a working device that would regularly water a tub of soil? Use your design thinking skills to assemble your own design (see Figure 11.68). There are at least five very different designs we know work successfully!

Design thinking

Figure 11.68 Can you build a working device from these items plus a micro:bit board?

Measuring soil moisture

The amount of water and nutrients in soil determine its conductivity. This can be measured by inserting two nails into the soil and supplying them with a voltage. See Figure 11.69. The soil will offer resistance and the micro:bit can be used to measure the current. We can read this measure using pin P0.

Producing and implementing

Moisture testing using general purpose code

1. Connect nails using crocodile clip leads to pins 0 and 3V.
2. Open the Python editor weblink to access your Python editor in a Chrome browser, as Chrome allows pairing of your micro:bit via USB.

Weblink
MakeCode Python editor

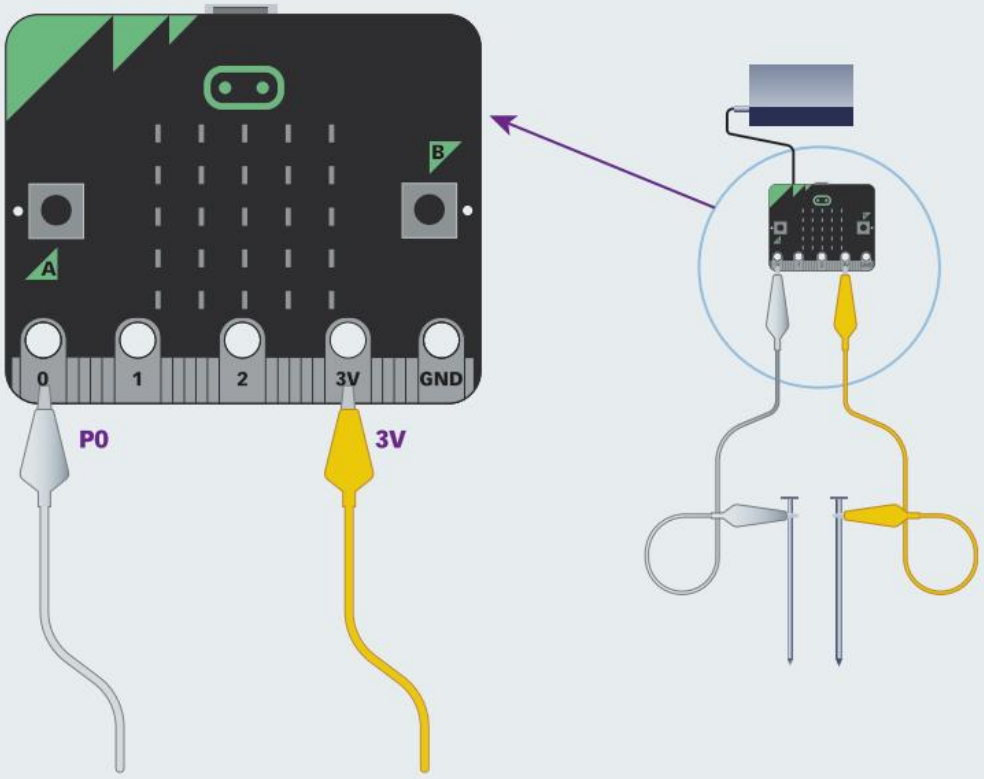


Figure 11.69 Setup for measuring soil moisture. Test soil conductivity using two nails connected by wires with crocodile clips to pins 0 and 3V.

3 Type the following Python version of this program using your Python Editor:

```
from microbit import *
while True:
    number = pin0.read_analog()
    display.scroll(number)
    sleep(1000)
```

4 Can you explain what the code does? Download this code to your micro:bit and confirm it works. If you don't have any soil, test by touching the nails together and holding them apart. Form a circle of people touching hands and then separate two people asking each to hold one of the nails. What is the measurement?

Saving energy using Python

This device would very soon drain the battery because it passes 3V through the soil constantly. Can you think of at least three ways to save energy?

Systems thinking

5 By only turning on a 3V pin when needed, we save a great deal of energy.

Swap the crocodile clip from the 3V pin to pin 1. By using code to set pin 1 to 3V (a value of 1023) it can now become our 3V pin for a very brief time then it can be switched off (a value of 0). This was not possible using micro:bit's dedicated 3V pin as this pin is always on.

6 We now implement this in the following general purpose code:

```
from microbit import *
while True:
    pin1.write_analog(1023)
    sleep(200)
    number = pin0.read_analog()
    sleep(200)
    display.scroll(number)
    pin1.write_analog(0)
```

Modify your earlier code, download and test it.

7 The visual block code in Figure 11.70 shows two other ways energy can be saved: by turning pin 1 on only when needed and by measuring only every 5 seconds.

8 Create Python code for at least one of these energy savers, download it and test the result.

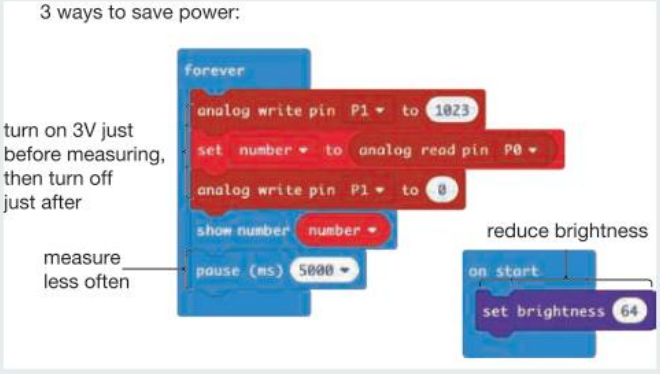


Figure 11.70 Three changes which will save energy shown using visual code.

Responding to dry and wet in Python

9 The following Python code tests if moisture measures less than 800 and responds by displaying an image.

```
from microbit import *
while True:
    pin1.write_analog(1023)
    sleep(200)
    number = pin0.read_analog()
    sleep(200)
    display.scroll(number)
    pin1.write_analog(0)
    if (number < 800):
        display.show(Image.UMBRELLA)
        sleep(500)
    else:
        display.show(Image.YES)
        sleep(500)
    sleep(5000)
```

Explain clearly what this code does before reading on and add internal comments to each part of the code to demonstrate your understanding of it.

The micro:bit now displays an umbrella image if moisture level is less than 800, otherwise it displays a tick. Note that short delays of 200 ms and 500 ms are added. Why do you think these are needed? What is the reason for a delay of 5 seconds at the end?

Add a servo to create a soil irrigator

10 We will add a servo to tip water when the moisture reading is less than a threshold value. Use 'crocodile to pin' cables to connect the servo to the micro:bit as shown in Figure 11.72.

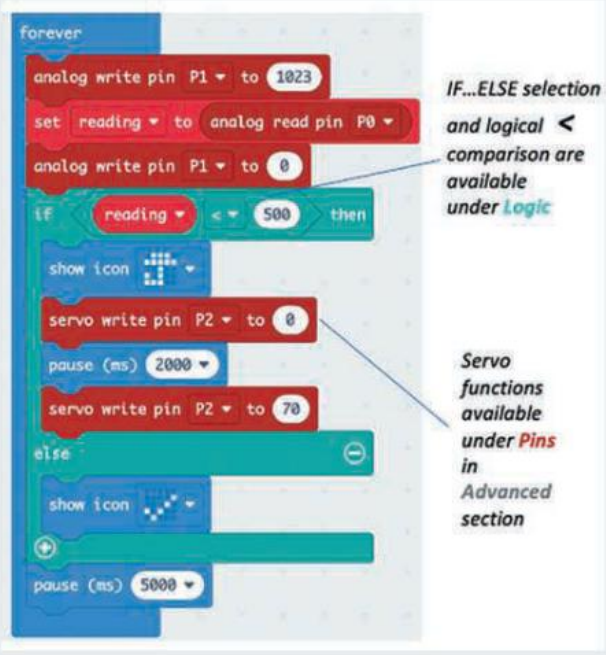


Figure 11.71 The micro:bit now gives visual feedback for both wet and dry soil. V1 and V2 micro:bits behave differently. You may need to increase the value of 500 here to 800.

Make sure:

a brown servo wire is connected to GND (ground) pin
b red servo wire is connected to 3V pin
c orange servo wire is connected to pin 2

Note you are still testing a loose tabletop version and are not yet using your actual irrigator design.

If you leave your micro:bit USB cable plugged in to the micro:bit as well as adding its battery pack you should have sufficient power. However servos are operating at the minimum of their voltage requirement here and if the batteries are a little flat the servo may not work. The solution is to power it separately as shown in the accompanying Knowledge builder.

Use the visual code version of the irrigator provided in Figure 11.71 to perform a 'tabletop test' as preparation for building your irrigator. Ensure connections to servo match the servo wire colours shown in Figure 11.72, and that the correct colours are connected to the correct micro:bit pins. Test your code. The servo should rotate 70 degrees every 5 seconds when the leads from pins 0 and 1 are not touching and stop when they are touching.

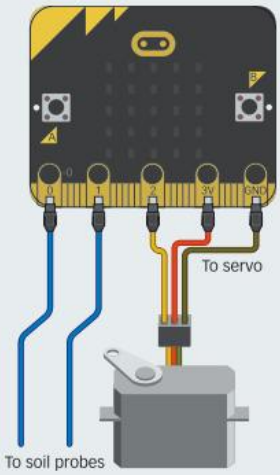


Figure 11.72 A micro:bit with leads correctly connected to the servo.

Controlling the servo using general purpose code

11 Use your Python development environment for this step.

Modify the Python code from step 9 to add the servo code to control watering, using Figure 11.71 as your guide. Translate the visual code in Figure 11.71 to Python. A version appears following the next Knowledge builder and some helpful suggestions follow.

We need to be able to control a servo using Python. This additional code is shown on the following page in red. The first section adds a servo function to translate the analogue pin control voltage into a servo angle. The final additional code then rotates the servo from 0 to 70 degrees and back again. Note the delay to allow for this rotation. You may need to adjust your servo angle values.

KNOWLEDGE BUILDER

Supplying power to hobby servos

Servo motors have three wires: power, ground and signal (see Figure 11.72). The power wire is typically red, and should be connected to the voltage pin on your board. The ground wire is typically brown and should be connected to a ground pin. The signal pin is typically orange and should be connected to a digital pin on the board.

This will usually work – but only just! Hobby servos are designed to operate on higher voltage than supplied by the micro:bit. For this reason, make sure you have BOTH the battery pack and the USB cable plugged in to the micro:bit.

If you want to make certain there is sufficient voltage supplying your servo, use a separate 4.5V supply (a battery of 3 AAA or AA 1.5V cells) and connect as shown in Figure 11.73.

Important! Make sure you connect both the brown wire of the servo and the negative terminal of the battery to the GND pin on the micro:bit. Why? Voltages are relative. Voltage levels only make sense when compared to a common base voltage. We therefore must share a common ground.

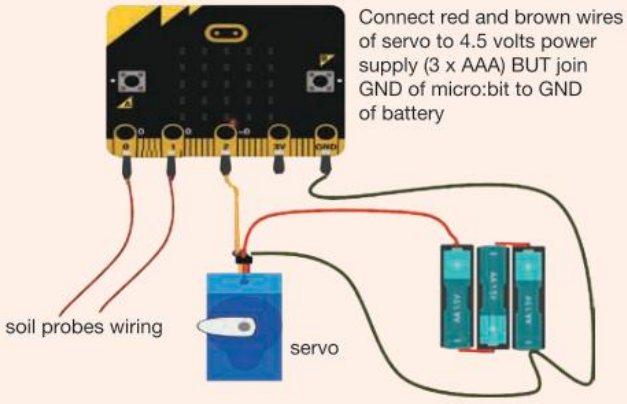


Figure 11.73 Use an additional 4.5 battery pack or similar to supply extra voltage to your servo, if needed.

Design thinking

Build the watering system

You are now ready to construct your automated watering control system.

12 Construct your irrigator and test your earlier design operates correctly. If you are having difficulties with a design, Figure 11.74 shows a simple one you can build.

- Fix servo to paddle-pop stick using an elastic band.
- Position the servo axis over rim of cup.
- Fix paddle-pop stick across rim of cup using elastic band or tape.
- Use a double servo horn (white plastic piece with servo) and attach a straw (fashioned as a scoop).

Large-style paper straws can be nicked halfway, bent up into a scoop shape and held by tape.

Figure 11.74 Cup, servo, paddle-pop stick and straw.

Python irrigation code

Here is a Python version of the irrigation code for you to check the version you created in Step 10.

```
from microbit import *
# define duty cycle for servo
pin2.set_analog_period(20)
def set_servo_angle(angle):
    duty = 40 + (angle * 4) / 9
    pin2.write_analog(duty)
while True:
    pin1.write_analog(1023)
    sleep(200)
    number = pin0.read_analog()
    sleep(200)
    pin1.write_analog(0)
    if (number < 800):
        display.show(Image.UMBRELLA)
        set_servo_angle(0)
        sleep(500)
        set_servo_angle(70)
        sleep(1000)
        set_servo_angle(0)
        sleep(500)
    else:
        display.show(Image.YES)
        sleep(500)
```

Testing and evaluating

13 Evaluate your final project against the criteria you listed at the start by writing a brief report.

- How can the hardware be improved?
- How can the code be improved?
- How fit were the materials for the purpose?
- Are there positive and/or negative social or ethical considerations?
- What did you learn?

Taking it further: Add remote control!

Visual code

Systems thinking

14 Figure 11.75 illustrates a visual coded version using two micro:bits to add remote control to your irrigator. The additional micro:bit remotely monitors the watering system micro:bit you constructed. The changing values on the transmitting micro:bit will be seen on the receiving one. The diagram shows the separate code for each of the micro:bits: the irrigating one and the monitoring one. However, you could even download the same code to both transmitter and receiver micro:bits, as their functions do not overlap.

Note that by adding a unique radio group channel number to both micro:bits, transmissions will not interfere with others in your class.

If you are coding visually, copy and test this code. Visual block code for these added features is provided in the Nelson MindTap resources.

General purpose code

15 Using Figure 11.75 as your guide, code the radio control in Python, using two micro:bits. The only additional MicroPython you need is to import the radio library, turn the radio on and send and receive the data using the following code snippets.

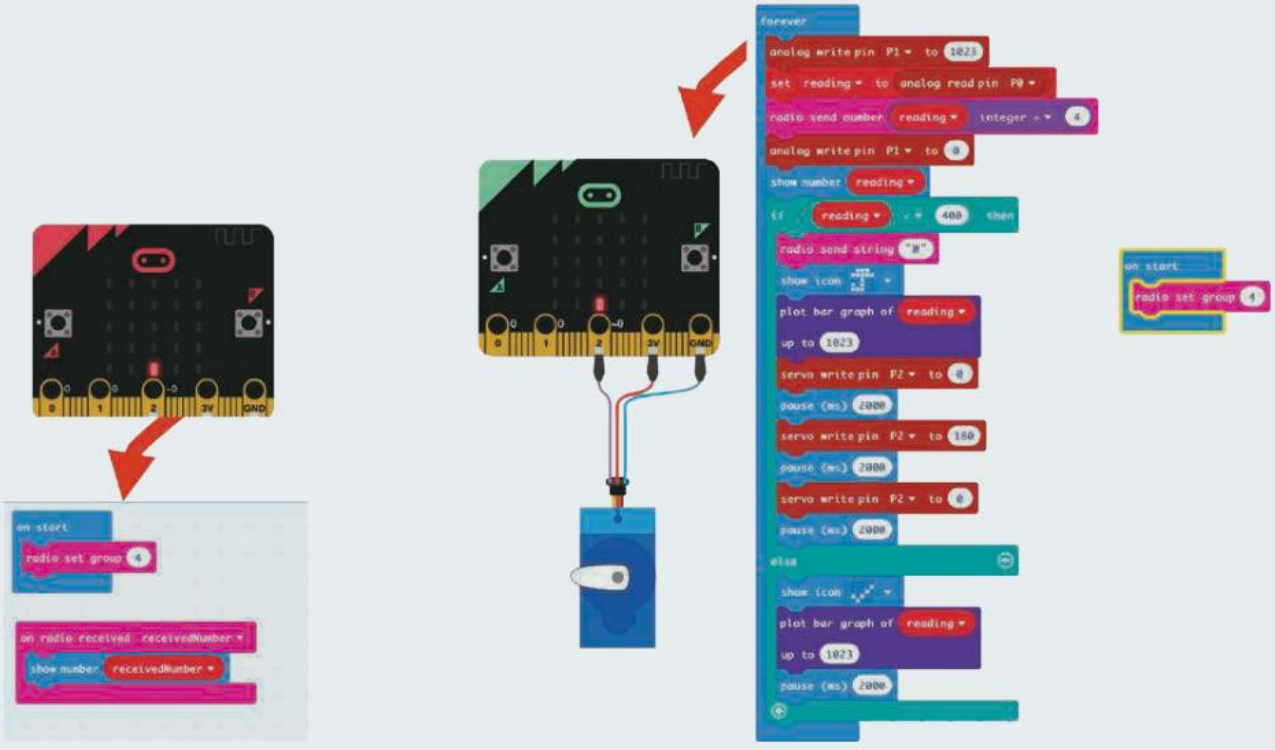
```
# radio module must be imported.
import radio

# radio must be switched on.
radio.on()

# add this to the receiver microbit
incoming = radio.receive()

# add this to the transmitter microbit
radio.send(str(number))
```

A complete Python solution is provided in the Nelson MindTap resources.

Further ideas for enhancements

Add further features to your project taking advantage of buttons A and B and gestures (using MakeCode's Input block library). Some suggestions:

- Display a bar graph to give visual indication of moisture.
- Add a feature to display moisture reading only when button A is pressed.
- Add ability to change threshold value without reprogramming by using gestures (tilt left/right).
- Add ability to reset default threshold value by using shake gesture.
- Maintain a count of the number of times the irrigator waters the soil and view result using button B.



Monitoring micro:bit

Irrigating micro:bit

Figure 11.75 Adding radio remote monitoring using two micro:bits and the radio function. Left shows transmitter code which controls the servo and transmits the data and right shows receiver code. which receives and displays the data.

11.12 Project 2: The amazing all-knowing wizard

Alamy Stock Photo/GRANGER - Historical Picture Archive

Figure 11.76 The Amazing Mechanical Turk chess player.

From 1770 until its destruction by fire in the 19th century, an amazing mechanical chess player fascinated everyone who went to see it. And this was before the discovery of electricity!

The 'Mechanical Turk' performance (Figure 11.76) travelled the world. Much later it was revealed a human chess player was hidden behind the fake clockwork mechanism inside the cabinet, despite the presenter first opening each door and illuminating the inside with a candle. For over 80 years, the Turk won most of the games it played and defeated many famous challengers (Napoleon Bonaparte and a founding father of the United States and inventor, Benjamin Franklin). If players tried to cheat or made illegal moves, its hand would sweep the pieces off the board! The Turk sometimes communicated by moving a finger to spell out messages on a letter board and baffled many spectators.

Project resource
The amazing all-knowing wizard

Your task

In this project, you create your automaton using two micro:bits. One micro:bit selects a question and the second controls a servo which points magically to the correct answer – across the other side of the room! Videos of the project in operation are supplied on Nelson MindTap.

Amaze audiences by letting them select any question. Secretly press button A, hidden in your pocket, the correct number of times to correspond with that question number. The wizard across the room will 'magically' move randomly then finally point to the correct answer!

(Note that this project also suits working in pairs.)

Materials required

- micro:bits (pack includes short USB cable and battery pack) (x 2)
- 'pin to alligator clip' jumper leads (available in local electronic stores) (x 3)
- paper clip, re-usable putty (such as Blu-Tack)
- question and answer sheets (supplied on Nelson MindTap)
- (optional) long 0.5m USB cable

Tracking your project development

We suggest that during the development of your projects you maintain a journal of your progress. It is suggested you only make brief notes and rough sketches.

Identifying and defining

How it works

You claim to have created an all-knowing mechatronic wizard which can understand a question and answer it. You ask the player to select from a list of questions. You secretly hold a transmitting micro:bit (in your pocket?) and use its button to select the question. Across the room a receiving micro:bit servo mysteriously rotates to point to the correct answer!

Researching and planning

A visual code prototype to test the concept

A note about coding: At secondary school, you are expected to do most of your coding in a general-purpose language (such as Python, JavaScript or C++). Here, we start with visual code as a springboard for such coding. It would be an advantage to complete the section 'Understanding programming' in Chapter 4 before commencing this project.

Systems thinking

1 First sketch a rough flowchart to meet the following requirements:
   - Button A of the transmitting micro:bit steps through questions numbered 1 to 8, and displays this integer.
   - Button B of the transmitting micro:bit transmits this integer by radio to receiving micro:bit which looks up array to identify and display corresponding answer.
   - Receiving micro:bit rotates servo arm to position to point at correct printed answer.

2 Devise a mathematical formula. Consider:
   - A typical hobby servo arm has a sweep of 180 degrees. Calculate the degrees between each of the eight positions which the wizard must point to (Figure 11.77). Create a function to output angles for inputs of integers 1 to 8.

Develop the code

Using visual code, we suggest you try to design as much of the code as possible before you look at our example code. Here are some suggestions to help you:

3 Open MakeCode. You require four blocks of code.
   - *on start* block to set radio channel
   - *function* block (located under Advanced) for a formula (see last step) to rotate servo arm to correct angle
   - *radio receive signal block*
   - *main program loop block to repeat the code forever*

Note: You can load the same code on *both* sender and receiver boards as each will respond only to their appropriate blocks of code. See Figure 11.75.

Start code

4 Use 'on start' block to select a radio channel from 1 to 255. Your teacher can assist issuing channel numbers to avoid conflicts.

Servo angle function

5 Join servo angle blocks to form a function to calculate servo angle using your formula from step 2 above. It receives a num value and rotates the servo through the correct angle. Note that MakeCode provides a specialised block to turn the servo angle a given angle. Search for 'servo' above the libraries.

Radio receive block

6 Use radio receive block to receive the question number and match it to differently numbered answer using a list (like an array) then move servo to correct angle and display the answer number. This will only occur on the receiver board even though the same code can be loaded on both.

Main program loop block

7 The purpose of a forever loop is to repeatedly listen for button presses and transmit them to the receiver micro:bit.

   Note we give the user visual feedback to confirm when button B is pressed successfully. Make certain your if ... then blocks are correctly nested.

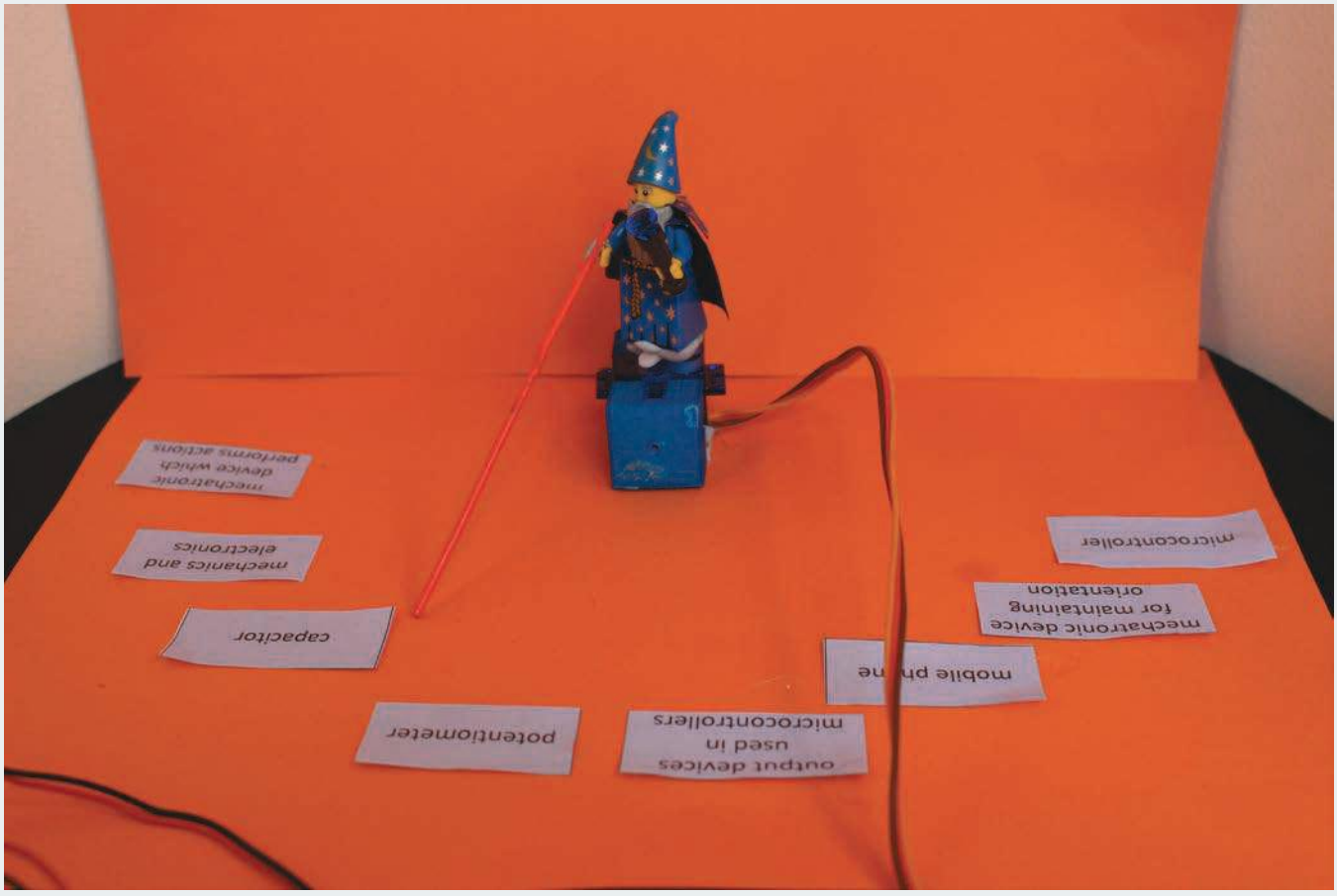


Figure 11.77 The automaton wizard mysteriously points at the correct answers. A video of it in operation is supplied on Nelson MindTap.

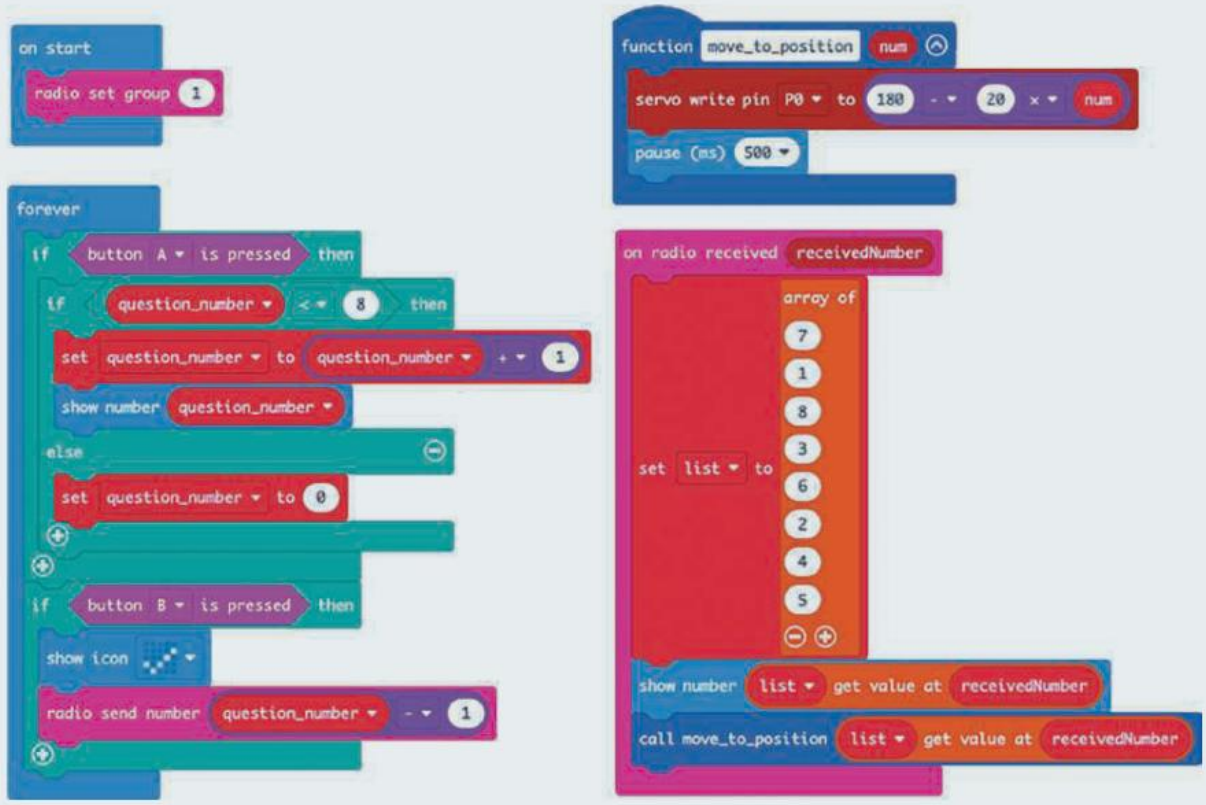


Figure 11.78 The block code version of the automaton created using the simulator. Transmitter and receiver code can be loaded together on both micro:bits as they do not conflict.

Test the code on the simulator

8 When you run the code using the simulator (Figure 11.78) and press button A, only one simulator board shows, the sender board, and a question number should appear on its LEDs. A servo will appear attached to this board, but it is not used here.

When you press button B, a second receiver simulator board and servo appear and radio transmission is indicated by yellow pulses. The answer for the question will show on this second board's LEDs and servo will rotate.

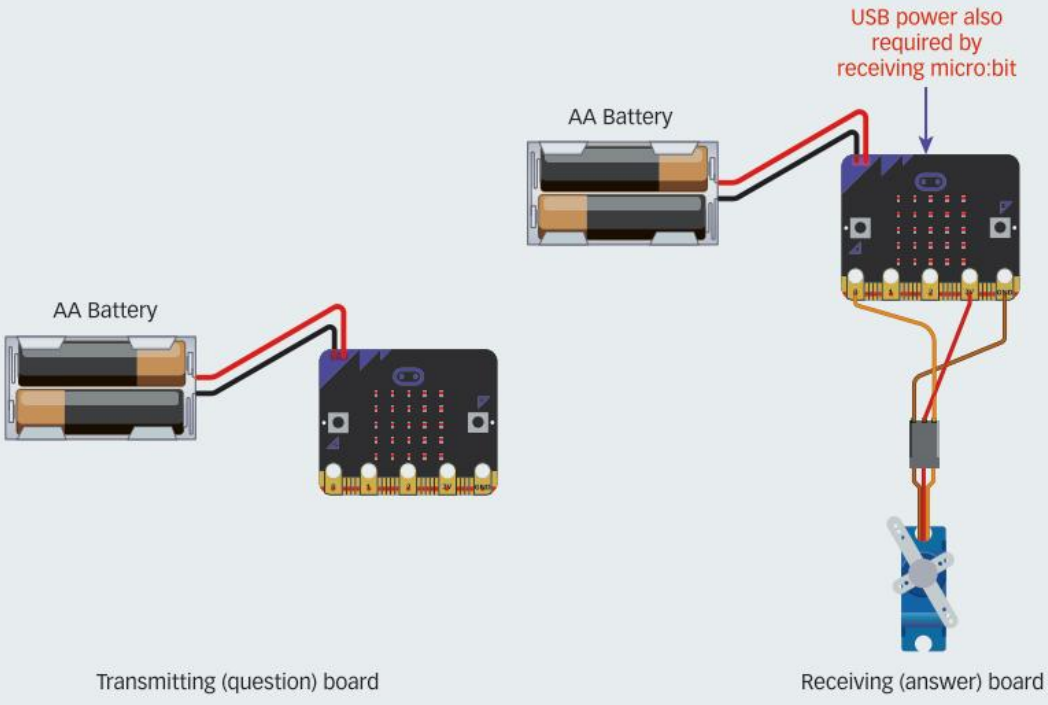


Figure 11.79 Wiring diagram for question board (left) and answer board (right). If your servo doesn't move properly, it likely lacks power. Refer to the Knowledge builder in Project 11.1.

Test the logic of your flowchart and formula above using MakeCode's Block code's simulator. Once this operates correctly, download block code to your physical setup and test a prototype.

Build your prototype automaton

Design thinking

9 Load the code on the two micro:bit boards. Then attach a servo to your answer board. Use re-usable putty to hold micro:bits and servo to a desk. At this prototype stage, do not worry about the actual questions and answers or the wizard character. The wiring is very simple and is shown in Figure 11.79. Note that the servo data wire is connected to pin 0.

Note we give visual user feedback to confirm when button B is pressed successfully.

Make sure you also plug in a USB cable to the receiver board for added servo power. If the servo still does not rotate smoothly, or if it 'shivers' swap to a 4.5 battery with 3 x 1.5V AAA cells. These methods should work OK but if not, refer to the earlier Knowledge builder which shows how to add more power to the servo.

10 Download the block code and test the device.

Producing and implementing

Coding the wizard using Python

Next we re-code the device using the general purpose Python language. Your hardware setup will remain the same. This time, however, we will create separate code for each of the boards.

Before coding in Python, select the Makecode Block Javascript tab and view the Javascript translation of your block code. This code is similar to the Python version. Note the need to add a function to tell the servo how to interpret angles. This is built-in to the visual code servo block.

Code the transmitting board

11 Open the micro:bit Python editor (Figure 11.80).

Although it is not necessary to type comments shown starting with #, it is good programming practice. Documenting code also makes it far easier to read.

Figure 11.80 Python editor for the micro:bit board with standard 'hello' code.

12 All micro:bit Python code must begin by importing specialised libraries to allow Python to control the micro:bit. Here, we will also need random number and radio communication libraries. Radio needs to be turned on before the program starts as, to save power, it is off by default.

- Type these first lines in your editor:

```
from microbit import *
import radio
radio.on()
radio.config(channel=1)
radio.config(power=7)
```

13 Behind the scenes in MakeCode, the blocks are actually built using JavaScript code, which responds to events (such as button presses and gestures) whenever they occur. To achieve this in a language such as Python we must use a constantly repeating loop to check each time if an event has occurred. It is important not to slow the loop with too many pauses or display statements.

- Add the following main loop code. Compare this to the visual block code and note how similar it is!
- The sender will secretly hide the transmitting micro:bit in their pocket. Questions start at 1 and each button A click will advance by one question up to 8 then revert to 1. We add a line to initialise the question number variable to 0.
- Note the final line: as our receiver code uses an array (always numbered starting at 0 instead of starting at 1) to find the answer positions, we need to transmit a number one less than the actual question number.

Computational thinking

```
# main program
question_number = 0
while True:
    if question_number == 8:
        question_number = 0
    # next question
    if button_a.was_pressed():
        question_number += 1
        display.show(question_number)
    # send request for answer
    if button_b.was_pressed():
        radio.send(str(question_number-1))
```

Weblink
micro:bit Python editor

After the final line add code to display a tick, as we did in the visual block code. Completed code for this first board is provided in the Nelson MindTap resources, called *automaton transmitting.py*.

Load code and test

14 Time to load your code!

- Load the transmitter Python code created in steps 11–13 to your transmitting (question) micro:bit. Note the first time you try to load Python code after using MakeCode, it will be slow to load as it is loading all of MicroPython. Also, you may need to disconnect and reconnect.
- Perform a preliminary test on your programming by repeatedly pressing button A. Button A will display a question number 1–8 and button B will send this to the receiving (answer) board (not yet coded).

Code the receiving board

15 Start code the same way by repeating step 12.

16 Add a line to create an array (a list) defining sequential questions matched to randomised answer positions:

```
list = [7, 1, 8, 3, 6, 2, 4, 5]
```

17 In MakeCode, visual blocks contain hidden functions that allow you to turn a servo a certain number of degrees. This is not available in MicroPython, however the following function (shown here for pin 0) can be used to calculate a standard servo's duty cycle given the degrees you want it to turn (see earlier Knowledge builder for an explanation of PWM).

- Add:

```
# converts angle to PWM duty cycle
def set_servo_angle(angle):
    duty = 40 + (angle * 4) / 9
    pin0.write_analog(duty)
```

Important note on coding: You should type this code – even if you have a pdf version of this text. Copying from a digital copy can introduce indentation errors. In Python you will already know how important indentation is. This is the way Python structures its code. Your code will not run if indentation is not exact. Fortunately, the micro:bit Python editor is clever and makes this obvious by adding green lines to show you indentation alignment and provides error messages to show syntax errors.

Computational thinking

18 We also need to add a function which calculates the servo angle for each of the eight answers. Can you explain how the following function works?

- Add:

```
# moves servo to given angle
def move_to_position(num):
    set_servo_angle(180 - 20 * num)
```

19 (Optional) Although there is no servo attached to the sender board, you might decide to add one later to have the sender (transmitting board) point to the questions. If so, add these last two functions to your transmitting board code. You will need to add a line of code to call the *move_to_position* function.

20 Add a function to receive question numbers via radio, look up the correct answer position (by referencing the list) and move servo to point at it.

```
# use list to locate answer position
def radio_check(msg):
    msg = int(msg)
    move_to_position(list[msg])
    display.show(list[msg])
```

21 Time for the main program! It is short and simple. Note we only call the `radio_check()` function after a message has been received.

```
# main program
while True:
    msg = radio.receive()
    if msg:
        radio_check(msg)
        sleep(500)
```

Load your code on the answer micro:bit. Completed code is provided in the Nelson MindTap resources, called *automaton receive.py*

Building and troubleshooting your device

Construct a protype automaton on your desk. Attach a servo to the second micro:bit as shown in Figure 11.79 by connecting its orange wire to micro:bit pin 0, red wire to 3V pin and brown wire to GND pin.

Is your servo 'shivering' or not moving consistently even after connecting both the battery pack and USB cable? If this still happens refer to the *Knowledge builder: Supplying power to hobby servos* in the first project.

Attach a pointer to the white servo arm. An unbent paperclip will do the trick. You will operate your automaton by secretly pressing button A to select a question. When you press button B, the receiver board's servo across the room will 'magically' point at different positions!

You may wish to design and construct a small puppet (robot or wizard?) for the answer servo – we used a LEGO wizard minifig attached to the servo arm (see Figure 11.77).

Design and print answer sheet

Now you have a working prototype, you can prepare and print the questions and answer sheets. An easy way is to type up answers then print and cut them out (see Figure 11.77). Stick down the answers so the pointer aligns accurately with the answers. Fix the servo/wizard to a desk with removeable putty.

Here is a test set of eight sample questions which match the answer sheet in Figures 11.77 and 11.81 using identical list positions as in the earlier code. This resource is provided in the Nelson MindTap resources.

Sender board sample questions

1. What fields are combined in mechatronics?
2. What type of board is a micro:bit?
3. What is an effector?
4. What everyday device allows us to explore many of the sensors used in mechatronics?
5. What electronic component features in an accelerometer?
6. What is a gimbal?
7. What is an actuator?
8. What electronic component forms an essential part of a servo?

A demo video of the working project in operation is provided on Nelson MindTap.

Testing and evaluating

Testing

Thoroughly test the device. Try sending questions from the transmitting micro:bit by repeatedly pressing button A to select the question and button B to send it. Does the receiving micro:bit servo turn as designed?

Make adjustments as required.

Evaluating

Evaluate your project against criteria you listed at the start by writing a brief report:

- (Class discussion) How could the behaviour of this automaton be improved?
- How could useability be improved for the operator?
- How could the equipment setup be improved?
- How could the code be improved?
- How suitable were the materials for the purpose?
- Are there any positive and/or negative social or ethical considerations?
- What did you learn?

Taking it further

- Add random slow servo movements to suggest your wizard is 'thinking' before locating the correct one. This will make a dramatic improvement to the overall effect.
  Hint: Use a counted loop which generates random answers 1 to 8. You will need the *random.randint(1,8)* function and will need to import *random* module at the start of the code. A solution with this feature is provided on Nelson MindTap, see *automaton receive extras.py*
- Add code so a simultaneous press of both buttons A and B (this is allowable on the micro:bit), or micro:bit's shake gesture, resets the question count to 0 in case you lose count on the secret micro:bit in your pocket.
- Produce multiple question and answer sheets for other topics.
- Eliminate the possibility of users predicting the pattern by creating two different answer patterns.

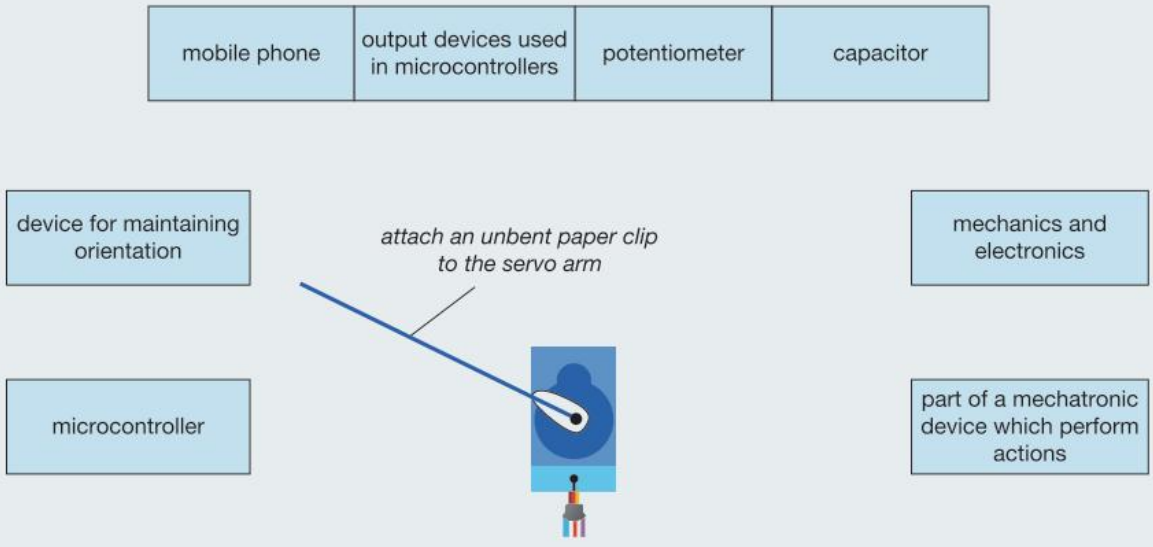


Figure 11.81 Matching answer sheet for the eight sample questions above. Note the order must match the list positions in your code.

12 Building robotic systems

FOCUS AREA: Building mechatronic and automated systems

Chapter outcomes

In this chapter, you will learn:

- the history of robotics
- the uses of robotics
- the types of robots
- to build and program a robot to complete a task
- about Asimov's laws of robotics
- to identify parts of a robot
- about the sensors and actuators used in robotics
- to identify the degrees of freedom of a robot
- about applications of AI in robotics.

Key terms

animatronics
automated system
contouring
controller
degrees of freedom
end effector
laws of robotics
pick-and-place
point-to-point
robot
robotics

9780170442787

IN ACTION

Farming with a Ladybird

ACFR, The University of Sydney

Figure 12.1 Farming with a Ladybird.

A world first robot called the 'Ladybird' (Figure 12.1) is being trialled among the onions, beetroots and spinach rows, of a farm in the New South Wales' central west.

It was a successful first trial for the University of Sydney's prototype, which cost about $1 million and a year's worth of scientific development.

'We've given it that name because it looks a little bit like one [a ladybird beetle], it has red covers with black spots from the solar panels.

'It's a solar powered, electric driven robot, with a variety of different sensors that we think are going to be useful for precision agriculture into the future,' said Dr James Underwood.

Dr Underwood is a Senior Research Fellow from the university's Australian Centre for Field Robotics and developed the sensory equipment on the robot.

The ground robot is purpose-designed for the vegetable industry, to help farmers collect intelligence on nutritional information, autonomous farm surveillance, mapping, classification and detection of pests and, eventually, autonomous weeding and harvesting.

'The first phase for us out here near Cowra has been to test the ability of the robot to drive itself around the farm.

'It's a fully autonomous vehicle, it's capable of driving up and down the rows of the farm with very good precision, it went really well and basically worked straight out of the box.'

The different sensors that Dr Underwood has developed will gather a wide range of farm intelligence.

'We've been testing what sort of information we get from a variety of different sensors that we have on board the platform.

'We have various different types of optical sensors such as stereo cameras, and hyper-spectral sensors and some laser range finders; with a combination of that data we can build up a 3D, very detailed type of picture, and map the farm down to the individual plant.

'It's our hope that with that data, and some algorithms that we design back in the lab, we can process that information to produce useful data for farmers to manage their crop.'

Peak vegetable industry body, AUSVEG, awarded The Ladybird's lead researcher Professor Salah Sukkarieh Researcher of the Year for his work, at their recent annual conference in Cairns.

Ed Fagan owns the farm in Cowra in central-west NSW where the Ladybird is being trialled.

The vegetable grower says that the Ladybird robot's intelligence gathering around pests and crop nutrition will be invaluable for farmers, 'A lot of the time in horticulture, if you're short of an element in the plant, by the time you see a symptom it's too late, they will be able to pick up a nutrient deficiency before we see any symptoms.

'Secondly, you can use it at night at 2 o'clock in the morning and go out and do an insect survey, so things like cutworm popping out at night time, slugs, worms, things like that.

'Instead of getting out of bed at 3 o'clock in the morning and wandering around with a torch and looking at about five square metres, this thing could do two or three hectares at night and then in the morning you can just see what you've got,' Mr Fagan says.

But Dr James Underwood says that it's about the big picture too.

'Ultimately, we hope that we can really maximise what you can get out of a unit of area of land in terms of growing produce, in terms of the quantity of yield but also the quality and food security,' she says.

Reproduced by permission of the Australian Broadcasting Corporation – Library Sales.
Alex Blucher © 2014 ABC

Weblink
Ladybird: part 2 - data and processing

Questions

Watch the YouTube video of the Ladybird in action.

1. Identify three industries where automated systems are commonly found.
2. Why do you think the impact of mechatronic devices on the agricultural industry has been relatively limited?
3. Outline the advantages and disadvantages of a farming device such as Ladybird.
4. Discuss a positive and a negative ethical implication of the widespread introduction of mechatronic devices in the agricultural industry.
5. Select three types of sensors on the Ladybird and research their functions for the agricultural industry.

12.1 What is the difference between a robot and an automated system?

There are many overlaps between these terms and in some cases a device could be described as both (Figure 12.2). Popular culture imagines robots as walking and talking humanoid domestic helpers. This would be a neat way to describe the difference between a **robot** and an **automated system**; however, robots often do not have the ability to move, such as those programmed to weld car parts together. The question is even further confused as some software is referred to as 'bots'!

Another approach is to use the term 'automated system' for systems designed to perform one specific task and the term 'robot' for systems that can be programmed to perform a variety of tasks. In the projects at the end of this chapter, you explore one of each by building a soil irrigation automated system and by programming a robot to escape a maze.

The Skill builders and projects in this chapter provide you with the opportunity to use a wide selection of popular classroom robots and to program them in a variety of computing languages.

Figure 12.2 Robots or mechatronic devices?

12.2 Robotics

Robotics is the science of the use and study of robots. The term 'robot' is usually applied to an automatically guided machine that is able to perform tasks on its own. Robots currently perform many functions, from making cars to defusing bombs. Children and adults play with toy robots, while vacuum-cleaning robots suck up dirt in a growing number of homes.

The use of robots

A robot can be programmed to perform tasks normally carried out by humans. These are often repetitive, dangerous, very precise or even impossible for people to undertake. Robots have the advantage of never getting tired, being able to work with higher accuracy than humans, and even being able to do things humans cannot do, such as working in extreme temperatures. When robots perform boring, repetitive tasks in place of human workers, employees are able to spend more time performing creative, less tedious tasks. However, the disappearance of whole categories of jobs which are more labour intensive or routine will have a significant continuing impact on society.

The history of robots

The earliest robots were built specifically to perform simple tasks on an assembly line. These types of robots are still widely used in manufacturing. Modern robots usually have sensors to accept data from the environment and are increasingly found in the area of artificial intelligence. Intelligent robots are designed to imitate human behaviour and thought processes. Their inputs may include the senses of vision, touch, hearing and smell. An intelligent robot must be able to do two things:

- obtain information from its surroundings
- carry out physical tasks.

Modern robots possess three characteristics:

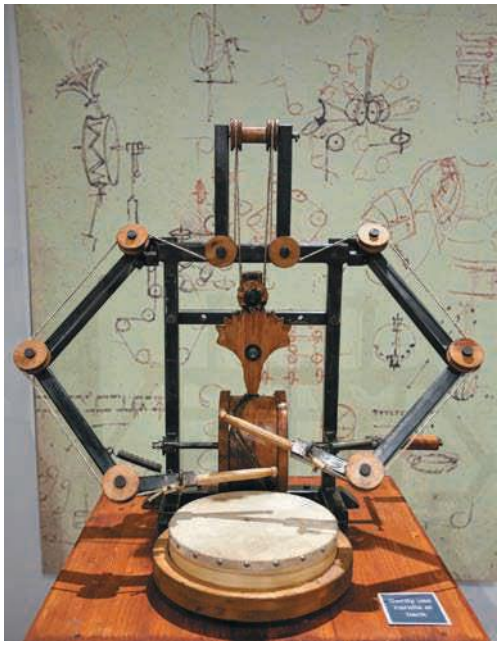
- programmability – they can be programmed to perform tasks like a computer
- mechanical capability – they can perform machine-like tasks in controlled environments
- flexibility – they can perform a variety of tasks by being reprogrammed or by responding to their external conditions.

INFOBIT

The word 'robot' comes from the Czech word *robota*, which means 'drudgery' or 'work'.

Mechanical, computerised robots have been around for less than 50 years but the concept of robotic humans is not new. The ancient Greeks spoke of 'mechanical helpers'. In the 15th century, Leonardo da Vinci, the famous Italian mathematician, artist and designer, drew plans for a mechanical drummer (see Figure 12.3). The drummer was for entertainment and it was designed to keep a perfect beat once its handle had been wound. (The project in the previous chapter involves building an automaton like this.)

During the 1920s, Karel Čapek wrote a play called R.U.R. (Rossum's Universal Robots). In Čapek's play, the Rossum factory produced mechanical devices that were called 'robota'. At first, these human-like machines were merely unintelligent slaves, but later, a scientist gave the creatures emotions. The robots eventually turned evil, killing the humans and taking over the world.



Newspix/Charles Brewer

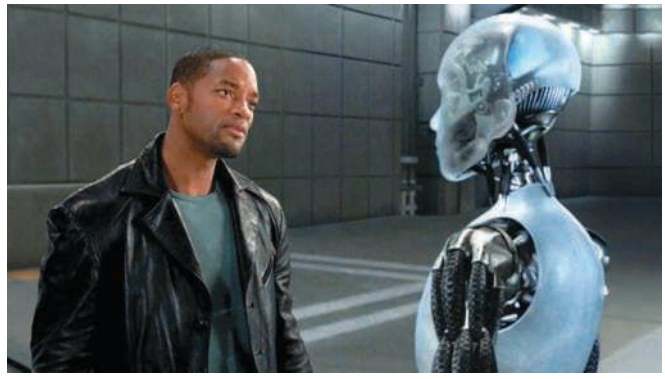
Figure 12.3 Leonardo da Vinci's drumming robot, created by engineer Gabriele Niccolai from da Vinci's sketches.

Following early instances in plays and science-fiction stories, robots started to appear in television shows, such as *Lost in Space*, and have since featured in many movies, including *2001: A Space Odyssey*, *Star Wars*, *Judge Dredd* and *Transformers*.

Asimov's Laws of Robotics

Isaac Asimov (1920–92) took a different view of robots from writers before him. He was the first to give robots relational characteristics in interacting with humans (Figure 12.4). Asimov thought robots should be regarded as a very important technological innovation. In 1942, he wrote a story about robots, *Runaround*, which contained his three **laws of robotics**. He later added a fourth law – the Zeroth Law. The laws are:

- Zeroth Law: A robot may not injure humanity or, through inaction, allow humanity to come to harm.
- First Law: A robot may not injure a human being or, through inaction, allow a human being to come to harm, unless this would violate the Zeroth Law.
- Second Law: A robot must obey orders given to it by human beings, except where such orders would conflict with the Zeroth Law or First Law.
- Third Law: A robot must protect its own existence, as long as such protection does not conflict with the Zeroth Law, First Law or Second Law.



Alamy Stock Photo/Pictorial Press Ltd

Figure 12.4 The ethical issues raised by Asimov's laws of robotics are explored in the movie *I, Robot*, named after an Isaac Asimov collection of short stories.

The robot industry

While the ideas had been around for some time, the construction and use of robots did not properly begin until the mid-1950s.

- In 1956, the first robot company was formed by George Devol and Joseph Engelberger.
- In 1959, computer-assisted manufacturing was first demonstrated at the Massachusetts Institute of Technology.
- In 1961, the first industrial robot, called UNIMATE, came online in a General Motors car factory.

Alamy Stock Photo/kolvenbach

Figure 12.5 In 1999, Sony built its first AIBO (robotic dog). The final version, released in 2006, could speak 1000 words, react to an owner's commands, take photographs and play music.

Alamy Stock Photo/Broccoli Photography

Figure 12.6 When robots first appeared in movies, they were often depicted as unfeeling villains with the potential to run amok and take over the world. More recently, robots have been portrayed as intelligent, social beings with a range of human-like emotions, as in the case of Pixar's robotic hero Wall-e.

CASE STUDY

An AI buddy

Alamy Stock Photo/Hugh Threlfall

Shutterstock.com/JCDH

Figure 12.7 (left) Anki's Cozmo, a programmable robot with built-in machine learning; (right) the Roomba robotic vacuum cleaner.

A companion robot everyone wants?

Anki, maker of Vector, a toy-like autonomous countertop robot, is shutting down, and hundreds of people are losing their jobs ... Sure, home robots like Vector are cute, but is this something consumers actually want? Apparently not enough customers for Anki, but here's what's far less clear: Beyond the success of Roomba, what on earth do we want sophisticated machines to do for us around the house?

There's perhaps no more challenging place to do that than in the pure chaos of the home, what roboticists call an 'unstructured' environment. Robots have for decades ruled the factory, because that's a 'structured' environment that's easy to master. They can travel along prescripted paths and perform highly repetitive maneuvers while facing few, if any, unexpected obstacles. In the home, robots have to contend with stairs and children and clutter on the floor.

The popularity of assistants like Amazon's Alexa didn't make the company's job any easier. Like the Roomba and its sweeping, these assistants do one thing very well. And they do it by sitting stationary on a countertop, which means they don't have to navigate the chaos of the home. If home robots can't yet grab you a beer from the fridge, fold laundry, or pick up literally any object, is there much value in their mobility? And if they're not as smart as Alexa, is there much point in making friends with them?

The makers of home robots also struggle to keep up with Amazon's and Google's ability to iterate on their voice assistants quickly. Hardware like Vector's is an entirely different story. If the company designs a robot and pushes it out to market and it's not working as desired, it could take years to redesign it.

The ultimate goal for home robots may not even be to build you a new best buddy but to make robots in a sense disappear

into the home. 'In the case of the Roomba, it's just a vacuum cleaner, we don't think about it much as a robot anymore,' says Cory Kidd, developer of Mabu, a medical robot that interacts with patients. 'And I think that's actually a success, when people think of these things in terms of their functionality and the problem they're trying to solve.' Think about the microprocessor: If you wear an old digital watch like me, you don't think of it as a computer because it's become more a part of you than something unique. But it's indeed a computer.

Still, what home robots do have going for them is that novel charm. This is where the human-robot interaction bit gets really interesting. Watching Kuri and Vector roll around beeping and booping and following your commands was a fascinating thing to behold. In essence, their physical presence allows for a bond with a machine loaded with far more charisma than what a stationary voice assistant can muster. That's the promise of home robots: The illusion of agency may not make them particularly useful at the moment, but they're forging a strange new relationship between people and machines. No longer are robots just tools—they're friends. And maybe one day (perhaps once they've sprouted arms and hands) they'll be so ingrained in the home, like Roomba, that we'll forget they're robots at all.

Vector may have failed to win enough hearts to survive, but Anki has made invaluable progress in fleshing out what this new human-machine relationship might look like. 'They were on the forefront of social robot design, and poured so much into getting the interaction right,' says MIT roboticist Kate Darling, who studies human-robot interaction. 'It was a beautiful product. But they had trouble growing it from nifty toy to something else.'

Source: 'R.I.P., Anki: Yet Another Home Robotics Company Powers Down', *Wired*, 29 April 2019.

ACTIVITIES

Identify

1 Define the term 'robotics'.
2 Identify the purpose of the earliest robots.
3 Identify the category of component in modern robots that enables them to accept data from the environment.
4 Name two characteristics of an intelligent robot.
5 Identify three main features of modern robots.
6 What was the name of the first industrial robot and in what year did it come online?

Analyse

7 Scientists have a very different view of robots from science-fiction writers and movie-makers. Describe some of these differences.
8 Discuss the relevance of Asimov's Laws of Robotics to the robotics industry today.
9 Describe a way to distinguish robots from automated systems.

Extend

10 Either write a short story involving Asimov's Laws of Robotics or summarise the plot of *I, Robot*, the movie starring Will Smith, loosely based on an Asimov story.

12.3 Types of robots

There are many different types of robots. One simple way to classify robots is according to whether they are stationary or mobile. Stationary robots are fixed in place, but their arms can reach out and manipulate parts and tools. These robotic arms have an **end effector** (Figure 12.8) attached to carry out a task.

Stationary robots are often found in assembly lines, where they perform simple tasks such as placing objects on a conveyor belt. Mobile robots, on the other hand, can move around in their environment and are not fixed to one physical location.

Industrial robots

Industrial robots have been around for more than 50 years, performing simple, repetitive tasks with superhuman speed and precision. They are now responsible for a remarkably wide variety of tasks, from assembling computers to popping frozen dinners into trays.

In the manufacturing industry, robots perform a variety of tasks (Figure 12.9), including:

- welding
- spray-painting

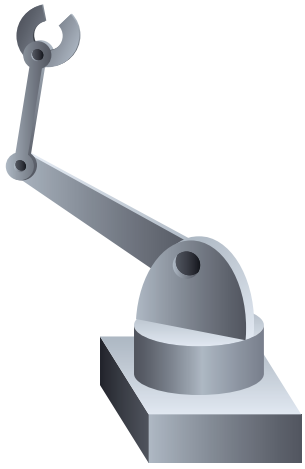


Figure 12.8 A robotic arm with an end effector attached.

- assembly operations
- palletising and material handling
- dispensing operations
- laboratory applications.

Robots with exchangeable tools

Some robots used in manufacturing are designed to perform one particular task, while others are designed with interchangeable parts, which make them more versatile. For example, a robot may act as a welder for one task, but by changing the end gripper, it may be turned into a spray painter. To reprogram a new robot arm, an operator simply selects from among choices that the robot itself offers; a worker can teach a robot's 'eyes' to recognise a new part in less than five minutes.

ThinkStock/Andrzej Thiel

Figure 12.9 Robots in manufacturing.

INFOBIT

Industrial accidents and the prevalence of some cancers have declined as a result of robots performing hazardous jobs in the workplace.

Mobile robots

Mobile robots usually move around on wheels, tracks or legs and began to appear in the 1980s in the area of nuclear science. In 1979, a nuclear accident at Three Mile Island in the United States caused a leak of radioactive material. The accident was a major concern because of the amount of radioactive substances released into the environment. This incident led to the development and use of special robots, called 'teleoperators', to handle the radioactive material. Today, mobile robots are used in areas such as security control, bomb disposal, planet exploration, transporting hazardous materials, and disaster rescue and recovery (Figure 12.10).

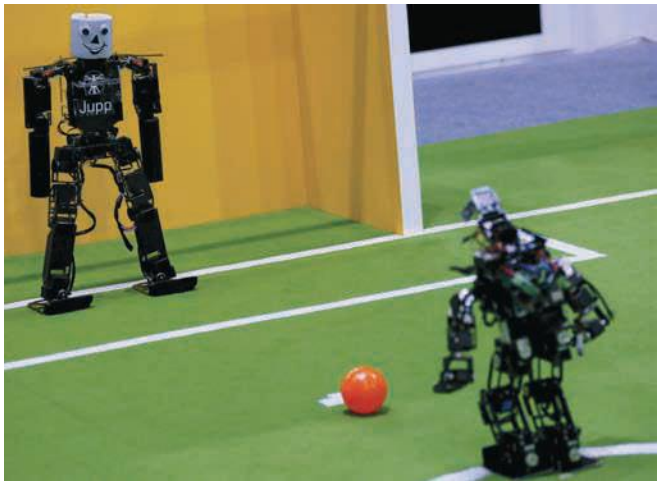
Hydronalix

Figure 12.10 This mobile robot is used to rescue distressed swimmers. Its sonar device scans for the underwater movements of swimmers in trouble and an in-built speaker allows a lifeguard on shore to calm and instruct the person being rescued.

Educational robots

Robots, or robotic kits, are used extensively in education. Robotic equipment in the classroom is used to teach and learn about programming and computer control. A well-known example is the Spike Prime robot system developed by LEGO. These comprise a brick control device, sensors and motors, which users assemble into models. Among other functions, the robots can be programmed to walk and to carry out a variety of tasks or missions. In addition, projects in this chapter will provide you and your classmates with the opportunity to collaboratively embark on a series of robotic challenges.

Another educational example is the programming of robots to play soccer in the RoboCup competition (Figure 12.11). Teams from all over the world enter the competition. The robots operate completely autonomously. The rules stipulate there must be no external control by humans.



Getty Images/Malte Christians

Figure 12.11 In the RoboCup Humanoid League, autonomous robots with human-like senses play soccer against each other. RoboCup is an annual international robotics competition that promotes robotics and artificial intelligence (AI) research. There are many other stages of the competition, such as RoboCup Rescue, RoboCup OnStage (Performance) and RoboCup Junior.

INFOBIT

Interactive 'social robots' called 'Petimos' have been designed to help protect children when they make friends on social networks. Petimos' primary function is to guard against cyber bullies and adults masquerading as children.

Domestic robots

Robotic toys include remote-control cars, remote-control boats, barking dogs and crying dolls (Figure 12.13). Other domestic robots perform boring and repetitive household chores, such as vacuuming, or provide services such as security surveillance (Figure 12.12).

In Japan, domestic robots have been developed which provide companionship and assistance to the elderly (Figure 12.14).

Shutterstock.com/lev radin

Figure 12.12 The K5 security robot developed by US firm, Knightscope.

Source: Wired

Figure 12.13 Many modern toys are considered robotic because they are programmed devices. They may react to light, sound and touch, using special sensors for input.

Source: Kompai

Figure 12.14 Kompai is a domestic robot that can record grocery lists via voice commands, call an emergency number for medical assistance and remind elderly people to take their medication.

CASE STUDY

The changing nature of work

Spill something in a nuclear reactor core and you can't just send in the janitor to clean it up. (Well you can, but just once!) What you can do – repeatedly – is send in remotely operated robots designed by Matt Cole (Figure 12.15). At 33, Cole is chief engineer of Colorado-based SA Technology, which earns more than $20 million a year making 'bots' for nuclear facilities.

For a kid who taught himself to weld, mill and lathe before he was old enough to vote, Cole believes it is a dream gig. 'We do the worst, the hottest, the weirdest jobs,' he says.

Like what? For starters, his creations mop up areas that are so radioactive they've been dubbed infinity rooms – a Geiger counter needle goes past 11!

Each task requires its own unique bot. For example, if a piece of nickel lodges in a pipe at a reactor plant and turns into the extremely radioactive and dangerous isotope cobalt-60, Cole and his team of engineers construct an agile bot that can extend a telescopic arm, spray liquid nitrogen to cool the pipe, and cut away the radioactive segment (Figure 12.16). Other bots are designed to plunge into giant waste tanks and hose down radioactive materials such as caesium-137.

SA Technology bots are at every major nuclear weapons facility in the United States, as well as most nuclear power plants and as more and more Cold War-era weapons facilities are being decommissioned, business has never been better.

Joshua Zaffos, 'Alphageek: Matt Cole Builds Robo Rooters that Scour Out Reactors', Wired magazine, June 2010

Wired/Joshua Zaffos

Figure 12.15 Matt Cole's robots perform tasks that are too dangerous for humans.

Alamy Stock Photo/ZUMA Press, Inc.

Figure 12.16 In order to reduce the safety risks to humans, robots can venture into radiation-exposed areas.

Questions

1. Describe the tasks carried out by Matt Cole's robots.
2. Are robots such as those created by Matt Cole the answer for solving toxic accidents and decommissioning nuclear weapons?
3. Automation and the development of robots for dangerous tasks are viewed by most people as beneficial to society. However, many workers in less dangerous jobs have been made redundant by robots. What is your response to this issue?

INFOBIT

During the 2011 nuclear crisis in Japan, many people were surprised that robots were not deployed to repair the crippled reactors. Despite being a technological powerhouse and a world leader in robotics, Japan essentially relied on a small team of engineers, firefighters and helicopter pilots to perform the task of cooling the reactors. Find out why.

Figure 12.17 The *Tobbie* robot won a 2018 Educational Toy of the Year award. It has an IR sensor for input, an audio buzzer and three motors as actuators, controlled by a micro:bit microcontroller board allowing it to be programmed using either block coding or Python.

SKILL BUILDER

Get to know your robot

Using a LEGO Spike Prime set, or equivalent, familiarise yourself with the kit and its programming environment. Most robotics kits supply introductory tutorials.

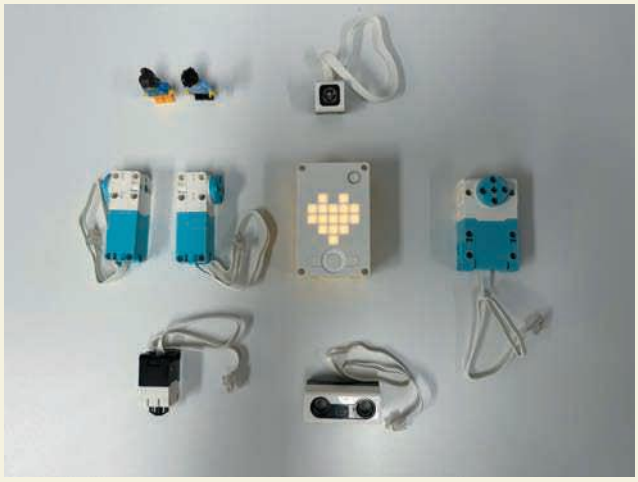


Figure 12.18 Sensors and actuators in a standard LEGO Spike Prime robot kit. The main hub is in the centre with a 3-axis accelerometer, 3-axis gyroscope and LED display; clockwise from bottom: ultrasonic distance sensor, touch/force sensor, medium motors, light/colour sensor, large motor.

1. Examine the components supplied in a robotics kit. Identify special components such as the light sensor, touch sensor, sound sensor, motors and lights.
2. Run the software you will use to code and operate the robotic equipment. For LEGO, this may be the *Spike* app, *Mindstorms EV3, MakeCode for Mindstorms, EV3 Classroom* or *EV3 LEGO MicroPython*.
3. Work through a tutorial to familiarise yourself with the hardware and software. Pay particular attention to the icons and buttons in the program.
4. Identify the output ports and the input ports. What names are they given in the program? What components are plugged into the output ports? What components are plugged into the input ports?
5. Use one of the many tutorials to build a robot using your kit.
6. Assemble and load the program in a tutorial to make your model functional.
7. Run the program and observe the model in action.
8. Write a program of your own for the model and run it.

ACTIVITIES

Identify

1. Identify the main difference between a robotic arm and a mobile robot.
2. Why are robots used in industry?
3. List four tasks performed by robots in manufacturing.

Analyse

4. Find an example of a robot used for security control. Describe the function of the robot in the system.

Investigate

5. Develop and conduct a survey on the use of robots in the home and in industry. Develop a chart to illustrate the findings and present them to the class. For example, a sample of questions for domestic robots might include the following options.
   - What would you want a domestic robot to do?
     - ☐ Housework
     - ☐ Gardening
     - ☐ Cooking
     - ☐ Security
     - ☐ Entertainment
     - ☐ Other
   - How much would you be prepared to pay for a domestic robot?
     - ☐ Less than $100
     - ☐ Between $100 and $500
     - ☐ Between $500 and $2000
     - ☐ Over $2000

   Questions relating to industrial robots could include the following suggestions.
   - How aware are you of robots being used in industry to manufacture goods?
   - Why do you think robots are used in industry?
   - What advantages do robot workers have over human workers?
6. Visit the RoboCupJunior Australia website and read about the Junior National competition.
   - a. Locate online two Australian schools that have entered the competition and read about their project.
   - b. Find out where and when the next RoboCupJunior competition will be held.
   - c. With your teacher, discuss the possibility of entering this competition.

12.4 The purpose of robots

The most common use of robots has been in manufacturing, where robotic equipment often replaces human workers on production lines. Robots are preferred because they:

- can operate equipment more precisely than humans
- can work 24 hours a day, 7 days a week, without getting tired or losing concentration
- can be cheaper to use over a long period
- can work in environments that may be too dirty or dangerous for humans
- can do work impossible for humans to perform.

Dirty tasks

In manufacturing, many jobs in factories are messy or dirty. Dirty tasks may include welding, grinding, spraying, moulding and casting. When robots are used to perform these tasks, it enables human workers to partake in more meaningful and creative pursuits.

Repetitive tasks

Robots are reliable workers, do not have emotions and therefore do not feel worthless when performing menial tasks. For most people, repetitive tasks performed in industry are considered very dull or boring. For example, a worker's only task may be to pick up an object from a conveyer belt and place it in a box. The robot can perform this task all day, every day, without getting bored.

Dangerous tasks

In manufacturing, robots often perform tasks that are very dangerous. Using robots for tasks involving extreme temperatures, for example, reduces the risk of workplace accidents. Apart from dangerous tasks in manufacturing, robots are being used to carry out other important but dangerous activities such as clearing landmines, helping in rescue missions and mopping up toxic leaks. Police robots are used to defuse and remove explosive devices. Some robots are tough enough to survive multiple blasts.

INFOBIT

The first known case of robot 'homicide' occurred in 1981, when a robotic arm crushed a Japanese factory worker.

Getty Images/selimaksan

Figure 12.19 Purpose-built robots include prosthetic arms that can handle heavy or delicate tasks.

Impossible tasks

Working deep under water, exploring live volcanoes at short distance or travelling to distant planets are impossible tasks for humans. Robots are often called on to perform underwater salvage missions to find sunken ships or planes. The *Titanic*, which sank in 1913, sat undiscovered on the ocean floor at a depth of 3811 metres for more than 70 years. At this depth, the ship was too deep to be explored by human divers. In 1985, a team of researchers, with the aid of a robot named Jason Junior (JJ), were able to locate the wreck of the *Titanic* (see the In action section of Chapter 4).

Underwater robots were also called on following the explosion and fire on the Deepwater Horizon drilling rig in 2010. Operating one and a half kilometres under the ocean's surface, they played a vital role in the fight to stop oil gushing into the Gulf of Mexico. In 2014, robot submarines were used to search for a missing Malaysia Airlines aeroplane.

PEOPLE

Animatronics engineer Grant Imahara

Grant Imahara (Figure 12.20) is an **animatronics** engineer, who has worked on such movies as *The Lost World: Jurassic Park, Star Wars Episode 1: The Phantom Menace, Terminator 3: Rise of the Machines* and *The Matrix Reloaded*. Animatronics is a multidisciplinary field integrating puppetry, anatomy and mechatronics. Before the term 'animatronics' became common, these creations were usually referred to as 'robots', however, these days robots are usually programmable, can perform a variety of tasks and do not necessarily resemble living creatures.

Grant was one of those kids who would disassemble remote controls and take all the wheels off toy cars. His life changed at the age of four when he received his first LEGO set. It introduced him to robots and provided a foundation for his interest in engineering. After graduating from university, he was asked to work for Lucasfilm, where he worked in the special effects model shop.

He made spaceships, miniature cities and robots. He was part of the R2-D2 development team for episodes one, two and three of *Star Wars* (Figure 12.21). Then the *MythBusters* opportunity came along and now he gets paid to blow things up. Grant has also written a book on building robots called *Kickin' Bot* and his combat robot, Deadblow, was featured on *BattleBots*.

Getty Images/Jason LaVeris

Figure 12.20 Grant Imahara is an animatronics and robotics expert who is best known for his work on the American television show *MythBusters*.

Alamy Stock Photo/JJM Stock Photography

Figure 12.21 In *Star Wars Episode I: The Phantom Menace, Star Wars Episode II: Attack of the Clones* and *Star Wars Episode III: Revenge of the Sith*, R2-D2's movements were controlled by Grant Imahara.

Questions

1. Identify the event that provided a foundation for Grant Imahara's interest in engineering.
2. Research and provide a job description of an animatronics engineer.
3. Research and explain the relationship between animatronics and robotics.

ACTIVITIES

Identify

1. Identify the industry where robots are most commonly used.
2. Outline the reasons why robots are used in manufacturing.
3. List four areas of robotic use today, apart from the manufacturing industry.

Analyse

4. Imagine you could have a robot that would do any task you requested – a companion to do all the work you find tedious or tiring. How do you think this might affect you as a person?
5. Describe types of jobs that would not be suitable for robots.

Extend

6. Underwater robots proved invaluable following the 2010 Deepwater Horizon oil spill (also referred to as the Gulf of Mexico oil spill). Research this environmental disaster, paying particular attention to the:
   - reasons why robots were used to contain the oil leak
   - difficulties encountered during the clean-up
   - tasks performed by the robots.
7. Investigate the ways in which robots are used to help people with disabilities.

12.5 Robots in action

Robots are used in many areas of our modern life to perform a wide range of tasks. In manufacturing, robots are used in assembly production lines. They are also used in all types of exploration (in space and on Earth), in medical science to perform operations, in agriculture, and to maintain and repair machinery.

Agriculture

In agricultural robotics, the location of individual plants is often used to identify plantation rows from aerial images. Knowing where rows are and the order in which they are best travelled is important in efficient detection of weeds and diseased plants, plant counting and correction of missing areas in sowing.

Drones and land robots can monitor each plant on a farm individually and this can reduce the amount of herbicides and pesticides that must be applied. Both the farmer and the robot can use data about the type and distribution of the weeds in the field.

CASE STUDY

Motion profiling in agriculture

Robotic management of crops usually begins with imagery from a drone converted into a data cloud of points representing the plants in a field. These are converted into points then into lines representing plantation rows (see Figure 12.22).

Shutterstock.com/Love Silhouette

Figure 12.22 This UAV (unmanned aerial vehicle or drone) was used for field monitoring (left) and the image it took (right) shows plants in green and weeds in purple.

Autonomous agricultural robots, such as the one in Figure 12.22, can water, plant, spray and harvest crops and identify the most efficient route and the exact location of each row.

A *motion profile* (Figures 12.23 and 12.24) describes a robot's movement at regular intervals using position, velocity, acceleration and heading (direction). If a robot is intended to follow a path this data is essential. Its *trajectory projection* indicates where the robot should head and *completion algorithms* describe the travel pattern which will cover the desired path.

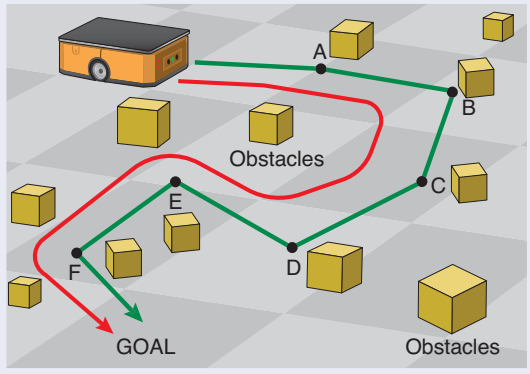


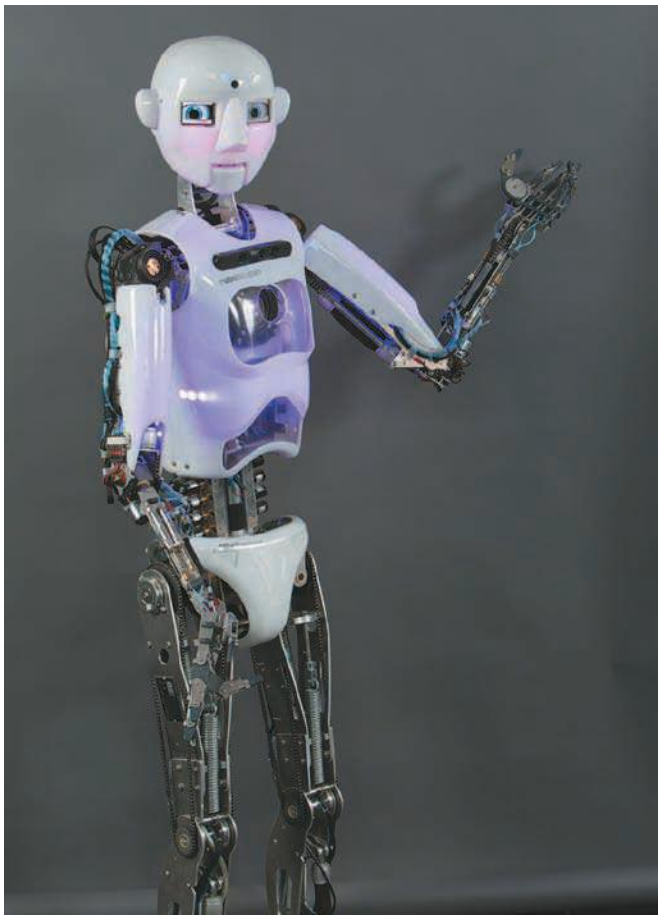
Figure 12.23 A *motion profile* completely describes a robot's movement at regular intervals using position, velocity, acceleration and heading (direction).

Professor Salah Sukkarieh, Australian Centre for Field Robotics, University of Sydney

Figure 12.24 In 2016, the Australian Centre for Field Robotics developed the world's first cattle station custom robot, *SwagBot*, an omnidirectional electric robotic ground vehicle, capable of navigating difficult cattle station environments through water crossings and steep ground.

Exploration

For centuries, explorers and scientists have been gathering data about Earth and our solar system through missions and voyages, often at great risk to themselves. Today, robots have enabled research and data collection to take place without directly involving humans (Figure 12.25).



Source: Questacon

Figure 12.25 RoboQ is an adaptive robot that receives input from sensors to make physical movements or produce vocal responses.

INFOBIT

Unlike hardware robots that we can see and touch, web robots (also known as web wanderers, bots, crawlers or spiders) are programs that traverse the web automatically collecting and analysing data.

Robots in the Antarctic

Compared to a century ago, the Antarctic is relatively accessible to scientists wishing to study its climate, ecosystems and glaciers. However, there are still places too dangerous, too remote or simply too expensive for humans and standard research equipment (Figure 12.26).

Robots in the Antarctic are involved in a range of research activities, including flying low over vast ice sheets and mapping the hidden realms below ice-covered lakes and seas. The types of robots include pilotless aircraft capable of flying pre-programmed routes or automated submarines that venture far below ice shelves at great depths.

US National Science Foundation/Josh Landis

Figure 12.26 Nomad was the first robot to explore an extreme polar environment. It is equipped with a sophisticated laser rangefinder and sensors, used to detect obstacles and alert the robot to dangerous terrain.

Robots in space

NASA researchers envision futuristic robots that act like people and work efficiently with astronauts. Building human-like robots capable of using their hands for intricate work is not new to space travel (Figures 12.27 and 12.28). The original Robonaut was conceived in the 1990s.

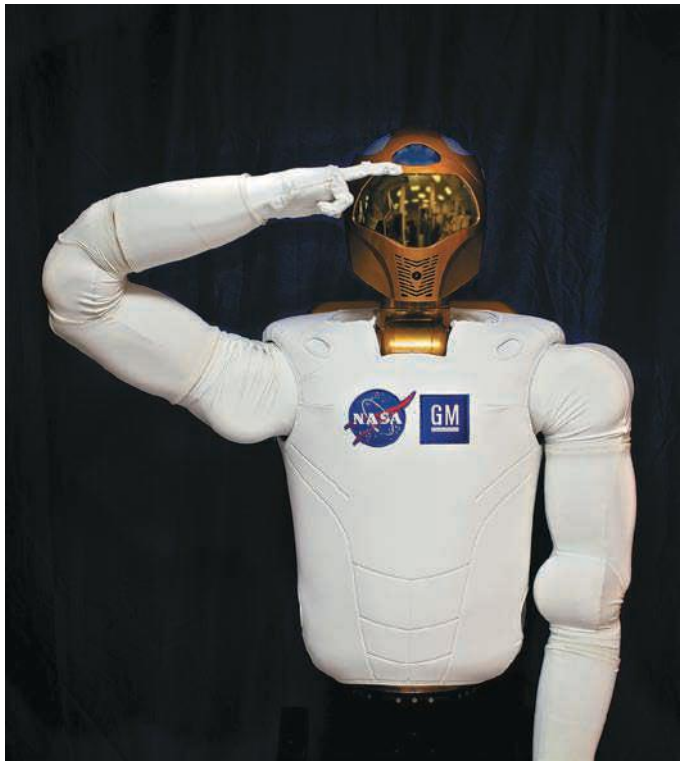
Robonaut 2 (R2), developed by NASA and General Motors, can lift and hold four times the weight of any previous robotic assistant. R2 is stronger, has more movements and is more technologically advanced than the original Robonaut, according to NASA. The flexibility of R2 allows it to use the same tools as astronauts, removing the need for specialised robot components.

The robot has two highly flexible arms and hands on a torso designed to be bolted onto NASA's Destiny laboratory on the International Space Station. Astronauts can replace an arm, a hand, or a limb as needed and will be able to attach other parts for future stages in its life. The greatest benefit of R2 may eventually be in performing jobs too difficult or dangerous for humans.

R2 is also destined for car assembly plants. It can lift approximately five kilograms with each arm, about four times that of other humanoid robots. Its nimble hands, fingers and opposable thumbs also enable it to perform more specialised or intricate jobs.

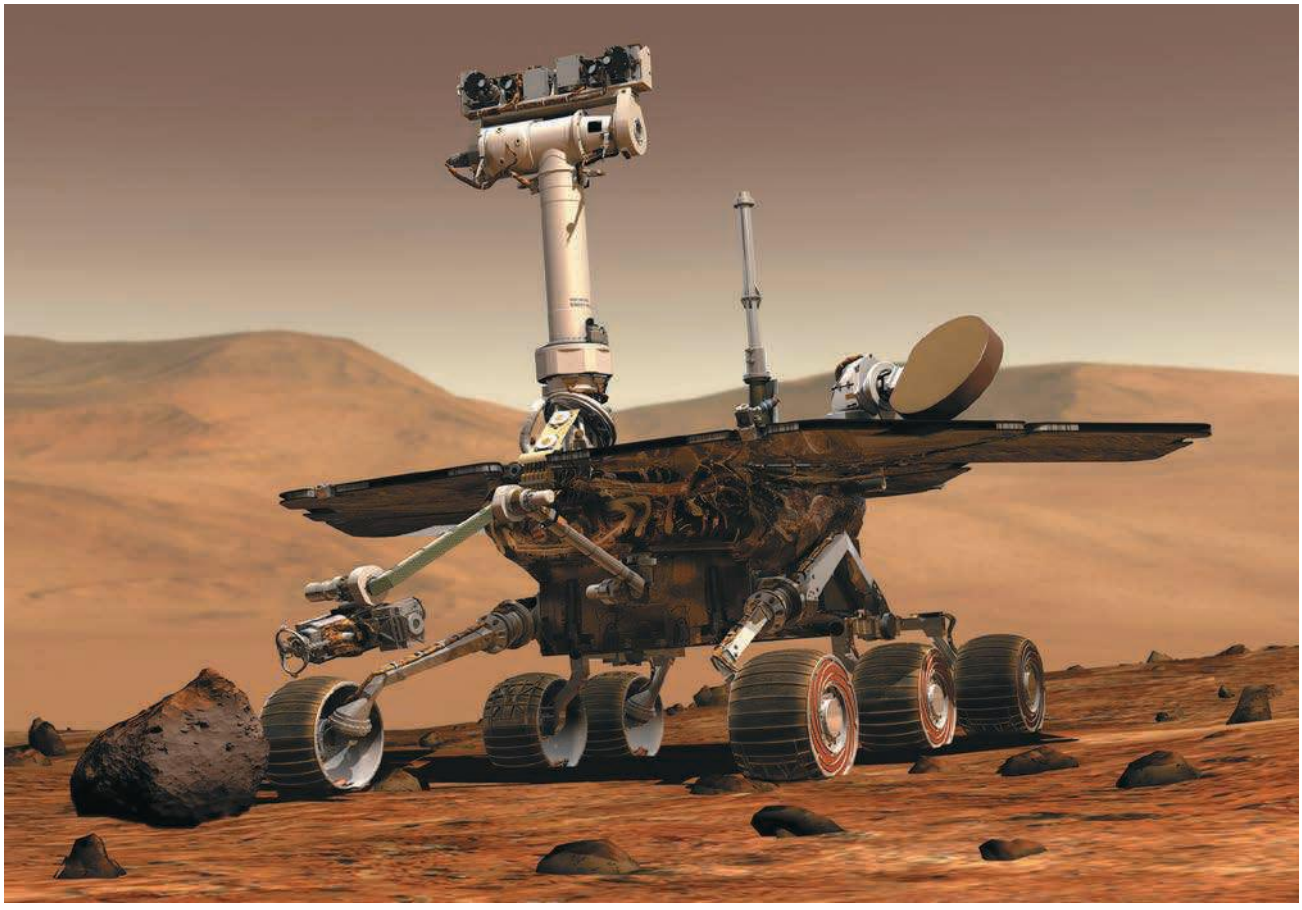
Use the Robonaut weblink provided to research the Robonaut and describe its purpose and the features that would be considered human-like.

Weblink
Robonaut



NASA Images

Figure 12.27 Robonaut is a human-like machine that, when fully developed, is expected to perform tasks and seek help only when it encounters problems it cannot solve on its own.



NASA Images

Figure 12.28 The Mars Exploration Rovers performed on-site scientific investigations of the 'red planet' until 2018.

Underwater robots

Exploration also takes place underwater where temperatures are at the other extreme to that of volcanoes (Figure 12.29) and pressure may be strong enough to crush the human body. The design of the latest underwater robots is closely based on the anatomy of crabs – known for their ability to walk both on land and under water. With the help of such amphibious robots, scientists have been able to study marine plant and animal life, as well as ocean currents and underwater ecologies.

INFOBIT

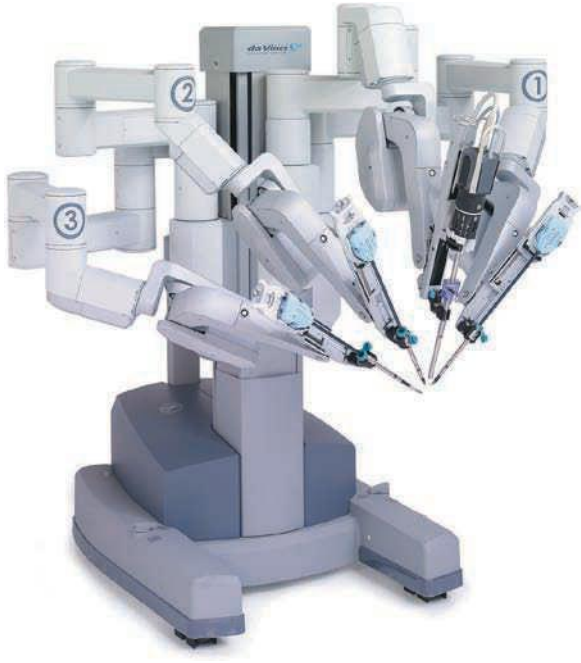
Underwater robots are often large and when they crash into things such as delicate coral they can cause a lot of damage. Robotic engineers in the United States are developing small, agile, fish-like underwater robots that will explore sunken ships and monitor coral reefs. They can also be used to move through urban industrial pipes to track down illegal pollution spills.

Alamy Stock Photo/Science History Images

Figure 12.29 Live volcanoes are extremely dangerous places for people to be near. Robots such as Dante II are able to travel into live volcanoes to collect gas, water and rock samples for scientific research.

Medical robots

Extensive research has been carried out into the use of robots in medical science. Researchers believe robotic technologies can improve existing procedures and provide new approaches to current problems.



Alamy Stock Photo/WENN Rights Ltd

Figure 12.30 Dr Roboto is a remotely operated surgeon featuring four arms.

One example is the use of robots to perform surgery through very small incisions, greatly reducing the risk to patients (see Figure 12.30). The previous chapter outlined developments in 'soft touch' actuators for such robots.

Manufacturing robots

Robots and computer-controlled systems have made a significant impact in the manufacturing industry, particularly in assembly lines. In the modern world, much of what is produced is done so on an assembly line. High-speed assembly lines produce cars, furniture, soft drinks, computers, razor blades, toasters and numerous other items. These goods are produced in massive quantities, and all have to be of the same quality.

In 1961, the first industrial robot was introduced to the assembly line at General Motors' vehicle manufacturing plant. The system allowed the company to manufacture cars quickly and at a low cost. The robot, called UNIMATE, was developed by the father of robotics, Joseph F Engelberger. The task of the huge robotic arm was to sequence and stack hot pieces of die-cast metal by following step-by-step instructions. It greatly improved the speed and accuracy of vehicle manufacturing. But this new technology resulted in changes to the nature of work in the car industry. Many jobs were replaced by robots. Workers who did retain their positions were required to retrain, since the remaining jobs required more technical skills.

Robots for maintenance

Robots are well suited to performing maintenance and repair work, such as on roads. Road surfaces tend to develop long cracks over time, which can make driving hazardous and therefore need to be repaired constantly. By taking preventative maintenance measures using robots, the durability of a road is significantly enhanced.

INFOBIT

Swimming robots are micro or nano robots that can swim when injected into the body. Their applications include removing parasites and breaking kidney stones.

Robots assisting with disabilities

The world's most sophisticated robots are now designed to support our growing population of elderly and disabled citizens (Figure 12.31). There is a range of household robots that can understand instructions and help with household chores such as taking a bottle from the fridge, using tongs to lift bread from the toaster onto a plate and carrying a tray of food from the kitchen to the dining table.

Source: Popular Science: A Bonnier Corporation Company

Figure 12.31 This wheelchair contains two robotic arms programmed to help users perform everyday tasks such as cooking, dressing and shopping.

SKILL BUILDER

Skill builder resource
Robotic attraction

Robotic attraction

For this Skill builder you will require a programmable robot with an ultrasonic sensor. You will develop a simple program to make the robot respond to the closeness of your hand.

When your hand is closer than 15 cm it is to move away. Between 15 and 30 cm, it is to move closer. Any greater than 30 cm and it is to stop.

Converting an algorithm to working code

Computational thinking

Mechatronics and robotics theory is very different to real-life. As you progress in robotics you will discover all sorts of unexpected physical and coding surprises which you will need to solve in order to make your projects work properly.

Here is one example written in structured English. You may at first think the following algorithm if coded correctly would work. Note the `while True` statement is needed to make the code repeatedly loop, as such a while statement is always true!

```
begin
while True:
    if sensor < 15
        move back
    if sensor > 15 and sensor < 30
        move forward
end
```

However, when the distance is greater than 30 cm robot motors will keep moving according to their last instruction as they have never been told to stop! Another problem is there is no instruction telling it what to do if the reading is exactly 15 cm – another logic error!

It is no use adding a stop motor instruction at the end, as the robot will jerk to a stop each time through this repeated loop. Remember also that lines of code execute very, very fast and the motors will not have time to start before they are told to stop again.

Here is a better solution using an `if..else if..else` statement, as the robot will only perform any action when its condition is true.

```
begin
while True:
    if sensor < 15
        move back
    else if sensor ≥ 15 and sensor < 30
        move forward
    else
        stop
end
```

Some coding environments do not offer `if..else if..else` statements. In this case you need to use repeated `if..` statements or nest `if..` statements inside one another.

Systems thinking

Practical problems when building robots

Physical issues can also create unexpected problems. On some robots the ultrasonic sensor may sense the surface in front of it and show unintended readings.

Which robot?

You can use any programmable robot with an ultrasonic sensor in this Skill builder. We used both Spike Prime and a micro:bit-powered Maqueen. (See Table 12.2.)

Design thinking

Designing

Design and build a simple two-wheeled robot with a front-facing ultrasonic sensor. We have used a LEGO Spike Prime (see Figure 12.32). The LEGO Spike app has illustrated step-by-step build instructions for various robots. We suggest the Drive Base 3 is a good start. Add sensors as required.

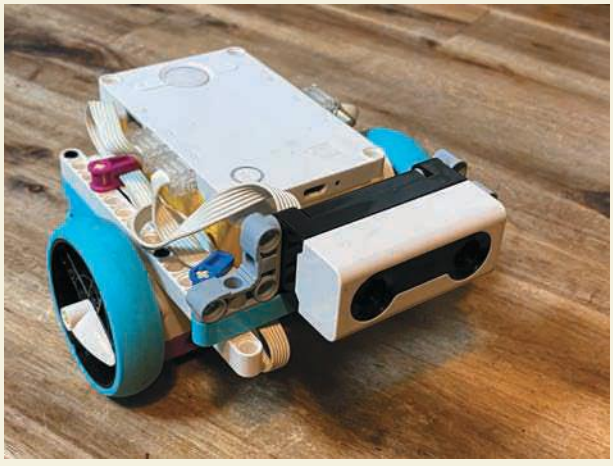


Figure 12.32 A simple build using Spike Prime with two motors and ultrasonic sensor.

DFRobot's Maqueen (see Figure 12.33) is also a great and inexpensive choice, already has an ultrasonic sensor and is coded and controlled by a micro:bit.

Figure 12.33 Here a Maqueen robot moves backwards when an object (such as a hand) is close. A direction arrow has been added to the micro:bit display.

Coding

Visual code

1 To code the Spike Prime, first download the Spike app.

To code the Maqueen, use the MakeCode app and be sure to download the Maqueen extension from MakeCode Extension library before you start coding (see Figure 12.34).

2 Start by creating visual code to display the robot's reading from its ultrasonic sensor, download it to the robot and check it is reading accurately. You may wish to fine tune the cut-off values suggested above.

Source: MakeCode

Figure 12.34 To code the Maqueen using the MakeCode app, download its extension shown here.

3 Create visual code to implement the above algorithm. Solutions for both the Prime and Maqueen are provided in the Nelson MindTap resources.

4 Try to improve your code by adding display direction feedback and ability to press a button to view distances.

General purpose code

If you have learned Python, use the Spike Prime app (V3.4 onwards) or Spike web app, selecting Python to develop your code.

For the Maqueen, convert your visual code to a form of Python using the Python tab at top of window.

Solutions are provided in the Nelson MindTap resources.

Weblinks
LEGO Education SPIKE App

MakeCode for micro:bit

SKILL BUILDER

A gesture-controlled robot

Skill builder resource
A gesture-controlled robot

If you have a Maqueen robot or other micro:bit controlled robot, such as the BitBot, develop code to control the robot using tilt gestures from a second micro:bit using its built-in radio. Tilting the micro:bit to the left or right will turn the robot and tilting forward or back will control direction. Speed can be changed using the A and B buttons.

Basic visual code for the transmitting micro:bit is provided in Figure 12.35. The receiving code can be added to this same program and loaded on both micro:bits as their functions do not overlap. Use the Python tab to view your code in general purpose code.

A complete solution is provided on Nelson MindTap.

Suggested challenges for improving the robot

- If you have a V2 micro:bit, add sound effects and use the touch logo to enable a 'supercharge' burst.
- Avoid collisions by adding an ultrasonic sensor and reversing the robot for a given number of seconds when it encounters an obstacle.

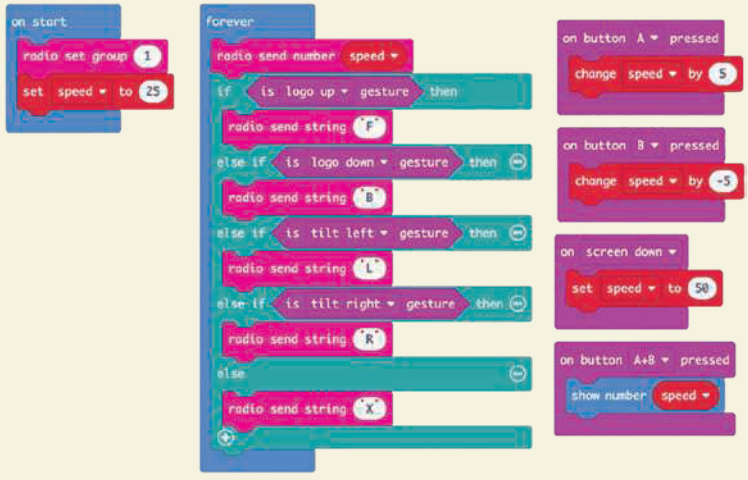


Figure 12.35 Transmitting visual code for a remote micro:bit-controlled robot.

ACTIVITIES

Identify

1. List three different areas of exploration in which robotic technologies are used.
2. Why are robots suited to space exploration?
3. Identify the type of creature that has been the inspiration for underwater exploration robot design.
4. Who is considered to be 'the father' of robotics?

Analyse

5. 'Scientists would be better off spending valuable funds on developing robots to help the disabled, rather than on exploring live volcanoes.' Do you agree or disagree with this statement? Justify your answer.
6. Discuss the ethical issues that arise from the introduction of computer-controlled systems to replace human labour.

Extend

7. Research some of the changes to the nature of work that began with the introduction of the first automated assembly line.
8. Research how advances in robotic and computer technology have enabled surgeons to perform operations remotely on their patients who may not even be in the same country.

12.6 Robot movement

Like other computer systems, robotic systems are composed of hardware and software. Movement is an important function of many robots and it is the hardware *and* software that facilitate this process.

Degrees of freedom

Robotic motion is described in terms of **degrees of freedom**. One degree of freedom is equal to one movement either back and forth (linear movement) or around in a circle (rotational). To understand how this works, consider the human arm. The human arm has six degrees of freedom (Figure 12.36), allowing the following movements:

- shoulder – up and down (one degree)
- shoulder – forwards and backwards (one degree)
- elbow – up and down (one degree)
- wrist – rotate (one degree)
- wrist – up and down (one degree)
- wrist – left and right (one degree).

Additionally, the operation of thumbs and fingers gives several more degrees of freedom.

In a robotic system, usually one robotic joint corresponds to one degree of freedom, but this depends on how the hardware is connected. Most jointed robots have six degrees of freedom that are similar to the human arm; an articulated arm type, for example, usually has six degrees of freedom (Figure 12.37). Six degrees of freedom is usually sufficient to conduct most simple tasks.

So, in a robot with an articulated arm, the following separate movements are possible:

- arm – up and down
- arm – lengthen and shorten
- body – rotate
- wrist – up and down
- wrist – rotate
- wrist – left and right.

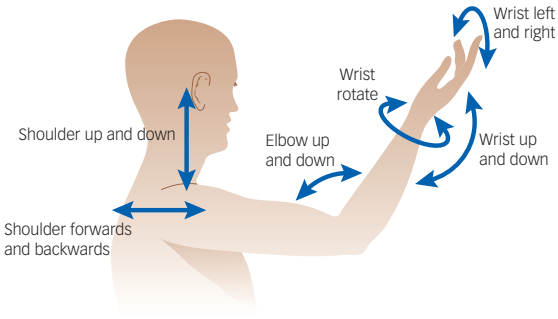


Figure 12.36 Degrees of freedom in the human arm.

Robots move in three ways: **pick-and-place**, **point-to-point** and **contouring**.

- *Pick-and-place:* This movement has only two positions per axis and suits loading and assembly functions in mass-production situations.
- *Point-to-point:* This movement is controlled from one point location in space to another point location. Each point is programmed into the robot's control memory and then played back during the work cycle. This type of movement is suited to tasks such as spot welding.
- *Contouring:* This movement allows the robot to follow a path that forms the shape of a smooth curve. The robot's gripper carries out a programmable path movement and performs variable functions on this path. Applications include spray-painting, continuous welding and grasping objects moving along a conveyor.

Systems thinking

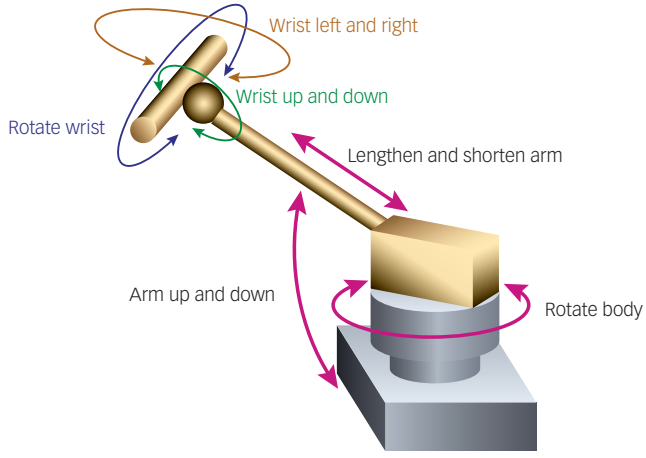


Figure 12.37 Degrees of freedom in a robotic arm.

Parts of a robot

The hardware used to control robotic systems can also be likened to human body parts. The human body can be considered to have five main physical components: body, muscle system, sensory system, power source and brain system. The physical components of a robot resemble that of a human (see Figure 12.38).

Robot sensors receive signals from the environment and pass them to the **controller**. In humans, our eyes act as sensors, collecting signals from our environment and passing these signals to our brain. The robot controller processes signals from sensors into output signals that enable movement or some other event to take place. Our brain processes signals

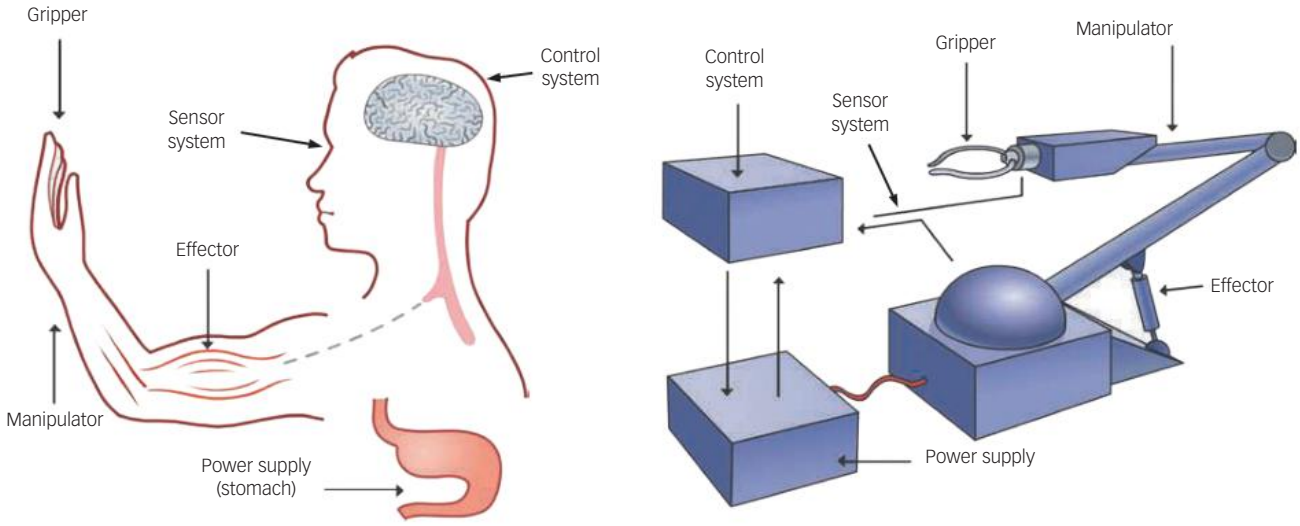
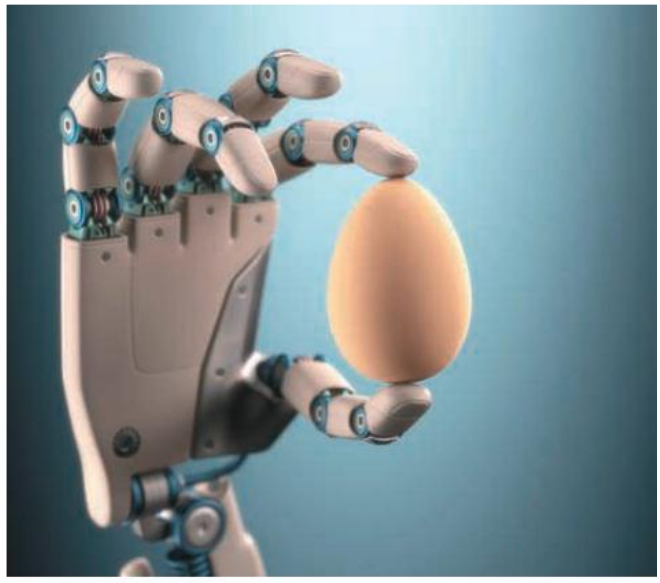


Figure 12.38 The human body and computer-controlled systems have similar components.

received from our eyes and sends messages to drive particular body parts. For example, when we see a step in front of us, the brain sends a message to lift the leg to climb the step.

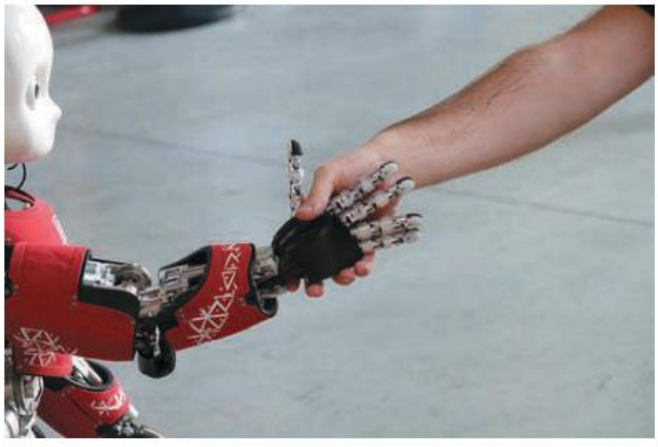
In a robot, these output signals are sent to actuators to tell them to carry out some action, such as switching a motor on or off, opening or closing a valve, moving an arm to a specified point or avoiding bumping into an object. Actuators are output devices, such as motors and end effectors, which convert energy and signals to motion.

End effectors are actuators or output devices for robots and are connected to the end of a robot arm. Different types of end effectors exist, and all perform a physical task. End effectors may be tools, vacuum grippers or magnetic grippers; all of which possess a specific set of commands (see Figures 12.39 and 12.40).



Alamy Stock Photo/Kiyoshi Takahase Segundo

Figure 12.40 A wide variety of end effectors can be fitted to a robot arm. Some end effectors are designed to imitate a human hand's sense of touch. They use acoustic and electrical resistance sensors to control the actions of the effectors.



Alamy Stock Photo/Michele D'Ottavio

Figure 12.39 A robot with a humanoid end effector.

Table 12.1 The physical components of a human body compared to the components of a robotic system.

Components of the human body system	Components of a robotic system	Functions of the robotic components
Body structure	Physical structure – gears, screwdrivers and levers (hardware)	Enables movable body parts
Muscle system to move the body structure	Actuators (motors) – electric motors, solenoids, pneumatic or hydraulic cylinders	Converts energy and signals to actions to move the body parts
Sensory system that receives information about the body and the surrounding environment: sight, touch	Sensor system – either direct contact: touch sensors, strain gauges or pressure sensors; or non-contact: infrared, light, sound, radar, sonar, laser beam or visual	Receives information about the robot's environment
Energy supplied by food and air to activate the muscles and sensory organs	Power supplied by mains electricity or batteries	Provides power to activate the actuators and sensors
Brain and nervous system that processes sensory information and tells the muscles what to do	Controller (brain) – microprocessor(s) and software or wired into the hardware	Processes information from the sensors and sends appropriate commands to the actuators

12.7 Software programs

All robotic systems use software to instruct the hardware on how to perform tasks. As a rule, industrial robots perform quite specific tasks. The software that drives the hardware is often a specialised control language. Other robots are designed to perform more general tasks, such as the LEGO robots constructed for home and educational use. These general-purpose robots are instructed by more general-purpose programming languages.

Programming languages may use different techniques to control robots in performing tasks. They fall into two categories: online programming and offline programming. These terms do not refer to the Internet.

Online programming: involves 'teaching' the robot the path it must take. This is done by manually moving the robot arm through the required motions. The path is recorded and a program is generated.

Offline programming: requires programming code to control the robot's motions before the robot is put to action.

Thinkstock/Denis Tabler

Figure 12.41 A magnetic gripper in action.

KNOWLEDGE BUILDER

Line follower

Here you program a robot to follow a line using just one sensor. (You will use these skills later in this chapter's maze – solving the robot project.) If a robot has two sensors, these can 'straddle' a line and code can easily be written so it can follow the line accurately. With one sensor we have to be clever ...

Basic single sensor edge-following robot

Computational thinking

If a human is asked to follow the edge of a line they can see the direction ahead and which side of a line they are on.

A robot IR light sensor pointing at a line receives only a single item of data: the amount of light reflected. The robot does not know which side of the line it is on. If it is programmed to turn in the correct direction when straying (because the sensor 'sees' the reflected light increase or decrease) it can be programmed to follow the edge of the line, using a zigzagging forward movement (see Figure 12.42). This is sometimes called the binary method of line following.

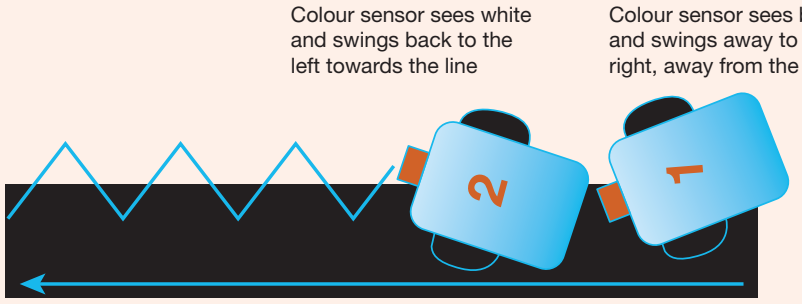


Figure 12.42 With only one sensor, a robot can follow a line simply by following its edge.

Suppose a robot is on the right edge of a line (see Figure 12.42). Presuming your robot has only one sensor, write a simple algorithm in pseudocode to swing it to the right (away from the edge) when the sensor is over the line and to swing it to the left towards the line when the sensor sees white. Try this before reading on.

The edge-following algorithm

When its sensor is over the black line, its reading drops below halfway (called the 'threshold value') and when over the white background, its reading rises above this threshold value.

Following is the algorithm written in structured English. The robot will zigzag forward:

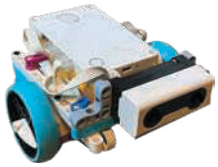
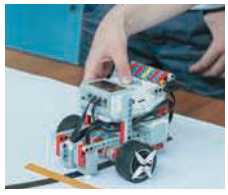
```
WHILE True
    IF sensor < threshold light value
    THEN
        Motor B OFF (or in slow reverse)
        Motor C ON
    ELSE
        Motor B ON
        Motor C OFF (or in slow reverse)

    ENDIF
ENDWHILE
```

We will later use this principle to write code for a robot to solve a maze.

Apart from LEGO's Spike Prime and EV3 robots, there are many cheaper robots in kit form you can discover online which you can build and program using visual or Python code, controlled using either a micro:bit or Arduino board.

Table 12.2 shows suitable robots we have tested.

Table 12.2 Suitable robots for both line-following and the maze project in this chapter.

Robot		Coding (free)		Comments
		Visual *Note: MakeCode extensions installed using Advanced > Extensions*	General purpose	
 Shutterstock.com/AlesiaKan	LEGO Spike Prime	Spike app	Spike app	• one app for visual code and Python • versatile • widely used
 Shutterstock.com/AlesiaKan	LEGO EV3 + colour sensor	EV3 Classroom MakeCode for EV3 Mindstorms (visual)	LEGO EV3 MicroPython	• no longer sold but • widely used • many schools have them
Shutterstock.com/MIA Studio	Maqueen	MakeCode (add the *Maqueen* extension – see text)	MicroPython using Mind+	• needs micro:bit • under $100
Alamy Stock Photo/Bartlomiej Magierowski	Build your own + Arduino + motor shield		Arduino	• needs Arduino • under $100

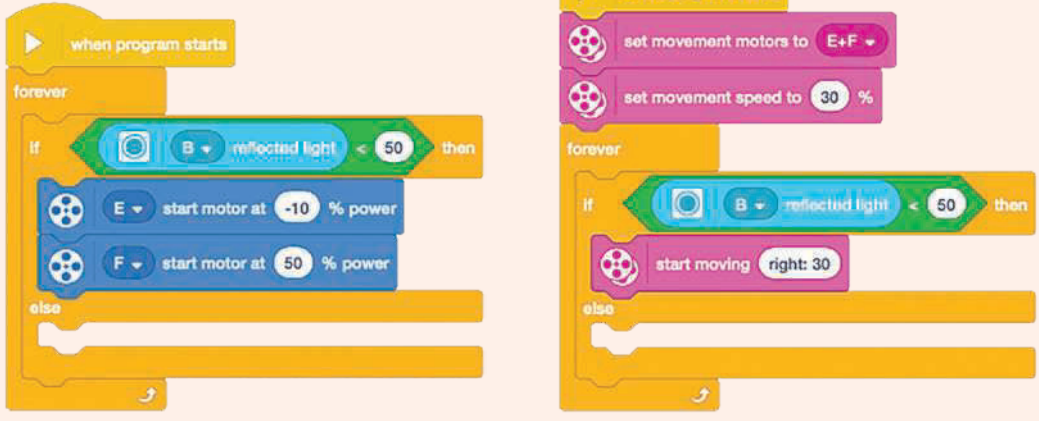
Weblinks
LEGO Education SPIKE App

MakeCode Mindstorms

Coding your robot

1. Select your robot. Refer to Table 12.2. Links are provided in the table. (If you would like to use an Arduino-based robot, instructions for building and coding are provided in the maze project at the end of the chapter.)
2. Use a white desktop or a large sheet of card and apply electrician's tape to create a track.
3. Use visual code to program your robot to perform edge-following. Figure 12.43 shows the start of the Spike Prime visual code (see Spike app weblink). This code presumes your right motor is connected to port F and left motor to port E. The blue motor blocks are in the 'More motors' additional library (you may need to add it). Add the final blocks missing from Figure 12.43.
4. Test and fine-tune your robot to follow a line edge.
5. If you have an EV3 or Arduino, recode using a general-purpose language such as MicroPython or Arduino.

Computational thinking



12.43 Two (incomplete) methods of line following on Spike Prime. (Left) Using the motors in the 'More motors' additional library; (right) using the blocks in the movement library.

SKILL BUILDER

RoboOlympics

Your task, in a group of 2–3, is to devise an event in which robots compete. Design, build and program a robot that successfully completes your event. Develop a scoring system for judging.

Alamy Stock Photo/PJF Military Collection

Figure 12.44 Two suggestions for RoboOlympic events.

The teacher and/or class then select events for a RoboOlympics competition (see also Figure 12.44).

Suggested challenges:

- Reach the top of an inclined slope. The steeper the slope, the more points.
- Speed race.
- Two robots play tug-o-war, or two robots push each other out of a circle (as in sumo wrestling).
- Follow a curved line with time limit.
- Continually move within an open rectangle or circle for one minute without leaving it or push cans out.
- Follow and escape a maze.
- Recognise different coloured objects.
- Move a ball into a goal.

RoboOlympic rules:

- Teams must select three events but may not participate in their own.
- A robot may progress to the next event only after completing each to the satisfaction of the team who designed it.
- Events may be completed in any order.
- A robot may have a maximum of three attempts for each event.
- You may have one trial run before being judged.
- A robot may be redesigned and reprogrammed before each event.
- Judges will award points out of 10 for each result – the best overall team score wins the Robo Olympics!

Game instructions:

1. Form a group (2–3) and devise an event. Decide roles (designers, builders, programmers, journal writers and testers). Your group establishes rules and scoring for your own event, builds a robot and proves it can be completed successfully.
2. Each group describes their event to the class, how it will be judged and scored. The teacher or the class selects four.
3. Build and program your group's robot to compete in your first event.
4. Judges from each group provide performance feedback.
5. Your group uses a collaborative tool (such as Google Docs) to compile a report on performances in each event, what you would do differently and the scores achieved.

CASE STUDY

Robot developer, Hugo Elias

When Hugo Elias was eight, rather than make him set the table for dinner, his dad would teach him how to wire a circuit with a knife and fork. By the time he was 14, Hugo had joined the local robot club, which met every Wednesday evening to tinker with robots and discuss the group's grand visions for the future. The group went on to become the Shadow Robot Company, which now makes everything from a pigeon-feeding robot to a robotic hand claimed to be as dexterous as a human one.

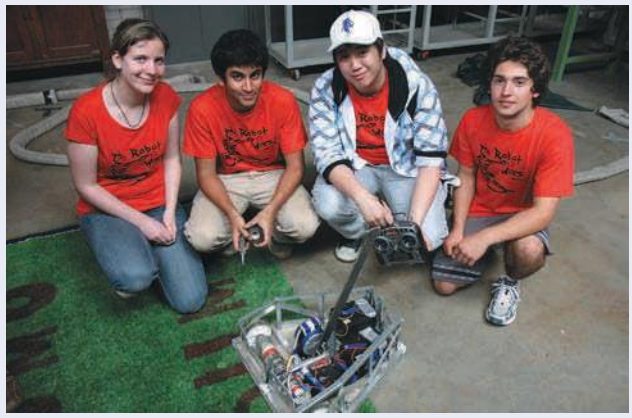
For Hugo, a typical day includes designing mechanical parts on a computer-aided design program, testing prototypes, writing software, wiring electronics and researching new ideas. He has a degree but believes this is one job where your academic qualifications are less important than practical experience.

'My degree didn't qualify me to do my job. Practically everything can be self-taught or learned on the job,' he says.

The best piece of advice Hugo can give budding robot developers is to build a portfolio of projects. 'I'm interested in seeing what projects people have worked on in their own time – competitions, and projects they have done for themselves,' he says.

For Hugo, building his portfolio meant taking part in competitions such as Robot Wars (Figure 12.45). Although each project takes a long time to complete, and can be set back by small mistakes, Hugo still regards his career as 'play' rather than work. 'If I won the lottery, I would still be doing this,' he says.

© 2010 New Scientist Ltd. All rights reserved. Distributed by Tribune Content Agency.



Getty Images/Andrew Hill

Figure 12.45 According to Hugo Elias, taking part in contests like Robot Wars is one of the best ways to launch your career as a robot developer.

Questions

1. Identify the tasks carried out by Hugo in a typical day.
2. To be an effective robot developer, Hugo states that 'academic qualifications are less important than practical experience'. Why do you think Hugo holds this belief?
3. Investigate the Robot Wars competition mentioned in the case study. Explain the purpose of the competition and describe the robots used by participants.

ACTIVITIES

Identify

1. Define the term 'degrees of freedom' as it applies to robotics.
2. Identify the number of degrees of freedom possessed by an articulated arm.
3. List three types of end effectors.
4. Name two ways that robots may be programmed.
5. List the main components of a robotic system and describe the function of each.

Analyse

6. How many degrees of freedom are in a human elbow joint? Describe these movements and draw a diagram to illustrate them.
7. Describe the difference between pick-and-place movements and point-to-point movements.

Extend

8. Research one type of end effector used by robots. In your research, include:
   - a. the name of the end effector
   - b. a picture or diagram
   - c. an explanation of the task it performs
   - d. a poster or digital presentation on the findings of your research.

12.8 Ethical issues in robotics

Machines outsmarting humans?

Consider these creations:

- a robot that can open doors and find electrical outlets to recharge itself
- computer viruses no one can stop
- predator drones that can drop bombs autonomously.

Impressed and alarmed by advances in artificial intelligence, a group of computer scientists is debating whether there should be limits on research that might lead to loss of human control over automated systems that carry a growing share of society's workload, from waging war to chatting with customers on the phone.

Their concern is that further advances could create profound social disruptions and even have dangerous consequences. As examples, the scientists pointed to a number of technologies as diverse as experimental medical systems that interact with patients to simulate empathy and computer viruses that defy extermination and could thus be said to have reached a 'cockroach' stage of machine intelligence.

While the computer scientists agreed that we are a long way from HAL (the computer that took over the spaceship in Stanley Kubrick's famous movie *2001: A Space Odyssey*), they said there was real concern that technological progress would transform the workforce by destroying a widening range of jobs, as well as force people to learn to live with machines that increasingly copy human behaviours.

The researchers dismissed the possibility of highly centralised super intelligences and the idea that intelligence might spring spontaneously from the Internet. But they agreed that robots that can kill autonomously raise important questions about military personnel in a distant control room being sheltered from the consequences of their actions.

There are other concerns too. What could a criminal do with a speech synthesis system that could mimic humans? What might be the implications if AI was used to mine personal information from smartphones?

Science Photo Library/US AIR FORCE

Figure 12.46 Predator drones, such as this one photographed in Afghanistan, still need a human hand to work – at least for now.

Chapter review quiz

CASE STUDY

OpenAI

OpenAI has trained AI language models using billions of pieces of data to such a level that it is now good at understanding and generating text when given a prompt. The prompt 'Suggest three names for a koala that is a superhero' delivered '*Super Koala*, *Captain Cuddles*, *The Mighty Marsupial*'.

OpenAI can even generate accurate lines of computer code in Python and other languages when given a simple task.

ChatGPT

ChatGPT is AI driven software powered by *OpenAI* that uses machine learning to respond to questions asked by users (Figure 12.47).

Shutterstock.com/FAMILY STOCK

Figure 12.47 *ChatGPT* can generate text which is convincingly human.

It can generate impressive responses to any fact-based query. Educators have expressed concern that complete essays on any topic can be passed off as the work of students, compromising any essay work required to be done at home. Many school systems around the world have blocked its use within the school.

Robot engineers are working on robots that will integrate such technology into their robots.

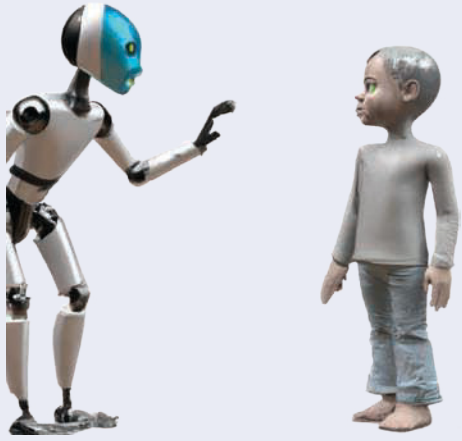


Figure 12.48 This image was generated with AI by Dall·E using only the description 'A robot showing its superiority over a human'.

Dall·E

DALL·E 2 is an *OpenAI* powered system that can create realistic images and art from any description provided in natural language (Figure 12.48). You can try this out yourself using its online website. Can you imagine ways this type of technology might be used in robotics?

Questions

1. Identify concerns researchers have raised in relation to advances in AI.
2. What legitimate concerns can you think of related to technological progress and the workforce?
3. Discuss the threat that AI systems such as self-driving cars, software-based personal assistants and robots pose to human jobs.
4. Do you think automated systems are benefiting humans and improving society in general, or are they doing the opposite?
5. Research the origins and goals of the OpenAI Inc. company online.

ACTIVITIES

Identify

1. Outline the difference between manual devices and automated systems.
2. Using an example, outline the purpose of an automated control system.
3. Identify three types of automated control systems.

Analyse

4. Describe the advantages and disadvantages of each type of automated control system.
5. Identify four periods in the development of automated systems.

Extend

6. Find one example (different from those mentioned in the text) of each type of control system – batch, continuous and discrete – and, for each example, discuss why you think this type of system is used.
7. Investigate and present a report on how modern homes are being equipped with a variety of technological tools that automate tasks.

Weblinks:
- Ladybird: part 2 - data and processing
- Robonaut
- LEGO Education SPIKE App
- MakeCode for micro:bit
- MakeCode Mindstorms
- Python
- Replit
- Maze generator

Skill builder resource:
- Robotic attraction
- A gesture-controlled robot

Project resource:
- Build a maze-solving robot

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

12.9 Project 1: Build a rescue robot

Your task

Rescue robots are autonomous or remotely operated devices designed to assist in emergency and disaster response scenarios. They can be used in hazardous environments where it is unsafe or difficult for human rescuers, such as in the aftermath of earthquakes, building collapses, or industrial accidents.

Many challenges in robotics involve building autonomous vehicles that can find efficient or safe paths around obstacles. These scenarios include autonomous cars, bomb disposal robots, warehouse management robots, pipe inspection robots, agbots (robots that till, weed and harvest crops) and home vacuuming robots. Sometimes such robots must avoid danger and at other times are relied on to remove it.

Your task is to build a robot which will follow a line (Figure 12.49) until it encounters an obstacle. It must avoid the obstacle and return to locate the line. When it encounters a red line, it must stop and locate a second obstacle but this time it must push it out of the way. Similar challenges to those outlined are often part of robot competitions such as RoboCup.

Materials required

You will need:

- A programmable robot
- Ultrasonic sensor
- A colour sensor
- Large white sheets of paper or card
- Electrical tape and/or coloured felt-tip pens

Identifying and defining

Selecting and preparing your robot

Your choice of coding environment will be influenced by your choice of robot. We suggest one of the following:

- LEGO Spike Prime
- LEGO EV3
- a micro:bit-controlled robot, such as the Maqueen
- build and code your own Arduino robot.

Select the robotic environment you will use and build your robot. It must have a front mounted ultrasonic sensor and a colour sensor underneath.

Computational thinking

Select your coding approach

You will commence by developing an algorithm for this problem and representing this using a flowchart (see Figure 12.51). Your choice of coding language may be determined by the robot you select. We suggest you create a working program using visual code and then code using a general purpose language such as Python. Once your code is working, some coding environments allow you to switch between visual and general languages.

Researching and planning

Create a line-following surface

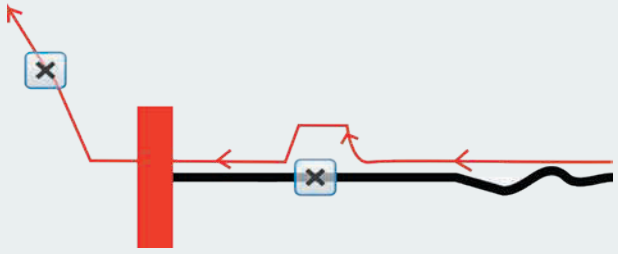


Figure 12.49 The robot follows the red lined path shown, avoids the first obstacle, then after crossing the red line, locates and removes the second obstacle.

Using Figure 12.50 as a guide, use black electrical tape to create your path using two large joined pieces of white paper or light, white card.

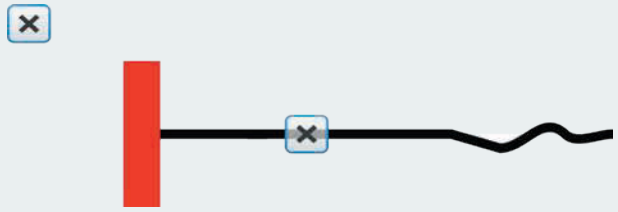


Figure 12.50 A suitable line design. The line should be drawn longer and include more turns.

Plan your algorithm

Draw a flowchart to represent the algorithm you plan to use.

This project needs three functions to be written: line following, obstacle avoidance and obstacle removal. We have split our flowchart into a main program which calls these three separate functions. Here we will give you a start by providing a main program flowchart in Figure 12.51. Your task is to develop code for the three functions yourself using the suggestions that follow.

9780170442787

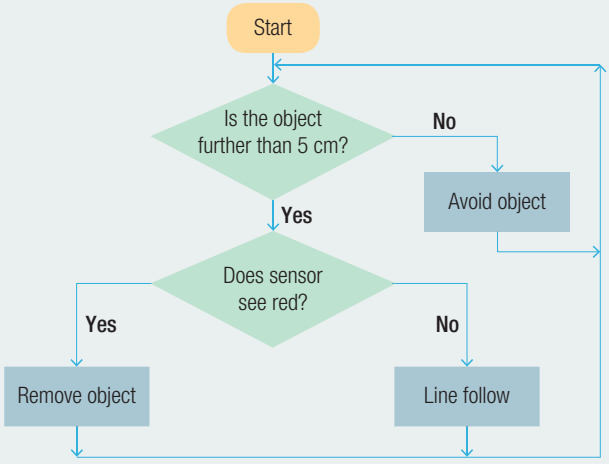


Figure 12.51 Flowchart for the main program code. As this algorithm never ends, we have omitted the termination end symbol.

Examine the main program

The logic used in the main algorithm shown in Figure 12.52 is explained here. If the ultrasonic sensor encounters an obstruction along the line it is following it must avoid the obstacle. Otherwise it keeps following the line unless its sensor 'sees' red and in this case it must locate an object in front of it and push it out of the way.

This main program will be in a forever repeating loop.

Producing and implementing

Define three functions

Use your coding environment to set up three required functions: line follow, avoid and remove. You will now code each of these in turn.

Develop a *Line follow* function

Develop and test code which causes the robot to follow the left edge of the black line. As you discovered in the Skill builder, the algorithm for a single sensor left edge line following is very simple:

```
WHILE True
    IF sensor < threshold light value
    THEN
        turn right
    ELSE
        turn left
    ENDIF
ENDWHILE
```

Computational thinking

Develop an *Avoid object* function

Develop and test code which moves the robot as follows:

- When the ultrasonic sensor gets within, say, 5 cm of an object, the robot stops
- It then turns right and moves ahead a short distance
- It then turns left and moves ahead a short distance
- It then turns left again and moves ahead a short distance
- It then turns right to meet the line again

Develop a *Remove object* function

Develop and test code that moves the robot as follows:

- After sensing red, the robot stops for one second then moves ahead a short distance
- It then keeps turning on the spot a few degrees each time until its sensor locates an object
- It then moves towards this object and pushes it

Testing and evaluating

Developing and testing your program

- Write your main program using the flowchart as a guide. Figure 12.52 provides a start using visual code
- You may need to define the ports your motors are on
- You may need to set an initial motor speed

General purpose coding

If you have successfully got the robot working using visual code, your task is now to code in a general-purpose language such as Python. Python coding is supported in the Spike app from V3.3.1 onwards.

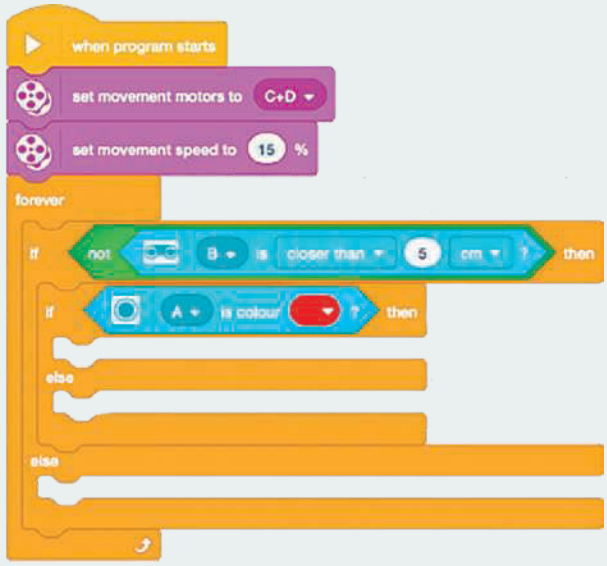


Figure 12.52 Main program visual code for Spike Prime. Your main program needs to call three functions, *line follow*, *remove* and *avoid*, which are missing from the three spaces shown. You also need to code each of these functions.

12.10 Project 2: Build a maze-solving robot

Project resource Build a maze-solving robot

Your task

This project has three stages. First you use a physical robot to successfully navigate a maze, followed by creating a virtual robot to solve a more complex maze. Finally, an optional stage where you develop an algorithm which calculates the shortest escape route from any maze!

In summary, no matter which robot you choose to use, the three stages in this project are:

- Stage 1: Build and code a physical robot to solve a maze using the 'left hand rule'.
- Stage 2: Program an on-screen robot prototype.
- Stage 3 (Optional advanced extension): Code the robot to calculate the shortest path. This is a form of AI (artificial intelligence) and in most cases our robot will find the solution faster than a human.

Identifying and defining

Many challenges in robotics involve building autonomous vehicles that can find efficient or safe paths. These scenarios include autonomous cars, bomb disposal robots, warehouse management robots, pipe inspection robots, agbots (robots that till, weed and harvest crops) and home vacuuming robots.

There are a number of approaches we could take for our project (Table 12.3).

Table 12.3 Approaches for maze solving.

	Maze-solving approach	Advantages and disadvantages
1	A wall-sensing robot in a physical maze with walls of cardboard or Corflute (corrugated plastic sheets)	A real world solution Maze is difficult to assemble Robot requires multiple sensors, is difficult to program and can act unpredictably
2	A physical maze but with 'walls' represented by black lines on a cardboard sheet	Easy to make and set up Robot is an edge-follower – easy to build and program
3	A virtual maze drawn by a computer using a virtual robot	No set-up Algorithm is easily tested independent of a physical robot Not a physical solution

Although you can construct a physical maze from corflute or cardboard, the second approach is more suitable in high school classrooms as physical mazes need to be set up and taken down. The later stage of the project suits a virtual robot, as building a physical robot for path shortening can require very challenging robot tuning.

Researching and planning

Stage 1: Create a maze-solving robot

A simple algorithm for solving any standard maze

A simple algorithm for escaping a simply-connected maze (one in which all walls are connected to its outer boundary) is to always keep one hand in contact with one wall of the maze (your left hand for example). This way you will always reach an opening, if one exists. If ever you visit a connected maze, such as Amaze 'n' Place at Berrima or Puzzling World's tourist attraction in Wanaka, New Zealand, make sure you test this algorithm (Figure 12.53)!

Alamy Stock Photo/Angus McComiskey

Figure 12.53 The Traquair House maze at Scotland's oldest inhabited house.

To understand how the left hand rule works, see Figure 12.54, where a connected maze is reshaped and reimagined as a circle. Trace the journey through the maze as you follow the inside of the circle's circumference. The maze walls can now appear merely as interrupting tentacles.

We see that if we could design a robot that always followed left-hand walls it would eventually escape any connected maze. The great news is you probably have already done this in the Skill builder in this chapter when you built and programmed a single-sensor edge-following robot. (If you skipped this activity, you will need it for the next stage of the project.)

Computational thinking

Edge-following robots use just one IR sensor and successfully navigate any connected maze. Their sensor is aligned to the right edge of a black line (as if it was a left wall). Confirm this works using Figure 12.55 to trace the inside left wall with your finger (even into dead ends!) from start to finish.

There are other more advanced algorithms for solving mazes which you may like to research.

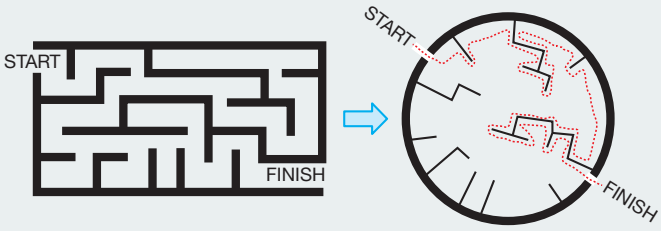


Figure 12.54 By re-shaping the maze into a circle, journeys can be seen as following the circumference with the inner walls of the maze looking like interrupting tentacles.

Create a maze

1. Represent your maze using lines. Figure 12.55 shows a simple maze, which is suitable for testing your robot.
2. Construct the maze on a large sheet of white cardboard using black electrician's tape to represent the walls. Make sure spaces between walls are wider than your robot's turning circle.

Figure 12.55 Suggested maze for a basic line follower using single sensor and two motors. Use electrician's tape on large sheets of cardboard.

Program your robot to solve any maze

3. Now that you know any standard maze can be solved by following the left-hand side of the start wall (line) you can use the edge-following algorithm from the earlier Skill builder. This algorithm is very simple:

```
WHILE True
   IF sensor < threshold light value
   THEN
      Motor B OFF (or slow reverse)
      Motor C ON
   ELSE
      Motor B ON
      Motor C OFF (or slow reverse)
   ENDIF
ENDWHILE
```

Producing and implementing

Select your robot

Your choice of coding environment will be influenced by your choice of robot. We suggest one of the following robots:

- LEGO Spike Prime or LEGO EV3
- a micro:bit-controlled robot, such as the Maqueen
- build and code your own Arduino robot.

Here is a guide for each of these robots.

Using LEGO robots

If you tried the last Knowledge builder in this chapter or the first project building the rescue robot, you will know how to get your LEGO robot to follow the edge of a line. Use this now!

1. Download the code to your robot and test using the maze. Adjust numbers to fine tune the motion.
2. Add light and sound.
3. See how fast you can get your robot moving through the maze.

Using a micro:bit robot

Another approach is to use robots controlled by the micro:bit as introduced earlier in the Skill builder (see Table 12.2). For both visual code and Python, you need the robot manufacturer's extensions to be able to use its sensors and motors.

1. Open MakeCode's web-based app. Load the appropriate extension for your robot (see earlier Skill builder) to allow the micro:bit to connect to your robot's sensors and motors.
   The code in Figure 12.56 uses both the Maqueen and MakeCode apps.
2. Add light effects (you may need to add the Neopixel extension).
3. Change values for speed. Your robot has to be able to negotiate 90-degree right and left turns. To do this, you will need sharp steering and reasonable motor speed.

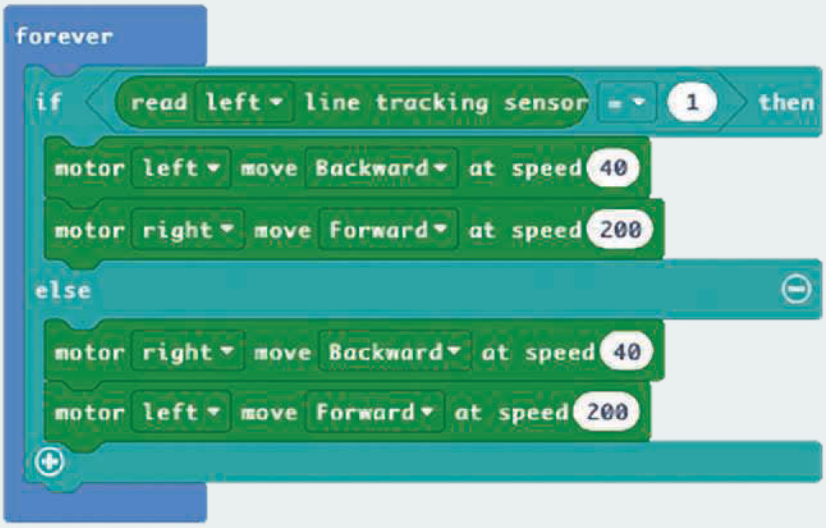


Figure 12.56 shows the visual code for edge line following Maqueen robot.

4 Connect the robot's micro:bit via USB cable and download the file to the micro:bit.
5 Plug the micro:bit into your robot and test the robot in your maze.

Using an Arduino robot

Materials required

Refer also to Figure 12.57.

- Arduino Uno board (or equivalent) and cable
- 2-motor robot chassis kit
- L298N bridge motor driver
- line trace sensor
- male-male (4) and female-male (6) jumper leads
- AA battery packs (6)
- (optional) slide switch
- Arduino IDE installed on computer

Design thinking

Building your robot

1 Assemble the robot chassis following the instructions provided with your chassis kit. The battery holder should be positioned near the swivel wheel to add weight to the rear. The IR sensor is mounted under the front with sensors approximately 10 mm above the floor. The Arduino board and motor shield can be attached top and/or under the chassis using thick double-sided tape.
2 Wire up as in Figure 12.58. Ensure the connections are exactly as shown. All wiring can be done without soldering or using a breadboard by connecting two wires to each of the +5V and GND terminals of the motor shield as shown.

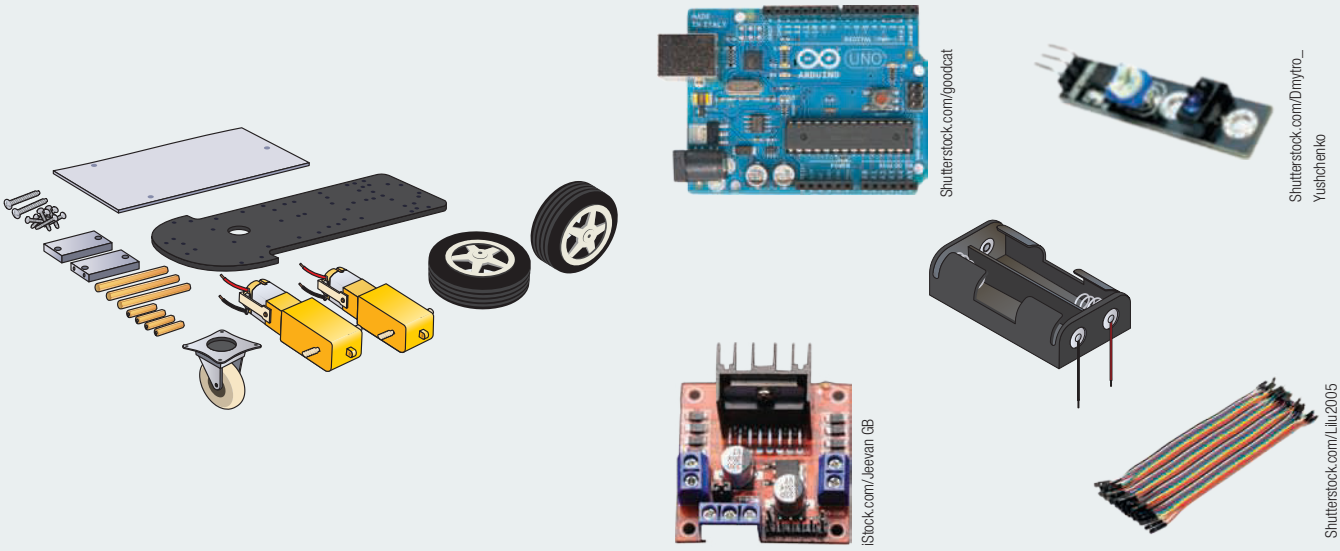


Figure 12.57 Parts needed for building your own Arduino powered robot.

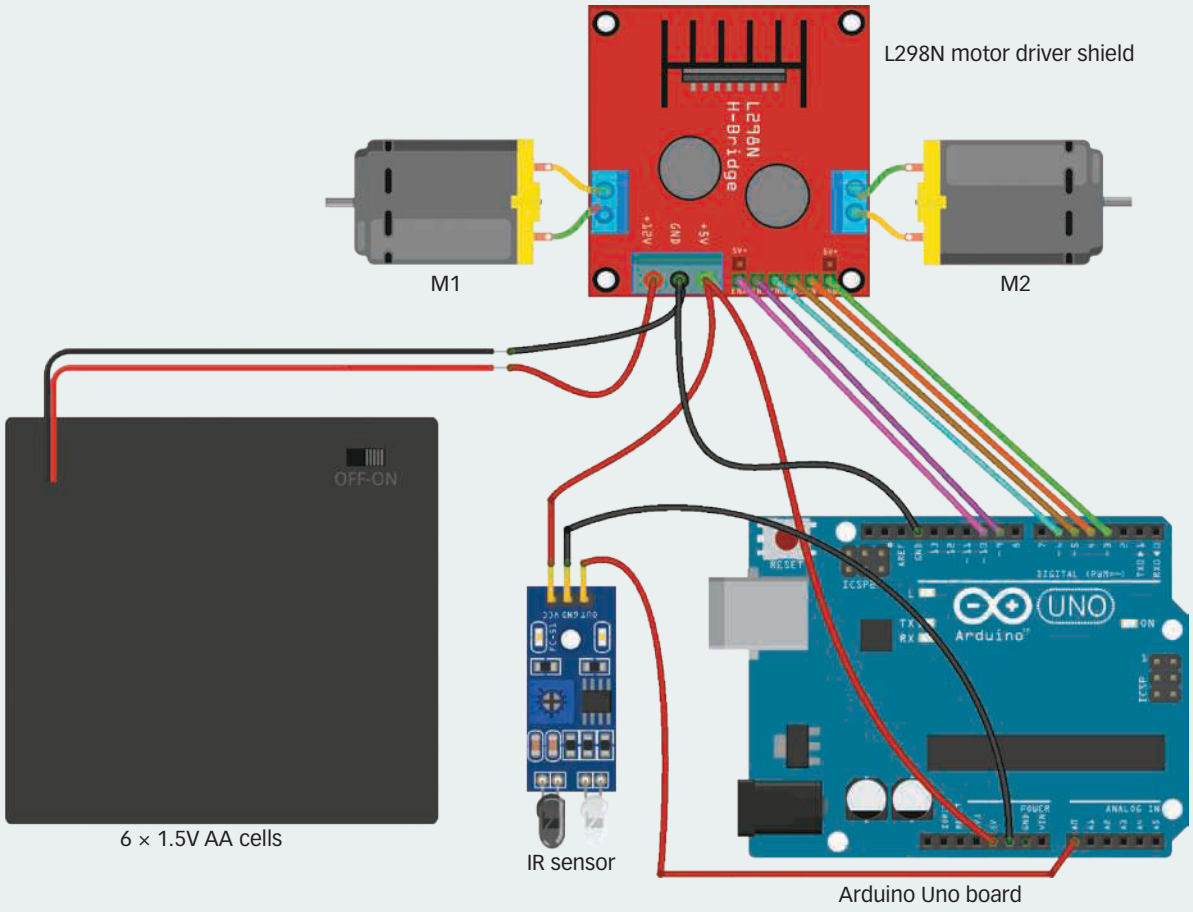


Figure 12.58 Wiring diagram for Arduino robot with IR sensor.

Coding the robot

3 Complete the following Arduino code where indicated.

```
//Please note: pin numbers may differ

// motor 1 pins
int enA = 10;
int in1 = 9;
int in2 = 6;

// motor 2 pins
int enB = 3;
int in3 = 5;
int in4 = 4;

// sensor pin
int sensor_pin = A0;

// initialise variable and constant
int reflection;
int ROB_SPEED = 80;  // set speed

// define function for right turn
void goRight()
{
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  analogWrite(enA, 1.4 * ROB_SPEED);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
  analogWrite(enB, ROB_SPEED);
}
 // define function for left turn
void goLeft()
{
complete this code
}

// set pin modes
void setup() {
  //Serial.begin(9600); //light level test

  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
}
```

```
// main program loop
void loop() {
  reflection = analogRead(sensor_pin);
  //Serial.print("light level = ");
   // use if testing light level
  //Serial.println(sensor_state);
   // use if testing light level

  if (reflection > 300){
    goLeft();
}
  else {
    goRight();
}
}
```

Testing and evaluating

4 Download the code to the Arduino board, debug and test your maze-solving robot. Update your documentation.

Stage 2: Build an on-screen prototype

Identifying and defining

Prototype a maze-solving virtual robot!

In this Stage 2 no physical robot is needed, as we build a working on-screen prototype of our maze-solving robot, coded using Python's Turtle module to act as our robot. This stage is in preparation for path-shortening later on.

Researching and planning

Design the algorithm

A physical robot has sensors attached so there is no need to tell the robot whether walls are ahead, left or right when it meets intersections, as it can sense these. However, an on-screen robot has no sensor and so needs logical rules to determine where walls are based on the direction it is pointing ('heading').

For an edge-following robot, the left-hand maze rule must obey the following rules:

- If there is a wall on the left and no wall in front then the robot travels ahead.
- If there is a wall on the left and a wall in front then the robot turns right.
- If there is a path to the left then the robot turns left.

For an on-screen robot, rules can be expressed in the following pseudocode:

```
IF there is wall at left THEN
   IF there is no wall ahead THEN
      move ahead one space
   ELSE
      turn right 90 degrees
ELSE
   turn left 90 degrees
   move ahead one space
```

1 Check that the bulleted rules are satisfied by this algorithm.

Open your regular Python coding environment (such as IDLE or Replit).

The code you will develop imports Python's built-in graphical Turtle module to simulate the robot's movement and to draw a maze. This module allows a turtle shape to be drawn by adding a graphical interface controlled by Python code. The on-screen turtle simulates a robot.

Producing and implementing

Define variables

We provide the main code and then set interesting challenges for you to code by yourself.

2 First import turtle, time and system modules.

```
import turtle
import time
import sys
```

Initialise variables

Imagine the maze as a grid, where each one unit grid square represents either a wall or a path. We can represent the maze as an array of terms, each representing a separate row. Each of these rows can be made of characters or spaces representing the maze.

We will use:

`+`	to represent walls
`o`	to represent start
`x`	to represent finish
"space"	to represent open path areas

Computational thinking

3 The maze shown in Figure 12.54 would thus be represented using the array shown in Figure 12.59. Check this yourself. We have given this array variable the name `pattern`. Note that each row is enclosed in double quotes and the whole is enclosed in square brackets.

Add this to your code by copying the pattern from the text file *maze ASCII pattern.txt* in the Nelson MindTap resources.

```
pattern =
[
"+++++++++++++++++++++",
"o  +        +      +",
"+   +  +++  +++++++++  +++",
"+      +           +     +",
"++++++  +++++++  +++++  +",
"+         +        +     +",
"+  +++++++++  +++++  +  +",
"+              +     +    +",
"+++  +++  +  +  +  +  +++++",
"+     +     +  +     +     x",
"+++++++++++++++++++++",
]
```

Figure 12.59 The maze shown in Figure 12.54 would be represented using the array shown here.

Weblinks
Python

Replit

9780170442787

4 We will define two separate arrays. One to store squares which are walls and one to store the finish. We set grid size to 25 units.

```
walls = []
finish = []
box = 25
```

5 In Turtle, a *screen* is the graphical space in which the turtle moves. We create it using the `turtle.screen()` method.

```
# the board
board = turtle.Screen()    # screen
board.bgcolor("black")     # background
board.setup(800, 400)      # dimensions
```

6 A 'turtle' is simply a pen for drawing. It can be any shape. We define walls by defining a turtle in the shape of a white square we will call 'maze'. Turtles can either leave a trail or not when they move. Here we do not want a trail when drawing walls but we will 'stamp' these white squares to draw walls.

```
# maze walls
maze = turtle.Turtle()      # define walls
maze.shape("square")        # define shape
maze.color("white")         # define colour
maze.penup()                # no trail
maze.speed(0)       # fastest drawing speed
```

7 We define the robot. We choose a turtle shape and set its heading to 0 degrees (i.e. pointing right).

```
# robot turtle
robot = turtle.Turtle()   # define turtle
robot.shape("turtle")     # define shape
robot.color("red")        # define colour
robot.setheading(0)       # point down
robot.penup()             # no trail
robot.speed(5)            # define speed
```

`drawMaze()` function

8 We define a function to draw the maze. Interpret this code using the `pattern` array above where each term is a collection of ASCII characters representing one horizontal row of our maze. Note the need for double parentheses on two of the lines when appending actual coordinates to a list of existing coordinates (which are already all inside parentheses)! Copy in this function.

```
def drawMaze(pattern):
    for y in range(len(pattern)):         # select next term in array
        for x in range(len(pattern[y])): # select next character in term
            board_x = -300 + (x * box)   # assign starting position
            board_y = 150 - (y * box)    # change to suit size of maze
            if pattern[y][x] == "+":     # stamp a wall, add to wall array
               maze.goto(board_x, board_y)
               maze.stamp()
               walls.append((board_x, board_y))
            if pattern[y][x] == "X":     # store the finish coords
               finish.append((board_x, board_y))
            if pattern[y][x] == "O":     # move robot to start
               robot.goto(board_x, board_y)
```

`direction()` function

9 This function is sent the `heading` parameter (from the main loop). We need it to check where the walls are, as the meanings of *left*, *right* and *ahead* will be changing as the robot rotates. This function also checks for a finish square. Copy this function.

```
def direction(heading):
    x_pos = round(robot.position()[0])
    y_pos = round(robot.position()[1])
    pos = (x_pos, y_pos)
    if pos in finish:            # quit the program
          print("Finished")
          sys.exit()
     if (heading == 270):             # robot points down
          left = (x_pos + box, y_pos)
          right = (x_pos - box, y_pos)
          ahead = (x_pos, y_pos - box)
     elif (heading == 0):             # robot points right
          left = (x_pos, y_pos + box)
          right = (x_pos, y_pos - box)
          ahead = (x_pos + box, y_pos)
     elif (heading == 90):            # robot points up
          left = (x_pos - box, y_pos)
          right = (x_pos + box, y_pos)
          ahead = (x_pos, y_pos + box)
     else:                            # robot points left
          left = (x_pos, y_pos - box)
          right = (x_pos, y_pos + box)
          ahead = (x_pos - box, y_pos)
     return pos, left, right, ahead   # return values for coords
```

We are now ready to use our main algorithm.

`moves()` function

In Python, we can use the values returned by the `direction(heading)` function in one line simply by using the following in a function.

```
pos,left,right,ahead = direction(heading)
```

10 Write the main function based on the algorithm developed earlier. Can you explain its logic?

```
def moves():
  pos,left,right,ahead=direction(heading)

  if left in walls:
    if ahead not in walls:
      robot.forward(box)
    else:
      robot.right(90)
  else:
    robot.left(90)
    robot.forward(box)
```

The main program loop

11 All that is left to do is to call the `drawMaze()` function and write the main program loop to run the program.

```
### main program ###
drawMaze(pattern)       # draw the maze
while True:             # main program loop
  heading = robot.heading()
  moves()
```

Testing and evaluating

Run your code. A complete version is provided in the Nelson MindTap resources.

12 Test and debug your code.

(Advanced challenge) Stage 3: Find the shortest path

It is now time to add a powerful function that will find the shortest path through the maze! This is a form of AI. This clever method for finding the shortest path through a maze works by first using the left-hand rule then eliminating all dead ends.

Computational thinking

We store each robot turn using R for right, L for left and D for dead end using an array. Any paths (not taken) to the right must be recorded (we use 'S') as the robot must later ignore these. Figure 12.60 illustrates a case.

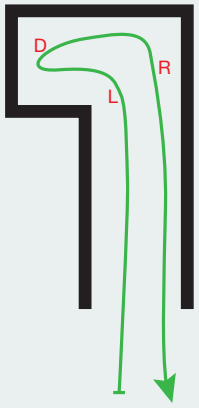


Figure 12.60 The letters LDR record the three turns using the left-hand rule. D stands for 'dead end' or a U-turn. Dead ends such as this will be removed from the maze path.

9780170442787

Examine Figure 12.61 to see how all dead ends can be replaced by a single letter.

The first two patterns and the last are equivalent to simple dead ends, the others can be replaced by a single turn. In this way, all combinations of three turns with a D in the middle can be replaced by a single turn!

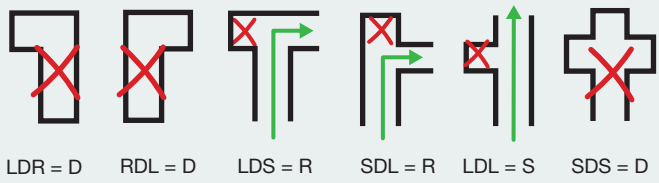


Figure 12.61 Removing all dead ends! To find the shortest path we repeatedly replace three letter combinations whenever there is a D in the middle by an equivalent overall turn. The first two patterns and the last are replaced by simple dead ends, the others can be simplified to a single turn, shown in green.

Design thinking

13 Examine this shortening process for the following simple maze (Figure 12.62). Copy the maze on paper and write out the letters for all turns from start to finish using the left-hand algorithm.

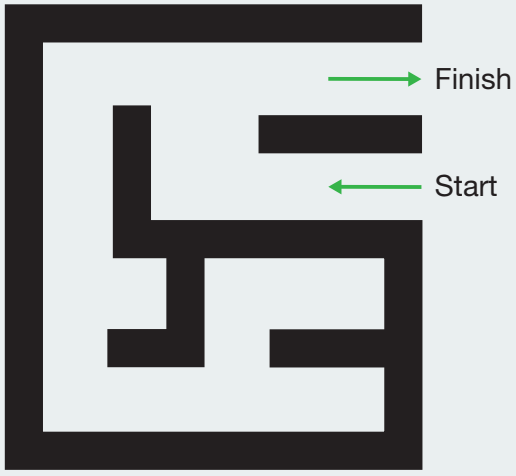


Figure 12.62 The left-hand rule results in 18 turns but the shortest path through this maze is only two right turns.

Check your turns are:

RLLLDLLLRDLLDSRSRS

Using the rules shown in Figure 12.61, confirm the shortest path solution for this maze is just two right turns: RR

14 Write an algorithm, using a flowchart or structured English, for a shortening function. You will need to create two functions. One will record a letter D, S, R or L to designate the type of turn. The second function will perform the shortening of the string of letters created, by looking for patterns whenever 'D' appears.

Then compare your effort with the code provided in the Nelson MindTap resources, *maze_turtle_shortpath.py*.

Examine this code carefully and understand how it works.

Extend this project

15 Modify your code:

- make the robot blue and create a blue trail for the left-hand rule maze run
- change the robot back to red when drawing the trail for the short path calculation
- make walls on the left of the robot go grey as the robot moves past them
- have the robot celebrate escaping the maze at the very end by 'dancing'
- find out how to use a robot (gif) image of your own to replace the turtle.

A working final version is provided as *maze_turtle_shortpath_leftwall* in the Nelson MindTap resources.

16 Use ASCII character mazes to generate your own ASCII maze, using a text editor to match our project's ASCII maze code. Use these settings:

- Type of maze: 'Text'
- Width and Height: 2

17 Check out our example using the above coded with a more complex maze. We added a dance, too! It's impressive just watching it: *complex_maze_turtle_shortpath_dance.py*

(Challenge extension) Path shortening using a physical robot

18 Building and programming a real robot to perform path shortening can be very difficult. You will be able to find many videos online of our robots doing this successfully.

For a physical robot to solve such a maze, you do not need to write code to draw a maze as we did for the on-screen maze (it already exists!) or to find which direction the sensors are facing (they will be attached to a forward-facing robot!).

However, to perform path shortening as we did in the last section, you need at least three IR sensors: one to check left wall, one to check right wall and recognise openings to the right, and one to detect walls in front as they occur. Unlike a virtual robot, a physical one does not normally store memory of sensor data and will not know what type of intersection it was.

Your challenge is to build a suitable robot and adapt the on-screen Python code for it.

Even if you don't tackle this part of the project, can you see there are new problems to solve?

- The robot must stop quickly and not overshoot intersections.
- Turns must be exactly 90 degrees.

Try to anticipate more!

Weblink
Maze generator

13 Creating games

FOCUS AREA: Creating games and simulations

Chapter outcomes

In this chapter, you will learn:

- the history of computer games
- social and ethical issues in gaming
- principles of games design
- tools used in games design
- what a game engine is
- to design and build games.

Key terms

artificial intelligence (AI)
console
core mechanics
functional requirement
game engine
games loop
joystick
massively multiplayer online games (MMOG)
non-functional requirement
SCAMPER
simulation

9780170442787

IN ACTION

The Creator of *Minecraft*

Photo: Alamy Stock Photo/TT News Agency; Background: Alamy Stock Photo/Iurii Vasenko

Figure 13.1 Markus 'Notch' Persson, the creator of *Minecraft*.

The best-selling video game of all time is *Minecraft*, launched officially in 2011. *Minecraft* is a sandbox game where players have unlimited freedom as to what they can create and the goals they select and is set in a blocky, randomly-generated 3D world allowing players to explore, build, and create using a variety of different blocks.

The game operates several different modes, such as survival mode, where players must gather resources and build structures to protect themselves from hostile enemies, and creative mode, where players have unlimited resources and can build without any restrictions.

In 2009, Swedish programmer Markus 'Notch' Persson (Figure 13.1) was a game developer working who in his own time and developing his own games at home. He had been playing another popular game, Dwarf Fortress. In the game, players help dwarf warriors build fortresses in which players control warriors, getting them to perform tasks such as cutting down trees, fishing, mining, or cooking, while fending off marauding monsters. He was also exploring a mining based game called Infiniminer. After a few days, he realised the digging, exploring, and building were better when viewed from an avatar's point of view and decided on adopting a first person view and adding role-playing elements to the game.

When asked how he goes about coding, Persson explained 'I kind of "code-sketch". I get started with a project by actually writing the code for it and getting something up on the screen. Then I play around with it and see if it's any fun, and change the parts that aren't.'

Persson was asked how he first started coding. He said, 'My father bought a Commodore 128 when I was seven years old, and we started subscribing to a computer magazine. It had program listings you could enter into your computer to get a silly little game or a fun effect and things like that. I started entering them, and noticed that they broke or did different things if you changed what you entered. I don't remember exactly how fast this process was, but I know I made my first own program when I was eight years old. It was an extremely basic text adventure game where you had to enter the correct sentences to move on to the next room ("open the door", "kill the ninja", that stuff).'

His girlfriend (and future wife), Elin, became his alpha tester. As each working version was completed, he sent it to her to play. He says her feedback was invaluable. 'Feedback is extremely important to learn what parts of the game people enjoy and in what general direction the player want the game to go.'

Persson's simple advice for those starting out is: 'Just make games for yourself and try to have a critical eye to what you do', he said. 'If you genuinely like the game there will be other people who like it as well.'

In 2014, Microsoft acquired Mojang Studios and Minecraft for $2.5 billion and Persson dropped out of involvement. Since then, the game has continued to be updated and maintained, with new features and content added regularly.

Sources: https://www.gamedeveloper.com/pc/interview-markus-notch-persson-talks-making-i-minecraft-i- and https://www.denofgeek.com/games/an-interview-with-minecraft-creator-markus-notch-persson/

Questions

1. Identify three practices Markus Persson uses in the process of game development.
2. Research what software Persson used when developing the game *Minecraft*.

13.1 The importance of games

In this chapter you will discover the history of computer games. You will identify features of a good game and use app and web software development tools to develop your own games.

The history of computer games will show you how the desire to develop ever more compelling games led to hardware and software advances such as faster graphics processors, more sophisticated software development tools and has refined the skills of software programmers. Finally, your skills will develop further as you build your own portable game in the project at the end of the chapter.

You will learn about games as **simulations**, create a handheld game in this chapter's project and build a virtual pet using a micro:bit in the following chapter. You will also develop an RPG game using OOP (object-oriented programming).

If you wish to develop a web-based game, *Chapter 16 Developing apps for the web* provides a project where you recreate a retro-style snake game which runs on your mobile device or in any browser.

Games are essential to human development. If you have ever watched young animals you will not need convincing that animals learn through play, just as we do. Games prepare our brains to handle the unexpected. Through games, we learn to take risks, develop physical skills and control, learn the importance of fairness ('fair play'!), the consequences of cheating and the ability to interact successfully with others.

Games have found their places in education, recreation, simulation and sports and fitness.

When you complete Chapter 15 you will even create your own collection of menu-driven simple educational games.

Recreation

The human race has been playing recreational games from the earliest times and evidence for this has been discovered in archaeological excavations. Senet is the earliest known board game and is at least 5000 years old (Figure 13.2).

There are computer versions of many well-known games (e.g. the popular mobile app *Words With Friends* is a version of *Scrabble*®) but despite the proliferation of computer games, physical board games such as *Chess, Monopoly*® and *Risk*® remain popular and new board games still sell well. There is a unique pleasure in manipulating objects in a physical game that is not available in a computer game, despite being less mobile.

Brooklyn Museum, Charles Edwin Wilbour Fund, 49.56a-b. Creative Commons-BY

Figure 13.2 Senet is the oldest known board game with fragments of playing boards uncovered in Egypt showing it was played at least over 5000 years ago.

As with physical games, recreational computer games are not only entertaining, they motivate, encourage social interaction and promote exercise. Games on video **consoles** offer dance apps such as *Zumba, Wii Sports, Ring Fit Adventure* and even *Pokemon GO* can get players moving.

Education

Games are such an effective way to learn they are used widely in both education and simulation. The word 'gamification' was coined to describe the use of game concepts to promote engagement in learning. Gamification involves adding such simple elements as point scoring, challenge, competition with others, rewards and incentives to an educational activity.

Games are an important tool for motivation in learning from kindergarten to higher education. Some notable educational titles over the years are: *Where in the World is Carmen San Diego?, Maths Blaster, Minecraft, Civilisation* and *Reader Rabbit*.

Games in the form of simulations are used in pilot and driver training. Simulation games such as *SimCity* and its spin-off *The Sims, Flight Simulator, Farming Simulator* and *Kerbal Space Program* are examples.

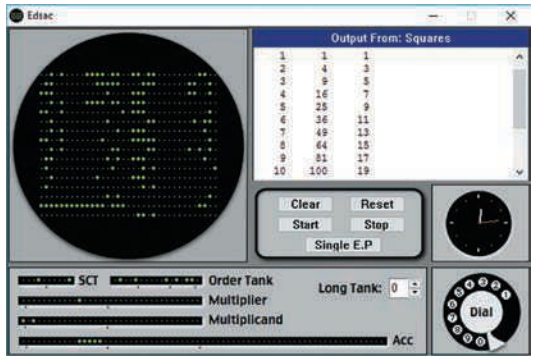
Some games that combine the elements of entertainment, educational and simulation. *Myst* was a ground-breaking example of a realistically simulated world involving the player in a graphic adventure to solve a mystery. It sold many millions and remained the best-selling computer game of all time until overtaken by *The Sims*.

13.2 A brief history of computer games

The earliest mainframe games

The first computer graphical computer game was a noughts-and-crosses game developed by AS Douglas in 1952 for his PhD degree at the University of Cambridge. He used a cathode ray tube to display output and a traditional telephone dial for input.

You can play a simulated version (called *OXO*) on a simulated 1948 EDSAC computer by downloading it via the weblink (Figure 13.3).



Source: Martin Campbell-Kelly

Figure 13.3 Try out OXO, the world's first computer game running on a simulated EDSAC computer.

Games written for the earliest computers in the 1960s and 1970s ran on large mainframe computers. In the absence of today's input devices, moves for simple games such as noughts-and-crosses were input on punched cards and became highly popular interactive science museum attractions. The earliest version of *Pong* appeared in 1958 with its output shown on an oscilloscope screen.

Another early example was *Spacewars* from 1962, which ran on the PDP-1 computer. You can experience a simulation of this game running in the original code (!) using its weblink. One game required users to launch a missile at a distant target by inputting its angle via a punched card. An angle of 0 degrees delivered the message: 'You idiot! You have just destroyed your own position!'

Weblinks
Edsac Simulator

Spacewar emulator

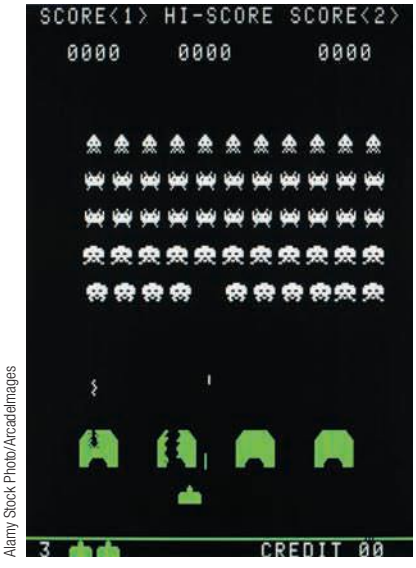
The era of arcade games and home consoles

With the invention of integrated circuits (ICs), the 1970s saw the appearance of video arcade machines in shops and arcades that played just one game. Dedicated consoles were also sold for home use. These were popularised by Atari's games of *Pong* and Taito's *Space Invaders*. We can learn how existing games supplied inspiration for new ideas by studying the history of *Space Invaders*. Its designer was inspired by Atari's already highly popular *Breakout!* game, which involved hitting a ball to demolish a wall of bricks and, by swapping its paddles for a gun and bricks for invading aliens, created a world beating new game (Figure 13.4).

You may be surprised to learn that these games had no CPU (microprocessor chip) as at that time they were too expensive. Instead, multiple ICs were hard-wired together and games ran using hardware rather than embedded computer programs.

Flickr.com/Sean Benham/CC BY-ND 2.0

Wikimedia Commons/Coentor/CC BY-SA 3.0



Alamy Stock Photo/ArcadeImages

Figure 13.4 The designer of *Space Invaders* took Atari's already highly popular *Breakout!* game and swapped paddles for a gun, and bricks for invading aliens.

Games drive innovation

The previous paragraph describes the challenge faced by Steve Wozniak, who later founded the Apple Computer company with Steve Jobs. He worked on the hardware for *Breakout* while both were working at Atari and created the game of *Pong* based on it. Many of the hardware features of Apple's first computer, the Apple I, were driven by Wozniak's desire to show off his game. He wrote:

> *'Basically, all the [hardware] game features were put in just so I could show off the game I was familiar with—Breakout—at the Homebrew Computer Club. It was the most satisfying day of my life [when] I demonstrated Breakout—totally written in BASIC. It seemed like a huge step to me. After designing hardware arcade games, I knew that being able to program them in BASIC was going to change the world.'*

Beginner's All-Purpose Instruction Code (BASIC) was a popular early programming language. The era of the personal computer had begun.

The early 1980s saw arcade games such as *Pac-Man*, *Donkey Kong* (Figure 13.5) and *Space Invaders* became hugely popular on personal computers and arcade games began to disappear. Personal computers such as the Commodore and Amiga were marketed specifically for their gaming potential.

Getty Images/Ibusca

Figure 13.5 Nintendo's *Donkey Kong* in which the character of Mario first appeared.

The Macintosh, the first computer with a GUI, offered a mouse in addition to the keyboard for games input. These early games introduced ideas that have become permanent features of games design, such as players having lives rather than timers and high scores as player incentives.

Programmable home consoles

The need to scale up the manufacture of silicon chips for games companies resulted in microprocessors dropping dramatically in price and in the late 1970s console games could be programmed on ROM cartridges and handheld LCD games (such as Nintendo's popular 1989 Game Boy) became possible. Throughout the 1980s, home consoles sold well with the Nintendo Entertainment System 8-bit home console selling in the millions and introducing plumber Mario to the world.

Consoles featured new ways for gamers to input via gamepads in place of keypads, **joysticks** and paddles and new ways to load games such as cartridges and later, CD-ROMs.

The number of bits a processor can handle determines the graphic detail, sound quality and the speed of a game. Competing manufacturers first launched 16-bit then 32-bit consoles. Sports-themed games became possible.

In the 1990s, consumers acquired their own desktop computers and PC games had access to the greatest graphics power and highest processor speeds. Popular games were sold on CD-ROMS. By the turn of the century, home consoles such as Sony's PlayStation, Sega's Dreamcast and the Nintendo 64 system became popular, followed in 2001 by Microsoft's Xbox and Nintendo's GameCube. A big advance occurred a few years later with Microsoft's Kinect system for the Xbox and the Nintendo Wii, and home gaming became physically interactive: players' positions could now be located in a room and could swing devices to emulate racquets and ball tossing (see Figure 13.6).

Shutterstock.com/robtek

Shutterstock.com/speedimaging

Alamy Stock Photo/John Muggenborg

Figure 13.6 Popular gaming consoles L to R: Playstation 2, Nintendo Wii and the Microsoft's first Xbox.

Online gaming

In the 1990s **massively multiplayer online games (MMOG)** began to be widely used on both personal computers and consoles. These games involve large numbers of players and virtual worlds that continue to exist today. MMOGs encompass many genres but most common are role-playing games, such as *World of Warcraft* with over 10 million subscribers.

The virtual economies of MMOG games often rival those of real-world countries and hundreds of thousands of people earn their main source of income harvesting their virtual resources (coins, gold etc.) for sale to other players in services known as 'power leveling'. The game *Second Life's* first millionaire acquired her wealth from profits made selling the game's virtual goods and established a company to sell these services.

The mobile revolution

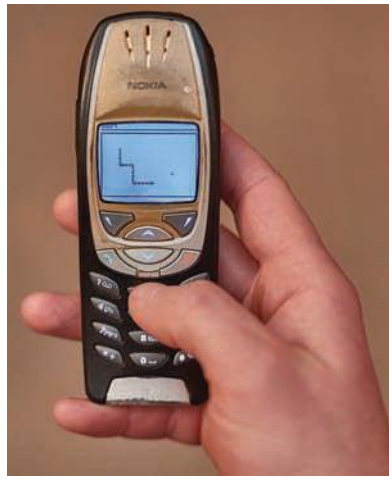
The earliest mobile phones had built-in games, in monochrome and on tiny screens. One of these, *Snake*, was popularised by an early Nokia phone (see Figure 13.7). You have the opportunity to design and build your own mobile web version in Chapter 16!

Mobile phones evolved into smartphones with larger screens and associated app stores, and thousands of developers created hundreds of thousands of handheld games for sale or free. As processing power increased and built-in sensors offered more possibilities for input, games became more immersive. Phone-based games apps are now played online between individuals and across the world with over a million available on the Apple App and Google Play stores.

Virtual reality

For many years it was thought virtual reality (VR) would be the next frontier in gaming. However, it was some years before issues of nausea-inducing latency (the delay between the action and the visual response), graphics power, resolution and price were solved. Meta's Oculus VR headsets, such as Rift and Quest, ushered in interactive 3D gaming with graphic realism to match real life. They introduced stereoscopic vision, stereo sound and positional tracking. You can explore many applications of VR, including those of games, in Chapter 8.

Is there a final frontier for gaming as it reaches for even more realism and player immersion?



Shutterstock.com/Peter Gudella

Figure 13.7 *Snake* playing on a Nokia 3210 from 1999, one of the most popular phones of all time.

Simulation

Although all games can be thought of as a type of simulation, there is a special category that creates real-world scenarios for training, testing or entertainment.

We will study simulation in more detail in the following chapter.

13.3 Computer games and AI

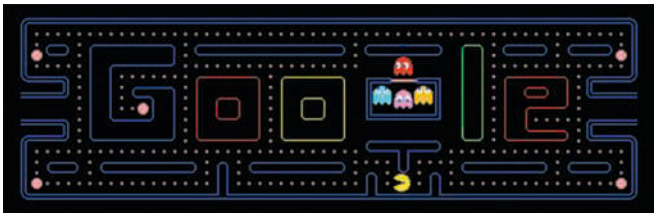
The usual dictionary definition of **artificial intelligence (AI)** is 'the ability of a computer to perform activities that are normally thought to require human intelligence'.

There is at least one problem with this definition. The biggest is the debate about what human intelligence actually is. In the video referenced in question 7 of the activities at the end of this section, Steve Wozniak discusses differences between human and machine intelligence. Should conscience and emotions ('emotional intelligence') be included in a definition of AI? Should we include the ability to adapt to changing circumstances, as humans do? Humans often make unexpected but useful connections leading to creativity and innovation. Can machines do this as effectively?

As technology becomes more sophisticated, many things that were once thought of as AI are no longer considered to be so. One example is text recognition (OCR) which is now not considered an example of 'true AI'.

A solution is to divide AI into two categories: weak and strong. Most computer games use weak AI. 'Weak' doesn't imply less useful – if a game's AI was too strong, it may make it unplayable!

The game *Pac-Man* illustrates this. You can play Google's mini version of *Pac-Man* (Figure 13.8). Use the *Pac-Man* weblink provided here.



Source: PAC-MAN™ & ©1980 BANDAI NAMCO Entertainment Inc via Google

Figure 13.8 The *Pac-Man* doodle from Google.

Each of the four ghosts' movements in the game is defined by a set of algorithms designed to give each a kind of intelligence. One ghost always turns left, another always turns right, one turns in a random direction, and the last turns towards the player. Individually, the movements of the ghosts are easily predicted, and clever players are able to avoid them. However, as a group, the pattern of their movement appears complex. Only one person, Billy Mitchell has ever beaten *Pac-Man*, and when he did it was because his game ran out of memory. See the Billy Mitchell weblink provided.

Designer Toru Iwatani wrote:

> *'To give the game some tension, I wanted the monsters to surround Pac Man at some stage of the game. But I felt it would be too stressful for a human being to be continually surrounded and hunted down. So I created the monsters' invasions to come in waves. They'd attack and then they'd retreat. As time went by they would regroup, attack, and disperse again. It seemed more natural than having constant attack.'*

Like *Pac-Man*, many games use AI to imitate human-like personalities in non-player characters such as laziness, fearfulness, aggression etc. Some games use basic chasing and evading movements, and others may use highly complex algorithms, machine learning and neural networks. Skilful use of AI can make a computer game more immersive.

Weblinks
PAC-MAN Doodle

Meet the Man Who Beat 'Pac-Man'

Deep Blue vs Kasparov: How a computer beat best chess player in the world

Watson and the Jeopardy! Challenge

AlphaGo Official Trailer

Google Labs

Prof. Alan Brown interviews Steve Wozniak

KNOWLEDGE BUILDER

Human intelligence challenges AI

Use the Internet and the links provided to research these famous human intelligence vs AI battles and copy and complete Table 13.1.

1996 and 1997: IBM's Deep Blue computer twice challenged world chess champion, Gary Kasparov

2011: IBM's Watson challenging three champions at the game of Jeopardy

2016: AlphaGo challenged a human professional at the ancient Chinese game of Go.

Table 13.1 Famous human versus AI challenges.

	Which won the game: human or AI?	Summarise the approach taken by the AI algorithm
IBM's Deep Blue		
IBM's Watson		
AlphaGo		

Taking it further: Explore some of Google's AI experiments, such as Quick! Draw! and Semi-Conductor.

ACTIVITIES

Identify

1 Describe two educational games you have used.

2 State some similarities and differences between games and simulations.

3 Identify four major stages in the development of gaming.

Analyse

4 Outline three social benefits of games.

5 Describe the challenges that have slowed the adoption of virtual reality in gaming.

6 Use the weblink provided to view Steve Wozniak, co-founder of Apple and the inventor of *Pong*, in this interview. He has some important things to say about games, computer design, software programming and education. Summarise the main points using bullet points.

Extend

7 (Class discussion) Discuss what form the future of gaming might take. Can you imagine what gaming may be like beyond VR or even holograms?

13.4 Issues in gaming

Legal issues

As with any other digital products, games developers must guard against copyright infringement. Gameplay, music, images as well as the 'look and feel' of a game may have copyright protection.

Copyright

Intellectual property is protected under international law, and by national laws. Large payouts have resulted from successful challenges by companies who believed games had stolen their ideas or assets.

Privacy

Other legal concerns involve issues of privacy. Online games collect huge amounts of data on players. Rightly or wrongly, the personality or preferences of a player may be judged from their game behaviours and then manipulated unscrupulously. The heartbeat rates of players in some games can be measured optically (by sensing changes in green light absorption by blood flowing under the skin) without the need for attached equipment. Could this data be used to challenge insurance claims? Could the data be on-sold to produce targeted ads? Player identity is a privacy issue. Should a player's identity be protected?

Data security

There are data security issues. Is it possible for a player's data to be deleted? For someone who has invested years playing one game, the unintentional or malicious deletion of that user's data would have a far-reaching impact on them. Should this be classified as theft? The value of the virtual assets in MMOG were mentioned previously. This has led to challenging legal problems many countries are yet to solve. The way governments and courts determine how to treat theft of a fantasy artefact from a game has implications for how the theft of personal digital assets stored in digital service (photos, emails, documents, books, music, movies, social networking interactions etc.) will be treated legally.

Developers must also take account of the nature of the content of their games and age appropriateness. In 2005, video games were added to the list of items that receive classification by the Australian Classification Board (ACB).

KNOWLEDGE BUILDER

Class activity: Social and ethical issues in gaming

Social and ethical issues involving gaming have received much attention over the years. From the following list, identify each issue as mainly social or mainly ethical. Select three issues from this list you believe to be most significant. Can you suggest others?

- Can exposure to and participation in in-game violence desensitise players over time?
- Can gaming develop cognitive abilities?
- Can games result in feelings of psychological trauma?
- Can gender stereotyping have negative influences in games?
- Can many hours playing video games 'rewire' our brains?
- Can gaming result in social isolation?
- Can some games features be considered gambling under state/territory laws?
- Are there any issues of cultural sensitivity relevant for games developers?
- Can online gaming fulfil unmet needs for social interaction?
- Can games be made more accessible for those with disabilities?
- Can gaming be addictive?
- Can gaming be a threat to personal privacy?
- Can microtransactions in a game lead to overspending or gambling addiction?

Note to teacher: We gain understanding and personal maturity by open and honest discussion of differing viewpoints. This class activity is in three stages and opens up controversial topics in gaming in a way that can lead to thoughtful discussion.

Stage 1: 10-question Oxford 'debate'

Place the words 'agree' and 'disagree' on opposite walls of the room and create an open space in the centre. As each statement below is read aloud, everyone in the class stands somewhere on an imaginary line drawn between the two walls which represents how strongly they feel about that issue. If they entirely agree they will touch the 'agree' wall, if they are undecided they will stand in the middle of the room, if they disagree strongly they will touch the 'disagree' wall.

Understand that a person might stand alone at one extreme because they interpreted a question differently or because they strongly disagree with everyone else. Make this clear at the start.

Read out each statement in turn, going on to the next one when all students have settled on their stance. Any person may ask for clarification, but at this stage no debate is allowed. Students will not be asked to defend their views during this stage.

The teacher records the outcome of each question by indicating on paper using horizontal lines the spread of the class. Of special interest later will be those questions where a marked split occurs.

Use these questions or substitute your own:

1. Repeated exposure to in-game violence can desensitise people to violence.
2. Repeated exposure to simulated game violence can result in real-life violent behaviours.
3. Games are guilty of racial and gender stereotyping.
4. Many hours playing video games can 'rewire' the brain.
5. Gaming can result in social isolation.
6. Gaming can fulfil unmet needs for social interaction.
7. Gaming can become addictive.
8. Gaming can threaten personal privacy.
9. My generation spends too much time playing computer-based games.
10. I spend too much time playing computer-based games.

(Optional) Stage 2: Discussion of the issues

Form discussion groups for selected questions. These groups are not to come to a conclusion about the issue but report back with a list of major points for their issue. Each group reads aloud their list.

Select one issue which was most interesting to the class during this stage.

Stage 3: Finding like minds

This point of this activity (sometimes called the 'Delphi decision-making technique') is to clarify different viewpoints. To start, every person tries to find one other person with whom they can agree on this selected issue. These form a group of four and finally of eight. This will result in large groups that will have different viewpoints but who agree among themselves. For each stage, a leader/teacher announces when time is up.

The process is as follows.

Preliminary viewpoint

Students individually write down their initial position on the chosen issue using a single sentence on a piece of paper or a mobile device.

Delphi 'pairs'

Each person now has five minutes to find one other person with whom they can come to agreement. They may wish to modify their stance to reach this agreement.

- If the two agree, they sit down together as a pair and write a new agreed statement.
- If a person can't find someone they can agree with, they remain by themselves.

Delphi 'fours'

- Pairs seek another pair to form 'fours'. The room has 3 minutes to achieve consensus between the two pairs or agree to make modifications to achieve consensus with any other group in the room. They move from group to group seeking agreement.
- If they agree they sit down together.

Delphi 'eights'

- 'Fours' of students seek out another group of four in an attempt to form 'eights'.
- The room has five minutes to achieve consensus or agree to make modifications to achieve consensus with any group in the room. They move from group to group seeking agreement.
- If they agree, then they sit down and write this agreed single statement.

Final stage

Teacher leads discussion by asking a spokesperson from each group to read their agreed statement.

Conclusion

The teacher or the class may want to debate the issues more fully at this point. Hold a brief class discussion to conclude.

KNOWLEDGE BUILDER

Accessibility

Accessibility means avoiding unnecessary barriers that prevent people with a range of impairments from accessing or enjoying your game. At least 20 per cent of casual gamers face accessibility issues. As one gamer who became a quadriplegic following an accident expressed it: *'When my injury happened, I got extremely depressed. Because I had no finger movement, I figured there was nothing I could really do any more. Eventually I found help to mod a controller. Thanks to remapping options I've been gaming ever since.'*

UI accessibility is discussed in further detail in the chapter *Designing for User Experience 1.* This features the case study and link to a revealing vlog by blind gamer, Steve Saylor.

There are many players who cannot play a game such as *Pac-Man.* Decide how *Pac-Man* could be modified to address the needs of those with impaired eyesight, hearing, motor skills, hand-eye coordination or with colour blindness. Use Table 13.2 to complete your own spreadsheet. Next, select another game. Use this table later to evaluate your own project.

You can download the excellent Game Accessibility Guide produced by game studios that will help.

Table 13.2 Game accessibility: complete a spreadsheet using the headings in this template.

Accessibility issue	Pac-Man	<another game>	<your project>
vision			
hearing			
colour blindness			
movement			
cognitive			

Weblink
Game accessibility guidelines: Excel checklist download

ACTIVITIES

Identify

1 Identify four legal issues relevant to gaming.
2 Identify one social issue and one ethical issue in gaming.
3 Identify one accessibility issue in gaming.

Analyse

4 Outline the Australian rating system used to classify computer games.
5 Provide a brief one-paragraph report on your experience of the topic in the *Knowledge builder: Social and ethical issues in gaming*.
6 Identify an issue you think is important in the world of gaming and justify your choice.

Extend

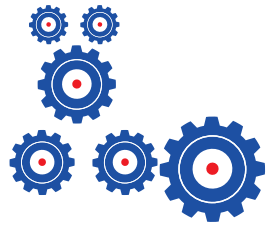
7 Use the Federal Government's computer game application for classification site to list what data a game developer must submit to receive a Level 2 classification for their game. Note costs involved.
8 Can repeated exposure to violent games desensitise players to violence? Explore evidence for and against and summarise your research.

Weblink
Australian Classification: Classify a computer game

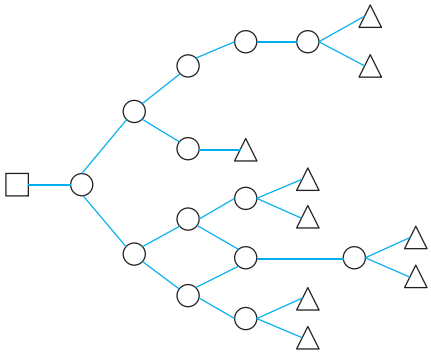
13.5 What makes a good game?

AUDIENCE

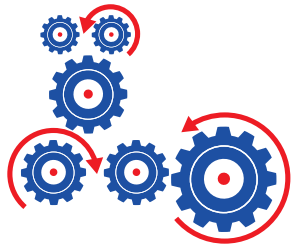
GENRE



GAME MECHANICS



NARRATIVE



DYNAMICS

Figure 13.9 Factors that make a game good.

Audience

Who is your game for? How will it be used? Keeping the needs of the user in mind is vital throughout the design process in any type of project.

If you have a particular goal in mind for your game, such as teaching an aspect of Ancient History through an adventure game or helping young children practice maths, it is important to survey the intended audience to establish their interests and needs. You could ask what type of games they like, or what they would most like the game to help them with.

Once you have this feedback, develop a first design and then develop alternatives. It can sometimes be hard to get past your first idea, yet allow for the possibility it may not be the best!

Genre

In developing a game of your own, an early decision will be to select a genre. Popular genres are: adventure (e.g. exploring a virtual world), platform, simulation, puzzle, maze (e.g. *Pac-Man*), strategy, role-playing and scrollers (e.g. *Space invaders*).

Your genre may be influenced by the hardware you will be using. It is likely your games projects will be developed for a personal computer and so input and output may be limited to keyboard, mouse, speakers or computer screen or perhaps a VR headset.

Game mechanics

The mechanics of a game describe the rules that govern the game, the actions that players can perform and the games' responses. In *Pac-Man*, possible actions are moving in one of four directions while eating dots and avoiding four ghosts. As mentioned previously, each of the ghosts behaves differently and they respond to 'power pellets'. The game increases in difficulty and collision with a ghost results in a *Pac-Man* life being lost.

Narrative

The narrative of a game describes its world or setting, its story, its characters and its goal. In *Pac-Man* the setting is a maze, the characters are *Pac-Man* and the ghosts.

The goal is to stay alive by avoiding ghosts, but the final goal has never been stated!

Dynamics

The dynamics of a game describe the way players interact with the mechanics of the game. This can involve the player deciding on a particular strategy or gameplay. In *Pac-Man* players can decide to focus on avoiding ghosts at all costs or on eating dots as quickly as possible to progress to the next level, or to maximise use of two 'warp tunnels' at the sides which slows down the ghosts.

Aesthetics

The aesthetics of a game describes its 'look and feel'. This can involve the sensory experiences for the player (visual, aural, haptic etc.), the appeal of the design of the game (its layout, artwork, colour palette etc.) or the emotion or pleasure it gives players.

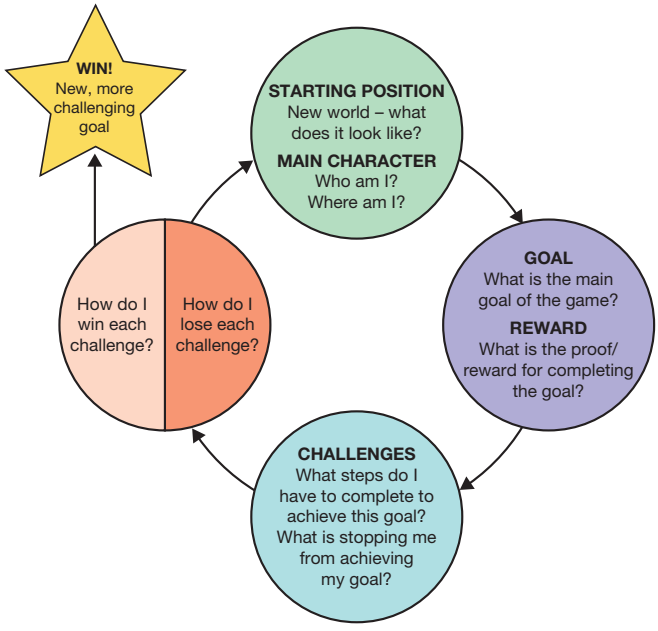
Pac-Man's designer, Toru Iwantani, intended his characters to be non-violent and 'cute'. He wanted to attract couples to make the games arcades of 1979 more 'family-friendly' than they had become. The process of eating the dots was intended to be satisfying and the audio effect was based on Toru noisily eating fruit. Many found the game addictive, often leading to local shortages in Japan of 100-yen coins!

13.6 What makes a computer game?

Core mechanics

Most games can be broken down into the following separate parts:

- game world
- characters
- **core mechanics**
- goal
- rewards and challenges.



Cathie Howe, Dean Tuttle, Simon Hutchison, Anthony Fennell.

Figure 13.10 The core mechanics of a game include the game's goal, rewards and challenges and are key features of a good game.

Games loop

The core mechanics, or game actions, are the main part of game code known as the **games loop**. This loop unites all the features shown in Figure 13.10. The core mechanics of a game describes the actions a player can perform – running, fighting, collecting, exploring, solving a puzzle, and so on.

Most application programs do nothing unless we ask them to, but a game program must keep running even when the player is not doing anything. The games loop code executes repeatedly. The loop contains functions designed to respond to particular situations, such as a collision between objects or a signal from a timer. It also contains 'listener functions' that are triggered by user actions such as mouse, touch or keyboard presses. Each time it executes it updates the state of the game.

A simple representation of a games loop using pseudocode might look like this:

```
while (user does not quit or lose)
    check for user input
    move the player's character
    create and move obstacles or enemies
    check for collisions
    draw graphics
    play sounds
    update score
end while
```

It is important that the games loop allows the game to run smoothly, regardless of user input.

ACTIVITIES

Identify

1 What are a game's 'game mechanics'?
2 Identify five key elements of effective game design.
3 Define the 'game loop'.

Analyse

4 What is the difference between a game's dynamics and its mechanics?
5 Games designers endeavour to balance challenge and reward. Describe the potential consequences of an imbalance.

Extend

6 This section provided background by the designer of *Pac-Man*. Research and report on the design history of a game you have played.

13.7 Game requirements

Functional requirements

Once you have established your game genre, decided on a concept for your game, and established the core loop, it is time to establish your game's **functional requirements**. Functional requirements describe what the game must *do*. They state a function or behaviour of the game. They do not specify the way the game achieves each of these – these are defined as non-functional requirements.

The functional requirements define the core mechanics of the game (see Figures 13.10 and 13.11).

Use the weblink provided to play the classic game *Space Invaders*. In this game, a list of functional requirements might be:

- After starting the game, the player selects a level of difficulty and starts playing.
- The player can move a spaceship to the left and to the right and can also shoot rockets.
- The goal is to destroy invaders. If a rocket hits an invader, the invader disappears and a crash sound is played.
- A game is finished when all invaders are destroyed. In this case the elapsed time for destroying all invaders is displayed.
- Each player has a personal score which can be compared with other players.
- While the game is running, music is played.
- The game can be paused, in which case the timer stops and the music stops.
- During the game, the player can always see the elapsed time and a counter of the destroyed invaders.

Weblink
Space Invaders

Cathie Howe, Dean Tuttle, Simon Hutchison, Anthony Fennell
Figure 13.11 A useful template for designing the core mechanics of your game.

KNOWLEDGE BUILDER

Functional test cases

An important activity when developing functional requirements is to develop test cases listing all possible inputs and their expected behaviours. Complete the spreadsheet template (Table 13.3) using the previous functional requirements for the *Space Invaders* game and then do the same for your own game.

Table 13.3 Functional test cases template.

Game	Test case	Expected behaviour
Space invaders	level of difficulty is selected	
	player presses left, right keys	
	rocket hits invader	
	all invaders destroyed	
	game is paused	
Your game		

Non-functional requirements

Non-functional requirements are no less important than functional ones. Whereas functional requirements are *what a game does*, non-functional requirements describe *how it does it*.

A game designer balances challenge and reward. It is the combination of goal, challenge and reward which results in an effective game. As the player (often in the role of an avatar) progresses through the game, they face challenges. These can be won or lost, and if won can often lead to a more difficult related challenge.

The setting or 'game world' is critical too. The more the player feels part of the world of the game, the greater their level of immersion will be.

One often quoted non-functional characteristic of a good game is 'low floor, high ceiling'. This means that the game is accessible for beginners but the challenges become greater as players progress. In other words, the game is easy to learn but remains challenging yet fun to master.

The design of its user interface (UI) is vital. It must offer a clear and uncluttered screen, accessible intuitive controls, clear feedback and the status of the game. (The design principles outlined in Chapter 8 on *Designing for User Experience* cover colour theory, intuitiveness, balance, interface conventions such as well-chosen icons, feedback and intuitive navigation.

The principle of avoiding cognitive overload, also described in that topic applies here, too.) Mental overload occurs when users are forced to face too much information at once.

KNOWLEDGE BUILDER

Non-functional requirements

Complete the Table 13.4 spreadsheet template to rate the non-functional requirements for the *Space Invaders* game and then do the same for your own game.

Table 13.4 Evaluating non-functional requirements.

Game	Non-functional requirement	Rating 1-5 ★★★★★
Space invaders	clearly understood goal	
	high accessibility	
	strong level of engagement	
	'low floor, high ceiling'	
	positive aesthetics	
	acceptable cognitive load	
Your game		

13.8 Designing a game

In the project at the end of this chapter, you will build an original game. You will already be familiar with some techniques (such as brainstorming and mind mapping) for generating ideas. Another way to develop a new game is to draw inspiration from existing games.

KNOWLEDGE BUILDER

Using games to generate new ideas

In this Knowledge builder you will analyse some existing games to evaluate them in terms of their mechanics, aesthetics and usability. We suggest two of the world's most popular retro games to start with (in a group or on your own). Search for an online version of *Tetris* and for Google's playable *Pac-Man* Doodle, issued for its 30th anniversary. Next, select at least another three games to complete a spreadsheet modelled on Table 13.5, including at least one more complex contemporary game.

Explanations for these table headings are given in the text above, in the section 'What makes a good game?'

Table 13.5 Evaluating existing games.

Game	Audience	Genre	Mechanics	Narrative	Dynamics	Aesthetics	Accessibility	Rating 1-5 ★★★★★
Tetris								
Pac-Man								

Using SCAMPER

Steve Jobs liked to quote: 'Great artists steal!', which he attributed to the artist Pablo Picasso. However, the credit should be given to poet T S Eliot: 'Immature poets imitate; mature poets steal; bad poets deface what they take, and good poets make it into something better, or at least something different'.

The writers of *The Lion King* state they were inspired by Shakespeare's *Hamlet* and by the biblical stories of Moses and Joseph. The writers moved the story to the African jungle, turning the original characters into lions. It wasn't a new story, but it resulted in an exciting and world-beating new movie and musical.

When using others' ideas for inspiration, avoid plagiarism. Be honest about other people's work and credit the works that inspired you!

SCAMPER is a lateral thinking technique which can help spark creative ideas for new products by using existing products as a mental springboard. SCAMPER is an acronym where the letters stand for:

(S) substitute
(C) combine
(A) adapt
(M) modify
(P) put to another use
(E) eliminate
(R) reverse

9780170442787

KNOWLEDGE BUILDER

Generating ideas using SCAMPER

Form a group of three or four, select a game from your list in Table 13.5 and apply the verbs above by asking the following questions. Use a project notebook to record ideas that emerge.

Substitute

Ask: What components can I substitute?

- Could you substitute this game world for another?
- Could you substitute the character(s) for others?
- What game rewards and challenges could you substitute?
- What game actions (mechanics) could you substitute?

Combine

Ask: Can I combine resources or use others' skills?

- Could you combine ideas in this game with another game to create something new?
- Could you combine your ideas with another group to create a new game?

Adapt

Ask: Can it serve another purpose?

- Can the genre of this game be switched (e.g. to a competition, a race against the clock, a puzzle or a memory test)?
- What other games is this game like?
- Could this game be used in a different context?
- What other products could you include as part of a new game?

Modify

Ask: Can something be changed or highlighted more?

- Could you change the shape, look or feel of this game?
- What could you add to this game?
- What aspects of this game could you highlight to create something new?

Put to another use

Ask: Can I use it differently?

- Could this game be used to do something different?
- Could the audience for this game be changed?
- How could this game behave differently in another setting?
- Could this game be coded using different tools to create something new?

Eliminate

Ask: Can I simplify?

- Could you simplify this game?
- What features, characters or rules could you eliminate?
- What could you reduce in importance?
- How could you make it faster, lighter or more fun?
- What would happen if you took away part of this game? What would you put in its place?

Reverse

Ask: Can I do the opposite?

- What could happen if you reversed or sequenced things differently?
- What could happen if you try to do the exact opposite of what the game is doing now?
- What character roles could you reverse or swap?

13.9 Tools for creating games

Games written for the earliest computers in the 1980s had to be coded 'from the ground up'. As the hardware changed rapidly, code could rarely be reused by later, even similar, games. Games were traditionally standalone apps built using standard coding languages such as C, C++ and Java.

There are now many tools available for creating games. One of the most popular was Adobe's *Flash*. Thousands of games made with *Flash* are still available on the web.

When mobile devices became popular, *Flash* presented a number of problems: it was not open-source, required a browser plug-in, used lots of battery power, was susceptible to hacking and security flaws, did not run on iOS or Android, was resource-heavy and incompatible with touch-based input. HTML5 now runs natively within web browsers and is fast. As a result, Adobe has replaced *Flash* with *Animate*, allowing web-based games to be written in HTML5 Canvas and using Javascript. (See Chapter 16.)

Weblinks
Unity Learn

Dev tried to make simple game in 8 different engines

Game engines

Game engines are specialist software that provide the tools game designers commonly need to build and run a game without having to write code from scratch. They provide tools for designers to import graphic assets created in other software; add lighting, audio, special effects, interactivity and animation and combine these to create a game environment. Common game engines (Figure 13.12) include tools for:

- physics
- input
- lighting
- rendering
- scripting
- collision detection
- AI
- libraries of pre-designed assets.

Games designers use game engines to avoid having to 'reinvent the wheel', as many games need the same elements. Without them, new games would be far more difficult and take a lot longer to create. Games engines include collision detection and physics engines which provide game objects with behaviours emulating the real-world environment. It is also possible to import pre-designed libraries of assets including textures, models, animations, project examples and even interactive tutorials.

Some game engines designed especially for beginners use drag and drop techniques and require no coding. Some have particular target audiences in mind: hobbyists and students or professional studios. Each offers features targeted at their intended user base. *Unity* is one of the more popular game engines and is widely used by both beginners and professional game designers. *Unity* provides the ability to build once and publish the game to multiple platforms. We introduced you to *Unity's* virtual world-building possibilities in the project at the end of Chapter 8.

If you would like to try creating a 2D game using *Unity*, follow the weblink provided and select 'tutorials'. One of the starter tutorials will lead you through this.

Creating a game using a game engine

There are many popular game engines you can use to create a game and each has strengths and weaknesses.

One developer created the same game using eight different game engines: *Unity, Unreal Engine, GameMaker, Godot, RPG Maker, Construct, G Develop* and even *Scratch*! He has made a video comparing the experience of building the same catching game using each of these engines, which is worth watching (see the weblink provided).

We recommend selecting one of the engines featured and trying out an introductory tutorial to build a simple 2D game. See the weblink opposite.

Shutterstock.com/monticello

Shutterstock.com/Wirestock Creators

Figure 13.12 Popular game engines.

SKILL BUILDER

Exploring *GameMaker*

1. Sign up for the free version of *GameMaker* and download it.
2. Select one of the introductory tutorials and work through it. We suggest starting with a simple arcade game such as that illustrated in Figure 13.13.
3. When you have finished this, have others test your game and suggest improvements.
4. Did you find this game authoring tool difficult to use? Write an evaluation of this software as a table of strengths and weaknesses.
5. What added features would you like to see?

Source: GameMaker

Figure 13.13 *GameMaker's* retro arcade game tutorial.

Despite the existence of such sophisticated tools, a good starting point is to build a first, simple game from scratch. In the project at the end of this chapter you will create a game using Python then adapt it to work on a micro:bit. You will have coded and created your own handheld game as well as polishing your coding skills!

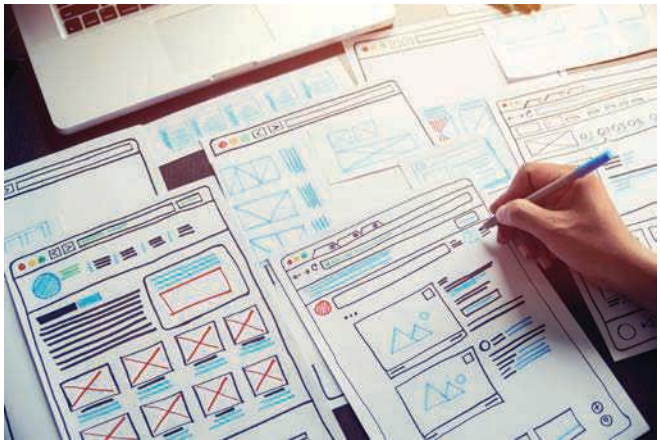
There are more game-based projects you can try: in Chapter 14 you code a virtual pet and an adventure game, in Chapter 15 you build a game collection and in Chapter 16 you build a web-based snake game.

A project notebook

Apart from any official or collaborative documentation, most developers keep an informal record of ideas and progress in a project (Figure 13.14). Here are some of the ways you could use a project journal or notebook:

- recording functional and non-functional requirements
- jotting down random ideas
- sketching user interface designs
- noting user feedback
- drawing flowcharts and storyboards
- noting challenges faced and overcome
- suggesting future improvements
- acknowledging help from others and any outside sources of ideas.

Why not start using a project notebook as you tackle the project in this chapter?



Shutterstock.com/Chaosamran_Studio

Figure 13.14 Most developers keep an informal record of ideas and progress in a project book or folder.

ACTIVITIES

Identify

1 State five possible functional requirements of a computer game of your choice.
2 State five possible non-functional requirements of a computer game of your choice.
3 Identify three tools for generating ideas for games.

Analyse

4 What features do games engines offer a game developer?
5 Outline how HTML5 Canvas can be used in web-based game development.
6 Play an online free version of the game of *Frogger* and identify its requirements, both functional and non-functional.

Extend

7 Devise a simple game idea, describe the game's mechanics and make a list of its functional and non-functional requirements.

Chapter review quiz

Weblinks:

- Edsac Simulator
- Spacewar emulator
- PAC-MAN Doodle
- Meet the Man Who Beat 'Pac-Man'
- Deep Blue vs Kasparov: How a computer beat best chess player in the world
- Watson and the Jeopardy! Challenge
- AlphaGo Official Trailer
- Google Labs
- Prof. Alan Brown interviews Steve Wozniak
- Game accessibility guidelines: Excel checklist download
- Australian Classification: Classify a computer game
- Space Invaders
- Unity Learn
- Dev tried to make simple game in 8 different engines
- 5x5 LightsOut
- Replit
- Python
- micro:bit Python Editor
- replit

Project resource:

- Creating a handheld game

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

13.10 Project: Creating a handheld game

Project resource Creating a handheld game

It is expected you will have already completed Chapter 4: An introduction to software development.

This project is a little more challenging than some others, but if you work carefully through what follows you will have success!

Your task

In the first stage of this project, using the general-purpose language Python, you will build a popular handheld game from the 1990s called *Lights Out!* for the desktop. Then, in the second stage you will develop an appropriate UI, move your game to be played on a micro:bit and adapt your code for it. Note that if you do not have access to a micro:bit you can still complete this stage of the project using the Python simulator, which is part of the micro:bit Python online software you will be using.

In Stage 2, you will learn a great deal about UI design and designing a game that takes UX seriously.

In the final stage, you will invent and develop a game of your own.

You will find this project easier if you have already studied the software programming chapters.

Summarising:

- Stage 1: Develop a screen-based prototype (using a guided tutorial)
- Stage 2: Design and build handheld versions for the micro:bit controlled by both buttons and tilting
- Taking it further: Design and build a similar game on your own using the knowledge gained.

(For those who would like a further challenge, there is the possibility of creating a web-based version using the JavaScript skills acquired by building a snake game in Chapter 16.)

Identifying and defining

Stage 1: Build a screen-based prototype (using a guided tutorial)

Lights Out! consists of a 5 × 5 grid of lights (Figure 13.15). When the game starts, a random pattern of lights is switched on.

Pressing any light toggles that light on or off as well as adjacent lights above, below, left and right. See Figure 13.15. The goal of the puzzle is to switch all the lights off. It's a lot more difficult than you think!

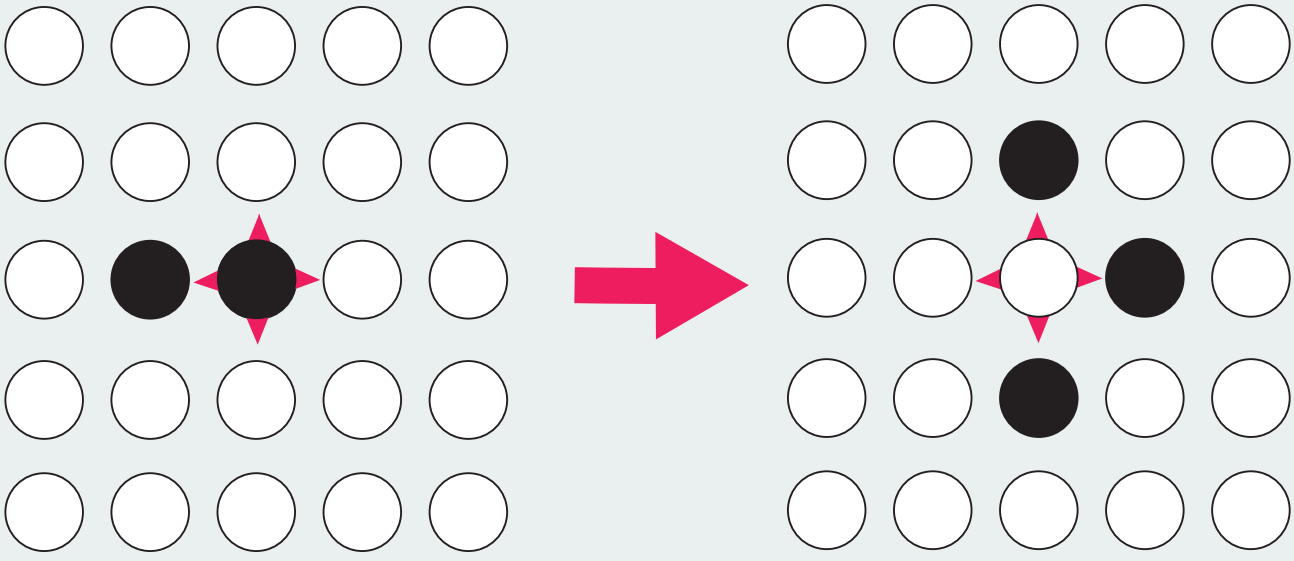


Figure 13.15 (L) The centre light is currently off. (R) After it is pressed it will turn on and then toggle the four lights adjacent to it (above and below). Toggling means swapping to the opposite: off to on, or on to off.

Researching and planning

Research the game

Before coding, it is important to understand the problem you are solving. Begin by researching an original version of this game. Try an online version of this game.

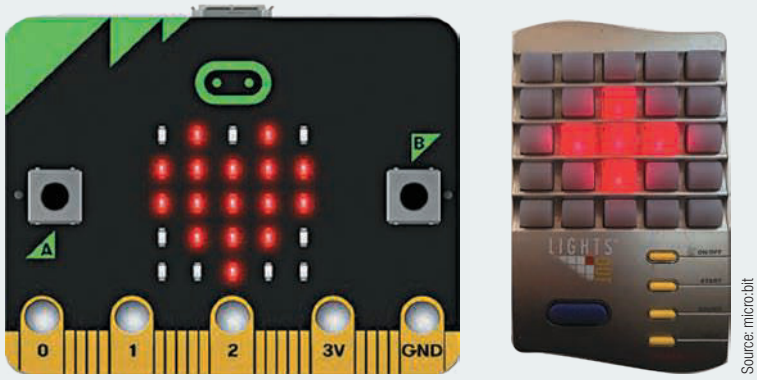


Figure 13.16 (Left) Code your own micro:bit version of *Lights Out!* (Right) Original handheld 1995 version of *Lights Out!*

Research and play other online versions and mobile app versions (Figure 13.16).

If you are interested in strategies to solve this game, the *Lights Out!* Wikipedia article provides its history as well as the following interesting observations:

- The order in which the lights are pressed does not matter, as the result will be the same.
- For the quickest solution, each light must be pressed no more than once, because pressing a light twice is equivalent to not pressing it at all.
- If you choose the wrong light just re-enter and you will reverse your move.
- Some patterns have no solution.

The solution

If a solution exists, it can always be found using 'light chasing'. Here's how:

- Lights in row 1 are turned off by toggling those below them. Repeat this for consecutive rows up to row 5. Row 5 is solved based on the following:
  - Inspect first 3 LEDs on bottom row.
  - If first is on, press 4th and 5th in the top row, else if second is on, toggle both 2nd and 5th in the top row, else if third is on, press 4th in the top row.
  - Repeat. Note you may need at least three repeats but this WILL solve it!

Computational thinking

Design the interface

Arrange your board as a 5 × 5 grid of 0s or 1s and describe each coordinate by referring to its row and column. This board shows a light as being on (a '1') in row 3, column 4.

rows \ columns	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	1	0
4	0	0	0	0	0
5	0	0	0	0	0

The interface for your first prototype will be limited, as you can only display printouts.

A board with all lights OFF would look something like the following, with each off light represented by a '0' and on by a '1':

```
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Design thinking

This shows the result when the light in row 2, column 2 is toggled to ON:

```
0 1 0 0 0
1 1 1 0 0
0 1 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Weblinks
5x5 LightsOut
Replit
Python
micro:bit Python Editor

Weblink
Lights Out! Wikipedia

Design the data structures

You now need to define variables to hold score and high score.

An array would be most suitable to represent the 'lights' using rows of 1s and 0s. (Note: we will refer to 0 and 1 throughout this project as 'lights' which are either off or on.)

We can identify each light by using x and y coordinates. Remember in computing, arrays are numbered beginning with 0, not 1!

Design the game loop

Here is where you decide the game's logic.

1 Draw a flowchart to represent the logic of the game.

Next, describe this logic using numbered steps before reading on.

Here is our version of the steps in order:

- Draw an empty board.
- Fill it randomly with a pattern to solve.
- Request values for row then column.
- Toggle both this and adjacent lights.
- Increment score.
- Check for a win.
  - If a win and if score is higher than high score, update high score, reset score, go to step 2.
  - If not a win, go to step 3.

Select development tools

Replit

2 Selection of appropriate software design tools can be a significant decision for a programmer. The right tool can make the process far smoother.

We suggest using repl.it (which allows you to code, catch errors and see results in the one view) or IDLE for prototype development in Python.

Producing and implementing

Start coding!

Build your code in stages, referring to the flowchart and steps above. IMPORTANT! If you are using the PDF version of this text you must type the code in by hand rather than try to copy and paste, as you will create syntax errors due to hidden spaces and tabs. Also remember that in Python, accurate indentation is critical!

We will now develop the code in careful stages. There are a few places where you have to complete a line of code on your own and these are clearly indicated. They are not difficult to do!

Import external modules

3 You will use Python's `random` function to generate new games. Enter this code:

```
import random
```

Define variables

Score and high score

4 Define global variables for score and best score. Add these next.

```
score = 0
bestScore = 0
```

5 Define a 2D array to represent our 5 x 5 grid of lights. Note that [3][4] refers to row 3, column 4. Here's how the array code looks. Take special care with the nested brackets! Python uses a list structure for this.

```
grid = [
[0,0,0,0,0],
[0,0,0,0,0],
[0,0,0,0,0],
[0,0,0,0,0],
[0,0,0,0,0]
]
```

Write this all on a single line:

```
grid = [[0,0,0,0,0],[0,0,0,0,0],
[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]]
```

Define functions

6 Referring to your game design, code each of the required functions.

Toggle lights function

Computational thinking

7 In the following code, the first line in the following (incomplete) toggle lights function toggles the selected light off or on (the `not` turns a 0 into a 1 and vice versa).

The next 4 instructions in the function toggle adjacent lights where they exist (above and below, to the left and right).

What happens if there is no light in one of these positions, as is the case where the selected light is at the edge of the board?

Complete the following for the toggle lights function and add it to your code.

9780170442787

```
def Toggle(row, column):
    grid[row][column] = not grid[row][column] # toggle selected light
    # toggle the left light
    if row > 0:
        grid[row-1][column] = not grid[row-1][column]
    # toggle right light
    if row < 4:
        complete this line
    # toggle light above
    if column > 0:
        complete this line
    # toggle light below
    if column < 4:
        complete this line
```

Board set-up function

8 We set up our board by toggling lights randomly 100 times. This ensures the game is possible, as some patterns cannot be solved. The logic of this is that if the board's pattern is created by toggling then it must be able to reverse the toggles and solve it! Add this to the code.

```
def SetUp():
    for n in range(0,100):
      r = random.randint(0,4)
      c = random.randint(0,4)
      Toggle(r, c)
```

This function toggles random lights 100 times.

Redraw board function

9 We now create a redraw function that prints the *Lights Out!* grid of 'lights'. Every light that is on is represented in the array by a "1" and so we print "1" and every light that is off by a "0", for which we print "0". These must be printed on the same row and printing moves to a new line at the end of each row with the final print statement. The Python print statement allows us to add `end = " "` to append a space after each 0 or 1. Add this to your code.

```
def DrawBoard():
    for row in grid:
        for q in row:
            if q:
                print("1", end = " ")
            else:
                print("0", end = " ")
        print()
```

Check win function

10 This function checks all the lights are off by adding all elements in the array and checking if this sum is 0. Clever, eh? Observe how this is achieved by using two nested counted loops to move through all the elements in the array.

We have left scoring for you as an activity for further development.

Add this to your code.

```
def CheckWin():
    tot = 0
    for cy in range(0,5):
        for cx in range(0,5):
            tot += grid[cx][cy]
    if tot == 0:
        return True
    else:
        return False
```

Note: `range(0, 5)` in Python means values 0 to 4 inclusive and does not include 5!

Code the game loop

11 You need a game loop to keep the game running. To do this, wrap it in a `while True:` (which is always true!). This way it will keep playing the game until the player quits the program.

Add this code.

```
# *** Gameloop ***
SetUp()
while True:
    DrawBoard()
    row = int(input("Row? ")) - 1
    column = int(input("Column? ")) - 1
    Toggle(row, column)
    CheckWin()
```

This game loop first draws the board by calling the `SetUp()` function and then asks the user for the column and the row of their move. We have modified the code so users can input their choice of row and column using numbers 1–5 rather than the indices that arrays use, which always begin with 0! We modify the input so players can use user-friendly 1 to 5 instead. This function then sends the light's position to the toggle function.

Testing and evaluating

Run the program

After checking your code, run the game.

It should print a random 5×5 board with some lights on and some off. Select the row and column of a light

that is on and press return. That light should go off and toggle on or off the lights to its left, right, top and bottom, depending on whether they were on or off in the last move. If it works, play it using the techniques listed at the start of this project.

If there are errors in your code, try to fix them yourself before looking at our online solution.

Helpful playing hints:

- If you type the wrong integer for a light's coordinates, re-enter the same values to undo!
- It doesn't matter what order you toggle lights.
- Use the 'chasing lights' technique, described in the 'Research the game' section above to solve each puzzle. It always works!

Fixing bugs

If your program crashes, fix syntax errors first. (Chapter 4 has a Python summary table and there are many online resources.)

A key tool for debugging logic is to use a debugging output statement. In Python, this is simply a `print` statement. Insert print statements wherever you want to check the values of variables or check if the program has reached that line by inserting `print("Here!")`.

At the moment if an incorrect coordinate is entered, the program crashes. Can you fix this? The final version *lightsout.py*. is available in the Nelson MindTap resources if you need help.

Can you identify other aspects of the code that can be improved?

Add scoring

Tell the user their score for each game and keep track of the high score. See the left column of Figure 13.18 if you need help with this. There is also a completed version in the Nelson MindTap resources, *lightsout_scores.py*

Other ideas

Here are some further suggestions to try, in increasing order of difficulty. Select just one or two to try.

- Add a welcome message and instructions.
- Add a congratulatory message for winners.
- Add a 'new game' message.
- Now for a UI issue. The board and user input are cramped. It is difficult to see where a new game begins. Fix this by adding some empty print statements: `print( )` in appropriate places.
- Print the number of moves made each turn.
- Ask the user if they wish to play again and quit if the answer is 'no'.
- If users enter values outside the range the program crashes. Modify the code to keep for input if values are integers outside 1–4.
- Use the 'chasing lights' technique, described in the 'Researching and planning' section above, to provide onscreen hints for the next move when the player has reached the last row.

Stage 2: Design and build a handheld Lights Out! using a micro:bit

This next stage requires you to use the knowledge you have gained to do more on your own. We point the direction. Don't worry, as full solutions are provided on Nelson MindTap should you need them. Even if you need to refer to these, you will learn a great deal in the process.

Here you adapt your code from Stage 1 for the micro:bit. You can do this stage even if you don't have a micro:bit by using the built-in simulator in the micro:bit Python editor. The micro:bit will be programmed using the same language, Python. Most of the code is identical. See Figure 13.18 to get the idea of what you will be doing.

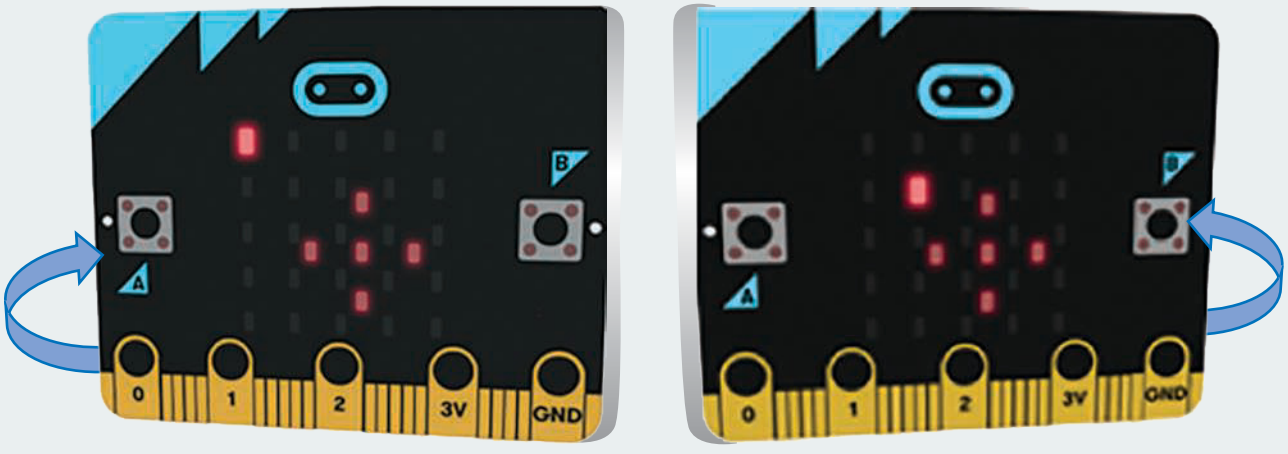


Figure 13.17 The selector LED ("playLED") is programmed to roll like a marble when the board is tilted. This brighter LED indicates the current position. In one UI version players move it by pressing buttons A or B and in another by tilting the micro:bit, confirming their choice by pressing buttons A and B or touching the micro:bit logo.

Select your development tool

For coding the micro:bit using Micropython, we suggest MakeCode's Python editor (see weblink).

Designing the UI and UX

Using a micro:bit offers great possibilities for improving the UX of the *Lights Out!* game. A key advantage are the options it offers for user input and display. An obvious improvement is it uses lights! And it's perfect that it has a 5X5 grid of lights!

Design thinking

Also, the micro:bit has an accelerometer, which can detect gestures including up, down, left, right, face up, face down, freefall and shake. Can you suggest ways these might be used for input to improve the UI for *Lights Out!*, and therefore the overall UX?

Selecting position

A brightly lit single LED light (we will name it *playLED*) could be used to indicate the currently selected position. This could be moved around the board either by using the buttons A and B or by tilting the board (Figure 13.17).

A problem this creates is that when the *playLED* is over the same position as a light, we won't be able to tell easily whether that light is on or off. Can you suggest a solution?

In what follows, we will develop two versions of a UI. The first uses buttons to input position and the second uses gestures.

You need a way to confirm our choice of position. Micro:bits can distinguish between single button presses or both pressed at once. We use button A to shift the LED to the right and B to shift it downwards. To toggle the lights at that selected position, both A and B buttons could be pressed at the same time or in a version 2 micro:bit, the top gold logo could be touched (see Figure 13.17).

Adapting for the micro:bit

Great news! Most of the code you developed in Section 1 for the screen prototype remains unchanged. Here we have used the scoring screen version from the Nelson MindTap resources, but you can choose to leave out scoring if you wish. We only need to define a few additional variables, replace the `DrawBoard()` function and replace just six other lines of code. Figure 13.18 shows all these changes.

Using buttons for input

This section will explain each of the changes we need to make, labelled A to D in Figure 13.18.

A: As we are now running Python on a micro:bit, it needs functions added to allow it to communicate with the board's components, such as its buttons, LEDs and accelerometer. We start by importing the micro:bit library of functions. The '*' just means 'all of them'!

```
from microbit import *
```

We define additional variables for the playLED position selector, its coordinates x and y and a variable to hold the pattern of LED lights on the board.

Weblink
micro:bit Python editor

B: This code is a UX feature only. This function has already chosen a pattern for the game by filling the array called `grid` with 0s and 1s. Can you see how this additional code turns on LEDs *slowly* to brightness 4?

C: This `DrawBoard()` function will be used during the game to light the LEDs corresponding to the grid array. It also displays a bright playLED used for selecting.

D: We make the playLED flash from brightness 9 to 4 when positioned over a lit LED and from 9 to 0 if over one that is 'off'. Can you explain this UI decision? How would the play be affected if the light did not flash but was on all the time?

Instead of asking the user for numbers for row and column, we need to use input from button presses. If button A is pressed, the playLED shifts right and if button B is pressed, it shifts down. If both are pressed together, the chosen light under it is toggled.

The code will not work if this last choice is not written first before checking the single buttons. Can you explain why the double button check must be done before the single checks?

Note: If you do not have a micro:bit you can use the Python simulator micro:bit. At the time of writing, it had no ability to simulate a simultaneous press on both buttons. Instead, replace the line of code in this section `if a and b:` with `if pin_logo.is_touched():` When playing, click the gold logo on the simulator using your mouse pointer.

We have decided to have our bright playLED selector light 'wrap around the edges' of the micro:bit. Can you identify how this is done?

In Python, `x % y` returns the remainder when x is divided by y. By writing `x = x % 5` we make the screen wrap in the horizontal direction. Can you explain how this works?

Test the program, fixing any errors. If you get stuck and need help try looking at the finished version provided on Nelson MindTap, named *lightsout_microbit_buttons*.

Understanding the code

Computational thinking

It is hard for anyone to hold the many ideas in this program in their heads at one time.

Go through your final code and insert brief comments (beginning with #) to explain what each section does.

Screen-based Lights Out game code

```
import random
# ***DECLARATIONS***

grid = [
[0,0,0,0,0],
[0,0,0,0,0],
[0,0,0,0,0],
[0,0,0,0,0],
[0,0,0,0,0]
]

# ***FUNCTIONS***
def SetUp():
    for n in range(0,100):
        r = random.randint(0,4)
        c = random.randint(0,4)
        Toggle(r, c)


def DrawBoard():
    for row in grid:
        for q in row:
            if q:
                print("1",end = " ")
            else:
                print("0",end = " ")
        print()

def Toggle(row, column):
    grid[row][column] = not grid[row][column]
    if row>0:
        grid[row-1][column] = not grid[row-1][column]
    if row<4:
        grid[row+1][column] = not grid[row+1][column]
    if column>0:
        grid[row][column-1] = not grid[row][column-1]
    if column<4:
        grid[row][column+1] = not grid[row][column+1]


def CheckWin():
    tot = 0
    for cy in range(0,5):
        for cx in range(0,5):
           tot += grid[cx][cy]
    if tot == 0:
       return True
    else:
       return False

SetUp()

# ***MAIN GAME LOOP***

while True:
    DrawBoard()
    row = int(input('Row? ')) - 1
    column = int(input('Column? ')) - 1
    print()
    Toggle(row,column)
    if CheckWin():
        print("Congratulations! You did it!")
```

Code changes for adaption to micro:bit

A

Add this first line to import microbit's Python module.

```
from microbit import *
```

Add a variable for playLED and set its brightness.

```
playLED = 9
```

Add variables to identify playLED coordinates (x, y) and store the current pattern.

```
x = 0
y = 0
img = Image('00000:00000:00000:00000:00000')
```

B

(Optional) Add a cool UI effect to the SetUp() function:

```
for r in range(0,5):
    for c in range(0,5):
        img.set_pixel(r, c, grid[r][c]*4)
        display.show(img)
        sleep(100)
```

C

REPLACE Drawboard print function with LED pixel functions!:

```
def DrawBoard():
    img = Image('00000:00000:00000:00000:00000')
    for m in range(0,5):
        for n in range(0,5):
            img.set_pixel(m, n, grid[m][n]*4)
    img.set_pixel(x, y, playLED)
    display.show(img)
```

D

We now need the A, B micro:bit buttons to control the flashing LED and to select positions!
REPLACE these highlighted lines from the screen version
Match alignment!

```
if playLED == 9:
   if grid[x][y] == 1:
      playLED = 4
   else:
      playLED = 0
else:
   playLED = 9
(a, b) =
(button_a.was_pressed(),button_b.was_pressed())
sleep(250)
if a and b:
   Toggle(x, y)
elif a:
    x += 1
elif b:
    y += 1
x = x % 5
y = y % 5
# we leave you to replace the print message
```

Figure 13.18 Screen version of *Lights Out!* Code, side by side with micro:bit version.

Using the accelerometer for input

Can you imagine how we could input positions just using the micro:bit's accelerometer?

One way would be to have the positional playLED respond when the micro:bit is tilted. The playLED could then act like a rolling marble. When the desired position is reached, that light could be toggled by pressing a button. If you have the previous code working, this is surprisingly easy to do. Now we will use a steady light rather than a flashing one for the playLED. Can you explain this choice in terms of UI?

Remove the following flashing code controlling the playLED from your *Lights Out!* micro:bit button version (in section D):

```
if playLED == 9:
   if grid[x][y] == 1:
      playLED = 4
   else:
      playLED = 0
else:
   playLED = 9
```

We need to control the speed when tilting the micro:bit. We increase the x and y positions only when the tilt is at enough of an angle and with a pause so it doesn't do it more than once. Here, the micro:bit's inbuilt accelerometer is used to measure this tilt.

Complete missing parts in the following code for 'dy' by recognising the pattern in the 'dx' statements. Replace the button-based code of section D in Figure 3.18 with this new version using tilting. Try to understand this code *before* reading our explanation following it. We can now just use button_a to toggle a move (easier!) as we no longer need to use buttons A and B to move the playLED position.

```
dx = accelerometer.get_x()
dy = accelerometer.get_y()
if dx > 300:
   x += 1
   sleep(200)
if dx < -300:
   x -= 1
   sleep(200)
if dy > 300:
   complete this line
   complete this line
if dy < -300:
   complete this line
   complete this line
x = x % 5
complete this line for y
if button_a.was_pressed():
   Toggle(x, y)
   sleep(200)
```

In this code you are replacing the button code with x and y accelerations named dx and dy. We don't want to make the tilting response overly sensitive, so set threshold values of 300. Experiment and see what values you prefer. When the threshold is exceeded, the x and y positions are incremented or decremented with a delay to prevent it doing it twice.

Test the program and fix errors. If you get stuck, try looking at the finished version provided on Nelson MindTap, named *lightsout_microbit_gestures.py*.

Taking it further

Now you have an opportunity to design and code a game of your own for the micro:bit based on what you have learned in this project. You should now know enough to do this on your own.

Here are some ideas as springboards:

- The light starts at the middle of the micro:bit LED array and can be rolled around by tipping the board. Points are gained as it slips through goal posts at the edges and lost if it hits obstacles.
- The light must be tilted to evade moving groups of four LEDs.
- A pedometer where your steps trigger the accelerometer and are counted.
- A balance and steadiness challenge (sensitivity of the tilt is increased) and players must keep the light away from a lit-up square border of the micro:bit.

14 Creating simulations

FOCUS AREA: Creating games and simulations

Chapter outcomes

In this chapter, you will learn:

- the difference between a model and a simulation
- the applications of models and simulations
- to use a simulation in prediction
- to use a spreadsheet to build and implement simulations
- how games are simulations
- to design, build and implement a simulation
- to create a virtual pet as a simulation
- to use object-oriented programming (OOP) to build a simulation game.

Key terms

class

game loop

instances

method

massively multiplayer online role-playing game (MMORG)

model

object-oriented programming (OOP)

simulations

9780170442787

IN ACTION

Virtual songlines

Games can open doors to new worlds. They can help us better understand the world around us, the world of long ago, or the distant world of an unfamiliar culture.

But how do you pass on these rich and meaningful stories to younger audiences who are accessing information in new and exciting ways?

As an engaging and immersive medium, videogames offer a key to sharing these stories.

Yarrer Gunditj woman and RMIT student Phoebe Watson and *Gamefroot* founder Dan Milward are working in the Indigenous game creation space to engage players with Indigenous stories and to share cultural knowledge.

Bilbie XR Labs, founded by Brett Leavy, designs and produces *Virtual Songlines*, a virtual world that creates simulated experiences of ancient Indigenous culture. Modern cities loom as ghostly overlays on original landscapes, all carefully researched. While exploring bush tracks players encounter scenes of hunting, camp life, fishing, dancing and play, all realistically recreated (Figure 14.1).

Among other projects, they have made survival games where you must learn cultural practices to survive. The creation of these realistic games goes beyond just entertainment and enjoyment. It has created a new format for storing information about Aboriginal and Torres Strait Islander cultures and gives anyone, including First Nations people who didn't have the opportunity to be taught about their culture or who don't know anything about their own connection to Country, the opportunity to experience first-hand what it was like to live before colonisation.

Phoebe Watson: 'Video games have the advantage of being more accessible than traditional games, as they have less mobility, geographic, or social bounds that prevent people from engaging. Video games can play an important role in preserving and teaching culture, and in closing the gap between Indigenous and non-Indigenous people.'

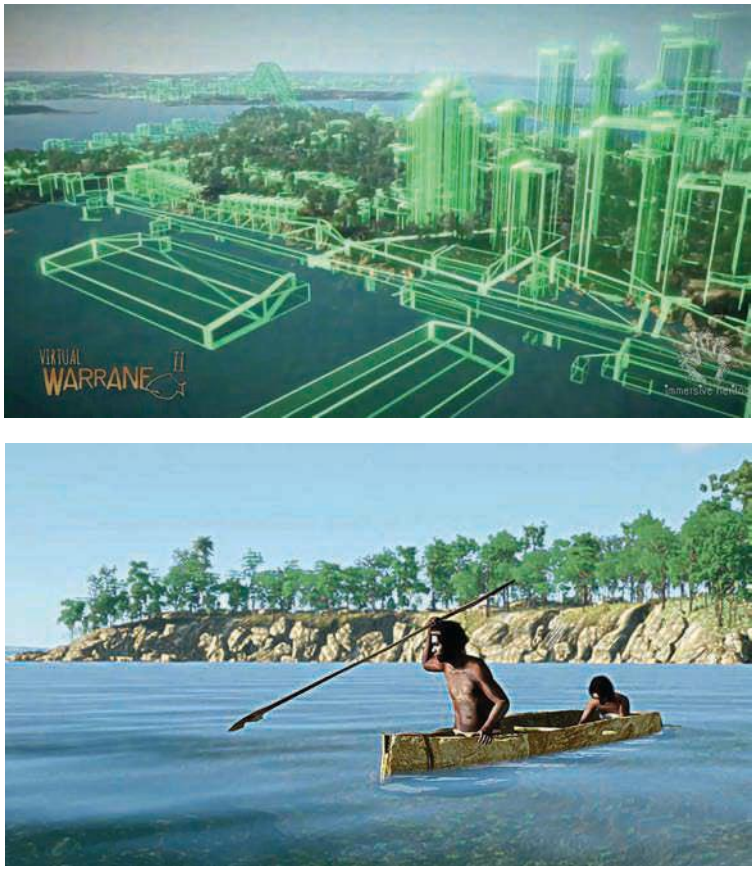
Source: Phoebe Watson and Dan Milward, 'Old stories, new tech: Indigenous storytelling in videogames', ACMI, 30 September 2019.

Questions

1. First explore the video walkthroughs and 360-degree scenes at *Virtual Songlines* and watch a walkthrough of a virtual journey around Sydney Cove at the time before the arrival of the First Fleet, *Virtual Warrane.*
2. Comment on the effectiveness of *Virtual Songlines* in 'preserving and teaching culture, and in closing the gap between Indigenous and non-Indigenous people'.

Weblinks
Virtual Songlines

Path 3 - Sydney Site



Virtual Songlines

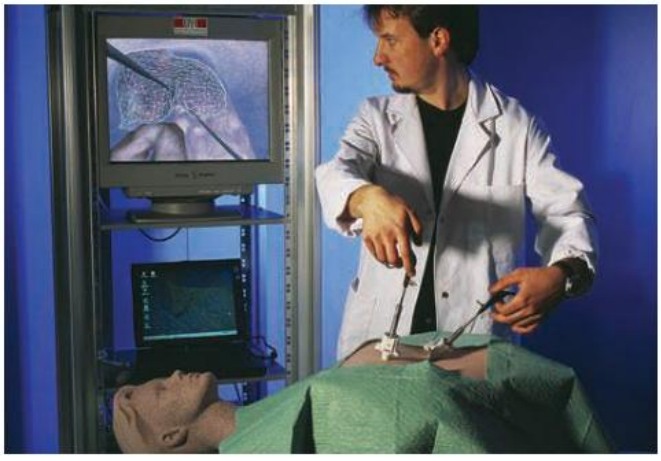
Figure 14.1 Scenes from *Virtual Warrane*, part of the *Virtual Songlines* VR experience.

14.1 Modelling and simulations

In this chapter you will begin by studying **simulations** leading to the creation of your own game-based simulation. This chapter offers practical experiences through a number of Skill builders and two projects. In the first project you build a virtual pet using a micro:bit and in the second, you create an interactive RPG adventure game using **object-oriented programming (OOP)**.

Skills and techniques used in games design, especially in those involving virtual worlds, are strongly related to the creation of simulations.

Computer models and simulations are used in diverse areas to help us gain a better understanding of the world in which we live. Will climate change increase the number and severity of cyclones in northern Australia? Is there likely to be another global financial crisis? Computer simulations and models help us make predictions and gain insights into natural, human and technological systems that would not be possible using traditional non-computerised methods (Figure 14.2).



Science Photo Library/Pascal Goetgheluck

Figure 14.2 Simulation devices are used to train surgeons and other medical professionals.

What is a model?

A **model** is a simpler version of a real-world object or idea. A model can be relatively simple, like a model aeroplane; or something more complicated, like a model of the Earth's climate. Modelling (creating a model) is a basic and common human activity, and from an early age many people learn to represent objects as a model. As well as being fun and enjoyable, building a model enables you to gain a better understanding of the object in question.

What is a computer model?

A computer model can generally be described as an electronic imitation or copy of an object or idea. Computers can be used to create models of all sorts of things, including cars, buildings, aeroplanes, hair dryers, personal budgets, bowling balls and kitchens (Figure 14.3). They have become an important tool in areas such as business, science, entertainment and education. The enormous improvements in weather forecasting over the past 30 years are due to sophisticated models and simulations as well as the expertise of meteorologists (Figure 14.4). The Australian Bureau of Meteorology uses models and simulations to produce regular weather reports and to predict unusual events such as cyclones.

Figure 14.3 Designing and engineering: Computer models are used to design, build and test a range of structures, including bridges.

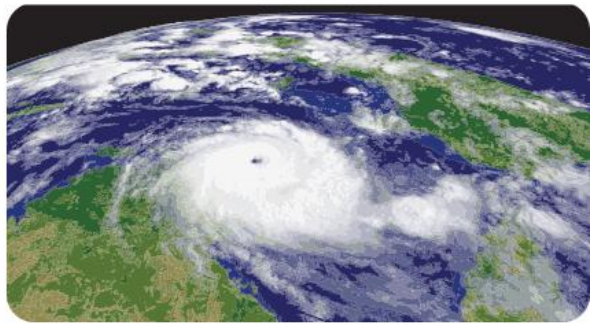


Figure 14.4 Weather forecasting: The enormous improvements in weather forecasting over the past 30 years are due to sophisticated models and simulations as well as the expertise of meteorologists.

Figure 14.5 Training and education: Medical simulations are used for virtual exploration of the human body and real-time interactive training programs for medical devices and surgical procedures.

Figure 14.6 Models and simulations are used in a range of areas.

What is simulation?

Simulation is the process of imitating real-life occurrences in order to test a model. A computer simulation, therefore, generally tests a computer model.

Testing the computer model of a house to see what would happen to it in an earthquake is an example of a computer simulation. Testing the structure of a computer-generated model of a bridge by running a computer-generated train over it is another example.

Purposes of models and simulations

The main purpose of a model is to allow us to establish an understanding of an object or an idea. The main purpose of a simulation is to test a model in order to predict how the real object or idea will react under different conditions (Figures 14.5 and 14.6).

ACTIVITIES

Identify

1 Identify the benefits of creating a model.
2 Identify the main purpose of a model.
3 Provide a definition of simulation.

Analyse

4 Explain how simulation is used in the design of structures such as houses and bridges.
5 Explain the relationship between a model and a simulation.

Investigate

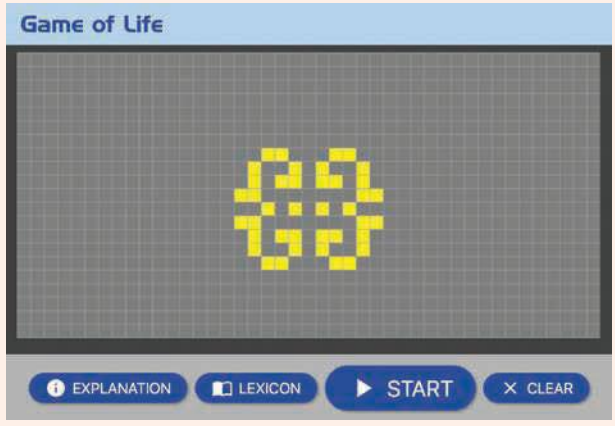
6 Investigate the purpose of models and simulations in one of the following areas: pollution, coastal erosion, population growth, vehicle safety.
7 Research and provide a definition of emulation.

KNOWLEDGE BUILDER

John Conway's *Game of Life*

In this Knowledge builder you will be playing and researching the *Game of Life*, a famous example of artificial intelligence (AI), which is both a game and a simulation. The *Game of Life* is really a 'cellular automaton', and was invented by Cambridge mathematician, John Conway.

This game has become widely known and was first discussed in an article by Martin Gardner and published by *Scientific American* magazine in 1970. The game consists of squares in a grid (Figure 14.7) which are either populated (yellow) or empty (grey). By starting the game, a square either disappears (dies), populates (is born) or stays where it is (and lives!). The game can be simulated on grid paper using pencil and eraser but a computer makes the simulation far more speedy and practicable.



Source: https://playgameoflife.com/

Figure 14.7 John Conway's *Game of Life*.

The four rules of the *Game of life*

The game is based on a 'Goldilocks principle' – a population size must be 'just right' – neither over nor under-populated.

After colouring squares to create a first 'seed' population each subsequent generation obeys the following rules:

- any filled-in square with one or no neighbours dies, as if by loneliness
- any filled-in square with four or more neighbours dies, as if by over-population
- any filled-in square with two or three neighbours survives.
- any square that is empty becomes populated if it has three neighbours.

Questions

Weblink
Game of Life

1 Open the *Game of Life* on your computer or mobile device using the weblink provided. Think about the rules and understand them clearly. Experiment and confirm each of the rules.

2 Starting with the first pattern in Figure 14.8 (the cross), test the accuracy of the nine following generations by drawing the first pattern as your 'seed' population and then figuring out each following generation on grid paper. What happens beyond the 10th generation?

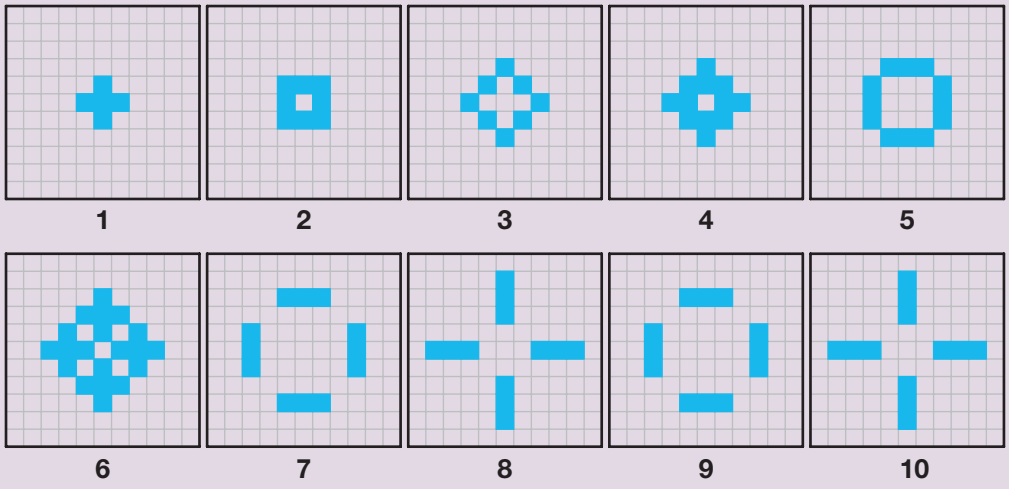


Figure 14.8 Generations of a seed pattern in the shape of a simple cross in the *Game of Life*.

3 Experiment with different patterns and see what happens in future generations.

4 In what way is the game an example of AI?

5 Experiment with predefined patterns using the drop-down menu. Record what happens to each of these.

6 Research the *Game of Life* and try to create a starting pattern to maximise its lifespan. It is like creating a race of people and keeping them alive for as long as you can! Discover how John Conway first came to invent this game.

7 What development happens most often in your game – do generations of creatures die out, stay static, grow larger or endlessly alternate?

- **a** Can you find a starting pattern that always dies out, no matter how many separate organisms there are to start with?
- **b** Can you find a pattern that stays exactly the same?
- **c** Can you find a pattern that switches between patterns; that is, a pattern that changes to another in the first step and then changes back to itself?
- **d** Try the following well-known patterns using the drop-down menu and state what happens to each:
  - pulsar
  - blinker
  - toad
  - glider.

8 Give your opinion about this quote from mathematician, Paul Callahan:

'We can see that [the Game of] Life, simple as it is to describe, exhibits much of the complexity of our own universe.'

9 Do you think you could code a computer program to play the *Game of Life*?

ACTIVITIES

Identify

1 Name three software games that are simulations.

2 State some similarities between games and simulations.

3 Identify a spreadsheet tool that can be used in simulations.

Analyse

4 Explain why John Conway's *Game of Life* can be considered a simulation game.

5 Outline three benefits offered by simulation games over their real-life equivalents.

6 Discuss the potential consequences of ultra-realistic VR-based simulation games.

Extend

7 Outline how a spreadsheet can be used as a simulator and explain how one of its features can be used for prediction.

8 Choose a computer simulation game used in industry training (e.g. aeronautics, military, retail) and outline its operation and claimed benefits.

14.2 The spreadsheet as a simulator

It is common to use spreadsheets as a simulation tool, particularly when a large data set is involved. If a model of a real-life situation can be controlled by a set of rules using mathematical formulae then a spreadsheet can first model and then predict what might happen, based on data you have or might generate.

To do this it uses a powerful feature called *Goal Seek*. A simple example could be the calculation of profit calculator from the sale of items. A company wants to know how many items it must sell to make a certain profit. The following Knowledge builder will show you how to do this.

KNOWLEDGE BUILDER

Using *Goal Seek* in a spreadsheet

Profit is calculated using this simple formula:

Profit = Units_Sold * (Selling_price – Cost_price)

We know the cost of a single unit and the selling price and we want to predict how many items must be sold to reach a goal.

Let us suppose we buy second-hand video games for $20 and sell them at 20 per cent profit –our selling price will be $24. We are aiming to save up $500 to buy a new phone. How many video games will we need to sell?

Fill in a spreadsheet with formulas to do calculations as shown in Figure 14.9.

Describe what each of these two cell formulas does.

	A	B
1	**Simple goal seeking example**	
2		
3	**Number of units**	
4		
5	**Cost price**	
6	**Selling price**	
7		
8	**Profit**	**=B6-B5**
9		
10	**Total revenue**	**=B8*B3**
11		

Figure 14.9 Set up your spreadsheet.

Of course, it is easy just to calculate directly, but using a spreadsheet we can keep modelling different scenarios without doing calculations! With more complicated scenarios, a spreadsheet can be even more fast and efficient.

Now add the data for 100 items. See Figure 14.10.

Spreadsheets have a tool known as 'goal seeking', which turns a simple spreadsheet like this into a simulated model of this aspect of the company and so becomes a prediction tool.

	A	B	C
1	**Simple goal seeking example**		
2			
3	**Number of units**	**100**	
4			
5	**Cost price**	**$20.00**	
6	**Selling price**	**$24.00**	
7			
8	**Profit**	**$4.00**	
9			
10	**Total revenue**	**$400.00**	
11			

Figure 14.10 Enter data.

1 Select *Goal Seek* from the menu bar of Excel *Tools > Goal Seek* (see Figure 14.11). If using Google Sheets, you will need to install its free extension named *Goal Seek.* Use *Extensions > Add-ons* from the menu bar to download and install this.

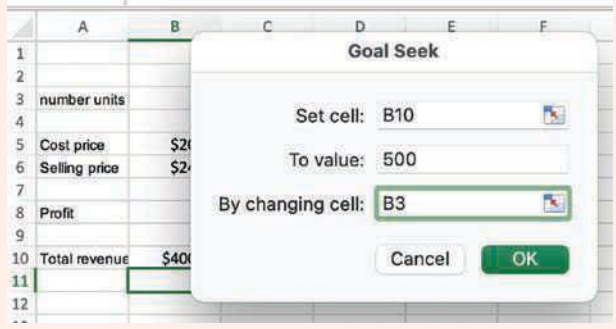


Figure 14.11 Use *Goal Seek*.

Suppose we wish to find out how many units we need to sell to make $500.

2 Enter B10 in *Set cell*:
Enter $500 in *To value*:
Enter B3 in *By changing cell*:
Press OK.

3 How many second-hand game units will you need to sell?

9780170442787

SKILL BUILDER

A chessboard problem simulation

Wikimedia Commons/McGeddon/CC BY-SA 4.0

Figure 14.12 The chessboard challenge.

In this Skill builder you are invited to use the power of the spreadsheet to answer a challenge first recorded in the 1300s. There are various versions, but in one the inventor of the game of chess requests payment from his ruler as follows:

The chessboard is to be filled with rice according to the following pattern: One single grain on the first square of the board, two on the next square, four grains on the next, and so on – doubling the number of grains for each square until the last one is filled (Figure 14.12).

(It is not recorded whether the inventor received a reward or was executed!)

1. Conduct an Internet search to find how many tonnes of rice were produced globally last year and record this. The easiest way to do this to use data from the Food and Agricultural Organisation (FAO) of the United Nations.
2. Try entering an integer in Excel or Sheets with more than 15 digits. Observe that any further digits appear as zeros. Find out why this limitation exists.
3. Your task in this project is to design and build a spreadsheet to find out if the rice challenge task is possible. Imagine that the chessboard is a huge one so that grains will fit inside each square, no matter how many there are!

Your spreadsheet should:

- show the number of grains on each square of the chessboard
- show the weight in metric tonnes on each square of the chessboard
- show the total grains on the chessboard
- show the total weight of the grains on the chessboard
- use conditional formatting to automatically have cells change colour and print a message on rows where the annual world rice harvest is exceeded
- calculate the cost of the rice in Australian dollars (AUD) for each square and for the total.

Use the following where required in your formulas:

- each grain weighs 0.025 grams
- convert USD to AUD using the current exchange rate
- there are 1000 grams in 1 kilogram and there are 1000 kilograms in a tonne.

4. Design the layout of your spreadsheet simulator, add columns and headings, enter the formulas required and add conditional formatting. You will need to use 'Fill down' to do this.
5. Build the spreadsheet and compare your answers to those of other students. Did you know spreadsheets have upper limits on the size of numbers they can work with and display? What is your conclusion?

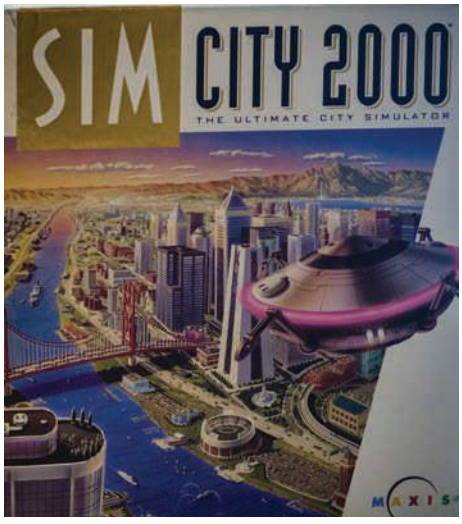
Chessboard square	Number of grains	Weight in metric tonnes	Value in AUD
19	262144	0	$4.39
20	524288	0	$8.78
21	1048576	0	$17.56
22	2097152	0	$35.13
23	4194304	0	$70.25
24	8388608	0	$140.51
25	16777216	0	$281.02
26	33554432	1	$562.04
27	67108864	2	$1,124.07
28	134217728	3	$2,248.15
29	268435456	7	$4.496.29

Figure 14.13 Extract from the simulation spreadsheet.

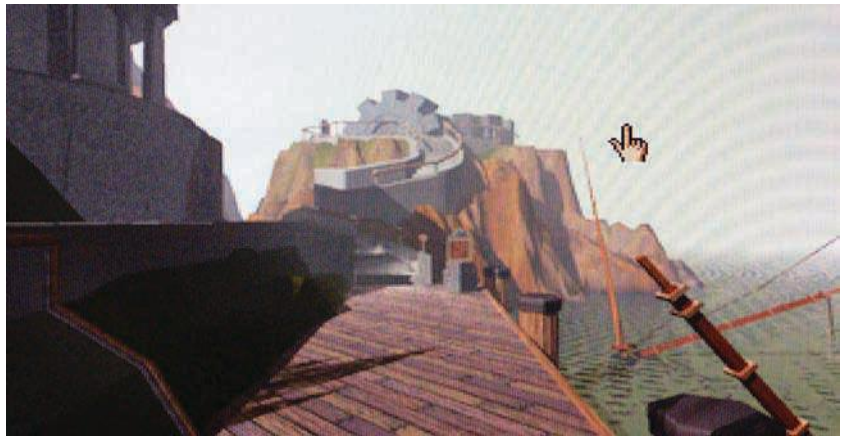
14.3 Games and simulations

Every computer game can be thought of as a simulation. Flight simulators are used for pilot training, battle scenarios are used to train soldiers and racing car games are popular forms of entertainment. Even if they do not involve real-world scenarios such as these, they are bringing an algorithm to life. *SimCity* is one of the most popular simulation games of all time, having sold over 5 million copies. There have been multiple editions of this simulation game but they all rely on the same principle: a player develops a city on undeveloped land and defines residential zones for houses, commercial zones for shops and offices and industrial zone for factories and public services such as schools, parks and fire stations. The player acts as mayor and sets taxes and budgets. Gameplay can centre on sustaining the city for the longest time, creating an appealing environment or making as much money as possible (Figure 14.14).

An example of a practical application of simulation involving virtual reality (VR) can be seen at UNSW's Advanced Visualisation and Interaction Environment (AVIE) VR theatre, where participants identify mining safety risks to earn points. A simulation of an underground coal mine section shows a series of hazards in the environment and trainees must answer questions and select the correct course of action to control the hazard (Figure 14.15). See it featured in the In action section at the start of Chapter 7.

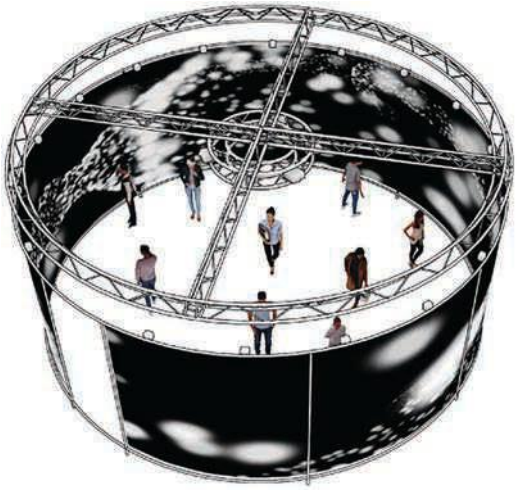


Shutterstock.com/melissamn



Alamy Stock Photo/ArcadeImages

Figure 14.14 Ground-breaking early computer games titles.



iCinema Centre

iCinema Centre

Figure 14.15 Simulated mining training at UNSW's AVIE VR theatre.

SKILL BUILDER

Simulating a game of chance using a spreadsheet

Shutterstock.com/Andrei Shumskiy

Figure 14.16 Your spreadsheet will be a lot faster than tossing these 1000 times!

Identifying and defining

In this Skill builder we simulate a game between three players, Olga, Jin and Elijah, using a spreadsheet. Two dice are tossed:

- if the *product* of the upper faces is a multiple of 6 then Olga wins
- if the *sum* of the upper faces is a prime number then Jin wins
- if a 6 or a double are tossed then Elijah wins
- if there is a tie then no one wins.

The question we need to answer is: Which player is more likely to win?

Researching and planning

The approach we will take is to first simulate realistically a single toss of the dice using a spreadsheet, extend the simulation to 100 tosses and finally to 1000 tosses. In doing this, we will see how useful the spreadsheet is in helping us solve the problem (Figure 14.16).

Before you start, use an Internet search to find the maximum number of rows permitted by your spreadsheet software.

Simulating a toss

1. Use two real dice between three people and run a few trials amongst yourselves using the rules above. Use a simple table drawn on paper to record results. Take a guess at which player's rule is the most favourable.
2. What built-in spreadsheet formula could be used to simulate the toss of a die?
3. Design a layout for your spreadsheet. Use the first column to write the number of the trial. Use only one row to start. Keep the next two columns free for the numbers resulting from tossing the dice. Add these headings to each of these first three columns: Toss, Dice 1, Dice 2.
4. In cell B2, use the built-in = RANDBETWEEN( ) formula to simulate the toss of Die 1 and fill right to cell C2 for Die 2. At this stage we will only perform a single toss (row 2) of the dice using the spreadsheet.

Improving the UI: Simulating a physical die using a spreadsheet

5. It is possible to get both Google sheets and Excel to create a simulated image of the top face of the dice! The secret is to notice that dice have a grid of nine dots. Set up a 3 x 3 grid of cells in columns to the right to imitate the top face of a die (you will need to narrow the column and rows). See Figure 14.17.

First, you need to find a symbol for a dot. There are various ways to get a suitable black dot. One way is to set the cell's font to Wingdings and use `CHAR(108)` in your formula to show that symbol. In the following steps you will use this to create two virtual dice.

Figure 14.17 The final result using Excel with a button and macro for tossing the dice. In place of the button, the spreadsheet can also be manually recalculated to simulate tossing the dice.

Your challenge now is to enter appropriate `=IF( )` formulas in each of the nine cells in a 3 X 3 grid pattern, which will cause them to print dots when they should, based on the number in cell B2.

Look at an actual die to decide the rule for each cell. For example, the centre cell of the die needs to only show a dot when 1, 3 or 5 is thrown. Use `CHAR(108)` to print the dot only if one of these numbers is tossed and to print nothing otherwise. The basic formula will be

```
=IF(OR(test 1, test 2, test 3…),
  CHAR(108), " ")
```

Thus for the centre dot this formula will be:

```
=IF(OR(B2=1,B2=3,B2=5),CHAR(108)," " )
```

Complete the others for both dice.

- Create a second animated die. Add a coloured background behind both dice to improve the UI.
- Retoss the dice by recalculating the spreadsheet.
  - In Google Sheets this is easy: insert a checkbox and click on it to refresh all dice random tosses.
  - In Excel, use the *Calculate Now* button on the ribbon under Formulas tab (small calculator icon) or the F9 key (Mac and Windows) or (advanced) add a button with a macro using Excel Developer tools.

A solution is provided in the Nelson MindTap resources named *dice toss*.

Producing and implementing

Calculating the winners

6 In columns D, E and F create formulas to decide if Olga, Jin or Elijah wins. The following points will help you, if needed:

- For Olga: check all possible products which are multiples of 6 (6, 12, 18, 24, 30 and 36). You need an IF statement to print 1 if Olga wins and 0 otherwise. Can you see how this formula works?

```
=IF(OR(C2*B2=6,C2*B2=12,C2*B2=18,
C2*B2=24,C2*B2=30,C2*B2=36),1,0)
```

- For Jin: check all possible sums which are prime (2, 3, 5, 7 and 11). You need an IF statement to print 1 if Jin wins and 0 otherwise.

```
=IF(OR(C2+B2=2,C2+B2=3,C2+B2=5,
C2+B2=7,C2+B2=11),1,0)
```

- For Elijah: check if either die is 6 or if the two dice are equal. You need an IF statement to print the number 1 if Elijah wins and 0 otherwise. If you need help, a solution is provided in the Nelson MindTap resources named *dice toss simulation*.

Toss multiple times and see if everything is working correctly.

Can you confidently say which of the players will win over time?

Creating a larger data set

7 Simulate 100 tosses of the two dice by filling down the existing formulas.

8 Add formulas at the end of each player's column to total their wins (by adding all the 1s) for the 100 tosses. See if you can print the winner's name in a separate cell and the word 'Tie!' when two or more players come first. If you need help, a solution is provided on Nelson MindTap named *dice toss simulation100*.

9 Which player seems 'luckier'?

10 Extend this now to 1000 tosses of the dice. You will need to adjust the formulas for the totals.

11 Which player seems 'luckier' now? Does this match your prediction at the start?

Testing and evaluating

12 Use the two tables shown in Tables 14.1 to calculate the total number of different ways each player can win. Who has the most ways? Does this match your experimental finding?

13 Write up a brief report on this Skill builder, and list the things you learned.

Extend this Skill builder

- If you have already studied programming, see if you can code this in a general-purpose language such as Python to perform the experiment for 10 000 and even 1 million tosses!
- Modify the game so that one die is tossed alongside one coin. Develop new rules for the three players and construct a spreadsheet to simulate your new game.

Tables 14.1 Tables list all possible outcomes for the dice experiment.

X	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	8	10	12
3	3	6	9	12	15	18
4	4	8	12	16	20	24
5	5	10	15	20	25	30
6	6	12	18	24	30	36

+	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

KNOWLEDGE BUILDER

Real world simulations

In this Knowledge builder you will experience some game-based simulations that address real-world problems.

Bushfire simulations

1 Use Table 14.2 and create your own spreadsheet comparing two bushfire training simulations by scoring each category out of 5. Determine the intended audience for each.

Select your own simulations online or use these examples:

- Darwin Rural Fire Simulator
- Concord Consortium.

Pandemic simulations

2 It is not always necessary to build simulations using sophisticated tools. In the link provided, examine how a designer has used the simple visual coding app *Scratch* to simulate the effects on the transmission of a virus such as COVID-19 in a community of people.

Try it out using the weblink and examine the code behind it.

Table 14.2 Bushfire training simulation comparison template.

	Simulation 1	Simulation 2	Comments
Ease of use			
Intuitiveness of interface			
User interface design			
Mechanics			
Accessibility			
User experience			
Educational value			

Weblinks
Darwin Rural Fire Simulation
Wildfire Explorer
Epidemic Simulation

SKILL BUILDER

Coding a micro:bit

You will be using the micro:bit computer in a number of activities throughout this course and in the guided project that follows. In this Skill builder you code a micro:bit board and encounter three computer languages: Blockly visual code, JavaScript and Python. However, you do not need to have an actual micro:bit to complete this Skill builder as the MakeCode website includes a realistic simulated one.

We start with *Blockly* visual code and then code in Python.

The goal is to bring your micro:bit 'to life' by lighting up a pattern of its 25 LEDs (light emitting diodes) to create a pulsing heart! Get ready to perform a heart transplant!

Flashing heart using a sequence structure

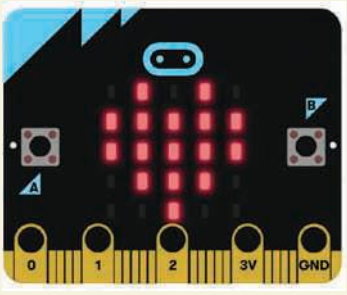
Here you create a simple 'flashing' heart. The code wraps a sequence of code inside an endless loop to keep the heart flashing.

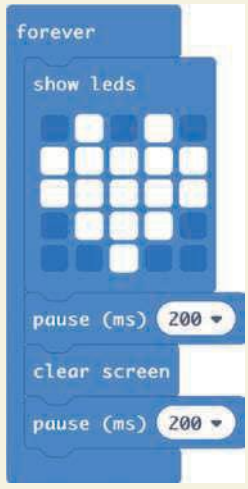
1 Open the MakeCode website and select the MakeCode editor.

2 Reproduce the code block shown in the *Blockly* panel of Figure 14.18. Check the code is working correctly by inspecting the built-in on-screen micro:bit simulator.

3 In the MakeCode visual Blockly editor, select the JavaScript tab and inspect its code.

Weblink
micro:bit Microsoft MakeCode





Visual Blockly

```
from microbit import *

while True:
    display.show(Image('09090:99999:99999:09990:00900'))
    sleep(200)
    display.clear()
    sleep(200)
```

Python

Figure 14.18 Visual and general-purpose programming languages producing a pulsing heart on a micro:bit.

4 To code the micro:bit in Python, use Google's Chrome browser to open MakeCode's Python editor. This browser allows pairing the micro:bit for easy downloads.

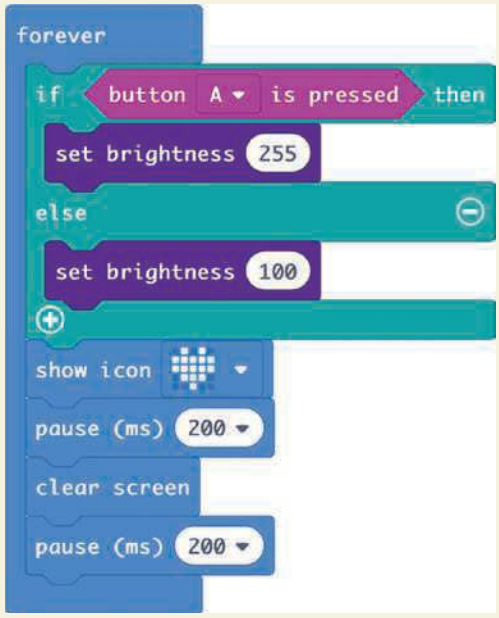
   Copy the Python code shown at right in Figure 14.18 then send this code to an actual micro:bit by connecting your micro:bit via USB (click the three dots beside the *Send to micro:bit* button) or use the built-in simulated micro:bit.

5 Compare the keywords used by the three languages for *pausing*. What fraction of a second is the pause in each?

6 By comparing the Blockly heart pattern in Figure 14.18 with the red Python code, write out an explanation, suitable for a user manual, describing the Python syntax for lighting LEDs.

Selecting a heart using a selection structure

7 Create the code shown in the Visual Blockly panel of Figure 14.19. Check your code is working correctly by holding down button A on the built-in micro:bit simulator.



Visual Blockly

```
from microbit import *

while True:
    if button_a.is_pressed():
        display.show(Image('09090:99999:99999:09990:00900'))
    else:
        display.show(Image('02020:22222:22222:02220:00200'))
    sleep(200)
    display.clear()
    sleep(200)
```

Python

Figure 14.19 Pressing button A causes your 'heart' to beat more strongly!

8 In the MakeCode Visual Blockly editor, select the JavaScript tab and inspect its code.

9 Use your Python editor to code the Python version in Figure 14.19.

10 Compare the different ways these three languages code the *selection control structure*.

11 How does the Python code make the heart appear fainter?

Pulsing heart using a loop control structure

You have already been using the loop structure in the previous examples as you needed to keep your heart flashing. Here you use a for loop (a counted loop) to make your heart's pulse far more realistic!

12 Using your Python editor copy the following code. (Before running the program and before reading further, describe what you think it will do.) Run the code.

```
from microbit import *

heart = "0x0x0:xxxxx:xxxxx:0xxx0:00x00"

while True:
    for count in range(0, 10, 1):
        display.show(Image(heart.replace('x', str(count))))
        sleep(200)
    for count in range(9, -1, -1):
        display.show(Image(heart.replace('x', str(count))))
        sleep(200)
```

We first defined an image for a heart with each pixel's brightness described using an 'x' instead of a number. We then used counted for loops (introduced earlier in this chapter). Describe how this program works using what you learned about the range function in the previous section.

(Spoiler alert!) The range function in Python works like this: `range(1, 10, 2)` counts *up* from 1 to 9 in steps of 2. If the step value was negative it would count *down*.

Extend this project

- Modify the code to make the heart beat faster.
- Modify the code so that pressing button A makes the heart smaller. (Hint: define an additional smaller heart pattern.)
- Modify the code so that pressing button B makes the heart beat faster.

14.4 Guided project 1: A virtual pet

Project resource
A virtual pet

Lots of us acquire a pet at some point in our lives but few of us get to create our own! In this project, you create your own personal buddy. After creating it, your aim is to keep it alive.
Imagine your creation as a prototype for a prospective toy manufacturer. Your design will demonstrate your concept to the manufacturer or a potential financial backer.

KNOWLEDGE BUILDER

The Tamagotchi

A Tamagotchi is like a digital or virtual pet (FIgure 14.20). Users feed it, play with it, give it drinks and much more. The goal is to have it grow up into a healthy adult! The name Tamagotchi is from the two Japanese words for 'egg' and 'watch'; however, together they sound like the Japanese word for 'friend'.

Alamy Stock Photo/jeremy sutton-hibbert

Figure 14.20 The popular Japanese toy Tamagotchi was first marketed in 1996.

Questions

1. Conduct some research online into the history of Tamagotchis and also to understand how they work.
2. App versions of the toy now exist for tablets and smartphones. Prepare for this project by downloading a free version of the My Tamagotchi Forever app and play it on a smartphone or tablet.

Weblinks
MindMup

micro:bit
Python editor

Identifying and defining

What does your pet need?

You will give your pet five everyday needs – the kinds of things required to stay alive or be happy. These might be food, comfort or friendship. As your pet's master, you will use the language Python to program a BBC micro:bit so you can meet its needs.

Think about the UI too. How will you let the user know they have fed their pet? Users will need to know how healthy their pet is. How will you provide this feedback? Would sound effects be useful for some feedback?

Also, a micro:bit doesn't look much like a pet. Can you design and build a body to hold your micro:bit so it appears more like a real creature?

1. Games designers find a diagram like that shown in Figure 14.21 helpful when working on a concept for a new game.
   Copy it or use the free MindMup (see the weblink) and complete the boxes by answering the following questions:
   - How does your game begin? (starting points? lives? location?)
   - What is the main character? (describe features)
   - What is the goal of the game? (what is the player trying to do?)
   - What is the reward for achieving this goal? (an animation? points? extra lives?)
   - What happens if the player fails?
   - What are the challenges/obstacles?
   - What do you do to win?
   - What do you do to lose?
2. Use what you have learnt from the Knowledge builder activity (opposite) to select 4–5 needs your virtual pet will have programmed into it (such as food, discipline).
3. State a goal for the game.

Researching and planning

Game design

Design thinking

4. When planning a project, it is important to know which technologies are suitable for achieving your goals. You will be using the BBC micro:bit as your hardware (Figure 14.22) and MicroPython for your programming software. MicroPython is simply Python with 'libraries' added to communicate with a micro:bit.
   Use the micro:bit's Python editor (see weblink) to write your Python code.

9780170442787

Design thinking

Figure 14.21 A useful diagram to help design a game.

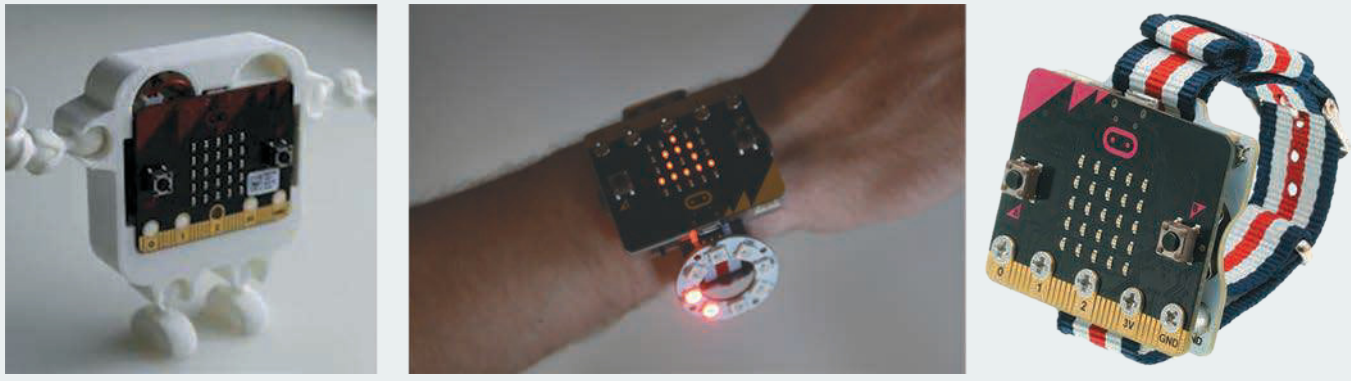
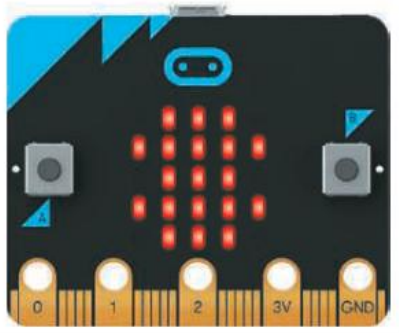
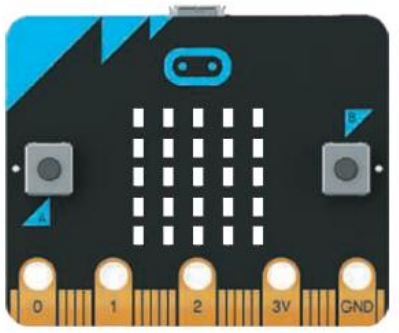
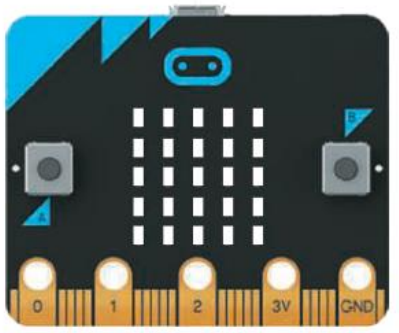
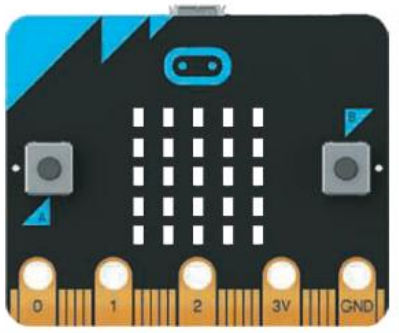
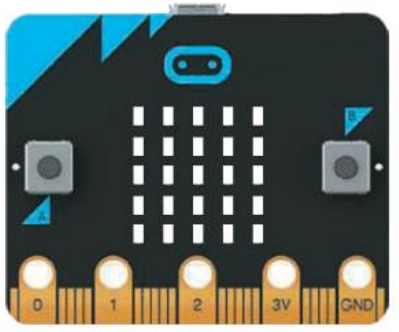


Figure 14.22 You can add features to your micro:bit to make it look more like an actual Tamagotchi.

Design the UI and user experience

5 IPO tables, like the one following, are used in planning computer programs. Copy and complete the table at least for food and water. Design icons for each of your animal's needs by shading the white dots of the micro:bit LED displays.

Input		Processing	Output		
Input variable	**Input control**	**Algorithm/rule**	**Display description**	**Animated?**	**UI display**
Food	The A button	Add 1 to food	Icon of hamburger	No	
Water	The B button	Add 1 to water	*Design your icon*	No	
Fitness	Choose a micro:bit gesture	Add 1 to fitness	*Design your icon*		
Your own variable	Choose a micro:bit gesture	Design a rule	*Design your icon*		
Your own variable	Choose a micro:bit gesture	Design a rule	*Design your icon*		

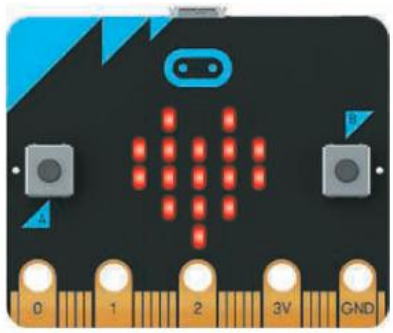
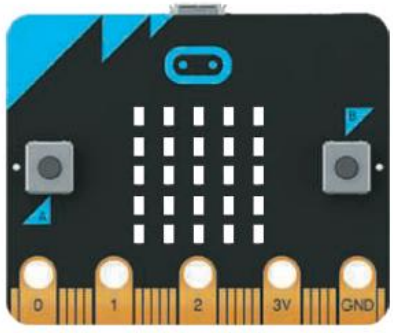
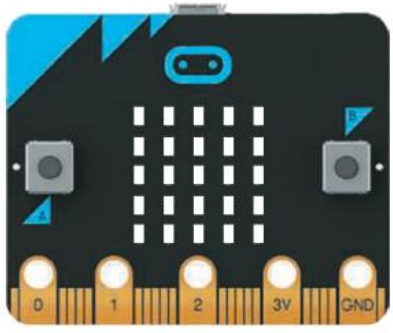
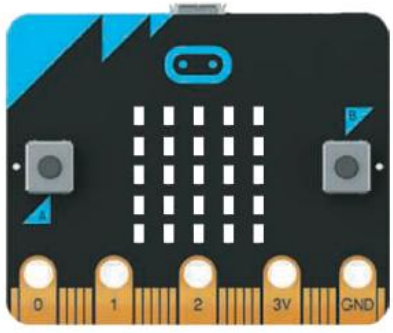
6 Players need to check the health of their pet and be warned when it gets unhealthy. Designing clear feedback, messages and alerts is important in software development. Users interact through your program's UI. This is called user experience (UX).

UX design includes decisions on colour scheme, audio feedback and navigation. Review the UI design principles in Chapter 7.

Copy and complete the following table to help plan your UX.

Design thinking

Computational thinking

Status		Processing	Output	
Status	**User access?**		**User feedback**	**User interface display**
Alive	*Automatic*	`while health > 0`	LED displays pulsing heart	
Health number	Press A + B buttons	`Health = food + water + ... ?` *Complete this rule using your pet's other 'needs'*	LED displays health status as integer	
Health warning	*Automatic*	*You decide*	*You decide*	
End of life	*Automatic*	*You decide*	*You decide*	

7 If this is your first time programming an image to display on a micro:bit, you should now complete the 'Skill builder' immediately preceding this project. You should also have completed Chapter 4. By now you have likely become quite familiar with Python.

MicroPython code for a microprocessor such as a micro:bit is in two parts. The first part of the code is for setting up and is run only once when the program first starts. The second part is a loop that keeps repeating. See a typical example at the very end of the Skill builder immediately before this project. Also, when you open micro:bit Python editor (see weblink) the default 'Hello' code is in this form.

Weblink
micro:bit Python editor

Producing and implementing
Starting to code

Use the micro:bit Python editor to develop your code. Here are the steps you will follow:

- Create variables to hold the values of your pet's needs (e.g. food and water).
- Create images or animations for each of your pet's needs and its health.
- Create a rule for your pet's health and a way to see it.
- Assign input controls to allow users to care for their pet and controls to allow users to check its health.
- Build a game engine that changes values while your game is running.

Create variables to hold values of the pet's needs

Computational thinking

8 We will only program hunger and thirst and track health in this outline and leave the rest for you. A happy pet will have values of 5 for both hunger and thirst.

Note that all MicroPython code for use on a micro:bit must begin with a line that imports the micro:bit Python files. We will also need the random function later, so import both of these now.

Start your code by typing this into the micro:bit Python editor:

```
from microbit import *
import random

food = 5
water = 5
```

9 Create images for each of the pet's needs. Create an image of a hamburger to represent food and of a water droplet to represent water. These are each single lines of code. Here are suggested designs:

```
img_food = Image("09990:99999:09990:
  99999:09990")
img_water = Image("00900:09090:90009:
  90009:09990")
```

Add a heartbeat

Create an animation to show your pet is alive. Use an animated heartbeat to show this. Defining it as a function will save writing it out in full each time you need it. Call this function `alive()`. Add this function to your code.

We have used a basic animation here, however, you can create your own for the alive animation function. Your program should now look like this:

```
from microbit import *
import random

food = 5
water = 5

img_food = Image ("09990:99999:09990:
  99999:09990")
img_water = Image ("00900:09090:90009:
  90009:09990")

def alive():
   bold_heart = Image ("09090:99999:99999:
      09990:00900")
   faint_heart = Image ("02020:22222:
      22222:02220:00200")
   heartbeat = [faint_heart, bold_heart]
   display.show(heartbeat, delay=100)
```

Later, in our main game loop, we will be using the following rule to keep track of the pet's health (don't use it in your code yet):

```
health = food + water
```

10 Write a function called `status()` to display the image of the hamburger followed by the food value, and the same for the water, which we will trigger when *both* buttons A and B are pressed at once.

Note the *values* for food and water are integers and since the MicroPython display only works with strings, we need to add `str()` to convert these.

Add this new function to your code:

```
def status():
    display.show(img_food)
    sleep(400)
    display.show(str(food))
    sleep(400)
    display.clear()
    sleep(1000)
    display.show(img_water)
    sleep(400)
    display.show(str(water))
    sleep(400)
    display.clear()
    sleep(1000)
```

11 Sadly, you need a function to display the 'passing of our pet' when its health drops to 0. You might play an animation or scroll a text message. For an animation, display changing LED patterns with a short pause between each one.

We are leaving this for you! The previous Skill builder should help here.

Name your function `the_end()` and add it to your code as shown.

```
def the_end():
    your animated images code goes here!
```

Assign input controls

12 Devices such as the micro:bit (and Arduino and Raspberry Pi) typically use code separated into two parts. You have now completed the first part. This first part runs once only. It is used to import libraries needed by our code (such as the random library in our case), initialise variables and define functions. The second part is the working program and usually this is enclosed in a loop to make it repeat endlessly.

We have not yet written the main looping section of the program. In games design this is called the game engine. We now create a loop that endlessly repeats by using a `while True:` statement as follows. Also, the time has come to calculate the pet's health. Type this code and add internal comments to explain what each part is doing.

```
while True:
    health = food + water
    a = button_a.was_pressed()
    b = button_b.was_pressed()
    sleep(200)
    if a and b:
        status()
    elif a:
        if food < 5 and food != 0:
            display.show(img_food)
            food = food + 1
            sleep(400)
            display.clear()
    elif b:
        if water < 5 and water != 0:
            display.show(img_water)
            water = water + 1
            sleep(400)
            display.clear()
    else:
        if health == 0:
            the_end()
        else:
            alive()
```

Run the code. Try pressing both buttons A and B together to see the health status. Work through the following explanation of this code and make sure you understand each step:

- We begin our loop with `while True:`, which forces all the code following it to loop repeatedly, as `True` is always true!
- We set variables `a` and `b` to capture button presses at the beginning of the loop.
- An `if` statement checks if *both* buttons A and B have been pressed at the same time and, if so, we display the health status of our pet.
- If only button A has been pressed, we feed our pet. If only button B has been pressed, we give our pet a drink.
- We then increase each of these values by 1, but to a maximum of 5. If no buttons have been pressed but health has run down to 0, our pet's life ends and we call `the_end()` function; otherwise, we show our pet is alive by calling the `alive()` function.

On your own: If food or water are already at their maximum of 5, then nothing happens if button A or B is pressed. Why has it been programmed this way? Can you improve user feedback by changing the code so a tick shows whenever they are at a maximum?

Game engine

13 Our pet's health never changes, so this is a very boring game but a great life for your pet! We need to reduce health. What are some algorithmic ways you could invent to cause your pet to become hungry or thirsty? Here are some ideas:

Computational thinking

- Reduce values over time.
- Reduce values randomly.
- Reduce values every pass through the loop.

Here we will choose to reduce them randomly. We previously imported Python's random maths module. Every time through the loop we will generate a random integer between 0 and 1000 using:

```
chance = random.randint(1, 1000)
```

When an integer less than 20 is generated, we will make our pet hungrier by reducing food value by 1 and we will reduce water value every time an integer greater than 960 is generated. However, we must make sure not to reduce these below 0! If the pet's life is too short, change these two values.

The code for this follows. We place this directly under `alive()` at the end of the code and in line with its indent. Make sure its indentation is correct.

Carefully examine the code and add internal comments to explain what is happening in each part:

```
chance = random.randint(0, 1000)
if chance < 20:
    if food > 0:
        food = food - 1
elif chance > 960:
    if water > 0:
        water = water - 1
```

Testing and evaluating

Debugging your code

Type the code you have developed and download then run it. Adjust your chance algorithm that reduces food and water, so that hunger and thirst are created often enough. Fix errors. Remember, you have only programmed food, water and health status.

```
from microbit import *
import random

food = 5
water = 5
# add 3 more needs here
img_food = Image("09990:99999:09990:99999:09990")
img_water = Image("00900:09090:90009:90009:09990")
# additional code here for more images

def alive():
    bold_heart = Image("09090:99999:99999:09990:00900")
    faint_heart = Image("02020:22222:22222:02220:00200")
    heartbeat = [faint_heart, bold_heart]
    display.show(heartbeat, delay = 100)

def status():
    display.show(img_food)
    sleep(400)
    display.show(str(food))
    sleep(400)
    display.clear()
    sleep(1000)
    display.show(img_water)
    sleep(400)
    display.show(str(water))
    sleep(400)
# add code here for additional needs
    display.clear()
    sleep(1000)
```

You invented your own `the_end()` function earlier, so you can use that now. We continue the code you have completed in what follows but with an advanced version of the end function. You can try this version or use your own. Can you describe what is happening here?

```
def the_end():
   heart_fade = "0x0x0:xxxxx:xxxxx:0xxx0:00x00"
   for i in range(9, 0, -1):
       display.show(Image(heart_fade.replace('x', str(i))))
       sleep(200)
   display.show(Image("90009:09090:00900:09090:90009"))
   sleep(200)

while True:
    health = food + water # modify this formula for additional needs
    a = button_a.was_pressed()
    b = button_b.was_pressed()
# add code to handle additional needs (use gestures?)
    sleep(200)
    if a and b :
        status()
    elif a:
        if food < 5 and food != 0:
            display.show(img_food)
            food = food + 1
            sleep(400)
            display.clear()
    elif b:
        if water < 5 and water != 0:
            display.show(img_water)
            water = water + 1
            sleep(400)
            display.clear()
# add code here to handle additional needs
```

```
    else:
        if health == 0:
            the_end()
        else:
            alive()
            chance = random.randint(0, 1000)
            if chance < 20:
              if food > 0:
                    food = food - 1
            elif chance > 960:
              if water > 0:
                    water = water - 1
# add code here to handle additional needs
```

Thoroughly test your code and make sure internal comments have been added to show your understanding of each part of the algorithm.

Extend this project

14 Your task now is to add code to handle more needs in addition to food and water. The grey commented sections in the code show where these should go. Some suggestions:

- Life is more than food, water and exercise! Add more needs for your pet. A good one would be `comfort`. Micro:bit has a little-known gesture that recognises a touch or 'stroke' on a pin of less than one second while holding the GND pin by registering a change in resistance on the pin. You may need to moisten your fingers a little – but it works!
- Improve your UX feedback.
  When the user presses button A or B with food or water already at a maximum of 5, nothing happens. Good programmers always give clear feedback. If you have not already done so, add code to display a tick if no food or water are needed yet.
- Add birthdays and make this game much more fun!
  At the moment, all the player does is care for their pet and try not to let it die. Can you code it to have 'birthdays', say every 60 seconds?
  One way to do this would be to add two new variables: `counter` and `day`, both initialised to 0. The counter would increment each time the `while True` loop was executed. Then every 100 (say), it would increment the value of `day` while resetting the counter to 0 and trigger an animation of a candle followed by the number of the birthday.
  A birthday example is provided in teacher resources with console readout for age and a candle animation each birthday!
- Prevent resurrection!
  There is another gameplay problem. Once the pet's life has ended, the user can simply press buttons for food and water and bring it back to life. Modify your code so this is not possible.
- Add 'play again' option.
  Add a gesture (shake?) to allow players to start a new game.
- Create better visual feedback – short animations could replace static images.
- The micro:bit Version 2 can play music!
  Add audio feedback whenever an important event happens by connecting a buzzer or earphones to your micro:bit.
- Adjust the value of the randomly generated `chance` integer to improve the gameplay.

Solutions are provided on Nelson MindTap.

14.5 Designing games using OOP

In this course you learn to code using an OOP (object-oriented programming) approach. Since this chapter is about creating games, it will be a good idea to experience creating one based on OOP principles. The focus of the remainder of this chapter is on using OOP to create a game.
Note: A complete introduction to OOP programming is to be found in Chapter 15 and so this is not provided again here. It is highly recommended working through that section of Chapter 15 before reading further.

Designing games

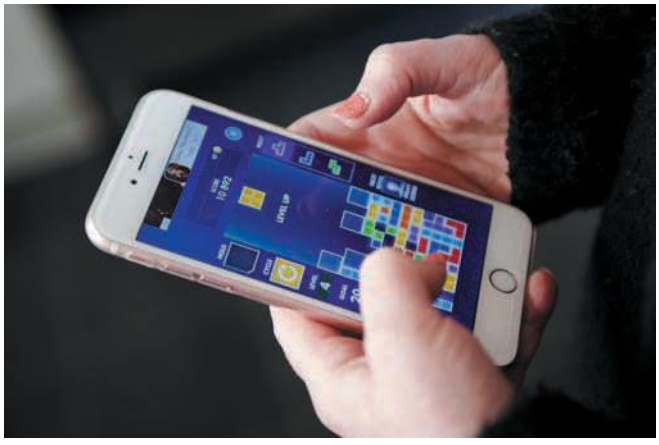
Most games are collections of objects occupying an imagined world, and as many objects are created from one basic 'parent' type (monsters, enemies, found objects, people), OOP can be a very suitable approach for coding games. Most game coding depends on OOP principles.

The game of *Tetris* was created in 1984 by Alexey Pajitnov and in 2012 and is on display in New York City's Museum of Modern Art (MoMA). It is one of the most popular handheld games ever created.

Play classic *Tetris* either by using the link provided or by using the free official *Tetris* mobile app (Figure 14.23).

Weblink
Play Tetris

Systems thinking



Alamy Stock Photo/kem bridges

Figure 14.23 *Tetris* played on a smartphone.

After playing for a while, identify how many different shapes are used in the game and complete Table 14.3.

Although there are seven different block shapes in *Tetris*, they all share the same behaviours (or methods). We could say they belong to the same 'class', which we could call 'block'. Identify three methods you believe might exist for this block class.

Table 14.3 Play *Tetris* and complete this table by identifying the shapes used in the game.

Instances (identify the different Tetrominos)	Properties or attributes

The seven blocks you identified in Table 14.3 are all ***instances*** of the block **class**. Each has at least three *attributes*: orientation, shape and colour. Each block has three ***methods***: it can move horizontally, vertically and be rotated.

The playing board could be considered another class, which we might call 'board'. If we wished, we could use it to create other boards as instances of this class or child classes of different boards.

There is actually an eighth 'ghost' block piece. This is the outline you see at the bottom which shows where your current piece is lined up, and how it will fit depending on your rotation. Can you describe its behaviour? Do you think it is another instance of the block class or separate from it?

Object oriented programming in MMOs

World of Warcraft (WoW) is an example of an MMO (**massively multiplayer online role-playing game**, sometimes referred to as MMORPG) released in 2004. WoW is played throughout the world and such games are multi-billion dollar industries. Players often write add-ons to such games using OOP approaches. OOP allows for programmers to maintain organisation in their programs, which is important in WoW add-ons, as these can conflict if they are not carefully structured.

As a simple example, an online video game designer might build a parent class called 'LivingThing'. This class might have attributes of name and health and methods including explore, run, fight and so on. Child classes such as Enemy, Animal and Player could be based on this LivingThing class and inherit its qualities as well as having its own: enemies might have weapons, whereas animals would not.

9780170442787

Once an enemy child class was defined, many types of enemies with differing characteristics could be created. One might move fast and not easily suffer damage. Another might be slow but have incredible attacking qualities. All of these, even though they have unique values, are still instances of the 'Enemy' class (see Figure 14.24).

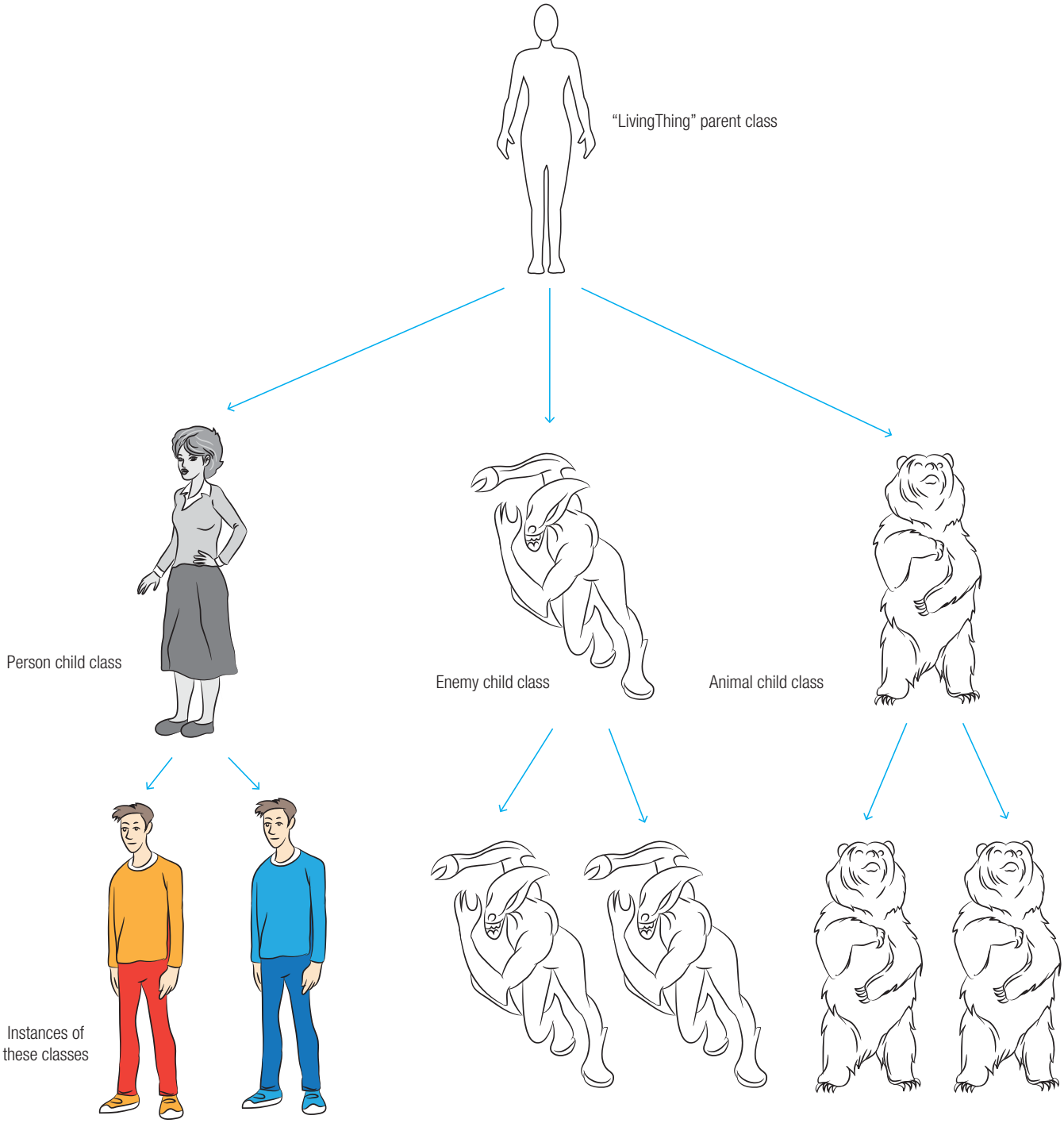


Figure 14.24 Planning an OOP-based game.

Other applications of OOP

Apart from gaming, other areas where OOP programming is widely used include AI and machine learning, OOP databases and simulation and modelling. One area of programming suited to OOP is the graphic user interface (GUI) design. A button, for example, in a GUI has attributes such as shape, size, colour, position and text label. Actions can happen to it such as being clicked, needing to be redrawn and being made inactive or active.

Thinking of GUI elements as objects defined by classes is helpful. The object's instance can describe its attributes and the actions that can be performed on it (its methods) and it can send other objects data describing these interactions.

Chapter review quiz

INFOBIT

Python is an object-oriented programming language, and you may be surprised to discover that in Python, *everything is an object.* This means integers, strings and functions are all objects with attributes, methods and initialisers. For example, when you type num = 5, Python creates a new object of type integer with the value 5 and assigns its reference to the name num and gives it defined behaviours.

ACTIVITIES

Identify

1 What is an MMO?
2 What OOP principle is particularly useful in games design?
3 Identify two OOP languages that could be used to develop games.

Analyse

4 Outline how the game of *Tetris* might be structured as an OOP game.
5 How could games developers use OOP to assist in creating improved or updated versions of their games?
6 Discuss advantages of an OOP approach over a procedural approach for the dice game (see the earlier Skill builder).

Extend

7 Research Atari's highly popular video console game *Asteroids*. (You can view a video of the original gameplay online and perhaps locate a retro version of the original game.) The objective of *Asteroids* is to score as many points as possible by destroying asteroids and flying saucers. The player controls a triangular-shaped ship that can rotate left and right, fire shots straight forward, and thrust forward. For the player's spaceship, identify at least three properties and three behaviours.

Weblinks:
- Virtual Songlines
- Path 3 - Sydney Site
- Game of Life
- Darwin Rural Fire Simulation
- Wildfire Explorer
- Epidemic Simulation
- micro:bit Microsoft MakeCode
- MindMup
- micro:bit Python editor
- Play Tetris
- Replit
- IDLE
- RPG extras map

Project resource:
- A virtual pet

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

14.6 Project 2: Building an OOP-based adventure game

Identifying and defining

Your task

If you were asked to create a role-playing game (RPG) at this stage of your programming experience, you would most likely design it using variables and functions to set up enemies and players. OOP offers a better approach.

We will guide you while you create a game using classes to represent the characters and the game itself.

The project is divided into three stages.

- Stage 1: we provide a guided project to develop a bare-bones version.
- Stage 2: you add additional code with some support.
- Stage 3: we suggest an exciting treasure hunt and other ideas to extend this project.

This RPG game is based on classes representing the game's characters (Figure 14.25). We help by providing basic methods so the game will work. You modify these and invent some of your own. This game provides lots of opportunity to be as inventive as you like and to exercise your OOP coding.

Figure 14.25 In this game classes are used to track characters' health, strength and fighting ability over time as they encounter various monsters.

Researching and planning

Stage 1: A basic version

Using classes

Since monsters and heroes in games share features, we will bundle these into a class called `LivingThing()`. Every time we want to create either a hero or a monster we just use this as a 'rubber stamp' and assign unique attributes to each of the game's characters.

In Python this 'super' class might look like this:

```
class LivingThing():
    def __init__(self, name):
        self.name = name
        self.health = 10
```

We examine this code now. The definition starts with the `class` keyword, followed by the class name `LivingThing` then followed by parentheses and a colon. Following this is a function, (called a *method* when part of a class). The one here looks a bit unusual:

```
def __init__(self, name):
```

The `def` here is stating that what follows is a *definition* of something. `__init__( )` is a special method called an *initiator* or *constructor*. Constructors are like blueprints for objects. When we 'call' a class, an empty new object is created. The `__init__` method is executed and its job is to prepare a new object with those attributes. This one defines one attribute: `name`. Any instance created based on this class will need a name. This function begins with `self` to show its variables belong to its instances and are not just local variables which disappear after the class is used.

We automatically assigned health of 10 to each creature.

Coding the game

Now we understand the concept of a class and its usefulness, we can create a bare-bones version of the game.

Select the Python editor you will use. We suggest either replit or IDLE (see weblinks).

Note: If you have a PDF version of this text, you should type code out rather than copying and pasting. In fact, code that is pasted will usually fail to run as it will contain unwanted hidden characters and incorrect indenting and spacing. A critical part of learning to code involves precision. Typing also will help you learn coding concepts far better.

Weblink
Replit

IDLE

Common issues students face while learning to code in Python are:

- indentation
- conditionals `if, else, elif, while` and `def` must be terminated with colons
- quote marks around strings must be correct
- correct capitalisation
- parentheses and braces paired correctly.

Importing external modules

1 Type the first two lines of your code:

```
from random import randint, choice
import time
```

This imports two modules whose functions will be used by your program (`randint` and `choice` are both members of the `random` maths module in Python).

Producing and implementing

Creating parent and child classes

Our approach will be to create child subclasses using `LivingThing` as a parent class. Two child classes will be created: one for the player and one for enemies and we will use these to create the actual living objects for the game (Figure 14.26).

2 Code the parent class:

```
class LivingThing():
    def __init__(self, name):
        self.name = name
        self.health = 10
        self.status = 'regular'
```

You can see from the `__init__` that instances of this class will demand a name, but will automatically be initialised with `health` of 10 and status of `'regular'`.

3 Create the two child classes:

```
class Player(LivingThing):
    pass

class Monster(LivingThing):
    pass
```

Both of these child classes inherit initial variables from `LivingThing`. `Pass` indicates no other code is yet written.

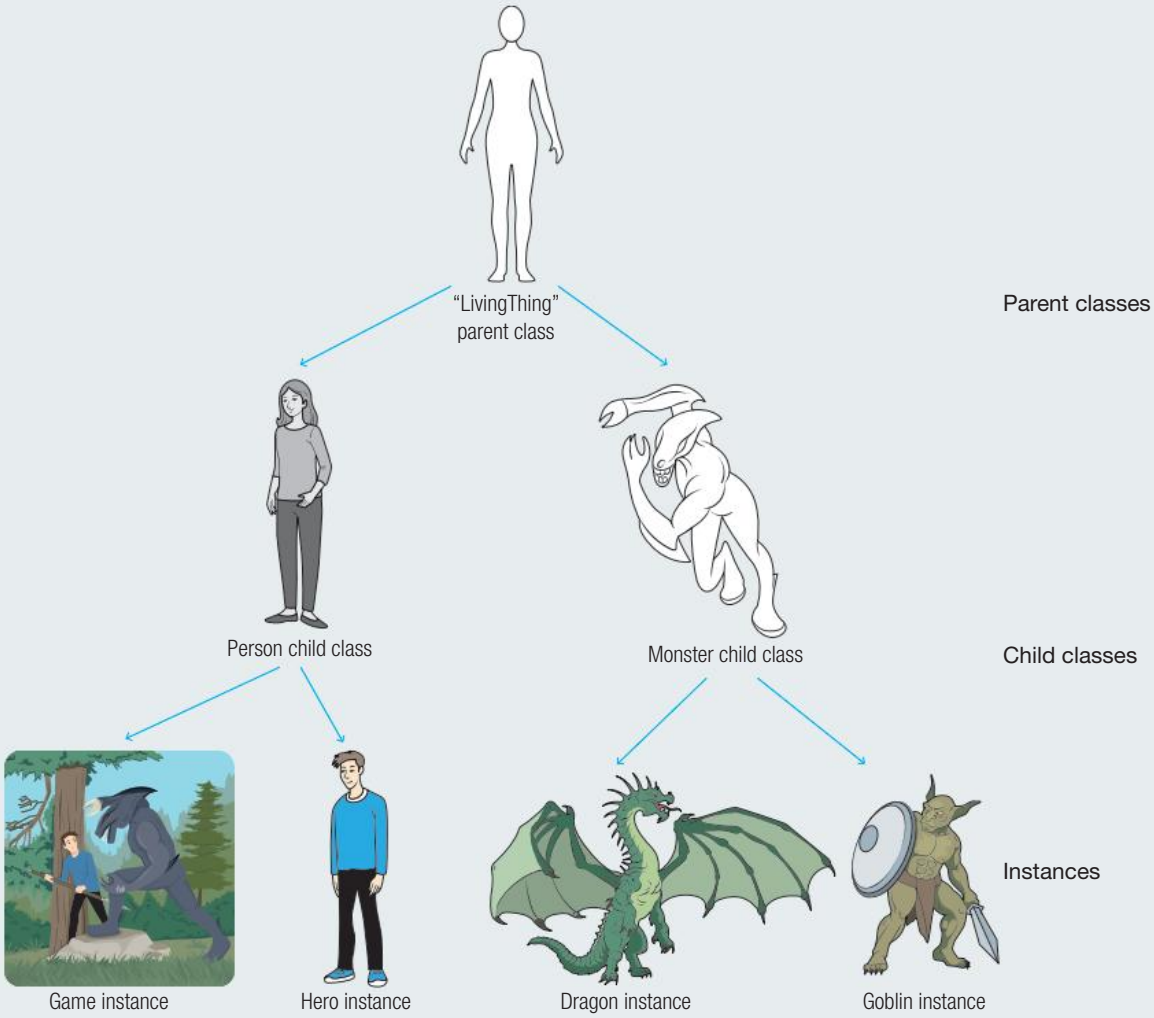


Figure 14.26 Planning an OOP-based game. A game designer builds two parent classes: LivingThing and Game. These define attributes and methods for the child classes. All instances inherit the methods and attributes of their classes.

9780170442787

Benefits of OOP

Here's where we begin to really see the benefits of object-oriented programming. If another programmer is reading the code, they know they can just look for the class `LivingThing` and find out everything they need to know about living things.

Further, they know if they update or add a feature to the `LivingThing` object then all objects created from it will gain the new features. This is much easier than trawling through a long procedural program trying to locate the code needing to be changed.

Defining methods

4 Living things get tired, get hurt and get healed! So we need to create methods for our `LivingThing` class so characters will be able to get tired, get hurt and be healed.

Type the following *inside* the `LivingThing` class after the existing code:

```
def tire(self):
    self.health = self.health - 1
def hurt(self):
    self.health = self.health - 2
def heal(self):
    self.health = self.health + 1
```

Notice how much easier it is to make a change to a class rather than to every object in a game! This is a key advantage of an OOP approach to programming. Interpret what each of these methods does.

Create a command dictionary

The player will type simple commands to play the game. These will trigger (or 'call') the appropriate methods. We will use Python's dictionary data structure (see earlier in this chapter) so the name (such as 'help' or 'stats') is matched to its correct method. Note we haven't yet written these methods – this comes next! Another approach instead of using a dictionary is that we could use a chain of if...elif statements to select methods.

5 Type this dictionary at the end of your current code (after the class definitions):

```
commands = {
'help': Game.help,
'stats': Game.stats,
'explore': Game.explore,
'run': Game.run,
'fight': Game.fight
}
```

Creating a game class

There is no rule that says we have to use classes for everything in this game. We could define the living things as classes then write all the code for the actual game as one large **game loop**. However, we have seen the benefits of classes so we will create a game class here to hold all the game's methods.

6 Add this to define a `Game` class.

```
class Game():
```

We will now add methods for this game class. At this stage, we will just set them up and leave them inactive by typing the word 'pass' in place of the code of each method and complete the code for each one later.

7 Type:

```
def help(self):
    pass
def stats(self):
    pass
def explore(self):
    pass
def run(self):
    pass
def fight(self):
    pass
```

Creating instances

8 We are now ready to create instances of our player, the two monsters and an instance of the game class we will call `RPG_game`. First we ask the hero's (your!) name.

Add these lines *following* the command dictionary:

```
name = input('Hi! What is your name?  ')
hero = Player(name)
RPG_game = Game()
```

Our hero has now been 'born' as an instance of `LivingThing`'s child class, `Player`.

9 Create two monster instances and supply them with names. Add:

```
goblin = Monster('Goblin')
dragon = Monster('Dragon')
```

Two monsters have now been 'born' as instances of `LivingThing`'s child class, `Monster`.

10 Add:

```
monsters = []
monsters.append(goblin)
monsters.append(dragon)
```

We have now created an empty list called `monsters` and added each monster to it.

11 As each game begins we will randomly choose one of the monsters as the enemy. You could create and add more to the list if you wish. Add this to the end of your program to choose a monster from the list each time:

```
monster = choice(monsters)
```

We are able to use a very convenient function, `(choice)`, available in Python's external random module imported in the first line of the program that randomly chooses one item from a list.

The game loop

12 Now we are ready to drive the game. First we deliver introductory instructions to the player (see code below).

Add:

```
print (' (type help to get a list of actions) ')
print (hero.name, 'enters a dark cave, searching for adventure')
print('and will soon face the', monster.name)
RPG_game.help()
while hero.health > 0 and monster.health > 0:
    line = input('What do you want to do? >> ')
    if line in commands.keys():
        commands[line](RPG_game)
    else:
        print ('I do not understand this suggestion.')
```

We used a `while loop` here to keep the game loop running until either hero or monster health drops to 0. We ask the player to choose an action: `help`, `stats`, `explore`, `run` or `fight`. We call that action using the `commands` dictionary of actions. The only time this loop ends is on the death of the player or monster. In that case we print final stats and add a farewell.

13 We need to handle the end of a game. Type this *after* the loop. Align it to the very left:

```
RPG_game.stats()
print("Game over")
```

Run your program and debug errors. Of course nothing much will happen as we have not written content for any of the five player methods. Debug your code carefully. Most problems are caused by incorrect indentation or punctuation.

Carefully check your file against the one provided on NelsonMindTap: *RPG_Stage_1_begin.py*. Note this also includes some completed methods.

Basic help method

This very simple method prints the valid choices for the player when they ask for it.

14 Replace the help method with:

```
def help(self):
    print('Your choices are:')
    for key in commands.keys():
        print(key)
```

Basic stats method

This simple method prints the player and monster statistics during the game. Run the code and this time type help when asked what you want to do.

15 Replace the stats method with:

```
def stats(self):
    print('You are', hero.name)
    print('with health of', hero.health)
    print('your status is', hero.status)
    print(monster.name, 'health is', monster.health)
```

Basic explore method

A neat way to decide when a monster comes on the scene is to toss a digital coin!

`randint(0,1)` could do this as it delivers only 0 or 1. These often equate to False and True in programming.

Suggestion: By replacing this with a biased coin (!) you could make the monster appear more often: `toss = randint(0,2)` is like tossing a 0, 1 or 2. We use this in the following.

16 Replace the explore method with this code:

```
def explore(self):
    if hero.status == 'confronted':
        print('Sorry! No time to explore, you face a monster!')
        print('You must fight!')
        RPG_game.fight()
    else:
        toss = randint(0, 1)
        if toss == 0:
            print(monster.name, 'confronts you unexpectedly')
            hero.status = 'confronted'
        else:
            print('You spend time building fitness by exploring')
            hero.heal()
```

Run the code and debug any errors. Can you interpret the code? Can you see how exploring unknown territory could be dangerous when monsters can randomly appear? Can you see how it handles a player who refuses to fight and keeps choosing 'explore'?

Basic run method

We need to penalise players for running from a fight when their status is set to 'confronted' (i.e. a monster has appeared). One way to do that is by having the monster outrun the player. To simulate this we generate random numbers for both player and monster based on their health and punish the player by reducing their health or increase monster health and reset the player battle status to normal. Examine the following code then add it to the game class.

17 Replace `pass` in the run method:

```
def run(self):
    if randint(0, hero.health) < randint(0, monster.health) and hero.status == 'confronted':
        print('It caught up with you so no choice but to fight.')
        self.fight()
    else:
        hero.tire()
        monster.heal()
        print('Your health has suffered by running.')
        print('The monster is stronger!')
        print('Your health is now', hero.health)
        hero.status = 'normal'
```

Basic fight method

Our approach here will be to simulate a fight and reduce both player and monster health (fighting is exhausting!) using the hurt method in the `LivingThing` class.

18 Replace `pass` in the fight method:

```
def fight(self):
    if hero.status == 'confronted':
      print(monster.name, 'attacks you')
      hero.hurt()
      monster.hurt()
      if hero.health <= 0:
          print('You were killed by the', monster.name)
      elif monster.health > 0:
          print('You only damaged the', monster.name)
      else:
          print('Victory! You killed the', monster.name)
      hero.status = 'normal'
    else:
      print('You are safe. Not a monster in sight anywhere!')
```

We have to consider players choosing to fight when no monster has appeared, so we place this method inside a conditional statement to check player status is 'confronted'. If not, no battle occurs. Status is set to back to normal after the battle.

19 Run the game and debug any errors. If you need help, a copy of this version is on Nelson MindTap called *RPG_Stage_1_end*.

Testing and evaluating

Basic gameplay example

Here is output from an actual game. See if you can stay alive a lot longer than this!

```
What is your name?  Hero
 (type help to get a list of actions)
Hero enters a dark cave, searching for
  adventure and will soon face the Goblin
What do you want to do? >> explore

Your health is now 16
Goblin confronts you
What do you want to do? >> fight

Goblin attacks you
You survived the Goblin
What do you want to do? >> help

Your choices are:
   help
   stats
   explore
   run
   fight
What do you want to do? >> run

A monster has appeared
Goblin attacks you
You survived the Goblin
What do you want to do? >> explore

Your health is now 2
Goblin confronts you
What do you want to do? >> fight
Goblin attacks you
You were killed by the Goblin
Game over!
```

Stage 2: Adding features to improve user experience

Now you know the game and how to use OOP, your task is to make the game amazing! We do this in the next two stages. Our bare-bones version requires no skill to play it and not much decision making from the player. It is driven purely by chance.

Imagine the following features:

- *convenience*: improve the UI by requiring only the first letter of choices and provide an option to replay
- *choice*: offer players battle weapons obeying rules similar to rock, paper, scissors
- *challenge*: add a puzzle that generates a random treasure map and create a treasure hunt.

Each of these suggestions will improve users' experience of the game.

Add a single letter response method

This one is easy!

1 Change the `commands` dictionary to use single letters only.

```
commands = {
    'h': Game.help,
    's': Game.stats,
    'e': Game.explore,
    'r': Game.run,
    'f': Game.fight
    }
```

2 Add a new `response()` method to the end of the Game class to respond to user choices. This will keep asking users for input until one of the valid letters is typed.

```
def response(self):
    print('Type the first letter of your choice.')
    line = input('What do you want to do? >> ')
    print()
    if line in commands.keys():
        commands[line](RPG_game)
    else:
        print ('I do not understand this suggestion.')
```

3 Delete the content in the Game class help method and replace it with this print statement listing the possible player actions.

```
print('Your choices are: help, stats, explore, run, fight.')
```

4 Replace the game's main `while` loop so it now calls this new method.

```
while hero.health > 0 and monster.health > 0:
    RPG_game.response()
```

Add a replay option

Because we have now created a method for user responses, adding a play again feature is easy:

- Ask the player if they want to play again.
- If answer is 'yes', reset health and call the `response` method to start new game.
- If the answer is 'no', exit program using sys.exit()
- If the answer is invalid, ask again.

5 We add this replay method to the end of the `Game` class:

```
def replay(self):
    print('Game over!')
    play_again = input('Do you want to play again? Y/N >>> ')
    play_again = play_again.upper()
    if play_again == 'Y':
        hero.health = 5
        monster.health = 5
        self.response()
    elif play_again == 'N':
        print('Bye for now! Back to the real world!')
        sys.exit()
    else:
        print('Invalid response.')
        self.replay()
```

6 Modify the final `while loop` to a while True loop containing an if...else statement. Now when a player or monster dies, player statistics are printed and the new `replay` function is called. Test your code so far and fix any errors.

```
while True:
    if hero.health > 0 and monster.health > 0:
        RPG_game.response()
    else:
        RPG_game.stats()
        RPG_game.replay()
```

Add weapons

A *Retiarius* was a Roman gladiator who fought using either a weighted net, a three-pointed trident or a dagger. It would be exciting to add battles with monsters to our game where weapons are chosen in a way similar to the game of scissors, paper, rock, an algorithm you may have developed previously in Chapter 15.

For the game to be more interactive, we will allow players to choose their weapon and write a method to do this.

Your first task is to define a list of weapons. Add this line under the monsters list:

```
weapons = ['net', 'dagger', 'trident']
```

Your second task is to write a function for the player to choose a weapon by using its first letter: n, d or t.

These will be the steps.

- First print an interesting message about discovering some old weapons.
- Then ask player for a selection and convert it to uppercase (in case they typed lowercase).

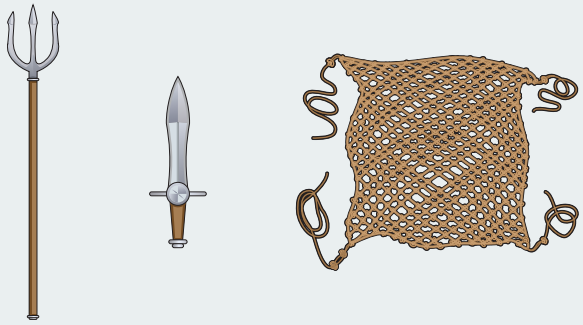


Figure 14.27 Typical weapons used by a Retiarius.

- Check it is a valid selection and if not keep asking.
- Assign the correct weapon name to `hero.weapon`
- Print message to player confirming their weapon choice.
- Short dramatic time delay while monster chooses its weapon.

Think about the gameplay and UX for this method. After the player chooses, the battle result is instantaneous. This is not very exciting at all. Introducing a three-second delay before revealing the monster's weapon changes the user experience completely. We imported Python's time module at the start of the project in preparation for this. You can now add time delays wherever you wish simply by using `time.sleep(number_of_secs)`.

7 Create a list of weapons for the monster to choose from randomly. Add this line after the import statements:

```
weapons = ['trident','net', 'dagger']
```

Guided by these points complete this `weapon_choice` function and add it to your Game class. Note: We have added a few empty lines to improve spacing for the printed output.

```
def weapon_choice(self):
    print(complete this line)
    select = input('Trident, Net or Dagger? Type first letter. ').upper()
    while select not in ['T', 'N', 'D']:
        print('Not a valid choice. Re-enter')
        print('Choose your weapon')
        select = input('Trident, Net or Dagger? ').upper()
    if select == 'T':
        hero.weapon = 'trident'
    elif complete
        complete this line
    elif complete
        complete this line
    print()
    print('Your weapon is a', hero.weapon)
    print('You hide your weapon. Monster holds a ...')
    time.sleep(3)
    monster.weapon = choice(weapons)
    print('…the monster has a', monster.weapon)
    print()
```

8 Run the program to debug your code. It will not yet do anything as you have not written code for this new battle method. The logic is:

```
START
    IF (player uses trident versus dagger)
    OR (player uses dagger versus net)
        OR (player uses net versus trident)
    THEN player wins
    ELSE IF both weapons same THEN tied
    ELSE player loses
    ENDIF
END
```

We apply this logic to a new fight method:

- if a win, call `hero.heal` and `monster.hurt()` methods
- if a tie, call `hero.tire` and `monster.tire()` methods
- if a loss, call `hero.hurt` and `monster.heal()` methods
- reset hero's battle status to 'normal'

The new fight method replaces only the first four lines of the original fight method.

9 Modify your fight method by deleting the first four lines then completing the missing sections in the following code:

```
def fight(self):
    if hero.status == 'confronted':
        self.weapon_choice()
        if (hero.weapon == 'trident' and monster.weapon == 'dagger') or ... complete line
            add a 'win' message
            hero.heal()
            monster.hurt()
        elif hero.weapon == monster.weapon:
```

```
            add a 'tied' message for when no one wins!
            monster.tire()
            hero.tire()
        else:
            monster.heal()
            hero.hurt()
            add a 'lose' message
        hero.status = 'normal'
        self.stats()
```

Add code to the end of the fight method to check if hero health is now zero. A completed version of this stage of the app is available on Nelson MindTap called *RPG_Stage_2.py*.

(Advanced) Stage 3: Treasure map

In this third stage of the project you have the opportunity to develop an exciting new method as well as add ideas of your own.

You find a treasure map but before you can use it, it blows away! You think you can visualise the path on the map. Will you find the path to the treasure and gain health or get lost and die in the forest? A complete version is provided in the Nelson MindTap resources, but see how far you can get on your own in this coding challenge.

Our approach to creating a random map is to join four horizontal paths by generating either left, middle or right vertical paths (see Figure 14.28). The player must remember the correct turns to reach the treasure (R is right turn, S is straight ahead and L is left turn).

10 A suggested approach is:

- create three string patterns to print numbers of spaces either side of a single vertical stroke. Then select one of these and print it three times. This will create either a left, middle or right aligned vertical pattern. Use these patterns:

```
M = ' ' * 21 + '|' + ' ' * 20
L = ' ' * 10 + '|' + ' ' * 32
R = ' ' * 32 + '|' + ' ' * 10
```

- create a list called `routes` to hold three variables representing these, another list `journey` to hold the correct pattern sequence and an empty string variable `wayback` to hold the player pattern. Use these:

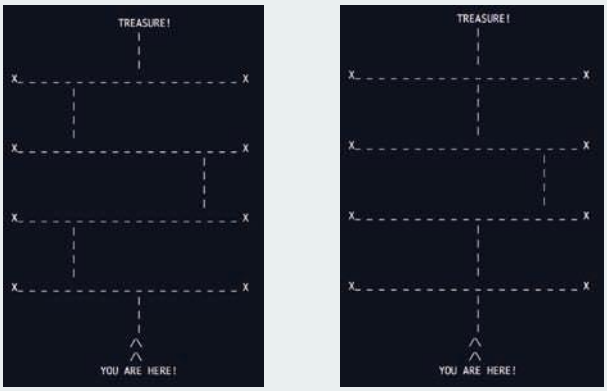


Figure 14.28 In the left map, turns will be LRRLLRRL. In the right map, turns will be SRLLRS.

```
routes = [M, L, R]
journey = []
wayback = ''
```

- print start and end lines (you will see these are both 'M' or middle patterns) and append M for these to the `journey` list
- randomly select either L, M or R from the `routes` list, print three times. Print a horizontal line. Append its corresponding letter to the `journey` list
- repeat last step *twice*
- pause program to give the player time to memorise the map then scroll the map upwards so it is hidden (by printing lots of empty lines)
- ask player to recall turns and compare these to the correct sequence. Reward player with health or penalise them by death
- *(Spoiler alert!)* What will be an algorithm for the correct turns: Think what would be a correct sequence if the next path is to the right of the current one? To the left? Straight ahead? The only possibilities are LR or RL or straight ahead (S). Write an algorithm to select pairs of entries in `journey` list and replace each pair by one of these. All M letters are thus eliminated.

Taking it further

Even if you found the last challenge tough, there should be an idea here you might like to try:

- add time delays
- add more story-based feedback for players
- add more monsters to choose from, alter the existing methods, add your own methods
- such as an eating or drinking involving danger or random pleasant events as rewards or dangers
- award more than one life to players
- add more than one monster at once
- add super powers
- add a time factor of day and night with fights more easily lost at night
- add regions so that the player can move to safer areas

After trying this last challenge on your own, play this online version of the game demonstrating the added extras or use the solution provided in the Nelson MindTap teacher resources. Examine the code to see how additional features were implemented. This version includes the random treasure map feature.

Weblink
RPG extras map

15 Developing apps and OOP

FOCUS AREA: Developing apps and web software

Chapter outcomes

In this chapter, you will learn:

- how to design and write computer programs
- how to use functions
- about the scope of a variable
- the history and differences between programming languages
- to use arrays, lists and dictionaries
- to manipulate strings
- to handle records and files
- about types of errors
- to detect and correct errors
- to use debugging methods
- the importance of documentation
- to design, develop and implement an app
- about object-oriented programming (OOP)
- to design, develop and implement an app using OOP.

Key terms

array
attributes
class
debugging output statement
declarative
dictionary
flag
imperative
key
list
methods
object-oriented programming (OOP)
stub
syntax

9780170442787

15.1 Designing and writing software programs

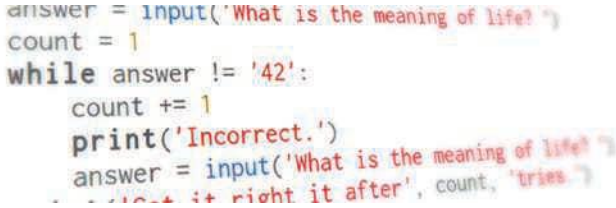


Figure 15.1

A student starting high school was overheard saying: 'Isn't it good all the important computer programs have been written?'

If this student chooses to study computer programming, it won't be long before they discover how wrong their impression is. As we address new challenges at the frontiers of medicine, space exploration, the environment, the arts, scientific research, data science, cybersecurity, artificial intelligence (AI), robotics and gaming, new software will need to be developed and existing software updated.

As so many aspects of our lives are now stored digitally online, attacks on privacy and security and large-scale cyber attacks will continue to threaten government and democratic processes. The demand for application programmers and data scientists outstrips supply.

The majority of us experience software as consumers: playing online games, browsing the web or using social media. In this course, you become a creator and not merely a consumer of software.

Most non-programmers believe writing software is a mechanical rather than a creative activity. However, as a senior high school computing student told a group of younger students: 'I want to tell you that coding is a type of art. There are no limits to your creativity and you can create anything you want.'

The computer is a tool and its hardware and software offer endless creative opportunities for invention, meeting human needs and providing great personal satisfaction.

Your programming skills so far

Chapter 4 introduced you to programming basics. It is very important you are familiar with its content before commencing this chapter.

In that chapter you learned about the relationship between a computer's hardware and software, types of computing languages, algorithms, flowcharts, pseudocode and **syntax**. You also learned the main components of computing languages:

- control structures
- variables
- operators
- assignment statements
- built-in functions
- data types.

You now have the foundation you need to explore the art, power and science of creating your very own applications!

15.2 Functions

Computer programs use *functions* to group related code. You have already used built-in functions such as `print()` and `range()` in your Python code. When you imported `randint` you were accessing an external function from a Python library. In this section you will learn to write your own functions.

Functions make code easier to understand and provide organisation. It is a way of hiding the complexity of how the input was changed to produce output (Figure 15.2). Another advantage is that once a function is defined it can be used again in other programs. This saves reinventing code. A programmer might write code to sort items alphabetically then by defining it as a function they can reuse it whenever sorting is needed. Functions are also called ***methods***, *routines*, *procedures* or *subprograms*.

Often programmers create libraries of functions to be used again in other programs.

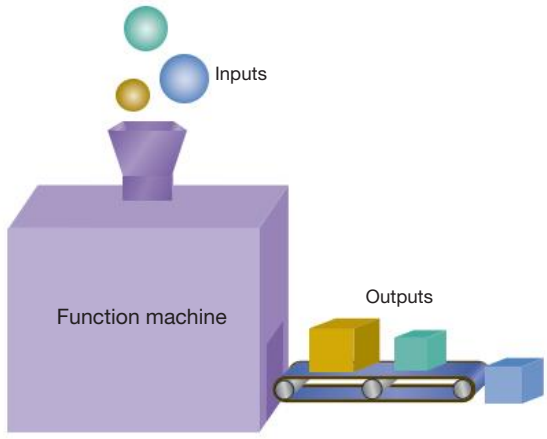


Figure 15.2

SKILL BUILDER

Writing and using functions

Comparing a function in two languages

Here we compare the same function written in both the Python and C++ languages. The function prints out the maximum value of two numbers. Note the similarities and differences. Which is easier to interpret?

Python

```
def max_val(num1, num2):
    if (num1 > num2):
        result = num1
    else:
        result = num2
    return result
print(max_val(5, 10))
```

C++

```
#include <iostream>
using namespace std;
int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
}
    else {
        result = num2;
}
    return result;
}
int main(void) {
          cout <<  max(5, 10);
}
```

Skill builder resource
Writing and using functions

Weblink
Replit

1. Copy and run one or, if you are able, both of these codes. Repl.it can be used to code in both languages. Observe that Python uses indentation to structure and separate its statements, whereas C++ uses curly brackets.
2. Write a function to print the minimum of two integers.
3. Write a function to print both the maximum and minimum of three integers.

Using functions

Functions in computing work in a similar way to functions in mathematics. However, in high school maths you are used to functions occupying a single line. For example, we might write $f(x) = x^2$.

If we assign the x a value of 3 then this function will return the value 9. Computing functions work similarly, but can be asked to do many steps, such as sort, print or generate a random number. Because of this they often occupy many lines.

In Python, functions are very easy to write. They look like this:

```
def myfunction(x):
```

They begin with the term `def` (short for *definition*), followed by the function's name (invented by you), followed by parentheses containing any parameters needed for that function's calculations. This all ends with a colon, followed by indented lines listing the steps the function performs.

4. Here is a common maths function:

```
def mycurve(x):
    y = x * x
    print(y)
```

The function calculates the square of a number. The function is a definition only and will not do anything until it is 'called' from a main program and supplied with a value for x.

Copy the code for this example into Python IDE and then call it by writing:

```
mycurve(5)
```

Is your result `25`?

Using functions

5 To use a function it must be defined *before* it is called.

Here is our earlier function again with a small change.

```
def mycurve(x):
    y = x * x
    return y
```

Notice the word `return`? We replaced the print statement to ask for the answer to be returned instead. The function now passes the result back to the main program. Whenever a function needs to return a value, this word terminates the function at that point, evaluates any expression after it (here `y`), and jumps back to after the place where it was called.

Type up this version of the function and call it by writing `mycurve(5)`.

You will not see any answer this time as it was only 'returned' but not printed. This time you need to *ask* for it to be printed by placing the function call inside a `print` statement. Try this call on this function:

```
print(mycurve(5))
```

6 Let's make the function user-friendly by introducing a fancier print statement. This is all part of designing a user-friendly interface. This time we will get the function itself to print the result along with a message so we no longer need to use the word `return`.

Copy and run the following code. Pay close attention to indentation and punctuation! Make sure you understand all the features.

```
def mycurve(x):
    y = x * x
    print('For this curve x =', x, 'corresponds to a y =', y)
mycurve(5)
```

SKILL BUILDER

Local and global variable scopes

Variables inside a function are not visible to the outside program. The function acts like a 'black box', which is sent input values and produces output. Its values are destroyed after the function finishes executing.

1 Can you explain why the following produces an error?

```
def sum(a,b):
    total = a + b
    return total
# call the sum function
sum(10, 20)

print(total)
```

When you run this program, it produces an error:

```
NameError: name 'total' is not defined
```

This error is about the *scope* of the variable. Variables defined inside a function only are *local* variables. The variable `total` that was defined inside the function is not recognised outside the function.

2 In Python, all variables defined outside functions, however, are *global*; we say they are *global in scope*. In other words, they can be used and recognised everywhere in the program, as well as inside functions. There is an exception, however. If the variable is *modified*, as it is in this case, Python will assume you meant it to be a local variable. If you wish to use a global variable inside a function and modify it, you must add *global* as a keyword. Eliminate the error by adding `global total` as the first line of the function and run the code again.

SKILL BUILDER

Skill builder resource
Using functions

Using functions

It is time to apply your knowledge of functions to create a small program. A friend who is a primary school teacher needs a program for students to practice multiplication tables. In this Skill builder, you will develop a program that offers them a choice of generating a multiplication table or practicing 10 random multiplication questions.

You will write functions to make your code easier to write and interpret.

You will create three separate functions. Solutions are available on Nelson MindTap.

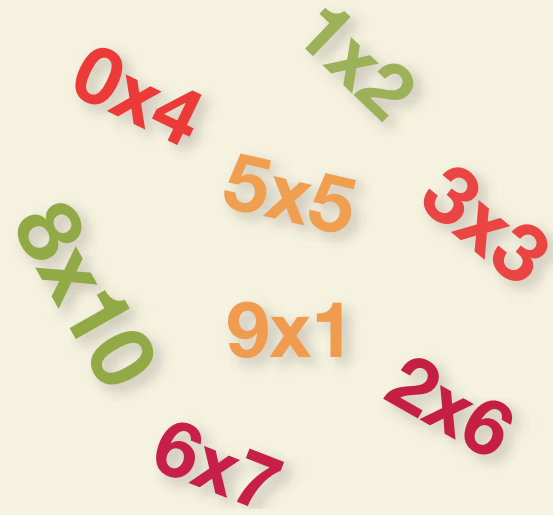


Figure 15.3

A multiplication table function

1 Write code that asks for a multiplication table from 1 to 10 and then prints it.

```
table = int(input('What multiplication table would you like? '))
multiplier = 1
for multiplier in range(11):
    print(table,'X',multiplier,'=',table * multiplier)
```

Copy and test the code. Note the input will be returned as a string so it needs to be converted to an integer using `int`.

2 Rewrite the previous code in the form of a function as explained in the last Skill builder:

```
def tableprint():
    code goes here
```

3 'Call' the function by typing `tableprint()` then test it.

A multiplication quiz function

4 Start a separate program for another function to run a multiplication quiz. This time we generate questions randomly!

This time you will need random numbers, so your first line should be:

```
from random import randint
```

This instructs Python to import its `randint` built-in function from the `random` library of functions.

Copy, save and run the code below, examining each line carefully to understand how it works. You will be building on this during the remainder of this Skill builder.

```
from random import randint
table = int(input('Multiplication table?'))
for question in range(1, 11):
    multiplier = randint(0,11)
    print(table, 'X', multiplier, '= ?')
    response = int(input('Answer?'))
        if response == table * multiplier:
            print(complete this)
        else:
            print(complete this)
```

5 Note the double equals signs `'=='` in the test condition. If you are not sure why, see Chapter 4.

Here is an outline of what happens in this code:

- The code loops 10 times.
- During each loop a random integer `multiplier` is generated and is used by the print statement to ask a multiplication question.
- The user is asked for their `response`.
- A decision structure `(if…else)` checks the `response` against the correct answer.

Rewrite this in the form of a function named `quiz()`.

Call the function and test it works correctly.

6 Copy and paste this additional function into your first program file following the `tableprint()` function and test it.

On your own

7 See how many of the following features you can add on your own to improve the function:

- There are 10 questions. Print the question number each time.
- Keep score of correct and incorrect responses or prevent the user progressing to the next question until their answer is correct.

Add a menu function

8 Create a function that acts as a menu to choose either table print, quiz or quit. After completing the chosen activity users should again be presented with this menu.

9 Include a 'play again' option.

Completed solutions for this Skill builder are provided on Nelson MindTap.

10 Extension: In the quiz, random questions might be repeated. Modify the code to prevent this. Hint: Use Python's `list` structure (see the next Knowledge builder).

Here is an example of its use:

```
mylist = []       # Start with empty list
  mylist.append(4) # add integer 4 to list
  if 4 in mylist:  # Test if 4 is in list
      do something
```

Check your program works and add an enhancement of your own. If you have a younger sibling get them to try it out – and ask for plenty of suggestions!

ACTIVITIES

Identify

1 What are three advantages of using functions in programming?

2 What does 'calling a function' mean?

3 What are parameters used for in writing functions?

Analyse

4 Outline the purpose of built-in functions in computer languages and give one example.

5 Compare techniques used for writing functions in two different languages.

Extend

6 Explain why functions in Python need to be written before they are called.

15.3 Programming languages

Figure 15.4 shows a selection of popular programming languages from the more than 2300 that have been invented to date.

The following programming languages are among the most popular at present (as measured by statistics gained from websites, forums, books, jobs and blogs):

- C
- C++ (pronounced 'see-plus-plus')
- Java
- PHP
- Python
- JavaScript.

Examples of programming languages

Different languages each have different strengths and weaknesses.

- Fortran was once one of the most widely used languages in science and engineering.
- COBOL is a business language and was behind the Y2K problem when many computers failed to recognise the change of date at the start of the new millennium. It was the first data processing language to resemble English and is still used by many banks.

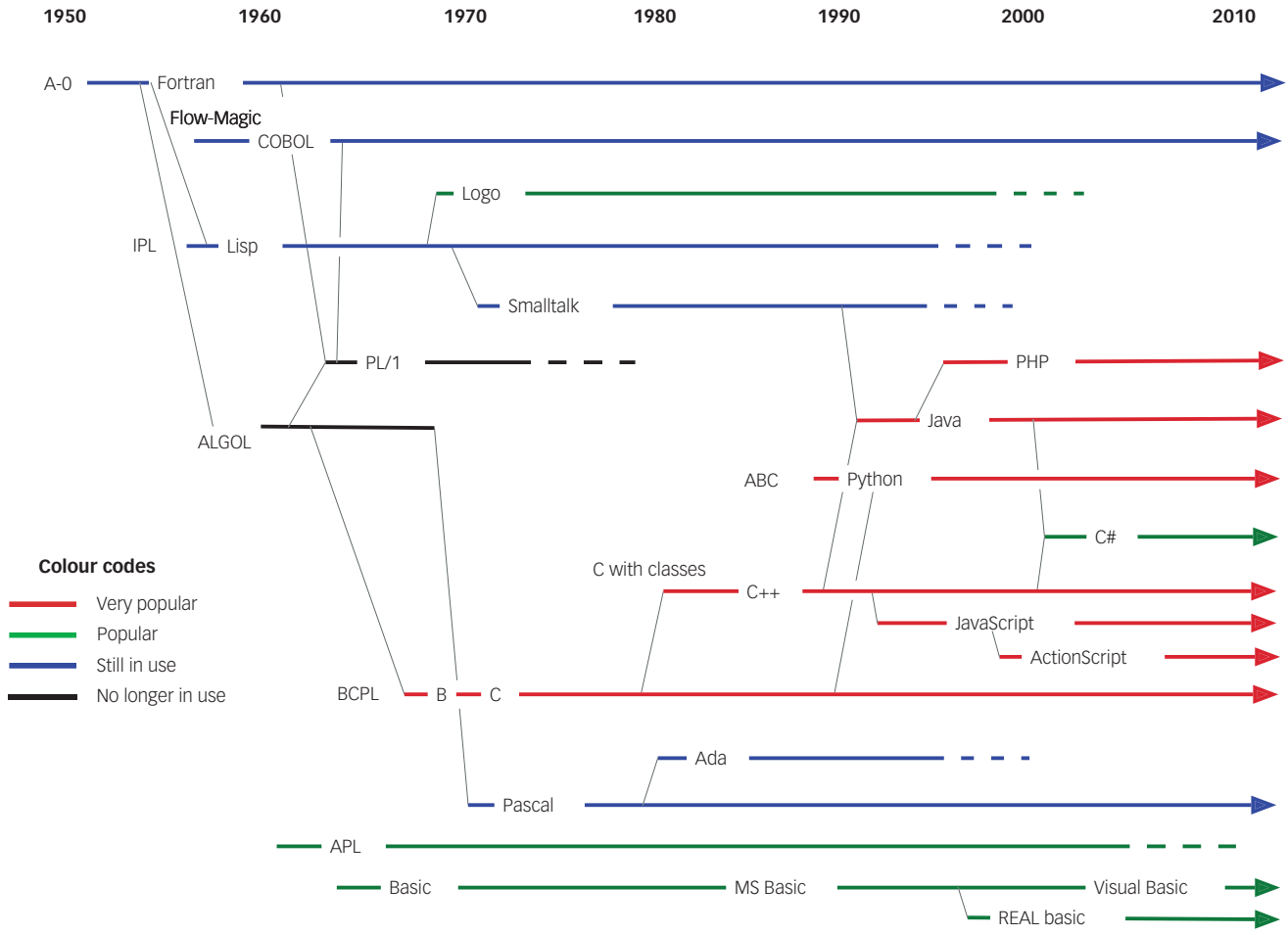


Figure 15.4 The development of computer languages began in the 1950s, with the most popular ones today shown in red.

- Logo was invented by Seymour Papert to help children learn programming by using graphics and simple instructions.
- ALGOL was designed by an international committee to be the world's first universal computer language.
- Pascal was designed by Niklaus Wirth and was widely used as a teaching language. It was designed to simplify ALGOL.
- BASIC was designed as a Beginner's All-Purpose Symbolic Instruction Code, which is where it got its name!
- C evolved from languages called A and B. Along with C++, it is one of the most used languages today.
- JavaScript is widely used in programming web pages.
- Python has an easy to learn syntax, is suitable for all platforms and is a popular choice for teaching programming in schools and universities. It is used widely by many large organisations and is the world's fastest growing language.

Families of languages

Languages can be grouped according to the methods (programming *paradigms*) they use. The two main groupings are:

- **imperative** – languages that specify how things are to be done using procedures or sequences of actions. Both procedural and object-oriented languages belong to this category. **Object-oriented programming (OOP)** defines self-contained 'objects' with their own rules and characteristics
- **declarative** – languages that describe what needs to be done but not how to do it. Subcategories here include functional and logic languages.

Choosing a programming language

It takes time to learn a programming language and still longer to master it, which is one reason why users strongly defend the languages they know.

Which language is best to learn? Should you choose to study the most popular language? Java runs on many platforms and is widely used, whereas Mathematica speeds up complex operations in mathematics but has a much smaller base of users. However, both languages are suited to their purposes. Your choice will depend on what you want to do.

Should you choose the easiest to learn? This may get you started, but you may end up investing time in a language without the power you need later on.

Fortunately, whichever language you choose, there will be enough overlap with other languages and your efforts will not have been wasted. Once you learn one language, it is much easier to learn another.

Perhaps one day far into the future there will be only one all-purpose programming language, which will be automatically generated by a programmer telling an AI tool what it is to do. This will make choosing a lot easier!

15.4 Complex data structures

A data structure tells the computer how to handle the data it will receive. The data structures you used in Chapter 4 were simple ones such as integers, strings and Boolean. These are known as primitive data types.

Programmers often need more complex data structures that hold more than one item of data in a unit or container. Examples of these include **arrays**, records, strings, files and dictionaries. We now examine some of these more complex data structures.

Arrays

Many programs need to hold a collection of objects. The most common structure used for this is the array (or **list**). Without this we would need a separate variable for each item! Many of the programs we write need to work with lists of data of the same type, such as lists of names, addresses or class marks in a test.

Arrays can be thought of as lists or tables of values of the same data type; for example, an array of strings or an array of integers.

Each *element* or *term* in a one-dimensional array is labelled with a sequential integer, called an index. Note that computer languages always start indexing at 0 (Figure 15.5).

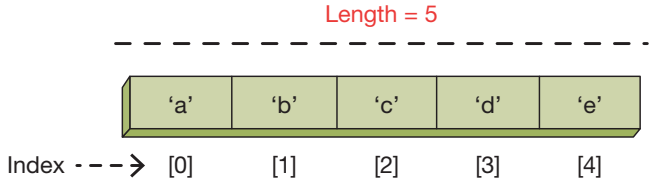


Figure 15.5 Parts of a one-dimensional array.

It is possible for an array to have more than one dimension, in which case more indices are required. For example, a two-dimensional array would be like coordinate points on a number plane, with each index written using a pair of integers.

Arrays are held in memory, just like single-value variables, so that processing the data in an array is very fast. Arrays are declared in the program so memory is set aside for it. In some situations, a programmer is not sure how big an array will be before running the program; for example, names in a school class. This is why most languages allow changes to the size of the array while a program is running.

Arrays are identified by a variable name, and many languages require an indication of what type of data the array holds. Each language has its own rules for arrays. Python uses the list data structure as an array.

KNOWLEDGE BUILDER

Lists and dictionaries

Two commonly used structures for collections of objects in Python are *lists* and *dictionaries*.

A great way to master these is to copy and run each of the examples shown in this Knowledge builder. We suggest you use the IDLE Python Shell interpreter in the following, although you can run them as small Python programs if you use `print` statements appropriately. Your teacher can show you these two approaches.

Lists

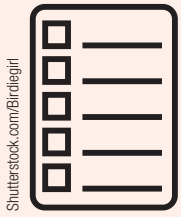


Figure 15.6 A list is a common Python data structure.

Python does not have a built-in array structure but instead uses a `list`. Square brackets are used to define a list in Python. Use your Python interpreter to type the following. Just type the code after the prompt `>>>`. This prompt is not to be typed again.

```
>>> myList = ['a', 'e', 'i','o','u']
```

Accessing list elements

Indices are used to reference elements in a list. In `myList` shown in the code above, we see the letter `'o'` is at position 3 (computers start numbering of indices at 0). We say the index of `'o'` is 3.

```
>>> myList[3]
'o'
```

For the `myList` example above the indices for the elements are detailed in Table 15.1.

Table 15.1 A simple list of five elements.

Element	a	e	i	o	u
Index	0	1	2	3	4

The index of any element is found using the `index()` built-in function (method):

```
>>> myList.index('e')
    1
```

You can check if an element is in a list like this:

```
>>> 'z' in myList
False
```

Adding an element

Add an element to a list using the `append()` method:

```
>>> myList = ['a', 'e', 'i', 'o', 'u']
>>> myList.append('z')
>>> print(myList)
['a', 'e', 'i', 'o', 'u', 'z']
```

Sorting

Sorted the elements in a list using the `sort()` method:

```
>>> myList = ['e', 'a', 'u', 'o', 'i']
>>> myList.sort()
>>> print(myList)
['a', 'e', 'i', 'o', 'u']
```

Removing elements

Remove an element at a given index position by using the `pop()` method:

```
>>> myList = ['e', 'a', 'u', 'o', 'i']
>>> myList.pop(2)
>>> print(myList)
['e', 'a', 'o', 'i']
```

Remove an element with a given value by using the `remove()` method:

```
>>> myList = ['a', 'e', 'i', 'o', 'u']
>>> myList.remove('a')
>>> print(myList)
['e', 'i', 'o', 'u']
```

List length

To find out how many elements are in a list use the `len()` method:

```
>>> myList = ['e', 'a', 'u', 'o', 'i']
>>> len(myList)
5
```

Looping through lists

You can use a `while` loop to display the elements in a list:

```
>>> item = 0
>>> while item < len(myList):
        print(myList[item])
        item = item + 1
```

You can also use this method to populate (add elements to) a list and stop when a blank line is entered. Try out this code using repl.it.

```
alphabet = []
print('Add a new letter to alphabet.' )
print('Blank space to end')
letter = input()
while letter != ' ':
    alphabet.append(letter)
    letter = input()
print(alphabet)
```

When you know beforehand how many times a loop will be repeated, this is called a counted loop. Python uses a `for` loop for this. You can use it to display all elements in a list. In fact, the `for` loop is designed specifically for working with lists in this way.

```
>>> for vowel in myList:
        print(vowel)
```

Weblink
Replit

9780170442787

Splitting strings into lists of words

It is possible to split a string up into a list. If it is a sentence, this will produce a list of words. Try this in repl.it.:

```
sentence = 'Will we really split up?'
words = sentence.split()
print(words)
```

Dictionaries

Python has another very useful data structure for collections of objects, which acts just like an English **dictionary**. In an English dictionary, every object has two parts: the word you look up followed by its meaning. Python dictionary objects also have two parts: a ***key*** and a *value*, which are separated using a colon.

Curly braces are used to define a dictionary structure in Python. Here is a dictionary of male and female sports teams. Type this into your Python interpreter:

Figure 15.7 Dictionaries are a very useful and widely used Python data structure.

```
>>> myDictionary ={'f_basketball': 'opals', 'm_basketball': 'boomers', 'f_soccer':
  'matildas', 'm_soccer': 'socceroos'}
```

Unlike lists, dictionaries are not ordered so no index is used.

Accessing dictionary elements

We access elements in a dictionary using square brackets, just as we do with lists:

```
>>> myDictionary['m_basketball']
'boomers'
```

You can check if an element is in a dictionary like this:

```
>>> 'm_soccer' in myDictionary
True
```

Strings

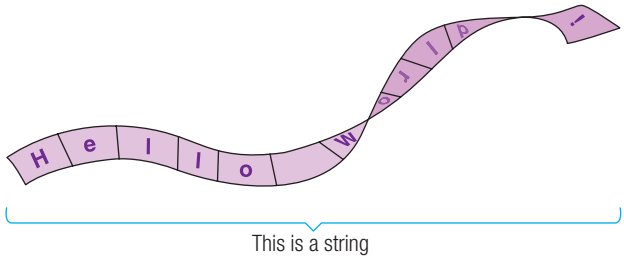


Figure 15.8 Strings are collections of keyboard characters including alphanumeric and punctuation marks.

Strings (Figure 15.8) are collections of keyboard characters (alphabetic, numeric, punctuation or whitespace) and are always written enclosed in single or double quotation marks.

It is likely you have produced error messages by accidentally omitting these while coding in Python!

The following Knowledge builder will develop your understanding of strings in Python.

KNOWLEDGE BUILDER

Strings

Copy and run each of the examples shown in this Knowledge builder using the IDLE Python Shell as in the last Knowledge builder.

Manipulating strings

Strings can be joined ('concatenated') by adding or multiplying:

```
>>> myString = 'aei' + 'ou'
>>> myString
'aeiou'
```

and

```
>>> myString = 'aei' * 5
>>> myString
'aeiaeiaeiaeiaei'
```

Strings can be accessed and sliced:

```
>>> myString = 'aeiou'
>>> myString[0]
'a'
>>> myString[1]
'e'
>>> mystring[2:4]
'io'
```

You can check if a character is in a string:

```
>>> 'z' in myString
False
```

You can change the case of characters using the `lower()` or `upper()` methods:

```
>>> myString.upper()
'AEIOU'
```

You can replace a character:

```
>>> myString = 'aeiou'
>>> myString.replace('e', 'z')
'aziou'
```

and

```
>>> myString = 'aeiou'
>>> myString.replace('eio', 'z')
'azu'
```

You can strip whitespace from the ends of a string:

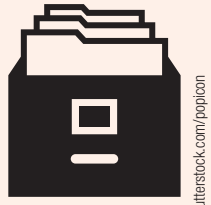
```
>>> myString = ' aeio u '
>>> myString.strip()
'aeio u'
```

Note that when you use `input()` in a Python program, the result is a string. You will often need to convert a string to an integer using `int()` to use it as a number, as you did in the earlier multiplication Skill builder.

```
table = int(input('What multiplication
  table would you like to practise? '))
```

Records

Many programs require us to deal with data about people or things, such as products in a shop. To describe the properties of each person or object, we need to collect together variables of different data types in one place. Different data types collected in this way is known as a *record*. A record for an employee will use text data for name and address and a number data type for salary. What data type do you think would be used for phone number?



Shutterstock.com/popicon

Figure 15.9 Related data of different types collected in one place are known as a *record*.

A record structure is a complex data type acting as a container for a number of different simple data types. In C++, it is called a `struct`. In Python, we would use a `class` structure. In the next chapter you will learn about Python's **class** structure.

The record structure allows for many different types of data to be included as parts of the one container.

Files

A file is an even more flexible data structure than a record.

Shutterstock.com/VectorsMarket

Figure 15.10 A *file* is any collection of data grouped under a single name.

In computing, a file is any collection of data grouped under a single name. Files can be organised in many different ways and there are various ways the separate items can be separated from one another. For example, in a spreadsheet file, the columns are often separated by commas and the rows are separated by the new line character. This file structure is recognised by most databases to allow the exchange of data between these two types of applications.

Files can be thought of as permanent containers for data. This is very useful for keeping the data ready for the next time we wish to use it.

The simplest type of file is a text file, which can be thought of as a long string of characters. Sometimes we write lines of text into a text file, indicating the break between one line and the next with a new line character.

We often need to write data into files. If we need to have data stored in a highly organised way, the best storage location is a database. If we only need to store text to log events or actions, we might use a simple text file for simplicity.

The simplest type of write data action we can do to a text file is to append text, which is similar to the way we write notes in a diary – we just add text where we left off last time. We might use an algorithm like this:

```
BEGIN
   open file for appending
   write line of text to file
   close file
END
```

SKILL BUILDER

Skill builder resource
Using files

Using files

Imagine you want to check if a word is already in an external file containing words.

We have provided two text files: `mini.txt` and `words.txt` on Nelson MindTap for you to use.

Note! Python needs to know where to find these external files. The easiest way is for you to place it in the same folder as your Python program file as Python will look there by default. We strongly suggest doing this. Otherwise, you will need to supply a complete path.

Reading lines from an external file is easy:

```
openfile = open('mini.txt')
```

The `mini.text` file contains these five words:

```
lists
dictionaries
strings
records
arrays
```

You will write a simple demonstration program to check if a word is in this list.

1 Copy these words from the external file into a Python list structure. We can open the external file and do this easily.

   The first line defines an empty list. We then use a for loop to read one word at a time and append the word to this list.

```
wordList = []
openfile = open('mini.txt')
for word in openfile:
   wordList.append(word)
print(wordList)
```

   Copy and run the code.

2 The printout will be:

```
['lists\n', 'dictionaries\n', 'strings\n', 'records\n',
   'arrays\n']
```

   Annoyingly, the new line character `\n` is printed after each word. We remove this by applying the built-in `strip()` method introduced previously.

   Replace `word` with `word.strip()` inside the `for` loop and run the program to test the result.

3 Replace the print line in step 1 by coding the following algorithm using Python:

```
while True
   ask user for test word
   if test word is in wordlist
      print 'Yes'
   else print 'No'
```

4 Replace the external file `mini.txt` with `words.txt`, which is a very complete list of English words. Use the program to find at least one word belonging to English that you have never seen before and one that you know exists but is not in our file!

A solution is provided on Nelson MindTap, named *files_demo.py*.

ACTIVITIES

Identify

1 Identify three factors to be considered when choosing a computer language.

2 What is the difference between an array and a record?

3 How do you declare an array in Python?

4 Select three popular computing languages and identify one strength of each.

Analyse

5 Outline the difference between a list and a dictionary structure.

6 Starting with an empty array called `colours`, write a simple program to add new colours if they have not already been added.

Extend

7 Find out how to check whether an integer is odd or even and write code to do this using the language you use in class.

8 Research why computer languages use indices for arrays starting at 0 rather than 1.

15.5 Error detection

Skill builder
Fixing syntax errors

When we design a program, we want it to work perfectly! We certainly don't want any errors in the program. Despite our best efforts, errors may be introduced into the program code. We will probably find some of them before we think the job is finished, but perhaps not all of them.

There are different types of errors made in programming. Understanding what they are and how they are introduced is helpful, as we all like to avoid making errors.

The main types of errors are:

- syntax errors
- logical errors
- run-time errors.

INFOBIT

Error-handling code is included to deal with potential errors during program execution. Error-handling may simply cause the program to terminate or quit with an error message or it may deal with the error, taking a different path of program execution.

Syntax errors

Syntax errors are like grammatical or spelling errors. In programming, we must obey the rules of the programming language or it will not run. Modern programming languages provide syntax checkers, which check the line of code you are writing as you write it and will warn you if you make a syntax error.

Syntax errors are the most common type of error made when learning a computer language. It is very easy to omit a colon or parentheses, put a space in the wrong place or fail to indent. For example, the following code written in Python:

```
for x in range(1, 11)
    print(x)
```

will produce this message:

```
invalid syntax
```

because Python's rules state that 'for' statements must end with a colon. The indent on the line for the print (x) is also an important feature of Python syntax.

The corrected code is:

```
for x in range(1, 11):
    print(x)
```

Syntax errors are very easy to miss, aren't they? Describe what the output will be for this two-line program. Were you correct?

SKILL BUILDER

Fixing syntax errors

This version of our guessing game code contains four syntax errors. You will see how easy it is to miss syntax errors – one is quite subtle!

Copy and run the game code and see if you can identify and correct the errors.

```
from random import randint
secret = randint(1, 100)
print(secret)
flag = True
while flag == True:
    guess = input('Your guess?: ')
    if guess < secret:
        print('Too low')
    if guess > secret
        print('Too high')
    else:
        print('Correct!')
        flag = false
```

A solution is provided on Nelson MindTap.

Logic errors

Logic errors in a program can be caused by:

- a failure by the programmer to understand the problem
- an incorrect algorithm.

Logic errors include:

- using the wrong mathematical formula, such as multiplying two numbers when you meant to divide – this type of error is hard to find, as the code will usually execute without complaint but the output data will be incorrect
- putting statements in the wrong order
- leaving out an essential step.

Imagine a banking program that added the withdrawn amount to your balance each time you withdrew money from your account instead of subtracting it. Each time you withdrew cash, your balance would get bigger. In your dreams only, of course!

The following Knowledge builder explains how desk checking can help discover logic errors.

9780170442787

KNOWLEDGE BUILDER

Error checking

Identifying logic errors in an algorithm before coding saves a great deal of energy later! In the guided project in Chapter 4, you developed a second, more advanced HiLo game. Refer to this algorithm now. Here you will use Table 15.2 to check for logic errors.

It is important to test all possible paths through an algorithm, including boundary values. Thorough testing should cover each of these tests:

In addition, tests should be added for `secret` when it takes on a value at each end of the range 0–100. That is, when `secret = 0` and `secret = 100`.

Table 15.2 Desk check table to check for logic errors in the number guessing game.

Path to be checked:	Suggest a suitable test value for `guess` given the `secret number = 30`	Expected outcome	Tick if algorithm is correct
`abs(guess - secret) = 0`			
`0 < abs(guess - secret) < 5`			
`abs(guess - secret) = 5`			
`5 < abs(guess - secret) < 10`			
`abs(guess - secret) = 10`			
`10 < abs(guess - secret) < 20`			
`abs(guess - secret) = 20`			
`20 < abs(guess - secret) < 40`			
`abs(guess - secret) = 40`			
`abs(guess - secret) > 40`			

SKILL BUILDER

Debugging logic errors

This program contains logic errors. Can you locate and correct them?

1 Copy the code and run it. As in medicine, the symptoms at first can be very confusing, making it difficult to locate the errors. Sometimes multiple errors interact.

Symptoms are:

- the code only allows a single guess
- the code reports *high* when it should report *low* and vice versa
- the program ends, reports both *Too high* and *Correct!* when a guess lower than the secret number is chosen.

```
from random import randint
secret = randint(1, 100)
print(secret)
flag = True
while flag == True:
    guess = int(input('Your guess?: '))
    if guess < secret:
        print('Too high')
    if guess > secret:
        print('Too low')
    else:
        print('Correct!')
    flag = False
```

Fix this code. A solution is provided on Nelson MindTap.

2 Add an option to play the game again.

Skill builder resource
Debugging logic errors

Run-time errors

Some errors will only show up when a program is executing. A common cause for run-time errors are attempts to perform an illegal operation. This might be as simple as:

- dividing a number by zero
- trying to open a file that no longer exists or has been renamed
- attempting to open a file using the wrong name or the wrong path
- performing a calculation that results in a very large number and then trying to assign it to a variable that is too small to hold it

Good programmers will try to avoid run-time errors using a range of strategies, including:

- careful desk checking
- peer review of the program code by someone who did not write it
- carefully testing program code against a range of data, including data that is deliberately wrong.

INFOBIT

A Commonwealth Bank customer found that more than $230 million had been deposited into his account! Someone had entered the account number into the field for the deposit amount, but the system didn't alert anyone. The bank apologised and corrected the mistake. What type of error is this?

15.6 Correcting errors and program documentation

Sometimes, despite our best efforts, a program does not work as it should, even after a program's algorithm has been desk checked and all syntax and run-time errors eliminated from the code. Programs are often very complex and so special debugging tools have been developed to locate hard-to-find errors.

It is extremely important you document how a program was designed and what changes have been made to it over time. There are many examples of valuable code that had to be abandoned because they were not properly documented.

Debugging tools

Most programmers use a special collection of tools to program, called an integrated development environment (IDE). This usually includes specialised debugging tools to help the programmers find certain types of errors. Most syntax errors are very easily found by the IDE debugger.

Debuggers allow us to do things like 'watch' the lines of code while they execute, stepping through the program line by line, and enabling us to see what happens to the value of each variable.

These are the main tools usually provided:

- Breakpoints: These are markers that signal that the execution of the code is to halt at that line of code. The programmer can then examine the values of the variables at that intermediate point in the code's execution.
- Program stepping: The program can be made to halt (using a breakpoint) and then the code can be executed one line at a time ('step by step'). The programmer can see if the lines of code and modules execute in the correct order and variable values change correctly.
- Variable tracking: Debuggers usually provide this tool. Each variable is listed and its value can be inspected at any point in the execution of the code. These values can also be compared to those predicted in desk checking.

Stubs

A **stub** is simply a temporary short piece of code, a 'filler', used to fill the place of code that has not yet been written or that is not yet working. It works by returning a dummy value used by the rest of the program to allow other parts of the program to work and be tested.

Flags

A **flag** is an even shorter piece of code, usually a single Boolean value, a 1 or a 0. It is like a flag used in a race; if it is down, the race has started! A flag allows programmers to see if a section of code has been executed, as it will change from one value to the other. The programmer examines the flag's final value. If it has changed, the code in that section must have been executed.

Debugging output statements

A **debugging output statement** is a print statement temporarily inserted by a programmer in the code. These can be used to print a message that the code has been executed (like a flag) and to print out the current values of variables.

Program documentation

When we write programs, it is very important that we properly document our work (Figure 15.11). Our program design will always include a clear statement of the purpose of the program and the algorithm that the program uses.

There are two main categories of documentation for programs:

- internal documentation within the program code. There are two main methods of internal documentation: using comments in the program code and using self-documenting variable names
- external documentation, such as help manuals.

Using comments in program code

Programming languages enable us to leave comments within the program.

The comments in the source code of the program are for the benefit of the programmer and any other person who may have to work on the program. Experienced programmers organise their comments neatly, so that they don't make it hard to read the program instructions.

Comments in Python begin with the symbol # and are shown in red, to make it easy to distinguish them from code. For example:

```
def sum(a, b):
    total = a + b; # total is a local variable.
```

When a program is compiled by the IDE, or interpreted by the language interpreter, comments are ignored.

Comments are used for:

- identifying the author of the program and the date it was last modified
- explaining what sections of code do
- marking parts of code that have been modified
- 'commenting out' code that the programmer may wish to suppress during testing.

Variable naming methods

Meaningful selection of variable names is a very important method of internal documentation, as it makes code easier to understand. The variable name can be made of multiple words that are concatenated (joined together), such as: `StreetAddress`.

External documentation

Outside our programs, we make use of other forms of documentation for different purposes and for different people. For the users of the program, we usually prepare a user manual – a guide on how to use the software we have developed.

We may also prepare a help file so that users can look up what they need to know by using the help menu of the software.

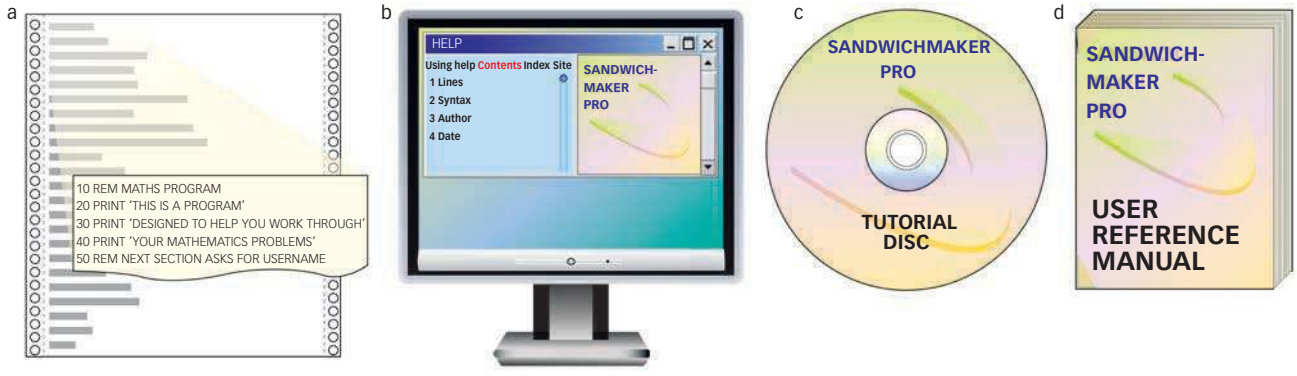


Figure 15.11 Documentation methods: (a) documenting program code with comments, (b) help files included with the software, (c) tutorials on disc or online, (d) user reference manual.

Chapter review quiz

ACTIVITIES

Identify

1. Identify three categories of programming errors.
2. At what point in the development of a program does a programmer discover a run-time error?
3. Identify three tools commonly found in a debugger.
4. Name two internal and one external types of documentation used by programmers.

Analyse

5. How can syntax errors be avoided?
6. How can logical errors be avoided?
7. What is one important feature of intrinsic documentation and why is it important?

Extend

8. Find out the meaning of the term 'computer bug' and research when the term was first used and by whom.

15.7 Guided project 1: A menu-driven game collection

Project resource
A menu-driven game collection

In this Guided project, you will develop a menu-driven collection of great games (Table 15.3). You will learn many skills: how to write your own functions, local and global variables, file handling, desk checking, error checking, interface design and Python's dictionary structure.

One of the strengths of this project is that the games you develop are arranged in order of difficulty. For your first game, you copy code provided for you. For your second game, we leave easier sections for you to complete. In the third, we provide the code but suggest features for you to add. For your fourth and fifth games, we only provide an algorithm and code snippets, and in the sixth you are completely on your own!

In this way you work as far as you can but still end up with a functioning project!

Identifying and defining

Outline of the guided project

Your program will offer players a number of games, from which they select one to play.

You start by developing a menu function, from which players select games. You then add your first game, the simple Hi-Lo game from Chapter 4. Next, you will add the advanced Hi-Lo game version also developed in that chapter.

You then program a game of *Rock, paper, scissors*.

Finally, you have the opportunity to develop an additional module – a game of *Simon* – and suggestions are given for other games for you to code on your own.

Note that completed versions are provided for students on Nelson MindTap should you need them. Solutions for Simon, Mastermind and Wordle are provided online for teachers. Algorithms are written in structured English and where code is provided, we have used Python.

Table 15.3 Your games project modules.

Project modules	What is provided	What you learn
Game selection menu	Algorithm and code template	Simple user interface design: creating a text-based menu
1 Hi-Lo simple	Algorithm and complete working code	Modifying code using functions
2 Hi-Lo challenge	Code base with sections to complete; student solution online	Adding features to existing code
3 Rock, paper, scissors	Algorithm and uncompleted code; student solution online	Refining algorithms and completing code
4 Simon	Algorithm and code snippets; teacher solution online	Developing code from an algorithm and debugging
Invent your own!	Suggested games and outline of mastermind game for you to code on your own; teacher solution online	Developing code for a game

Design thinking

Researching and planning

Developing the menu

iStock.com/Marija Arsonova

We need users to select a number from a menu to choose a game. Their choice will then call the corresponding game function.

Our algorithm will look something like this:

```
print main menu of available games
ask user for choice
call the function for game chosen
at end of game return to main menu
```

You learned earlier that whenever an action is to be performed repeatedly it is helpful to express it as a function. Then, whenever needed, that function can be called, rather than having to copy and paste the code over again.

So, we first create a function for a menu using just two games.

A function to print the menu

1 As most of the games will use Python's random function, make this the first line of your code:

```
import random
```

Our first step will be to write a function to print the menu of games. We will call it `print_menu()`. We already know from Table 15.3 the names of the games we will offer. We start with two games then if you develop others they can be added easily.

Copy this first function into your project file. Save your whole project using the name *games_collection.py*. We will refer to the complete project file as *games project*.

```
def print_menu():
    print ('1. Hi-Lo simple')
    print ('2. Hi-Lo challenge')
    print ('3. Rock, paper, scissors')
    print ('4. Simon')
    print ('5. Quit playing games!')
```

The game loop

We need this code to continue running as long as a user wants to play. To do this we place code for the game choices inside a `while True` loop and use the `pass` instruction as a placeholder for code yet to be written.

2 Copy and complete the missing section in the following code and position it immediately following the print menu function you created in step 1. Add comments to explain important features (as we did in the first line).

```
# this is the game loop
while True:
    print_menu()
    choice = int(input('Enter choice [1-5]: '))
    if choice == 1:
        print ('Hi-Lo simple selected')
        pass
    elif choice == 2:
        print ('Hi-Lo challenge selected')
        pass

    complete missing code for the other choices

    elif choice == 5:
        break
    else:
        input('Invalid choice. Try again...')
print('Thanks for playing!')
```

3 Save your project. Of course, there are no games yet to run. Notice the `break` statement in the code for choice 5. In Python, a `break` statement terminates a loop and moves on to the next statement outside it so we will use it to allow the player to end.

Run the code in step 2 and identify any errors. Check this menu is printed:

```
1. Hi-Lo simple
2. Hi-Lo challenge
3. Rock, paper, scissors
4. Simon
5. Quit playing games!
```

4 Let's make this printout prettier! You have learned printing an item like a single character multiple times is as easy as multiplying it by an integer: `print (30 * "-")` will print 30 hyphens in a row.

Using this approach, modify the print_menu() function to appear more like this (much prettier!):

```
-------------------- MENU --------------------
1. Hi-Lo simple
2. Hi-Lo challenge
3. Rock, paper, scissors
4. Simon
5. Quit playing games!
----------------------------------------------
Enter your choice [1-5]:
```

The complete code so far is:

```
def print_menu ( ):
    print (30 * '-' , 'MENU OF GAMES' , 30 * '-')
    print ('1. Hi-Lo simple')
    print ('2. Hi-Lo challenge')
    print ('3. Rock, paper, scissors')
    print ('4. Simon')
    print ('5. Quit playing games!')
    print (75 * '-')

## game functions will be coded here ##

# the game main loop
while True:
    print_menu()
    choice = int(input('Enter choice [1-5]: '))
    if choice == 1:
        print ('Hi-Lo simple selected')
        pass
    elif choice == 2:
        print ('Hi-Lo challenge selected')
        pass
    elif choice == 3:
        print ('Rock, paper, scissors selected')
        pass
    elif choice == 4:
        print ('Simon selected')
        pass
    elif choice == 5:
        break
    else:
        input('Invalid choice. Try again...')
print('Thanks for playing!')
```

Producing and implementing

Adding the first game: Hi-Lo simple

iStock.com/VikiVector

5 For the first game function we provide the code and you turn it into a function.

You have code for this game already! In Chapter 4 you created a simple Hi-Lo guessing game. In Python, functions must be defined before they are called or the program will report it doesn't recognise them. For this reason, function definitions of all games must be placed before the main loop that calls them. Position this function as indicated by:

```
## game functions will be coded here ##
```

Add the following code for this simple game, naming it now as a function called `HiLoSimple()`. Make sure that following the first line of the function definition, all the code is indented correctly. Make sure you have imported the random module as your first statement in your main program as stated towards the start of the project!

```
def HiLoSimple():
      secret = random.randint(1,100)
      flag = True
      def guessRequest():
         newguess = input('Your guess?: ')
         newguess = int(newguess)
         return newguess
      while flag == True:
         guess = guessRequest()
         if guess < secret:
            print('Too low')
         elif guess > secret:
            print('Too high')
         else:
            print('Correct!')
            flag = False
```

You will notice we created a function called `guess_request()` using the game's original code. We have done this to illustrate that functions can be nested inside other functions.

6 Add the call to the `HiLo` function to replace the word pass in the menu function in choice 1 as shown here:

```
if choice == 1:
   print('Hi-Lo simple has been selected')
   HiLoSimple()
```

Run this program and check `HiLoSimple` works correctly when users select choice 1.

After fixing any errors, add at least five internal comments (beginning with #) to explain what each part of the code does.

Adding the second game: Hi-Lo challenge

For this second game function we provide code and suggest enhancements for you to complete on your own. In the Skill builder in Chapter 4, section 4.9, we introduced a more advanced version of the Hi-Lo game with descriptive words used to describe how far guesses are from the secret integer (similar to the 'pin the tail on the donkey' party game). See also the Knowledge builder in this chapter, section 15.5.

Work through the following code to understand it.

```
def HiLoChallenge():
    secret = random.randint(1, 100)
    print(' For debugging number is', secret)
    flag = True
    count = 0
    def play_again():
            play = input('Would you like to play again? ')
            if play in ['Y','y', 'yes', 'Yes', 'T', 'True']:
                flag = True
                print('Here we go again')
                HiLoChallenge()
            else:
                flag = False
                print('Bye-bye')
    while flag == True:
        guess = input('Hello! Guess my number between 1 and 100 ')
        guess = int(guess)
        count = count + 1
        if guess == secret:
            print('Correct! You took',count,'guesses')
            play_again()
            break
        elif abs(guess-secret) <= 5:
            print('Boiling')
        elif abs(guess-secret) <= 10:
            print('Hot')
        elif abs(guess-secret) <= 20:
            print('Warm')
        elif abs(guess-secret) <= 40:
            print('Cold')
        elif abs(guess-secret) > 40:
            print('Freezing')
```

Observe:

- `abs()` in Python returns absolute value, or size of a number, without a sign. Thus `abs(x) = 5` means x equals either 5 or –5 and so `abs(x) < 5` means all values *between* 5 and –5. This is the same as writing: `(x < 5) AND (x > -5)`

On your own:

- Write additional code for 'Sizzling', 'Very hot', 'Very warm' and 'Very cold'.
- Add code to count the number of turns and report this to users.
- Describe a strategy for playing this game. Play the game and check it runs correctly before moving on to step 8.

7 Code your `HiLoChallenge` function after game 1.

8 Add the call `HiLoChallenge()` in the menu function to call the function by replacing the word `pass`.

9 Run the game package and check choices 1, 2 and 5 all work correctly.

Extension: At present players may enter integers that are out of range. Add code to repeat the input request when this occurs. For another challenge, repeat this request if entry is non-integer. See the Nelson MindTap solution if you need help.

Adding the third game: rock, paper, scissors

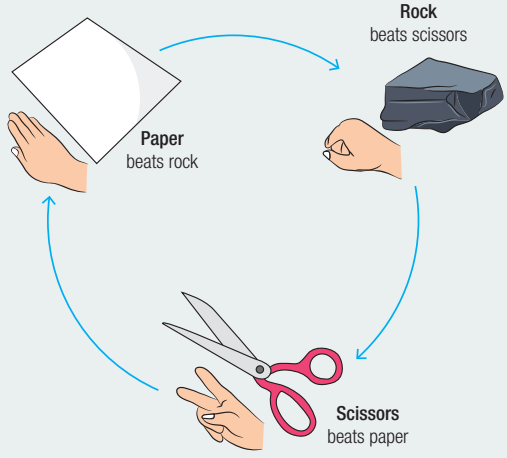


Figure 15.12 Rules for rock, paper, scissors.

For this third game function, we provide an algorithm in the form of a flowchart and you complete parts of the code.

10 Begin by playing rock, paper, scissors with a friend. See Figure 15.12 for the rules.

Computational thinking

INFOBIT

Using image pattern recognition: Researchers at the University of Tokyo created a robot with a hand that plays rock, paper, scissors. Their robot wins every time using a high-speed camera to identify within one millisecond which shape the opponent's human hand is making as it swings forward, then it rapidly responds with a winning move, without the opponent noticing the tiny delay.

Weblink
Janken (rock-paper-scissors) Robot with 100% winning rate

Develop the algorithm

Flowchart

Design thinking

11 Examine the logical flow using this flowchart.

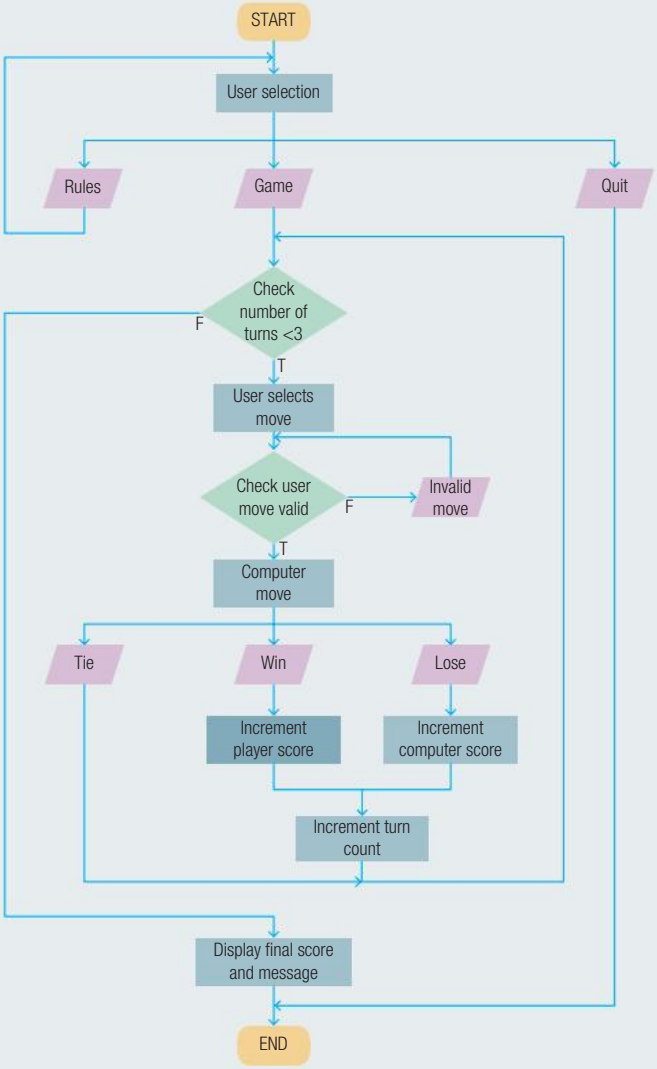


Figure 15.13 Flowchart for rock, paper, scissors game allowing three turns.

IPO chart

Complete Table 15.4 showing input, processes and output. Refer to Figure 15.13 if you need help.

Table 15.4 IPO table for rock, paper, scissors game.

Input	Processes	Output
User selects scissors, paper or rock	Check validity user choice	Win or lose
computer move		
player move		
player wins		
number of turns		

HUMAN BEINGS AND RANDOMNESS

Although rock, paper, scissors is considered a fair game, programmers have written algorithms that track a human player's choices and search for patterns. Using these patterns, the algorithm can make a prediction of an opponent's next move. In this way it gains an advantage, as human beings, no matter how hard they try, do not play with perfect randomness. An international competition is held annually and the best players can predict their opponent's moves with great accuracy!

Design an algorithm

Computational thinking

Often when programming we can devise a shorter way of achieving the same result. Writing a shorter and more efficient algorithm can be important in large-scale applications. If we can reduce the number of times the computer has to perform a decision test, it might make a huge different to the load on the computer or on the processing time. Imagine how critical this is for Google searches! Thinking about the algorithm above, can you eliminate some tests?

9780170442787

Here is an inefficient algorithm for the game. First work through the algorithm while thinking how it might be simplified:

```
START
select rules, play or quit from menu
initialise turns and scores to zero
WHILE turns is less than or equal to 3
  user selection
  WHILE user selection is not valid
     user selects either scissors, paper or
       rock
  computer randomly selects scissors, paper
    or rock
  IF user selection is same as computer
    selection THEN tie
  ELSE IF user is scissors and computer is
    rock THEN player loses
  ELSE IF user is scissors and computer is
    paper THEN player wins
  ELSE IF user is paper and computer is
    scissors THEN player loses
  ELSE IF user is paper and computer is rock
    THEN player wins
  ELSE IF user is rock and computer is
    scissors THEN player wins
  ELSE IF user is rock and computer is paper
    THEN player loses
  ENDIF
  IF player wins THEN increment user score
    and total turns
  ELSE IF player loses THEN increment
    computer score and total turns
  print outcome, turns and scores
print message and declare winner
END
```

12 What if the algorithm only handled winning moves? All others could be assumed to be losing moves or ties. Rewrite the algorithm using pseudocode this way. If you need help, a version is provided on Nelson MindTap.

Desk check

13 Now you have an algorithm, draw up a table and desk check the logic, showing expected outputs for all inputs. Remember every path through a program should be tested (the inefficient algorithm will help here). A solution is provided on Nelson MindTap.

Coding the algorithm

14 This requires Python's random module to be imported. Your first line will be `import random`. Define a function and initialise local variables needed by it.

```
count = 0
player_score = 0
computer_score = 0
```

15 Create an array data type (a list in Python) to define player choices:

```
choices = ['rock', 'paper', 'scissors']
```

16 Create a `while` loop to keep playing the game for three turns.

17 Below is a version with the scissors section of the code completed and the remainder left for you. The code for winning moves with paper and rock is similar to that provided for scissors. The complete version is available online. Copy this and complete the code where indicated. Add this new game function following the last game.

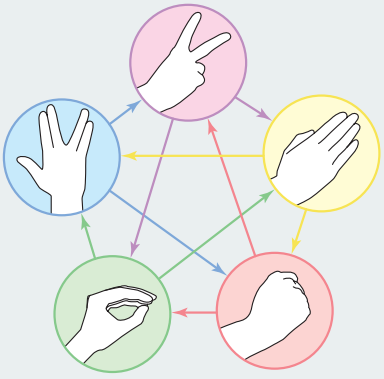
```
def RPSPlay():
    # playing against the computer
    # initialise variables
    count = 0
    player_score = 0
    computer_score = 0
    # initialise game choices as a list data type
    choices = ['rock', 'paper', 'scissors']
    print (66 * '-')
    print('Player VS Computer')

    while count < 3:
        player = input('Type rock, paper or scissors ')
        while player not in choices:
            print('invalid input')
            player = input('Type only rock,paper or scissors ')
        computer = random.choice(choices)
        print('Computer has chosen', computer)

        # player wins with 'scissors'
        if player == 'scissors' and computer == 'paper':
            print('Scissors cuts paper! You win!')
            player_score += 1
            count += 1
```

```
add code to handle player wins with 'paper'
add code to handle player wins with 'rock'
            # losing combinations
            elif player != computer:
                print(computer, 'beats', player +'. You lose!')
                computer_score += 1
                count += 1
            # a tie
            else:
                print('A tie!')
add code here to print number of turns, and running score totals
        # reaches here when turns reach 3
        if player_score > computer_score:
add code here to congratulate for win and handle loss
```

18 Add all the code to your main menu-driven game program in choice 3. Consider the user interface design and add empty print statements to tidy the output.

If you are using Python, the complete code is available online if you need to troubleshoot your code.

Computational thinking



Winning moves

- Scissors cuts paper
- Paper covers rock
- Rock crushes lizard
- Lizard poisons Spock
- Spock smashes scissors
- Scissors decapitates lizard
- Lizard eats paper
- Paper disproves Spock
- Spock vaporises rock
- Rock crushes scissors

Figure 15.14 Scissors, Paper, Rock, Lizard, Spock: a variation on rock, paper, scissors.

Extend this game

19 A great way to develop your thinking and coding skills is to make improvements to an existing game. Here are some suggestions for improving rock, paper, scissors. Try some of your own as well! You will find versions of solutions to these challenges on Nelson MindTap.

- Adding a time delay to simulate the computer 'thinking' about its move will dramatically enhance user experience. In Python, import the module called 'time' at the top of the whole program: `import time`.
- Then, use its `sleep()` function like this: *time.sleep(delay_in_seconds)*.
- Add a function to print the rules.
- Modify your code to allow players to type only the first letter of their choice: S, P or R in upper or lower case.
- The game continues for three turns even though the first to two is a win. Have the game finish when a player reaches two.
- A modified version of this game is Scissors, Paper, Rock, Lizard, Spock, which was featured in the TV series *The Big Bang Theory*. The logic is summarised in the diagram in Figure 15.14. Modify your code to play this version of the game.

Creating the fifth game: Simon

Alamy Stock Photo/Chris Willson

Figure 15.15 A toy version of the game Simon.

For the fifth game function we provide an algorithm and you develop the code by yourself.

Have you ever played a game where you have to remember a sequence that gets longer every time until finally you get it wrong? Your task is to recreate this game!

A popular version of this game is a toy called '*Simon*' (different to 'Simon says!'), with four coloured buttons, each making a different sound when pressed. Simon lights up its buttons in a random order, after which the player has to repeat the order by pressing the buttons. Here we

replace the buttons with random integers; however, you could print names of colours instead if you like!

Less guidance has been offered for this game to make it a more challenging one than the others.

21 Our approach will be to display a pattern of random integers, adding one more each time. Following a delay, we scroll this pattern off-screen by printing multiple empty lines and then ask the user to recall the sequence.

Here is the algorithm:

```
create an empty list called sequence
set a flag to true
WHILE flag is true
    create new random integer
    add to end of list
    print new sequence
    time delay
    scroll pattern out of sight
    ask player to input sequence
    IF correct THEN
        set flag to true
        print success message
    ELSE
        set flag to false
        print fail message
    ENDIF
ENDWHILE
```

Here are some clues to assist you coding:

- Use a `while` loop to keep the game repeating as long as the player is remembering correctly.
- Create a new number each turn with `next = random.randint(1,9)`.
- Add an integer each time to your pattern using `sequence.append(next)`.
- Create two lists, one to hold the actual sequence and one for the player's guesses.
- Apply UI principles here, even for this simple input action. The most natural way to ask someone to type a sequence would be typing with spaces between each integer. The following code snippet uses the features of lists introduced in the Knowledge builder in section 15.6 to separate each integer and append each to a list. Examine these features carefully:

`input()` is a function to receive data from the user.

`split()` is a function that looks for spaces and splits the string input into separate items.

`int()` is a function to turn string input into integers.

```
print('type integers with spaces between')
myguess = input()
myguess = myguess.split()
playersequence = []
for i in myguess:
    playersequence.append(int(i))
```

- Check if the two lists `playersequence` and `sequence` are identical.
- Consider that users may not have time to see the pattern. Slow this down using a time delay. In Python, import the module called '`time`'. Then use its `sleep()` function as follows: `time.sleep(time_in_seconds)`

22 Test your code thoroughly, and once you have your code working, turn it into a function and add it to your main program.

23 Here are some further suggestions for your code:

- Add a message to report the count of the items in the pattern each time.
- Modify your code to ask the player if they want to play again.
- Allow players a choice for the range of integers.

Now is the time to thoroughly test each game in your project. Have others play and evaluate them. Maintain your documentation and respond to suggested improvements.

Testing and evaluating

Extend this project

Here are some suggestions for other games (listed in approximate order of difficulty):

- Nim
- Noughts and crosses
- Battleships
- Mastermind (see Figure 15.16)
- (Challenge) Wordle (popular daily word game based on Mastermind). Use the knowledge you gained when handling the *words.txt* file of English words, then eliminate all but the five-letter words from it – or create a smaller selection manually. If you get stuck, check out our version on Nelson MindTap.

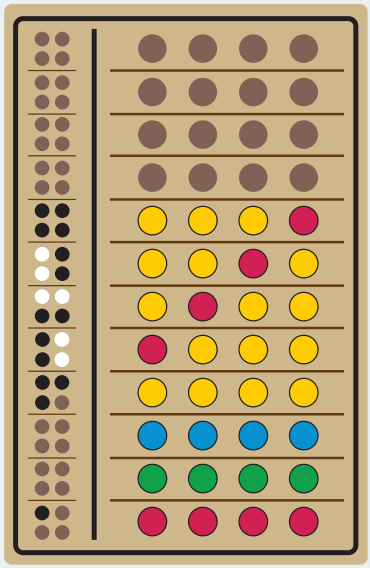


Figure 15.16 In Mastermind, players crack a hidden secret code by discovering the correct colours and their correct order. Each correct colour in the wrong order is given a white marker while each correct colour in correct order is given a black marker. Follow the logic through the eight turns of this example.

15.8 Object-oriented programming

So far, our programming has been procedural. Procedural programs represent problems as a collection of instructions, much like a cooking recipe lists ingredients and then the method. Procedural programs contain variable declarations and functions, which organise the program. This is a simple and direct way to provide instructions for a computer.

The real world, however, is made up of objects. These objects have a certain appearance and certain behaviours: in the case of a leaf, its colour, size and growth rate; and in the case of a car, its colour, model and engine characteristics. Object-oriented programming (OOP) languages recognise this and use a 'virtual object' called a *class*, which defines an object's features and behaviour in one neat package.

This class is then used as a template to create actual objects of that type for use in the program. For example, in a game where there are various enemies, a single class called `Enemy` might define the properties and actions for enemies of any kind and then be used to create an ogre, a zombie and any other baddies the game's designer wants.

Examples of OOP languages include C++, Java and Python. In this chapter we will be using Python; however, all languages follow the same principles and once you learn one language it is not difficult to transfer your knowledge.

The OOP approach

By now you have seen how a procedural approach to coding can actually result in long slabs of code and make interpretation difficult – especially for a new programmer!

In OOP, data relating to an object is bundled inside its class. By collecting all data relating to an object in one place and grouping it in a class, a programmer no longer has to worry about the detailed operation of different objects and can just use them when needed.

The concept of a class

Grouping related data (**attributes**) and functions (methods) is a feature of OOP known as *encapsulation*. Put simply, a class's *attributes* define its properties and its *methods* describe the things it can do.

This class becomes a template for the programmer to use. For example, if a class describes what a circle is with a formula to find its area, then when a programmer needs a circle of a particular radius, they can use this class to make one. They do not need to know the formula to calculate its area as that method would be packaged inside the circle class. The actual circle it makes is called an *instance* of its class.

Another way to think about this, is that in OOP, classes act like machines manufacturing objects in a factory. Factory operators do not need to know what happens inside their machines, they just need to press the right buttons; all the detail is hidden.

The analogy of a motor vehicle factory that produces cars is commonly used to explain OOP. Figure 15.17 shows the design or blueprint of a 'virtual car'. At this stage no actual car exists.

All cars, no matter what model they are, share common characteristics. We can imagine a 'super' class called 'car', which produces all car models. The class would describe the attributes (model, colour, chassis) and methods (ignition, braking) cars possess to save us re-entering them each time a new car is made.

Any car would have certain properties shared with all cars, such as the property of colour, but some others that are unique to it, such as autonomous driving. All we do is define a class called Car, which acts like a design blueprint for making any kind of car. Creating an instance is like starting the production line and seeing the first actual car roll off.

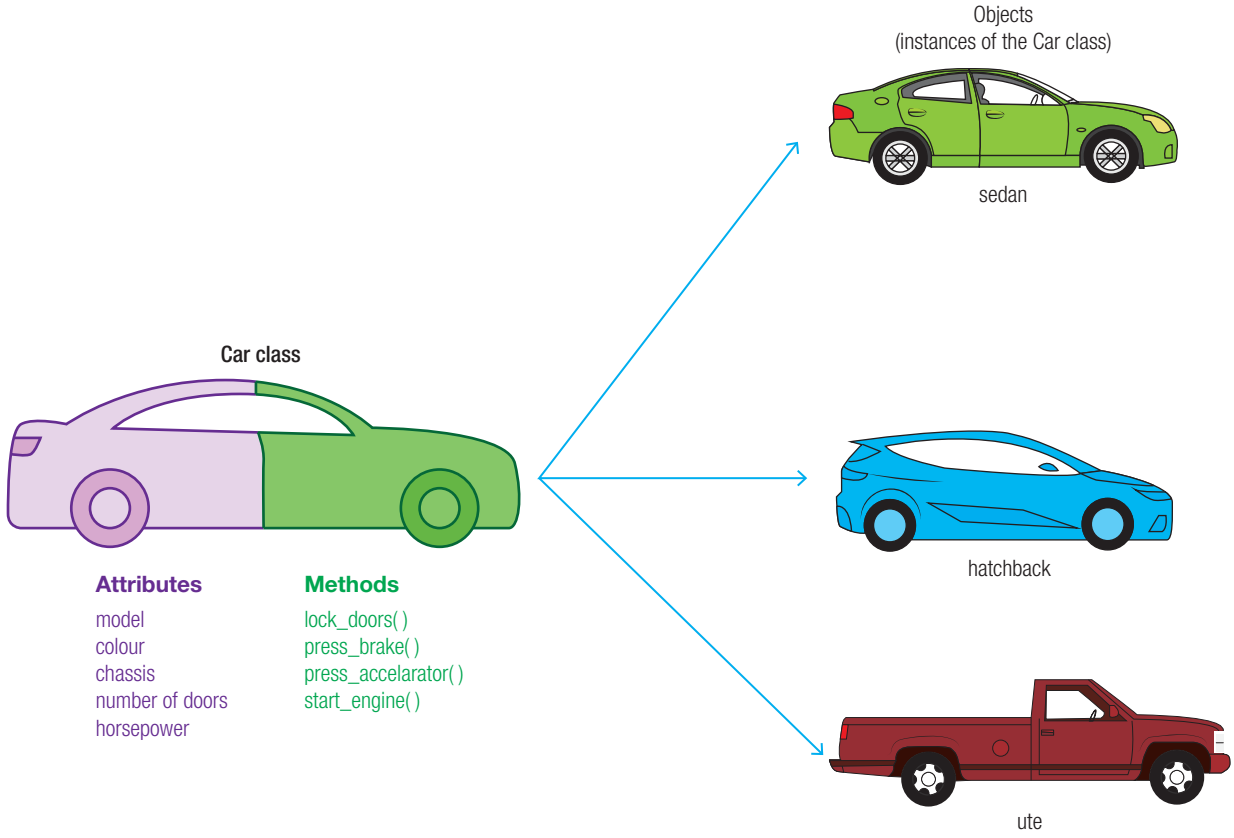


Figure 15.17 The OOP approach to programming is based on defining classes, which have *attributes* (the object's properties) and *methods* (what the object can do). Here we imagine a 'super' class called `Car` to which all car models belong.

INFOBIT

One programmer has written: 'My first struggle with OOP was getting it in my head that methods are like functions and attributes are like variables. It seems ridiculously stupid now, but each time I had to automatically translate that in my mind. It is a lot like learning a spoken language: it starts off as head knowledge you have to consciously translate, but at some point, you realise you are finally not doing a translation in your head each time.'

The benefits of OOP

So far, you have been using a procedural approach in your programming and have begun to use functions to organise these better. OOP offers three further key advantages beyond those of functions:

- In a very long program it can be challenging keeping track of the many functions. There also may be many variables scattered throughout the code, and many may relate to one another. OOP collects related code together in one place.
- OOP reduces the need for repetition as the one class can act as a blueprint to create many instances.
- OOP hides the detail of the attributes and methods of an object, allowing programmers to focus on the big picture.

Because OOP is a different way of thinking, it takes a while to get used to. Instead of first breaking a problem down into lots of functions, as you have been doing so far, objects are identified and their characteristics become part of that class. For the project at the end of this chapter you will be using OOP to design a role-playing adventure, but first you need to learn the syntax your programming language uses for writing OOP code.

Systems thinking

Systems thinking

15.9 Classes, attributes and methods

As explained, OOP is an approach to programming that groups all data connected with an object into packages or 'containers' called *classes*. These classes describe an object's properties (called *attributes*) and behaviours (called *methods*).

The *attributes* of a class are the object's characteristics.

The *methods* of a class define what it can do.

A class object for a pet might describe its:

- name, age and species as its *attributes*
- sound and what it eats as its *methods*.

A class object for a circle might describe its:

- fill colour, stroke colour, radius and centre as its *attributes*
- formula for circle area as one of its *methods*.

Design thinking

By now you are probably wondering what OOP code looks like in Python.

Defining a class using Python is simple: the word 'class', followed by its name followed by a colon. Its code after that is indented:

```
class Car():
    more code
    more code
    ...
```

Other languages use similar syntax. Make sure you know the way to define a class in the specific language you are using.

Defining a class

We now have written a car class but it doesn't do anything! By adding attributes and a method to our car class we define what a car is and what it can do.

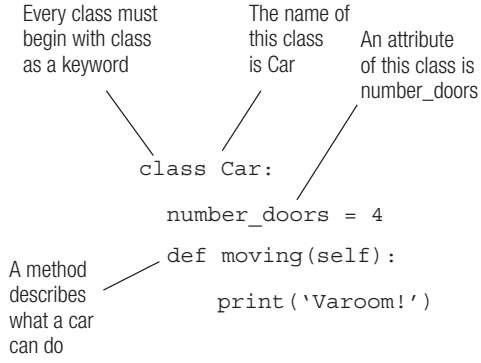


Figure 15.18 Adding attributes and a method to our car class.

See Figure 15.18. Here `number_doors` is a *class variable*, it is an *attribute* of our class `Car`. Here `moving()` is a thing our car can do, it is a *method* of our class `Car`. Methods are just regular functions written inside classes. They use Python's function syntax `def` in the usual way (the unexpected word `self` is explained in the next section).

Instances of a class

We will now create an *instance* of a car. If we write `mycar = Car()` and run it, we have 'manufactured' our first car! See Figure 15.19.

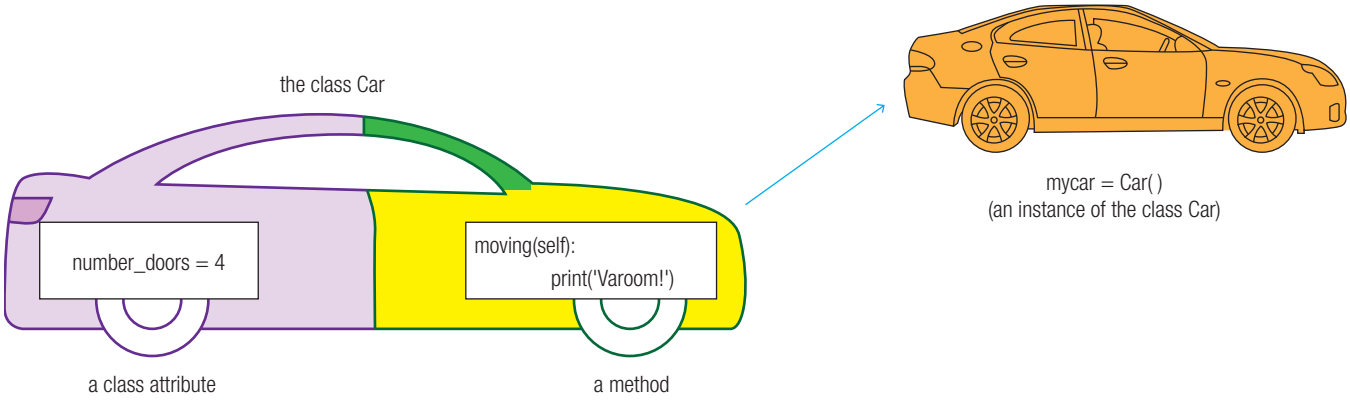


Figure 15.19 In this figure, `number_doors` is a *class variable* (describing an *attribute*) of the class `Car`. `moving(self)` is a *method*, which applies to all instances of the class `Car`. It is an action our car can do.

We include parentheses when calling a class because we sometimes need to provide (pass) values into our class. Here we have just made a plain car. Up to this point no cars existed (except in the mind of the car designer!).

In OOP language, we created an *instance* or *object* of the `Car` class with the name `mycar`. If we wanted to make more, we could repeat this but use new names:

```
mycar1 = Car()
mycar2 = Car()
```

and so on.

Instances inherit all the characteristics of their class. Notice the word `self` in our class method definition?

When an instance needs to use a method (such as `moving`), it passes its instance name (in this case, `mycar`) to the class method. This is the reason you must always place the word `self` first inside the parentheses of any class methods. If you forget, you would get an error. This is because the class would think this method belongs only internally (because its parentheses would be empty). It would be the same if you tried to pass an argument to an ordinary function that had no passing parameters defined inside its parentheses.

The following Skill builder coding activity lets you try these ideas for yourself.

SKILL BUILDER

Skill builder resource
Creating a car

Creating a car

We now know enough to build our own first car using Python OOP code!

1 Type out the Car class definition provided in Figure 15.19.

```
class Car():
    number_doors = 4
    def moving(self):
        print('Varoom!')
```

2 Manufacture your first car by creating an instance of it:

```
mycar = Car()
```

In step 1 you built the factory that makes cars by creating the `Car` class and in step 2 you manufactured your first car – the `mycar` instance of the `Car` class.

You're ready to start your car's engine up for the first time! OOP uses dot notation to call methods of the instances of a class. The name of the instance is followed by a dot, then the name of its method, followed by parentheses, similar to calling a function. In this case call the `moving` method to start the car.

3 Type `mycar.moving()` and run the program…what happens? Did you 'hear' the engine?!

Class attributes and instance attributes

In the earlier class definition for car, we defined a class attribute or class variable setting the variable `number_doors` to four. Such attributes are owned by the class itself. This means every car we make will have four doors.

Many cars, however, such as hatchbacks, have other numbers of doors. A hatchback has five, a ute may have two and a DeLorean has three! We deal with this using a second type of attribute called an *instance attribute*. Instance attributes are owned by instances of a class and not by the class. Unlike class attributes, instance attributes are not shared. Every instance or object has its own copy of that instance attribute. That is, for two different instances, instance attributes can be different values.

Using `init` and `self`

Examine the following version of the car class:

```
class Car():
    def __init__(self, number_doors):
        self.number_doors = number_doors
    def moving(self):
        print('Varoom!')
```

In this new version we can decide how many doors each new instance of our car has. In the first version above, all cars came with four doors! To do this we have used a strange new method with an unexpected name: `__init__`. Note the double underscores ('dunders') either side. This function is a special one called (in Python) an initialiser function. It is used to make this method special to avoid potential conflicts (it is similar to a constructor used in other languages).

In Python, instance variables, such as `number_doors`, must be included in the initialisation function `__init__()`. This assigns a parameter for the number of doors to all instances of our car class, allowing for different values to be passed as we create different cars.

The `init` function automatically runs whenever a class is used to create an instance. At first it is hard to get used to using such a strange name like `__init__`. The following will create a five-door instance of a car using our new car class:

```
myCar(5)
```

The second unexpected term is `self`. You must always include the word '`self`' inside the parentheses of `__init__()` as well as in any instance variables following. In our example, `self.number_doors` means that once we created our car using this class, we can access the number of doors using dot notation.

`self` delivers (or 'passes') attributes in the `__init__` method to all instances when they are created, ready for values to be given to them. Once the object or instance is created, it replaces the word `self` with the instance. Without the word `self` the class would treat the variables as belonging to the class only. The method `moving` used `self` for the same reason, to show it is a method belonging to instances of the class.

Now when a car instance is created using the class Car, we can decide how many doors that car has. This is much better than all cars having exactly four doors!

The following Skill builder will help you understand and use these ideas.

SKILL BUILDER

Skill builder resource
A simple OOP example: Using instance variables

A simple OOP example: Using instance variables

Time for you to own your own customised car! Copy the following OOP code into Python to create a more detailed car class. Code the last line as one unbroken single line.

1

```
class Car():
def __init__(self, name, number_doors, colour, model, cost):
    self.name = name
    self.number_doors = number_doors
    self.colour = colour
    self.model = model
    self.cost = cost
def moving(self):
    print('Varoom!')
def description(self):
    print(self.name,'is a', self.number_doors, 'door', self.colour, self.model,
      'costing $', self.cost)
```

2 Create a car with two doors by entering:

```
myCar = Car('Bertie', 2, 'cream', 'VW Beetle', 2500)
```

Notice that five parameters are being passed here to the class's init function.

3 Type the following and run your code:

```
myCar.moving()
```

All instances of the class Car will have this property! Did your engine start?

4 Type the following and run the description function. Don't forget the parentheses:

```
myCar.description()
```

Do you like this description of your first car?

5 We use dot notation to access instance variables in the same way as we did for instance methods. Type this and run the code again:

```
print('The number of doors in my car is', myCar.number_doors)
```

Does your car have two doors?

6 Now create another car instance for a friend and print out its description.

Overview: writing a class in OOP

To summarise, a basic class in Python looks like this:

```
class ClassName():
    def __init__(self, variable1, variable2, ...):
        self.variable1 = variable1
        self.variable2 = variable2

    def method1(self):
        ...
```

When writing your class definition use the following steps. If you are confused, match these statements with the example above. Also, completing the next Skill builder should help you make sense of this.

- Write the word `class` followed by its name, parentheses and a colon. If you are using class variables, define these next without using `self`. Class variables are not the same as instance variables. They belong to the class itself.
- Write a special function at the start of a class called `__init__(self)`
- List instance variables as the `__init__` method's parameters after the word `self`
- Declare these instance variables in the body of the `__init__` method and add `self.` (with a dot) in front of each of them: e.g. `self.model`
- Add any methods and include `self` as a parameter in parentheses for any methods instance variables need to access.
- Use `self` followed by dot notation whenever you need to call an instance method in your main code.

Examine the code for the Car class above and carefully identify these features.

SKILL BUILDER

A basic OOP example: Company employees

Imagine a program written for an animation company. We will use a class definition to describe its employees. To make this example simple to follow, we will restrict data required to name, surname and salary and we will add a *class variable* `count` to keep track of the number of employees, as this belongs only to the class, not instances.

We first initialise `count` to 0 but use the `__init__` method to increment the value of `count` every time an instance (a new employee) is created. This is a common technique to count the number of instances created by a class. Notice how class variables are used with the class name and dot notation.

Skill builder resource
A simple OOP example: Company employees

1 Type the following:

```
class Person:
    count = 0
    def __init__(self, name, surname, salary):
        self.name = name
        self.surname = surname
        self.salary = salary
        Person.count += 1
```

2 Define two employees by creating new instances of this Person class.

Add:

```
walt = Person('Walt', 'Disney', 120000)
nick = Person('Nick', 'Park', 100000)
```

3 We created two employees. Now test the code by typing:

```
print('Number of employees created is', Person.count)
print(walt.name, walt.surname, walt.salary)
```

4 Check output is:

```
Number of employees created is 2
Walt Disney 120000
```

5 You can make this class more useful to the finance department by adding the method `pay()` to the class in order to calculate pay using the annual salary. The class now becomes:

```
class Person:
    count = 0
    def __init__(self, name, surname, salary):
        self.name = name
        self.surname = surname
        self.salary = salary
        Person.count += 1

    def pay(self):
        pay = '$' + str(round(self.salary/26,2)
        return pay
```

6 Type on two single lines:

```
print('Fortnightly pay for Walt is',
  walt.pay())
print('Fortnightly pay for Nick is',
  nick.pay())
```

Run the code to check the output is:

```
Fortnightly pay for Walt is $4615.38
Fortnightly pay for Nick is $3846.15
```

7 In the `pay()` method you just added, explain:

- why `self.` is needed in front of the word `salary` in the calculation
- what the round `()` function does
- why there is a 2 inside the round `()` function
- why this needs `str()` in front of it.

8 Create a third employee.

9 Code and test an additional class method and name it `get_details()`, which prints the following details for any employee. To run this method for 'walt', type (noting the lowercase!):

```
walt.get_details()
```

- Name + surname
- Annual salary is $ …
- Fortnightly pay is $ ...
- Number of employees is …

ACTIVITIES

Identify

1 Identify three approaches to programming.

2 To what does the word 'object' refer in the phrase 'object-oriented programming'?

3 What is a class in OOP?

Analyse

4 Outline three advantages of using an object-oriented approach over a procedural approach to programming.

5 Explain the difference between a class and an instance (or object) in OOP.

6 Explain the difference between class variables and instance variables.

Extend

7 Invent your own everyday OOP example, similar to the car example in this chapter, to illustrate the concept of classes and objects. Write some simple code with a class definition, create instances and run methods.

15.10 Parent classes, child classes and inheritance

Earlier in the chapter, we introduced the OOP concept of *encapsulation* in which classes are used to group an object's data (attributes) and functions (methods). As well as encapsulation, another important concept in OOP is *inheritance*. This refers to new classes being able to be created by using previous ones. These are known as *children* of the *parent* class and they inherit all their parent's features. Of course, just as in real families, they can still have unique features!

SKILL BUILDER

Parent and child classes: an improved car factory

We now rewrite our code by creating child classes of the `Car` class. Each car model can now benefit from this principle of inheritance. We could create child classes with different numbers of doors. The number of doors would now be 'hard coded' into each child class's definition. By using child classes, we no longer need to state the number of doors each time.

In our earlier example, `number_doors` was written as a *class variable* of the `Car` class (as in Figure 15.19). Now we will write it as an *instance variable* of child classes. We have repeated the 'moving' method in the child classes for a unique engine sound for each one. This demonstrates how a child class method with the same name as its parent's class method overrides it.

1 Start with this code:

```
# Parent class
    class Car:
        def __init__(self, colour):
            self.colour = colour
        def moving(self):
            print('Varoom!')
# Child classes
    class Hatch(Car):
        number_doors = 5
        def moving(self):
            print('Hmmmmm!')

    class Ute(Car):
        number_doors = 3
        def moving(self):
            print('Grrrrrr!')
```

Each hatchback will now have five doors and each ute will have three doors.

2 Create instances of these two new child classes. You must supply a value for colour as this is an instance variable required by their parent class (Figure 15.20).

```
myhatch = Hatch('red')
myute = Ute('blue')
```

3 Predict the output from the following code then run the code to test your answers.

```
print('My hatchback is', myhatch.colour)
print(str(myhatch.number_doors))
myhatch.moving()

print()
print('My ute is', myute.colour)
print(str(myute.number_doors))
myute.moving()
```

Were you correct?

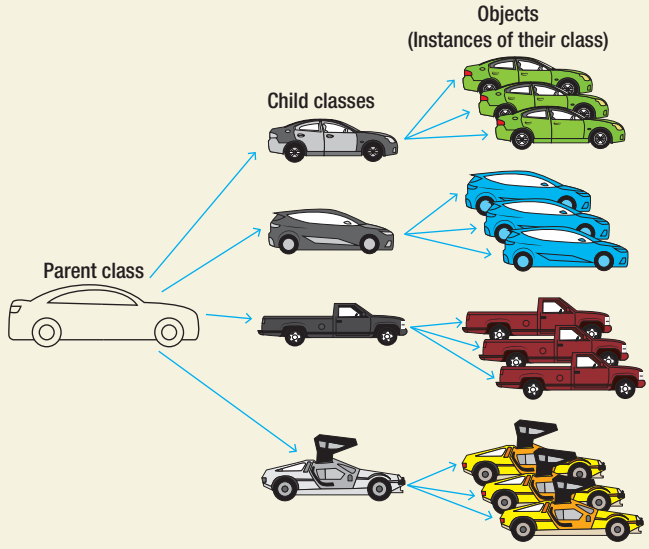


Figure 15.20 Using inheritance our car analogy uses parent and child classes. The only actual cars are the instances of the child classes, the others are just their class definitions.

4 Create two new child classes to create an instance of a two-door sports car or a truck, then run your own test code to output their names, door count and engine sounds.

Using inheritance, we could have made an even more impressive factory by creating a super class 'Vehicles', along with many other child classes apart from cars, such as motorbikes, bicycles, buses, motorised scooters, drones, hoverboards and more!

Skill builder resource
Parent and child classes

15.11 Three key OOP concepts

OOP shares many features in common with other styles of programming you know already:

- *variables* to store data
- *data types* such as integers, characters, lists and strings
- *procedures* such as functions
- *control structures* such as sequence, selection and repetition
- and *modules*.

OOP has three unique features, however: *encapsulation*, *inheritance* and *polymorphism*. You have encountered the first two already. We will now summarise these important concepts.

Encapsulation

Encapsulation is a central OOP idea (think: 'sealing up something in a capsule'). It means methods and properties belonging to a class are contained inside it (Figure 15.21), not scattered throughout a program.

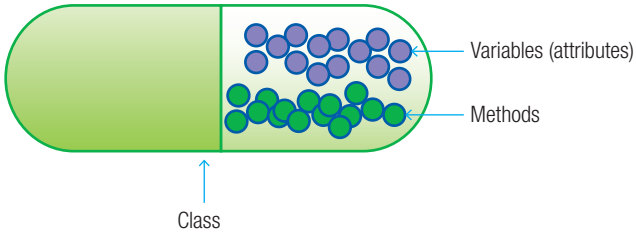


Figure 15.21 Encapsulation means methods and properties belonging to a class are contained inside it and not scattered throughout code, just as ingredients of a medical capsule are contained within it.

This is why the word 'object' appears in the name OOP! The detail of the way methods work are hidden in the class definitions. This is different to function-based procedural coding, where a programmer sees all the functions used spread throughout their code.

Systems thinking

We described an example of encapsulation previously as: a Circle class that uses position and radius of a circle in a method for calculating area. The details of the area formula are hidden in its class definition and a person needing a circle's area would not see how the method calculated it, just its area.

Inheritance

Examine Figure 15.22. In OOP we can define new classes based on previous ones. Child classes inherit the methods and properties of their parent class. An example would be a *Rectangle* class with a child class of *Square*. A *Square* will inherit all the definitions of a *Rectangle*. This might include side length and a method for finding area.

There might also be an even more general *Shape* class acting as a parent for all the other child shapes. These might each inherit attributes from the *Shape* class (such as the position where they are to be drawn on a computer screen).

Polymorphism

Polymorphism (from the Greek meaning *many forms*) means methods are inherited but can also be redefined by the child classes using them. An example would be a *Shape* parent class with an area method already defined. Child classes such as *Rectangle*, *Rhombus* and *Circle* could be given their own formulas for calculating area.

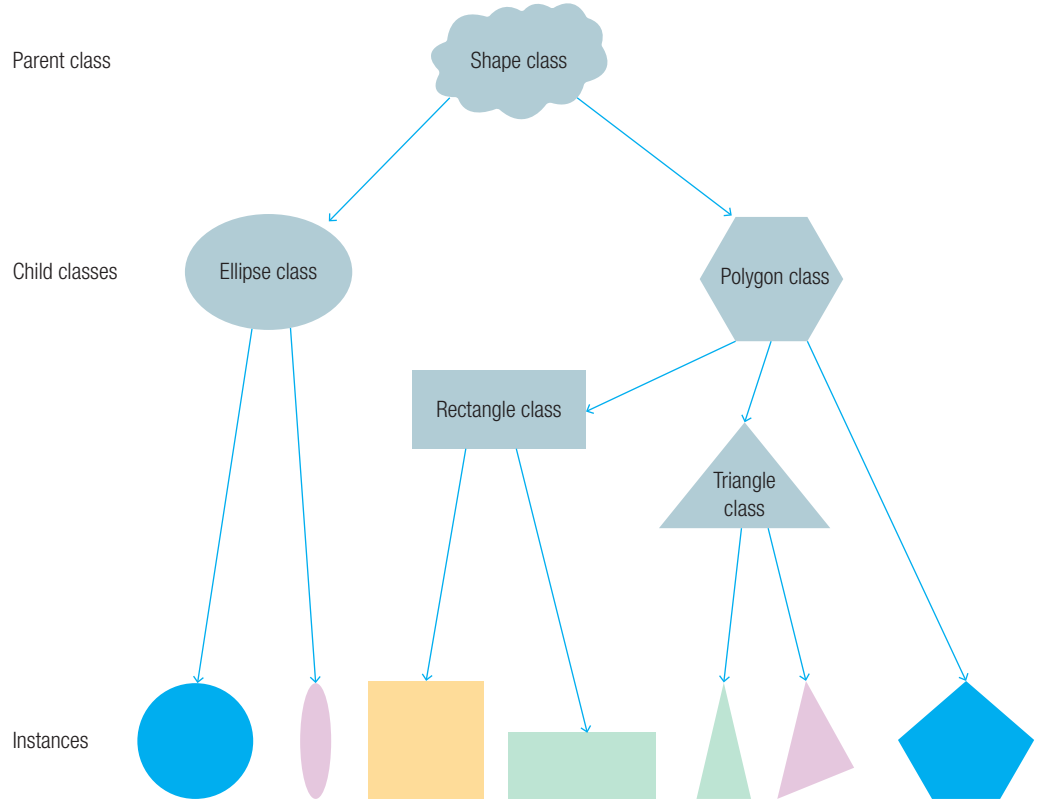


Figure 15.22 This shape parent class has been used to create child classes for other classes; e.g. the polygon class has been further used to create child classes for rectangle and triangle classes.

ACTIVITIES

Identify

1 Explain the term encapsulation in OOP.

2 Explain the term inheritance in OOP.

3 Explain the term polymorphism in OOP.

Analyse

4 Refer to the shape OOP example illustrated in Figure 15.22 and propose attributes and methods for each of the circle and the hexagon.

5 Identify the features of procedural programming used by the OOP approach and outline how each feature might be used.

Extend

6 Research the syntax for defining classes in both C++ and Java and compare these to Python syntax.

Circle		Hexagon	
Attributes	**Methods**	**Attributes**	**Methods**

SKILL BUILDER

Skill builder resource
Developing code using procedural and OOP approaches

Developing code using procedural and OOP approaches

The project at the end of this chapter involves creating an OOP-based simulation comparing various dice games using different rules of play. In this Skill builder you first use a procedural approach, then a functional one and finally an OOP approach using classes to code a dice game. The skills you develop in this Skill builder will come in handy for the project.

Dice game: a procedural approach

First code and run this *procedural* version of the dice game.

Most people learn to code using a procedural approach. So far you all the coding you have written has been done this way. Here is an example of a procedural program for a two-player dice game written using Python.

1 First explain what this program does.

```
import random
player1_dice = []
player2_dice = []
for i in range(3):
    player1_dice.append(random.randint(1, 6))
    player2_dice.append(random.randint(1, 6))
print("Player 1 rolled", str(player1_dice), "giving a total of", sum(player1_dice))
print("Player 2 rolled", str(player2_dice), "giving a total of", sum(player2_dice))
if sum(player1_dice) == sum(player2_dice):
    print("Draw")
elif sum(player1_dice) > sum(player2_dice):
    print("Player 1 wins")
else:
    print("Player 2 wins")
```

2 Type up this code using your Python IDE and run it.

Weblinks
Python

Replit

The output should be similar to:

```
Player 1 rolled [2, 1, 4] giving a total of 7
Player 2 rolled [5, 1, 6] giving a total of 12
Player 2 wins
```

3 Why do we need the double '= ='?

4 Modify the code to allow the game to be repeated without restarting it.

5 Modify the code to allow the game to be repeated exactly 100 times.

Dice game: a functional approach

You have learned already that we use functions to make program code more modular and readable.

6 Following is the same program organised using functions. Although it is longer, it is now much easier to identify the various parts of the overall algorithm. It is still, however, a procedural approach. Identify what each function does, then type up and run the code.

```
import random
# functions
def roll():
    dice_roll = []
    for i in range(3):
        dice_roll.append(random.randint(1, 6))
    return dice_roll
def tot(player):
    total = sum(player)
    return total
def printout(name, player):
    print(name, "rolled", player, "giving a total of", str(tot(player)))
def winner(a, b):
    if tot(a) == tot(b):
        print("Draw")
    elif tot(a) > tot(b):
        print("Player 1 wins")
    else:
        print("Player 2 wins")
#main program
player1 = roll()
player2= roll()
printout("Player 1", player1)
printout("Player 2", player2)
winner(player1, player2)
```

7 Modify the code to allow the game to be repeated without restarting it.

8 Modify the code to allow the game to be repeated 100 times.

Dice game: an OOP approach

It is time now to write an OOP version of the same program.

Weblink
Replit

9 We will use this in the project where you will see how helpful OOP is in longer programs. Examine the code carefully, compare it to the previous function version, then type it up and run it. Compare the variation between output print statements.

```
from random import randint

class DiceToss:
    def __init__(self, name):
        self.name = name

    def set_roll(self):
        self.roll = [] # clear last tosses
        for i in range(3):
            self.roll.append(randint(1,6))
        return self.roll
    def sum_roll(self):
        return sum(self.roll)
# main program #
def winner(a, b):
    if a.sum_roll() == b.sum_roll():
        print("Draw!")
    elif a.sum_roll() > b.sum_roll():
        print(a.name, "wins!")
    else:
        print(b.name, "wins!")

# create two player instances of the DiceToss class
player1 = DiceToss("player1")
player2 = DiceToss("player2")

# roll dice and print results
print("player 1 rolled", str(player1.set_roll()), "giving total of", player1.sum_roll())
print("player 2 rolled" + str(player2.set_roll()), "giving total of", player2.sum_roll())
# print winner
winner(player1, player2)
```

This OOP version was the longest but don't let that discourage you! In the following project you will need many dice players and so having a class to create them instantly will make this approach the fastest! Modify your OOP version to allow this game to be replayed.

Weblinks:
- Replit
- Janken (rock-paper-scissors) Robot with 100% winning rate
- Python

Skill builder resource:
- Writing and using functions
- Using functions
- Using files
- Debugging logic errors
- Creating a car
- A simple OOP example: Using instance variables
- A simple OOP example: Company employees
- Parent and child classes
- Developing code using procedural and OOP approaches

Knowledge builder icon:
- Fixing syntax errors

Project resource:
- A menu-driven game collection
- Challenge: Simulating 100 000 tosses!

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

15.12 Project 2: OOP-based dice game simulation

Identifying and defining

Can you win using this powerful dice game simulator?

In this project, you will use OOP by defining classes, creating modules and running a dice game simulation. You will explore various games where players sum dice to see who has the highest total then code the game to run it on a computer. You will then create a simulation for playing hundreds of thousands of games in only a few seconds and create new games with built-in cheating! You will discover by the end how efficient OOP can be.

By the end of this project, you will have a good working knowledge of OOP and understand its advantages over other approaches.

You will start with a basic game throwing four dice, but using three versions:

- players remove their lowest die and toss again
- players remove their lowest die and toss until it is higher
- players toss again if total is less than 10.

Can you guess which version produces a winning strategy?You will compare results by giving players the different rules then simulate them competing against each other. After your code is working you tackle some challenges – and you may think up others. Having real-life dice is not essential but may come in handy!

Here's the plan of attack:

- First, you will create a procedural solution.
- Then, you will create classes, methods and attributes and build an OOP solution.
- Finally, you will save your class definitions in a separate file and import this as a single module (this is very simple!). Modularisation is an important idea in programming as it allows code to be reused as well as organising it logically.

Our players (Figure 15.23) will be Olga, Ben, Jin, Elijah, Charlotte and Toby. (Combine their initials!)

Shutterstock.com/Denis Cristo

Figure 15.23 A parent DiceToss class is used to create child classes to represent different dice game rules. Dice players Olga, Ben, Jin, Elijah, Charlotte and Toby become instances created using child classes.

Coding dice using a procedural approach

Decide the following preliminary design issues before starting coding:

- What coding tool (IDE) will you use? We suggest replit here (see weblink).
- What programming function could simulate the toss of a die?
- How many dice does each person toss (see above)?
- How will you store the players' dice tosses?
- How will you total the dice?
- How will you decide which player wins (note: it could be a tie)?

Start by examining this procedural version of the code, which we developed in the earlier Skill builder in this chapter, but now using player names (Figure 15.24).

Computational thinking

Weblink
Replit

```
from random import randint

olga = []
ben = []

for i in range(4):
    olga.append(randint(1,6))
    ben.append(randint(1,6))
print("Olga rolled",str(olga))
print("Ben rolled",str(ben))

if sum(olga) == sum(ben):
  print("Draw!")
elif sum(olga) > sum(ben):
  print("Olga wins!")
else:
  print("Ben wins!")
```

Import randint from a module called random

Create a list data structure to store dice numbers

Dice are thrown by looping to create four random numbers 1 to 6 and appending these to the lists

To decide who wins, we first check for a tie; if not a tie, we check if Olga's sum is greater than Ben's

Figure 15.24 A procedural approach to dice tossing.

Explaining the code:

- From a module called `random` we imported the `randint` function. We use this to generate random integers representing numbers on tossed dice.
- We choose a list data structure to store the dice numbers for each player.
- Four random dice numbers are created and appended (added) to each list. These are summed using `sum(list_name)`.
- To decide who wins, we check for a tie and if not, we check if Olga's sum is greater than Ben's. Why do we not also check if it is less?

1 Type up the code and get it running. Now modify your code to print Olga's and Ben's totals.

2 Test out changing the number of dice thrown to 10 and rerun the code.

A solution is provided on Nelson MindTap: *dicegame_procedural.py*.

Researching and planning

Using OOP by creating a dice toss class

We need to invent games that use different dice rules. If we used the procedural approach above, we would end up with very complex code with lots of variables and would still have to simulate at least 100 000 tosses for each of the dice rules! So, now it's time for an OOP approach!

You have already studied OOP and classes, methods and attributes. Refer to the earlier summary on how to define a class in Python.

Here's what we do:

- Start the same way as for the procedural example by importing `randint` (see following).
- Define a class describing what happens when multiple dice are tossed. The class acts like a rubber stamp or blueprint and is used to build the dice game variations.

We will name the class `DiceToss`. Type the following into your chosen Python IDE:

```
from random import randint
class DiceToss:
```

Notice the word `class` is not capitalised, but the class name `DiceToss` uses caps (here two capital letters in the one word – like a camel hump). This is standard practice when naming classes. Notice there are no parentheses in this class because no parameters are passed. Later on, we will need to pass parameters.

3 In writing the definition for this `DiceToss` class, we need a data structure to hold dice tosses. We could choose separate integer variables, a string of characters, a `list` or a `dictionary`. The most suitable type however is Python's `list` data structure, which we also used in the last section for our procedural approach.

You might think we would write:

```
class DiceToss:
    roll = []
```

You learned earlier in the chapter that using a class attribute like this will create a problem, as classes act like copiers and every instance produced would share the same dice list. Not very suitable! To avoid this, we place `roll` inside that strange looking `__init__(self)` function explained earlier in this chapter. This creates an instance attribute instead. Now every instance can produce a different set of numbers for the roll parameter:

```
from random import randint

class DiceToss:
    def __init__(self):
        self.roll = []
```

A reminder of the purpose of that quirky `__init__` function may be helpful. This special function is a class's initialiser method and runs each time `DiceToss` is used, to create a new object. Here it creates a new dice toss list for each instance it creates.

We also need the word `self` with dot notation, `self.roll`, when listing the instance variables. If we just wrote `roll = []` then the roll list would remain empty because it would never receive instance values. It would turn into a class variable. The word '`self`' creates a new list variable for each instance.

On a roll

We have created our `init` method along with a list to hold dice, now we need to roll them!

4 Add a roll method and call it `set_roll()`. Notice we use Python's `randint` function inside a counted loop to generate four dice numbers. Notice also the `self` parameter added to every method needed by instances.

```
from random import randint

class DiceToss:
    def __init__(self):
        self.roll =[]

    def set_roll(self):
        self.roll = [] # clears for repeats
        for i in range(4):
            self.roll.append(randint(1,6))
```

DiceToss class

5 Separating methods that *get* data from those that deliver or *set* data is considered good practice in OOP. So here we do the same! All code in Figure 15.25 forms the DiceToss class and fully describes the tossing of four dice.

Writing such a class is an example of encapsulation.

```
from random import randint                    Import randint from random module
                                              to generate dice toss numbers

class DiceToss:                               Define a DiceToss class
  def __init__(self):                         __init__ method creates
    self.roll = []                            a new list for each instance

  def set_roll(self):
    self.roll = []                            set_roll method adds
    for i in range(4):                        four dice tosses to empty list
     self.roll.append(randint(1,6))

  def get_roll(self):                         Getter method returns output of
    return self.roll                          set_roll method
```

Figure 15.25 All code in the indented class block forms the DiceToss class and contains attributes and methods that fully describe the tossing of four dice.

Producing and implementing

Creating instances

6 Creating *instances* of the DiceToss class, is like giving four dice to different players and getting them to roll them. Type the following code to create two players as instances of the dice game and roll their dice. This code forms the start of your main program code following the earlier class definition.

```
olga = DiceToss()
ben = DiceToss()

olga.set_roll()
ben.set_roll()
```

Completing the basic DiceToss program

When you run this code, you won't yet see results as we have not written any instructions to print. We will fix this now. The remainder of the code forms the main program and is the same as the earlier procedural one, except now we call a `get_roll` method for our four dice tosses.

This is the completed code and the hard work is over!

7 Type up the Figure 15.26 code and get it working.

```
from random import randint

class DiceToss:
  def __init__(self):
    self.roll = []

  def set_roll(self):
    self.roll = []
    for i in range(4):
      self.roll.append(randint(1,6))

  def get_roll(self):
    return self.roll

olga = DiceToss()
ben = DiceToss()

olga.set_roll()
ben.set_roll()

print("Ben rolled" + str(ben.get_roll()))
print("Olga rolled" + str(olga.get_roll()))

if sum(olga.get_roll()) == sum(ben.get_roll()):
  print("Draw!")
elif sum(olga.get_roll()) > sum(ben.get_roll()):
  print("Olga wins!")
else:
  print("Ben wins!")
```

Class definition

Print the output of set_roll method

Main program

Sum integers in roll list, check for a tie, else determine winner

Shutterstock.com/Denis Cristo

Figure 15.26 The completed code for our basic DiceToss class.

Examine each line of the code carefully and understand how it works.

8 Did your code run correctly? Try changing the number of dice to 10 and re-run it!

A completed solution is provided on Nelson MindTap: *dicegame_oop.py*.

Although we were simulating four dice, the game is not very exciting as players just toss the dice without variation. Now we will add more interest!

Dice tossing gets interesting using inheritance

Wikipedia lists over 30 dice games. Look them up! You will now discover the power of OOP!

Our OOP code can test different types of dice games easily and run them hundreds of thousands of times to see which ones win. You could even simulate someone cheating using a loaded die, or replacing a dice with a 6 on each side (Figure 15.27).

You will use the OOP principle of inheritance to create child classes of the DiceToss class. This are just like a real child who learns from their parents, but wants to try out new things. Your child classes will already know how to toss dice (from their parent) but will make up their own dice tossing rules (just like a child would)!

Shutterstock.com/Doug Stevens

Figure 15.27 By defining a child class with a new method we can simulate a biased game where a die has six on every side!

Creating a first child class: ReToss

9 Use `class` as the keyword and name a child class `ReToss` with the DiceToss parent class by inserting this parent class name inside its parentheses:

```
class ReToss(DiceToss):
```

The child class `ReToss` you have now created inherits the three parent class methods: list initialisation, `set_toss()` and `get_toss()`.

We will give it an additional dice game method too: `lose_lowest()`. This removes the lowest dice and replaces it by tossing it again! Many popular dice games are based on this idea.

Check your code against the following:

```
from random import randint

class DiceToss:
    def __init__(self):
        self.roll = []

    def set_roll(self):
        self.roll = [] # clears dice
        for i in range(4):
            self.roll.append(randint(1, 6))

    def get_roll(self):
        return self.roll

class ReToss(DiceToss):
    def lose_lowest(self):
        self.roll.remove(min(self.roll))
        self.roll.append(randint(1, 6))
```

Test this for five tosses using a player called Jin:

```
jin = ReToss()
for n in range(5):
        jin.set_roll()
        print("jin dice before re-place " + str(jin.get_roll()))
        jin.lose_lowest()
        print("jin dice after re-place " + str(jin.get_roll()))
```

Creating a second child class: RePlace()

10 The lose_lowest() method will not always improve the total, and may even produce a lower one! Can you explain why? We will create another child class `RePlace()` to address this issue. Here we want to discard the lowest number but keep tossing until we get a higher number.

Our first attempt is this:

```
class RePlace(DiceToss):
    def replace_lowest(self):
        self.roll.remove(min(self.roll))
        while randint(1, 6) <= min(self.roll):
          randint(1, 6)
        self.roll.append(randint(1, 6))
```

Can you identify a logic error here (there are actually two!)?

Test this for five tosses using a player called Jin:

```
jin = RePlace()
for n in range(5):
      jin.set_roll()
      print("jin dice before re-place " + str(jin.get_roll()))
      jin.replace_lowest()
      print("jin dice after re-place " + str(jin.get_roll()))”
```

Hint: Does `randint(1,6)` stay constant?

Try to correct this logic error before proceeding.

Our second attempt is this:

```
class RePlace(DiceToss):
    def replace_lowest(self):
        old_min = min(self.roll)
        self.roll.remove(min(self.roll))
        toss = randint(1, 6)
        while toss <= old_min:
          toss = randint(1, 6)
        self.roll.append(toss)
```

Can you identify a second logic error here that will lead to an infinite loop?

Hint: What rare four dice toss event known as 'boxcars' would cause this algorithm to fail? Why does this cause the algorithm to endlessly loop at the `while` statement?

Debugging output statements

One method of debugging programs is to insert a print statement that regularly outputs the values of variables at that moment in the code.

11 First add this output print statement inside your while statement, after the line:

```
toss = randint(1,6)
print(toss, old_min)
```

Set up a new instance of this child class, a player called *test*, who tosses 1000 times using the RePlace() class and see if this reveals the problem. Add this code and run your program.

```
jin = RePlace()
for n in range(10000):
    jin.set_roll()
    print("jin dice before re-place " + str(jin.get_roll()))
    jin.replace_lowest()
    print("jin dice after re-place " + str(jin.get_roll()))
```

What rare toss makes the printout 'hang'? Can you explain why? Try to correct this logic error before you proceed.

Putting it together

Your code now contains three classes describing three different games:

- `DiceToss()` is the parent class: a regular game tossing four dice
- `ReToss()` is the first child class: lowest die removed and re-tossed
- `RePlace()` is the second child class: lowest die removed and re-tossed repeatedly until higher

12 We have corrected the logic error below. Identify and describe the issues this corrects.

```
class DiceToss:
    def __init__(self):
        self.roll = []

    def set_roll(self):
        self.roll = [] # clears current dice
        for i in range(4):
         self.roll.append(randint(1, 6))

    def get_roll(self):
        return self.roll

class ReToss(DiceToss):
    def retoss_lowest(self):
        self.roll.remove(min(self.roll))
        self.roll.append(randint(1, 6))

class RePlace(DiceToss):
    def replace_lowest(self):
        old_min = min(self.roll)
        if old_min != 6:
         self.roll.remove(min(self.roll))
         toss = randint(1, 6)
         while toss <= old_min:
            toss = randint(1, 6)
         self.roll.append(toss)
```

Adding code for the main program

First new instance: Ben

13 You have now defined your classes and can write the main program. These are the steps:

- Create a new instance, a player named Ben, using the `ReToss` child class.
- Call the `replace_lowest()` method for Ben's toss method.
- Print what Ben's dice toss looks like both before and after the `retoss_lowest()` method is applied.

Here is the code:

```
ben = ReToss()
ben.set_roll()
print("Ben dice before re-toss" + str(ben.get_roll()))
ben.retoss_lowest()
print("Ben dice after re-toss" + str(ben.get_roll()))
```

14 Repeatedly run this code. Observe the change each time in his dice numbers. Can you get this code to repeat 10 times? (Note: Do not repeat the creation of the class each time!)

Second new instance: Jin

15 Now create a second player instance, Jin, this time using the `RePlace` child class. The steps are the same as for Ben.

16 Repeatedly run the program. Both player's results will be printed. Observe the changes each time in the dice numbers for each player. Describe what happens as a result.

A completed version is available on Nelson MindTap called *dicegame_oop_fungames.py*.

Using modules

It is helpful to use modules when coding large programs. We will set up our program using a module containing our class definitions (you have already had experience importing modules such as Pythons' `random` module). It is extremely easy to do this in Python.

Create the external module

17 Separate your class definitions into an external file (module) this way:

- Create a folder called Dice_games to hold everything.
- Save the code in step 12 with its three class definitions, `DiceToss()`, `ReToss()` and `RePlace()` inside this folder and name it *dice_module.py*.

Create the main program code

18 Now create a main Python file and import this new module. Then add code to create instances for each player's class – the different game strategies played by each of Ben, Jin and a third player, Olga, who plays an ordinary game, and finally a few lines to sum their dice and see who wins each time!

Add this module import instruction as the first line of your main code:

```
from dice_module import *
```

The `*` means everything should be imported.

19 Write the main program code shown here. Olga uses the regular rules, and Ben and Jin use their new rules.

```
from dice_module import*

# creates instances of a regular game and two new games!
olga = DiceToss()
ben = ReToss()
jin = RePlace()

# each player rolls their dice
olga.set_roll()
ben.set_roll()
jin.set_roll()

# the two new dice games!
ben.retoss_lowest()
jin.replace_lowest()

# which new game approach wins?
if sum(ben.get_roll()) == sum(jin.get_roll()):
   print("Draw!")
elif sum(ben.get_roll()) > sum(jin.get_roll()):
   print("Ben wins!")
else:
   print("Jin wins!")
```

Save this main program inside the same folder as your module. Call it any name you like. We used *main_dice.py*.

20 Try running the modular version of your project by running this main code. It will import your outside module automatically. The code in the module acts just as if it was already part of the main program!

The online resources on Nelson MindTap have a working version in a folder called *Dice_OOP_modular* if you get stuck. Run the file *main_dice.py*.

Testing and evaluating

Optional extra challenges

- Improve output by adding code to print dice rolls for each player before and after their special moves, as we did in step 13.
- Select either Ben or Jin and use real dice to play against them!
- Add code to ask if a replay is desired.
- Add code to print a special message whenever a toss is 'boxcars' (all dice showing 6 as in Figure 15.28) or 'snake eyes' (all dice showing 1).

Project resource
Challenge: simulating 100 000 tosses!

Challenge: simulating 100 000 tosses!

You are now ready for the big time! So far you need to restart the program every time you want to simulate a throw. Wouldn't it be great not to have to do this and simulate 100 or even 100 000 tosses in the one run?!

Using our OOP modular approach means it is easy to do this. We don't even need to touch the *dice_module.py* file in the folder.

Remembering Ben and Jin's different rules, answer *before* you continue:

- Which player, Ben or Jin, do you expect will win the most games (get the highest total on the dice)?

Try modifying the code to ask the player how many times they want it to run. If you face a roadblock, type up the following code (you have most of this already), save it in the same folder as your module and run it. This version adds code allowing the player to select the number of dice toss simulations and variables to hold player scores.

Spoiler! If you get really stuck, we have supplied the complete code on Nelson MindTap: *main_dice_bigsimulation1.py*.

Dreamstime.com/Vrabelpeter1

Figure 15.28 What are the chances?! One of the outcomes you will be able to test using your OOP project: how many times can you expect all six dice to come up 6 in 1000 tosses?

```
from dice_module import*

# set up rules for players
ben = ReToss()
jin = RePlace()

# initialize player scores
ben_score = 0
jin_score = 0

# initialise game count and number of games
number_of_games = int(input("How many repeats of the dice simulation?"))
games_played = 0
print()
print("Simulation running")
print("==================")

# loops desired number of times through the gameplay
while games_played < number_of_games:
# players given dice
    ben.set_roll()
    jin.set_roll()
# game rules applied and results handled
    ben.retoss_lowest()
    jin.replace_lowest()
    if sum(ben.get_roll()) == sum(jin.get_ roll()):
      pass
    elif sum(ben.get_roll()) > sum(jin. get_roll()):
      ben_score += 1
    else:
      jin_score += 1
    games_played += 1

# final totals
print("Simulation complete")
print("-------------------")
print("Final scores")
print("------------")
print("Ben won:" + str(ben_score))
print("Jin won:" + str(jin_score))
if ben_score == jin_score:
    print("Game was drawn")
elif ben_score > jin_score:
    print("Ben won most games")
else:
    print("Jin won most games")
```

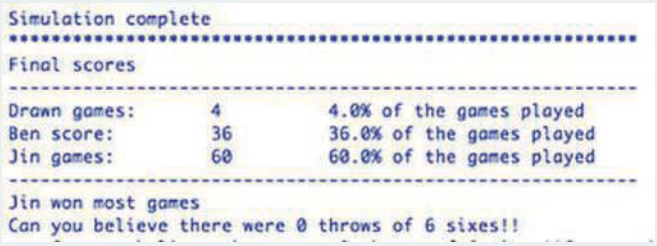
- Did you guess who won the most games?
- Can you explain why?

The output so far just prints the overall number of games won. It would be more interesting to see the result of each game.

Add the following print statements to this main program. Initialise a `drawn_score` variable to 0 beneath the section of the code named 'initialise player score'. Select 10 for the number of simulations and run the code.

```
if sum(ben.get_roll()) == sum(jin. get_roll()):
   print("Draw!")
   drawn_score += 1
   pass
elif sum(ben.get_roll())> sum(jin. get_roll()):
   print("Ben wins!")
   ben_score += 1
else:
   print("Jin wins!")
   jin_score += 1
game_number += 1
```

- What effect do they have on the run time of your code? Why?
- Comment out the last three print statements. Run the simulation to play 100 000 times. Note: the simulation takes approximately one hour to run with the print statements and ten seconds without them so you should definitely comment them out!
- Why do Ben's and Jin's wins not add up to 100 000? Do you think they could ever do so?
- Time to improve the user interface! Improve the user experience by redesigning your screen output so it is easier to read. Modify output to include percentages of games won.
- Challenge! Notice the final line stating the number of times a player threw all sixes! Can you add code to achieve this?


Design thinking

```
Simulation complete
**********************************************************
Final scores
----------------------------------------------------------
Drawn games:        4          4.0% of the games played
Ben score:          36         36.0% of the games played
Jin games:          60         60.0% of the games played
----------------------------------------------------------
Jin won most games
Can you believe there were 0 throws of 6 sixes!!
```

If you get stuck, we have supplied the complete code for this fancier version on Nelson MindTap in the folder *Dice_OOP_modular_bigsimulation* (Run *main_dice_simulation2.py*).

Extending this project

Six friends get together, invent a four-toss dice rule for themselves that all agree on (see Figure 15.29). Your task is to write child class definitions for this new scenario. Pit players against each other and record the results over 100 simulations. A solution is provided for teachers only! Before you start, guess which strategy listed will win the most games over 1000 simulations.

Here is an outline of the stages you will need to complete:

- Write child classes of the DiceToss class, each with a new method that will simulate the players shown in Figure 15.29, adding them to the module *dice_module.py*. Hint: to check if a number is odd, Python's `%` operator gives the remainder after repeated division. Thus, number `% 2` will always equal 0 if number is even.

Shutterstock.com/Denis Cristo

Player	Olga	Ben	Jin	Elijah	Charlotte	Toby
Game class name	DiceToss	ReTossOdd	TooLow	DiscardTwo	AllEven	Biased
Game description	Normal play	If dice total is odd, retoss all dice until total is greater	If total is less than 20, all dice are retossed once only	Two lowest dice are retossed once	Retoss only odd dice until all dice are even	Make two of the dice biased

Figure 15.29 By creating a parent DiceToss class, child classes can be created to represent different dice game rules. Each game involves tossing four dice.

- Create an instance of each new class using the player names in Figure 15.29. Here is a child class that reproduces Jin's "Too low" dice game rule. We included two debugging output print statements. This code is added to the module *dice_module.py* and an instance of it called jin is created in the main program:

```
class TooLow(DiceToss):
    def tool_low(self):
        print("Toolow before", self.roll)
        save_total = sum(self.roll)
        if save_total < 20:
         self.set_roll()
        print("Toolow after", self.roll)
```

You need to call the parent class's `set_roll` method for Jin to roll her dice. Then call the child class's new method `too_low` to apply the new rule:

```
jin.set_roll()
jin.too_low()
```

- Complete the six child classes including a new method for each player.
- Thoroughly check your final code using debugging output print statements to help identify errors.
- Write code to print out the winner of a single simulation.
- Modify your code to allow multiple simulations.
- Design a user interface to print percentages of games won by each player over repeated simulations.
- Would you expect results for 10 trials to be similar to 100? Perform these simulations to confirm or disprove your answer.
- Match two players against each other 100 000 times and note which strategy wins. Note: remove repeated print statements as they slow the program, retaining only those needed to print final results.
- Invent and code a dice strategy of your own.
- Finally, answer the earlier question: how many throws of six sixes would you expect in every 1000 tosses? Code this to check your answer.

16 Developing apps for the web

FOCUS AREA: Developing apps and web software

Chapter outcomes

In this chapter, you will learn:

- what HTML is
- the history of HTML
- how to use metatags
- to build a webpage using HTML
- what CSS is
- the advantages of using CSS
- to build a webpage using CSS
- what JavaScript is
- the uses of JavaScript
- to add interactivity to a webpage using JavaScript
- to build a webpage using JavaScript
- to use the canvas element to create animations
- to build a web-based quiz using HTML, CSS and JavaScript
- apply UI principles in designing an interactive website
- to design and build a web-based snake game using JavaScript.

Key terms

canvas
CSS
DOM
HTML
IDE
JavaScript
metatags

9780170442787

In the last chapter and in Chapter 4, you built desktop apps using both procedural and object-oriented programming approaches. In this chapter you create interactive apps for the web. In this chapter's Guided project you create an interactive web quiz that runs in any desktop web browser or mobile device. In the project you will build a web-based retro snake game.

We begin by introducing HTML, CSS and JavaScript, including animation using the HTML canvas element.

16.1 Apps for the web: HTML, CSS and JavaScript work together

When a web developer (you!) builds a website they are writing code for a browser to manipulate words, images, audio, video and interactivity.

The developer's goal is to provide a functional website to users and deliver an appealing experience (UX). One of the strengths of the web is that three languages with three separate tasks all work together to achieve this. Figure 16.1 illustrates these roles.

- **HTML (HyperText Markup Language)** describes the basic structure of a site. In human terms it provides its skeleton.
- **CSS (Cascading Style Sheets)** describes the web page's appearance: its formatting, layout and presentation. In human terms it is like a 'makeover' – compare this to a person's clothing, hairstyle and appearance. Just by changing the CSS file a webpage can look completely different, yet have the same content. (See the earlier section on Zen Garden.)
- **JavaScript** is a full programming language. It is used to provide web pages with its behaviours such as its interactivity. It is responsible for special effects, password creation, interactive games and animations (such as the web-based snake game project at the end of this chapter). In human terms it is like the brains behind the web app – delivering movement, logic and responsive behaviour.

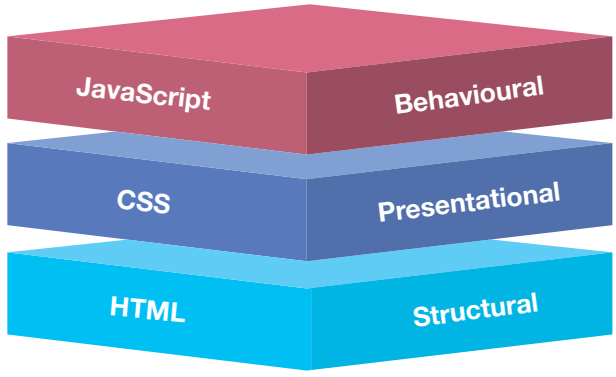


Figure 16.1 In interactive websites, three languages, HTML, CSS and JavaScript, perform separate roles.

16.2 What is HTML?

Suppose all the information stored on computers everywhere were linked. Suppose I could program my computer to create a space in which everything could be linked to everything. These were the earliest questions Tim Berners-Lee, creator of the World Wide Web, says he asked himself.

At first, all Internet communications were text only and the Internet just meant email, but with the introduction of graphical user interfaces (UIs) on home computers there was a reason to invent agreed rules (protocols) to allow images and sounds to be exchanged as well. It was during the late 1960s that Ted Nelson first thought of the idea of words with attached explanations. Later, Sir Tim Berners-Lee imagined that documents might be able to be linked electronically (hyperlinked) to help researchers.

Instead of an index that exists separately to the text, he imagined an index that was part of the text itself. Important words could act like buttons and when pressed would take the reader on a journey where they could read more. To do this he invented the HTML markup language. Hypertext markup language (HTML) code describes how text and graphics can be displayed and how the elements can carry these embedded hidden links.

A reader of Australian history could press the words 'Governor Macquarie', for example, and they would be taken to a page with more detail. They might become

interested in his influence on the architecture of early towns such as Windsor. They might then click on 'Windsor' and read about the architect John Watts. HTML usually presents these links as blue underlined words, but they may also be images or icons. When a mouse is clicked over these areas a request is launched for the linked item, which is then delivered for viewing. You will meet Sir Tim Berners-Lee a few times while you complete the activities in this chapter!

HTML is a markup language (*not* a programming language) because it consists of tags (not control structures) describing how a browser displays any text positioned between the tags.

Markup languages are easy for humans to understand, unlike some other computer codes because their tags are ordinary English words with obvious meanings. Before computers had graphic user interfaces with menus, even word processors used markup languages to indicate styles of text, such as bold or left justify instead of the menus and palettes you use now.

HTML has existed since the first day of the World Wide Web and the first browser.

HTML essentials

Paired HTML tags

HTML tags are usually paired, one at the beginning of the element to be defined, and another at the end. Both tags in the pair are wrapped in less than (<) and greater than (>) signs but end tags include a forward slash (/).

Here is an example:

```
<p>HTML code is very useful.</p>
```

This tag pair defines a paragraph.

HTML code can be used to make words bold, define a paragraph, create a button, provide a link to a URL or another file or fetch and deliver images for a web page.

They can even be nested inside another pair:

```
<p>HTML code is <strong>very </strong>
  useful.</p>
```

The browser will display:

HTML code is **very** useful.

We will now dissect the most important parts of this HTML code:

- The opening tag: Contains the element name, here , wrapped in opening and closing angle brackets. This states where that HTML element starts to take effect.
- The closing tag: This repeats the element name but is preceded by a forward slash . This states where the action ceases.
- The content: This is the content of the element on which the HTML tags act.
- Nested tags: We have shown a second HTML element nested inside our paragraph tags. It has its own opening tag and closing tag

Type the last HTML example into a text editor (such as *Notepad* on Windows or *TextEdit* on a Mac). You may need to set the preferences to *not* render text files as HTML. If you save it using a filename using the *.html* suffix, such as *index.html*, it will open in your default web browser, not the text editor and will appear as shown above. Original HTML code such as this is called *source code*.

You can inspect source code for any web page element by right-clicking on it in your browser and choosing 'Inspect element' or view the entire source code by selecting a 'View source code' menu item usually found under 'Developer' in your browser's menu. Load any web page and try this.

Common HTML tags

Here are some of the most common tag pairs in HTML with a description of what each one does:

Common HTML meta tags

<html> encloses the HTML document code </html>

<head> the metadata for the document </head>

<body> the body section of the document </body>

<title> the title of the document </title>

<h1> heading </h1>
<h2> sub-heading </h2>

 a paragraph

 defines a section of the document

 groups inline elements in part of the document

Common HTML attributes

Attributes of an HTML element are used by CSS or in a script (e.g. JavaScript) to identify elements or provide additional information.

id
Identifies individual elements; e.g.:
<h1 id="first_id">Quiz</h1>

class
Used to identify groups of elements. Elements can belong to more than one class; e.g.:
<h1 class="class_name">Quiz question</h1>
<h1 class="class_name">Quiz results</h1>
<h1 class="class_name" class="keep">Quiz results</h1>

href an attribute of <a>
Used with a hyperlink; e.g.:
Quiz
a defines a hyperlink
href specifies the hyperlink's URL

src an attribute of <img>
Provides the location of an image and (optional) alternative text in case the image can't be seen; e.g.:
<img src="image.jpg">
<img src="image.jpg" alt="This is an image">

style
Used to add styles such as color, font, size to elements; e.g.:
styles an element.

KNOWLEDGE BUILDER

Choosing a text editor

There are many code editors for creating websites using HTML, CSS and JavaScript. Although you can code using any regular text editor (as you did in the last section using *Notepad* or *TextEdit*), we recommend you use a web-based **IDE (Integrated Development Environments)** such as *replit*, *CodePen* or *jsFiddle*, which offer syntax highlighting, auto-completion of code, compiling and debugging.

In this chapter we will use *replit* as it not only provides an in-browser coding tool, it reports errors, allows image folders to be added and even hosts your project online! The *replit* interface can be seen in Figure 16.2.

CodePen calls its projects, 'pens'. A visit to *CodePen's* site is worthwhile to see some amazing examples. Look under 'Picked pens' and try some out. Find a simple 'pen' and try changing the CSS to see what happens.

Weblinks
Replit

CodePen

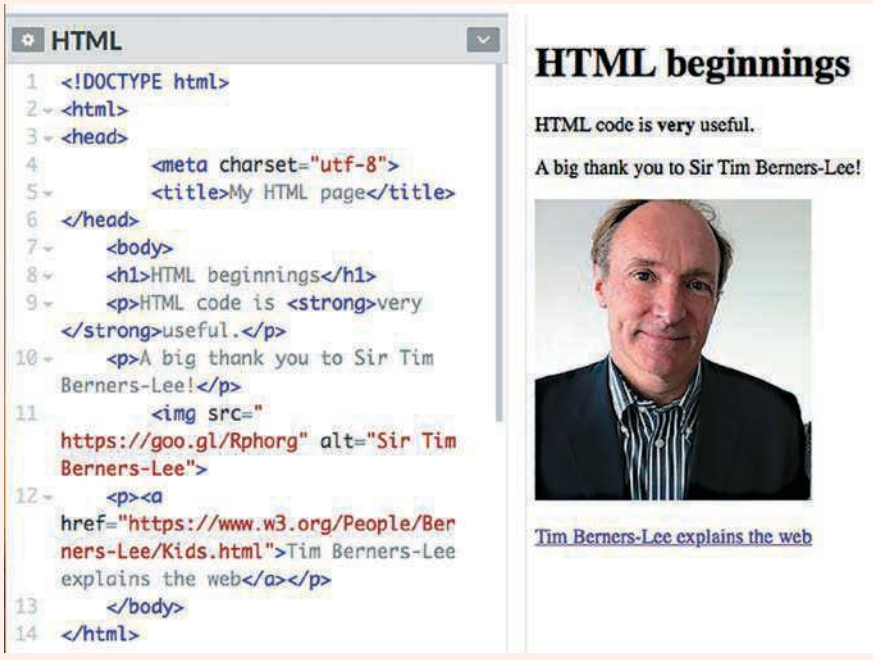


Photo: Flickr/cellanr/CC BY-SA 2.0

Figure 16.2 The *replit* web editor interface showing an HTML file, including a CSS file (see next section).

SKILL BUILDER

A first HTML page

HTML basics

A web page HTML has the following basic structure:

```
<html>
    <head>
        <title>webpage title</title>
    </head>
    <body>
        content of webpage ...
    </body>
</html>
```

head section

body section

the HTML document

Skill builder resource
A first HTML page

HTML pages begin with the word `doctype` followed by `head` and `body`. Notice the indenting, which is not essential, but makes it easier to see where tag pairs begin and end.

Web pages often include other formal elements to set the display size to match the type of device viewing the page. We omit these for now.

```
<!DOCTYPE html>
<html>
<head>
    <title>My HTML page</title>
</head>
<body>
    <h1>HTML beginnings</h1>
    <p>HTML code is <strong>very
      </strong>useful.</p>
    <p>A big thank you to Sir Tim
      Berners-Lee!</p>
</body>
</html>
```

1 Replace *replit's index.html* HTML file with the code above. Save it (as) then 'run' it to inspect the result in your browser. You should see a duplicate of Figure 16.3.

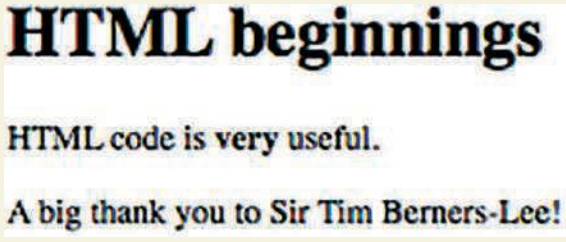


Figure 16.3 Basic HTML output.

Images

As well as pairs of tags, HTML uses single tags for some special purposes, where paired ones would not make sense.

One important tag is `img src` meaning 'image source'. It allows you to place images on your page from other web sites.

You provide the URL of the image inside quotation marks and provide alternative text to display in the browser if the image is not available.

Here is an example:

```
<img src = "https://tinyurl.com/
  tblthumbnail" alt = "Sir Tim
  Berners-Lee">
```

2 Add this code to your *index.html* file just above the closing tab of the body section. Open the *index.html* file to view the placed image. See Figure 16.4.

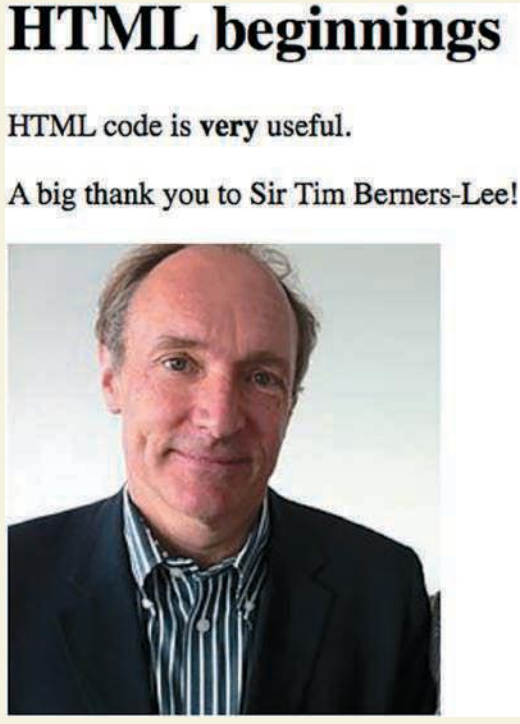


Figure 16.4 HTML beginnings including image.

Anchors

The final essential element of HTML is the one that gives the World Wide Web its power and was the central concept in its creation. It is called an *anchor* and is written as an `<a> </a>` tag pair.

This special pair of tags provides the ability to link images or words on one web page to other pages. These linking items are called 'hypermedia' and the linking text is referred to as 'hypertext'.

It was this concept that Sir Tim Berners-Lee, the inventor of the Web, saw as the main purpose of HTML code, to interconnect the world's knowledge by linking related items.

3 First wrap an `<a> </a>` tag pair around words you want to trigger the link. `<a>` stands for anchor then add `href =` followed by the URL (Uniform Resource Locator or web address) of the web page (href stands for *hypertext reference*).

Type this in the body section of your *index.html* file using your text editor and save it:

```
<a href="https://www.w3.org/People/
  Berners-Lee/Kids.html">Tim Berners-Lee
  answers questions about weaving the
  web</a>
```

This turns 'Tim Berners-Lee explains the web' into hypertext and links it to the URL address inside the quotation marks.

The HTML code in your text editor should now look like this:

```
<!DOCTYPE html>
<html>
<head>
    <title>My HTML page</title>
</head>
<body>
    <h1>HTML beginnings</h1>
    <p>HTML code is <strong>very
      </strong>useful.</p>
    <p>A big thank you to Sir Tim
      Berners-Lee!</p>
    <img src="https://tinyurl.com/
      tblthumbnail" alt="Sir Tim
      Berners-Lee">
    <p><a href="https://www.w3.org/
      People/Berners-Lee/Kids.html">Tim
      Berners-Lee answers questions
      about weaving the web</a></p>
</body>
</html>
```

4 Open your *index.html* file in a browser and check it looks like Figure 16.5. Notice the title at the top.

Photo: Flickr/cellanr/CC BY-SA 2.0

Figure 16.5 The HTML displayed in a browser showing the new hypertext at the bottom.

Notice the coloured hypertext link now created at the bottom of the web page? Click this link because it is well worth reading Sir Tim Berners-Lee's explanation of how he wove the web in his own words.

It is pretty cool to see the text gain super-powers!

Internal or external CSS and JavaScript?

In the following pages we create both CSS and JavaScript files. If you are using *replit*, empty versions of these two files are automatically generated. However, these can also be placed inside an HTML file.

CSS can be handled by HTML in three ways:

- *Inline* by using the style attribute inside any HTML element tags:
  `<h1 style="color:purple;">My purple heading</h1>`
- *Internally* by placing it between `<style>` and `</style>` tags in the `<head>` section of the HTML file
- *Externally* by using a `<link>` element in the HTML file to link to a named external CSS file:
  `<link rel="stylesheet" href="style.css">`

JavaScript can be handled in two ways:

- *Internally* by placing it between `<script>` and `</script>` tags as part of the HTML file.
- *Externally* by naming the external `src` source file as part of the `<script>` and `</script>` tags:
  `<script src="myScript.js"></script>`

Going further with HTML

This is a good time to learn more HTML. There are many good tutorials on the web but a great place to start are the Mozilla tutorials.

Complete the 'Getting started' and 'HTML' sections to extend your knowledge of HTML. As well, W3Schools has an easy-to-read reference with interactive examples.

Weblinks
Structuring the web with HTML

W3Schools: HTML Tutorial

ACTIVITIES

Identify

1 What are 'tag pairs' in HTML?
2 What do the letters CSS stand for?
3 How do both HTML and CSS differ from JavaScript?
4 Describe the purpose of an anchor element in HTML.
5 What is `id` used for in CSS?

Analyse

6 Describe the distinct purposes of each of HTML, CSS and JS.
7 What are IDEs and what advantages do they offer over plain text editors?
8 Describe typical scenarios where internal or external CSS and JavaScript might be preferred.

Extend

9 Use *replit* or similar to build a simple web page of your own.

16.3 CSS essentials

You have now learned some HTML essentials and know how to create a simple web page. The first websites used HTML tags to describe the style (colour, alignment etc.) of text in a web page. These needed to be inserted every time a change occurred. Web designers wanted to separate the content (HTML) of a page from its layout. The ability to define styles and reuse them made a huge difference for web site designers. This is called CSS and stands for Cascading Style Sheets.

Like HTML, CSS is a description language, not a programming language. CSS is used to change the font, colour, size, background and spacing of your content or to split it into multiple columns. It can even be used to create animations!

Here is a snippet of CSS:

```
p {color: purple;}
```

This code would apply a purple colour to all sections of an HTML page enclosed by a pair of `<p> </p>` **meta tags**.

CSS can be written as part of your HTML file or saved as a separate external file. It is a good habit to write it as a separate file so you can then reuse it.

A CSS rule is made up of two parts:

- *Selector* names the elements in your HTML document to which that rule will apply.
- *Properties* describe how that element is to be styled, for example purple text with a border.

```
p {color:purple; text-align:center}
```

The CSS box model

CSS is based on rectangular boxes. It uses the concept of nested boxes to apply its rules. Each element is imagined as being inside a box, with that box's content, padding, border, and margin built up around one another like a big warehouse crate packed with smaller boxes. The outer boxes control styling for the inner boxes (Figure 16.6). This is why CSS has the word *cascading* in it.

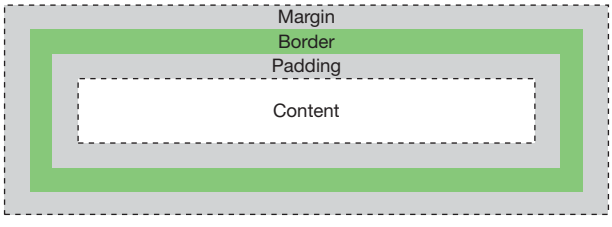


Figure 16.6 CSS uses this box model to decide how to apply its rules for CSS padding, borders and margins.

The box model allows you to add borders around HTML elements and space between them.

- Content: The content of the box consisting of text and images
- Padding: The transparent area around content
- Border: The border around both padding and content
- Margin: The transparent area outside the border.

Common CSS tags

Table 16.1 lists some common CSS tags with a description of what each one does.

Table 16.1 Common CSS tags.

Common CSS properties and values		
CSS property	**CSS example**	**explanation**
margin	`margin: 20px`	*Defines space between element's border and its parent*
padding	`padding: 20px;`	*Defines space between content and border*
border: size style colour	`border: 20px solid purple`	*Select from* `none, dotted, dashed, solid, double`
background-colour	`background-color:antiquewhite`	*Uses colour name or hex value*
font-family	`font-family: "Times New Roman", Times, serif`	*Font name in quotes if more than one word. Here if Times New Roman is unavailable will try Times, then will try generic fonts*
font-size	`font-size: 40px`	*Can use pixels (px), ems or per cent*
font-style	`font-style: italic`	`none, italic, oblique`
font-weight	`font-weight: bold`	`normal, bold`
text-align	`text-align: center`	`left, right, center, justify`
left, right, top, bottom	`position: relative; left: 70px;`	*Sets position but must state absolute or relative first*
height, width	`width: 200px; height: 200px;`	*Specifies height and width of an element*
colour	`background-color: purple; color:orange`	*Uses colour name or hex value. In this example, both a purple background-color and orange text color*

SKILL BUILDER

Adding CSS to your first web page

If you are using a web editor such as *CodePen* or *replit*, a CSS file is likely already provided (see Figure 16.7), however it will be empty. If instead you are using a plain text editor such as *Notepad* or *TextEdit*, make sure both HTML and CSS are saved inside the one folder.

1 In the previous Skill builder you created a simple web page (Figure 16.7). Now we will use CSS to control its appearance. We need to tell the HTML file where to find the CSS file. Add the following code to the head section of your HTML file. This tells it where to locate the external CSS file:

```
<link href="style.css"
  rel="stylesheet">
```

This is what these terms mean:
`href`: hypertext reference

`rel`: states that the relationship between the HTML document and the linked one is a stylesheet.

Save your changed *index.html* file.

2 Make sure the CSS file is named *style.css* and add the following code to it.

```
h1 {
    font-family: Arial, Helvetica,
      sans-serif;
    font-size: 42px;
    display: inline;
    color: blue;
    background-color: rgb(166, 191,
      230);
}
p {
    font-family: Arial, Helvetica,
      sans-serif;
    font-size: 18px;
    color: rgb(0, 0, 0);
}
```

Inspect carefully each of the properties shown. The `inline` property states that this CSS applies only to the element `h1` and not the whole width of the page. Omit it and note the difference after reloading the HTML file.

Does it now look like Figure 16.7?

Delete the background colour property for the h1 heading and add the following to your CSS code:

```
body
{
display: inline-block;
background-color:antiquewhite;
padding: 20px;
border: 20px solid purple;
margin: 20px;
}
```

Photo: Flickr/cellanr/CC BY-SA 2.0

Figure 16.7 Your HTML page with CSS applied. At this stage we are not concerned with how good (or bad!) it looks.

Here CSS is being applied to the entire *body element* of the HTML. The body element has set its *width* and *height* automatically (this is the default when it is not specified) and added a padded area, then a solid purple border 20 pixels wide and a margin around the whole element. The background colour is defined – there are over 100 named colours to choose from or hex values may be specified.

The result is shown in Figure 16.8 and the complete final version is available in the Nelson MindTap resources.

Photo: Flickr/cellanr/CC BY-SA 2.0

Figure 16.8 Adding more CSS features.

Skill builder resource
Adding CSS to your first web page

Weblinks
Learn to style HTML using CSS

W3Schools: CSS Tutorial

As the browser renders code, it works out what styles are to be applied to each 'box', knowing that if they are contained within other boxes they will take on its rules as well.

4 Use *replit* with the same HTML code as before, and experiment with CSS properties listed altering their values. Don't forget to end each with a semicolon! Try deleting the display property and see how the body content now stretches to fit the display size.

5 The best way to learn is to do! *replit* and *CodePen* are great tools for learning how HTML and CSS work together as they allow you to immediately see the effects of adding and changing your CSS code.

Learn CSS

This is a good time to dig deeper into CSS. There are many tutorials on the web but a great place to start is Mozilla

Extend your knowledge of CSS by completing as much of the tutorial as you need.

W3Schools also has an easy to read reference with interactive examples for CSS.

ACTIVITIES

Identify

1 Describe *selectors* and *properties* in CSS.

2 Identify five features of a web page that could be controlled using CSS.

3 Write CSS for making all the h1 heading text in an HTML page purple with font size set at 30 pixels.

4 Outline how CSS can be used as part of an HTML file rather than as an external file.

Analyse

5 Explain the basic structure of the CSS box model by using a diagram.

6 Describe the effect of the following CSS on the body of an HTML page:

```
body
{
display: inline-block;
background-color:antiquewhite;
padding: 30px;
border: 40px solid blue;
margin: 50px;
}
```

Extend

7 Construct a table to compare features of the following three IDE web development tools: *repl.it*, *CodePen* and *jsFiddle* (refer to the Knowledge builder at start of this chapter).

16.4 JavaScript essentials

The time has come to learn some JavaScript. By now you have mastered the essentials of HTML and CSS. Whereas HTML and CSS were about web content and design, JavaScript is about interactivity. In the guided tutorial following, you learn to use JavaScript while creating a quiz app, but first we cover the basics. Note: Do not confuse JavaScript with the computing language Java. They have different histories and different purposes.

We will use the website you created learning HTML and CSS and add interactivity to it using JavaScript.

JavaScript basics

Here you will learn JavaScript by using it, however it is helpful to use a reference table of essentials so you have these in one place. The chapter *Introduction to software programming* covered programming basics and our code examples featured the general purpose language Python. For convenience, we provided a table summarising the main features of Python. If you compare that table with Table 16.2, you will notice common features. Learning one language helps you learn others!

As JavaScript is used in combination with HTML, the first section of this table deals with accessing the HTML code.

9780170442787

Table 16.2 JavaScript essentials.

JavaScript essentials	Code examples *Note JavaScript is case sensitive:* `newVariable` *is not the same as* `newvariable`*!*
Input and output	

Include JavaScript in an HTML page	`<script type="text/javascript">`
	`  //JS code goes here`
	`</script>`

Call an External JavaScript File	`<script src="script.js"></script>`

Accessing HTML elements via name or ID	`let getElement = document.getElementById("myID");`
	or
	`let getElement = document.querySelector ("h1");`

Changing HTML content	*The following require* `getElement` *to be accessed first (as shown above)*
	To return text only:
	`getElement.textContent = "content";`
	
	To return actual text without inner tags:
	`getElement.innerText = "content";`
	
	To return entire HTML content:
	`getElement.innerHTML = "content";`

Write directly to HTML document *Usually used while testing as replaces all existing HTML.*	`document.write("content");`
Prompt *Prompts user via dialog box. Returns a string.* *You can use this method for small amounts of information.*	`let firstName = prompt("Please provide your name", "default value");`
Using the console *Outputs values to browser's built-in console while developing code.*	`console.log("The value id ", myVariable);`
Operators	
subtraction	`-`
multiplication	`*`
division	`/`
raise to power (exponentiation)	`**`
remainder (modulus)	`%`
assignment	`=`
test for equality	`===`
less than	`<`
greater than	`>`
greater than or equal to	`>=`
less than or equal to	`<=`
not equal to	`!=`
`and` True if both are true	`x && y`
`or` True if *at least* one is true	`x || y`
Control structures	

Selection (Conditional)	`if (num === secret){`
	`  alert('Well done')`
	`}`
	`else if (num < secret){`
	`  alert('too low');`
	`}`
	`else{`
	`  alert('too high');`
	`}`

Repetition
`while` loop
	`while (i < 10){`
	`do something;`
	`}`

(Continued)

Table 16.2 (Continued)

Counted repetition
`for` loop
	`for ( i = 0; i < 10; i++){`
	`do something;`
	`}`
	
	or for array elements:
	`for ( i = 0; i < myArray.length; i++){`
	`do something;`
	`}`

Other	

Comment in code	`/* Multi line`
	`comment */`
	
	`// One line comment`

Random numbers	`num = Math.random();` *returns 0≤ random number<1* *ie from 0 up to, but not including 1*

Writing functions	`function circleArea(r) {`
	`let area = Math.PI * r * r;`
	`return area;`
	`}`

Declaring a variable
`let` *has block scope*
	`let newVariable = 43;`
	
	or
	`let newVariable;`
	`newVariable = 43;`
	
	for strings use quotes:
	`let newVariable = "somestring";`

Calling functions	`total = circleArea(4);`

Interaction	*Common examples:*
	`onclick`
	`onmouseenter`
	`onkeydown`

SKILL BUILDER

Skill builder resource
Adding JavaScript interactivity

Adding JavaScript interactivity

1 Using *replit* open the HTML file you created earlier in this chapter. We have provided the final version of the HTML and CSS files in the Nelson MindTap resources in case you do not have them, titled *Javascript_interactivity_begin*.

Preparing the HTML file

2 Before starting a JavaScript (.js) file add the following code immediately before the end of the `<body>` section in your HTML file to tell it the location and name, `script.js` of this file. `src` stands for source.

```
<script src="script.js"></script>
```

This allows the HTML file to locate the .js file, so JavaScript can interact with it.

The reason we place the `<script>` element near the end of the HTML file is browsers read code in the order it appears. If the JavaScript loads before the HTML with which it is meant to interact, it may not work properly.

3 Replace this line in the HTML file:

```
<p>A big thank you to Sir Tim Berners-
  Lee!</p>
```

with these two lines:

```
<button>Introduce yourself!</button>
<p id = "personal"></p>
```

The first line creates a button and puts the label `Introduce yourself!` on it. The second line creates a paragraph with an `id` of `"personal"`, but which is empty; i.e. it contains no text yet between the tags.

Your HTML code should now look like this:

```
<!DOCTYPE html>
<html>
<head>
    <title>My HTML page</title>
    <link href="style.css"
      rel="stylesheet" type="text/css"/>
</head>
<body>
    <h1>HTML beginnings</h1>
    <p>HTML code is <strong>very
      </strong>useful.</p>
    <button>Introduce yourself!</button>
    <p id="personal"></p>
<img src="https://tinyurl.com/tblthumbnail"
  alt="Sir Tim Berners-Lee">
<p><a href="https://www.w3.org/People/
  Berners-Lee/Kids.html">Tim Berners-
  Lee answers questions about weaving
  the web</a></p>
<script src="script.js"></script>
</body>
</html>
```

Adding interactivity with JavaScript

4 Type these as the first lines of your new .js file:

```
let myButton = document.
  querySelector("button");
let getPersonal = document.
  getElementById("personal");
```

There are a few ways to identify elements of the HTML file. Here we meet two main ones. In the first line, the `querySelector()` function finds the first element in the HTML document with a name that matches `'button'`, and then stores it in the variable `myButton`. When you want JavaScript to do something to an HTML element, naturally you need to tell it which one.

The second line instead uses `getElementById()`, which identifies the element in the HTML document which was given the `ID` of `personal` and stores it in the variable `getPersonal`.

5 Add this function named `setUserName()` followed by a button function that triggers or calls it, when the button is clicked:

```
function setUserName() {
    let myName = prompt("Please enter your name.");
    if (!myName) {
        setUserName();
    } else {
        getPersonal.textContent = "Hi " + myName +
          "! Great to meet you -  ask me anything about the web...";
    }
}
myButton.onclick = function() {
    setUserName();
};
```

Run this code. You should see the image shown in Figure 16.9 before pressing the button.

Flickr/cellanr/CC BY-SA 2.0

Figure 16.9 Adding interactivity with JavaScript.

Press the button and a personalised message should show. Use the link below the image to find answers to your questions!

Tidying up

6 Try opening the HTML file in a regular bowser. Do this by saving the three files (index.html, style.css and script.js) in a single folder then double-clicking the index.html file. To save these files in CodePen use Export, download as zip, then open the 'dist' (distribution) folder. In replit use the drop down dots beside Files and download as zip. You will see the border stretches awkwardly to fill the width. This is where CSS is really useful as we can modify the appearance of a page without recoding the HTML, as was necessary in the early days of the web.

Set the margin property to `auto` to horizontally centre an element within its container. The element will then take up whatever width was specified, and the remaining space will be split equally between the left and right margins.

The meanings of padding, borders and margins are shown in Figure 16.6. Note: The width of an element does *not* include its padding, borders or margins!

7 Change the CSS for the body to:

```
body {
    display: inline-block;
    background-color: antiquewhite;
    padding: 20px;
    border: 20px solid purple;
    width: 400px;
    margin: auto;
}
```

The result of this exercise is provided in the Nelson MindTap resources, named *Javascript_interactivity_button*

8 We decide we want the button to be prettier – try styling it using CSS. Add the following to the end of your CSS file and try playing around with the values.

```
button {
      font-family: sans-serif;
      font-size: 20px;
      background-color: rgb(124, 193, 221);
      border: 0px;
      border-radius: 10px;
      padding: 10px 30px;
      cursor: pointer;
      margin-bottom: 20px;
      width: 50%;
      margin-left: 25%;
      margin-right: 25%;
}
```

Weblinks
Grok Academy

Tutorials for complete beginners to the Web

Swapping images

Now for more interactivity! We create a second function to swap the image when clicked.

9 First, add the following line at the beginning of your script.js file:

```
let timsImage = document.querySelector('img');
```

Using your exported files, create a folder inside your enclosing project folder and call it *images*. We have provided two images of Sir Tim Berners-Lee in the Nelson MindTap resources. Drag these into this new *images* folder and add the following function and its function call (the word `swap()` at the end) to your JavaScript script.js file.

```
function swap() {
    let mySrc = timsImage.getAttribute('src');
    if(mySrc === 'images/Tim_Berners-Lee_2012.jpg') {
        timsImage.setAttribute('src','images/Tim_desk.jpg')
    } else {
        timsImage.setAttribute('src','images/Tim_Berners-Lee_2012.jpg');
    }
}
timsImage.onclick = function(){
    swap();
}
```

Try figuring out how this works before reading on. The function `swap()` checks which image is the current source (`src`) and swaps it for the other one when the image is clicked. Double click on index.html and try this now.

Congratulations! You have created your first interactive web page using raw coding. This is what web developers do!

A final version of this activity is available online, called *JavaScript_interactivity_final*

In the Guided project towards the end of the chapter you will create an interactive Quiz using Javascript where you will add more interactivity and put into action many of the things you have just learned.

Learn JavaScript

This is a good time to learn more JavaScript. We suggest you try out Grok learning's interactive JavaScript module before tackling the project. You will also find an introductory Space Invaders JavaScript module to try! See Figure 16.10.

There are many excellent JavaScript tutorials if you want more practice. We recommend Mozilla's JavaScript tutorials (along with their HTML and CSS tutorials).

Source: Grok Learning

Source: Mozilla

Figure 16.10 Two online resources for learning JavaScript: (a) Grok Learning (b) Mozilla's JavaScript tutorials.

16.5 Canvas essentials

HTML5 added a feature called **Canvas**. It was first introduced by Apple as part of their WebKit and today all major browsers support it. `<canvas>` is an HTML element which can be used to draw graphics and create animations using the JavaScript scripting language, allowing games to be written for browsers.

In the project at the end of this chapter you will develop a snake game from scratch using HTML5 Canvas and JavaScript in a guided project. It is a great way to master JavaScript.

First complete the following Skill builder to learn to use canvas.

SKILL BUILDER

Using canvas with JavaScript to animate

Setting up canvas

1 We will use *replit* in the following but other code editors such as CodePen can be used. From now on we will refer to this as the 'editor'. You will see HTML and JavaScript files in the left Files panel.

Canvas is part of HTML but is controlled by JavaScript.

2 You need to first set up a canvas element in a web page. Create a canvas element, give it dimensions, an ID and a black border by adding this line to the body of your editor's HTML file. Your HTML page will then appear as:

```
<!DOCTYPE html>
<html>
    <head>
<canvas id="learningcanvas" width="500"
  height="500" style="border:1px solid
  #000000;"></canvas>
   </head>
   <body>
      <script src="script.js"></script>
      </body>
</html>
```

3 In your editor's JavaScript (JS) file, which will start off empty, add the following three lines:

```
//DEFINITIONS SECTION
const cvs = document.getElementById("learningcanvas");
const ctx = cvs.getContext("2d");
```

Here we identify our canvas element in the HTML document using its id `"learningcanvas"` and define this as the constant `cvs` (we could choose any name) as we don't want to have to write `document.getElementById("learningcanvas")` each time! Then, because there are a number of ways to use canvas we tell the browser to use its 2D canvas methods.

Drawing a rectangle

In canvas, unlike in mathematics, the origin is the top left corner, with the y axis facing downwards. This is mainly because early computer displays fired electron beams starting from the top left position. See Figure 16.11.

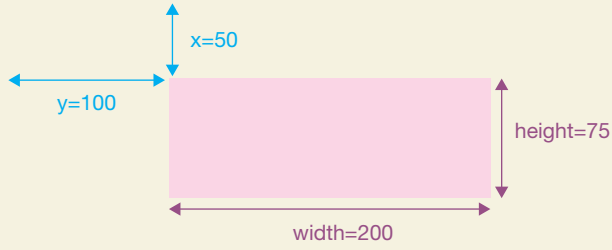


Figure 16.11 Drawing a blue rectangle on canvas using JavaScript.

To apply colours to shapes, we use

`fillStyle = color`

to set the style when *filling* shapes and

`strokeStyle = color`

to set the style when *outlining* shapes.

4 To draw a filled rectangle with its top left corner at (x,y) add use `fillRect(x, y, width, height)`

Add the following code to your JS file to draw the blue rectangle shown in Figure 16.11 to the canvas:

```
ctx.fillStyle = "blue";
ctx.fillRect(50, 100, 200, 75);
```

Press the "run" or "save" button (depending on your code editor). You should see the rectangle in Figure 11.

Note that whenever you set `strokeStyle` and/or `fillStyle`, these new values remain the default until they are changed. The value for `color` can be any CSS permitted colour, gradient or pattern, which are listed on the Web.

Skill builder resource
Using canvas with JavaScript to animate

5 To draw a border around a rectangle with its top left corner at (x,y) we use `strokeRect(x, y, width, height)`

Add this to your code to add a purple border to the blue rectangle (Figure 16.12) and run it.

```
ctx.lineWidth = 20;
ctx.strokeStyle = "purple";
ctx.strokeRect(50, 100, 200, 75);
```

Figure 16.12 Adding a purple border.

A full list of properties and methods for using canvas, along with tutorials for HTML, CSS and JavaScript can be found on Mozilla Web Docs site.

Weblinks
Learn web development

Tearable cloth

6 Replace the contents of your JavaScript file with the following.

```
        // DEFINITIONS SECTION
var c = document.getElementById("learningcanvas");
var ctx = c.getContext("2d");
let x = 0;
let y = 0;
function myRect(x, y){
    ctx.fillStyle="blue";
    ctx.fillRect(x, y, 50, 50)
}
```

The first two lines are the same, then we set up x and y variables for the coordinates of a rectangle and draw it on the canvas:

Following the animation approach explained above we will add a `mainloop` function. Our main loop just does one thing: increments x and y each time through the loop. We will follow this function statement with a call to kickstart it. What do you think you will see?

7 Add this function to your JavaScript file:

```
function mainloop(){
    x += 5;
    y += 5;
    myRect(x, y);
    requestAnimationFrame(mainloop);
    }
requestAnimationFrame(mainloop);
```

Run the program and you should see Figure 16.13!

Each time through the animation loop the rectangle is redrawn but never erased. This creates a 'trail'.

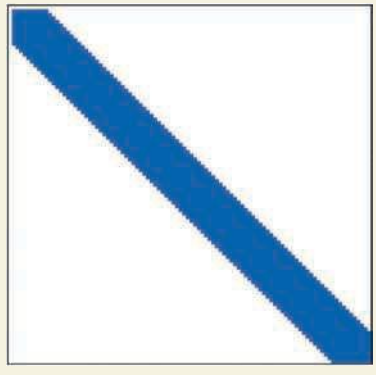


Figure 16.13 Square leaving a trail.

8 Add the following code as the first line of the `mainloop` function to erase the large 500 x 500 rectangle, which will also erase our blue one:

```
ctx.clearRect(0,0,500,500);
```

Animating canvas

Once a shape is drawn using canvas, it stays there. For this reason, when we create a game where objects change position, we need to repeatedly update the frame.

INFOBIT

Check out this impressive interactive canvas animation.

`requestAnimationFrame()` is used to call (refresh) an animation frame at regular intervals.

The following illustrates the idea. We draw a rectangle on the canvas and then animate it.

```
function mainloop() {
    // code goes here
    requestAnimationFrame(mainloop);
    }
//start the main loop
requestAnimationFrame(mainloop);
```

This last line 'kickstarts' the code by calling the main loop function named mainloop() and the last line in the loop function keeps calling itself every time to keep the animation going.

Run the program and you should see the blue square animated along a diagonal path as in Figure 16.14!

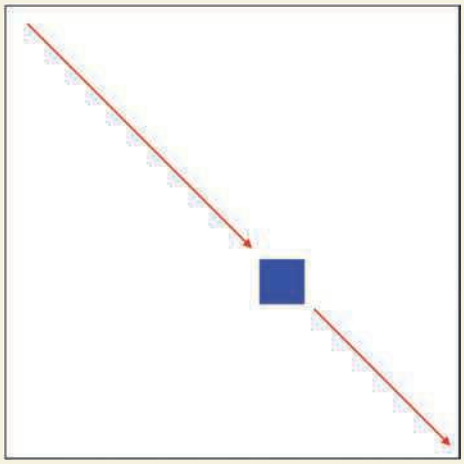


Figure 16.14 An animated square.

Adding bounce

We have supplied a more advanced version of this animation on Nelson MindTap named *bounce_demo*. Run the demo.

Open and examine its code and read the internal comments and see if you can understand how it works to produce the result shown in Figure 16.15.

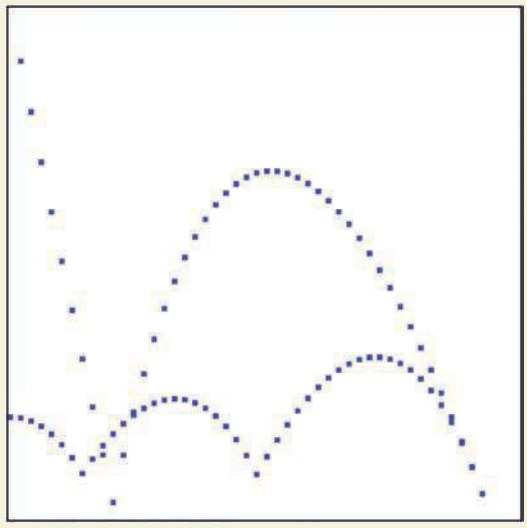


Figure 16.15 A square with real bounce!

The projects at the end of this chapter use these canvas skills to create a quiz game and a retro game of snake where we bring together the three components of web development covered in this chapter: HTML, CSS and JavaScript.

The quiz game is a guided project and will consolidate your knowledge of JavaScript.

Weblink
Bounce Demo

ACTIVITIES

Identify

1. What is the HTML canvas element used for?
2. Which of HTML, CSS or JS is used to define the canvas element? Write the code for a canvas element 500 x 500 with a black border.
3. Write JavaScript for drawing a purple rectangle 150 x 250 pixels at canvas screen coordinate (100, 200).

Analyse

4. Briefly outline the process for animating an object using canvas.
5. Describe three practical applications for the canvas element.

Extend

6. Explore mini canvas projects which used canvas by searching the word in *CodePen*. Then select three and describe in broad terms how each effect was achieved.

CASE STUDY

Software developers

Shutterstock.com/PR Image Factory

Figure 16.16 Software developers with dual degrees can bring more than just technology expertise to a business.

Florence is a good example of the modern generation of software developers. She has a degree qualification in software development, as well as a master's degree in finance.

Personnel with dual qualifications are able to provide:

- business skills to understand what a business is doing and where improvements can be made
- technical software development skills to envisage how these improvements can be made.

People like Florence have the capacity to add great value to a business – typically bringing a broader vision for the benefits of technological change and growth in a company than those who are 'just business people' or 'just technologists'.

Florence works for a large website software development company that specialises in integrating mobile telephony and SMS with web and other applications, and in mobile computing applications.

Questions

1. Several universities offer combined degrees with computer science as one component. Find out what combinations are offered at three universities.
2. In your own words, suggest to what extent software developers have to be 'inventors' of the future.
3. On your school's Open Night, a visiting primary age student was overheard to say to his father: 'Dad, isn't it great that all the important computer programs have been written?' If asked, what would you say to him?

Weblinks:

- Replit
- CodePen
- Structuring the web with HTML
- W3Schools: HTML Tutorial
- Learn to style HTML using CSS
- W3Schools: CSS Tutorial
- Grok Academy
- Tutorials for complete beginners to the Web
- Learn Web Development
- Tearable Cloth
- Bounce Demo
- Quiz Demo
- JS Fiddle
- <video>: The Video Embed element
- A basic snake game
- Google Maps snake game
- Snake game with scores and pause
- snake_onscreenbuttons
- snake_mobile
- Game of Life

Skill builder resources:

- A first HTML page
- Adding CSS to your first web page
- Adding JavaScript interactivity
- Using canvas with JavaScript to animate

Chapter review quiz

Nelson MindTap

To access resources above, visit **cengage.com.au/nelsonmindtap**

16.6 Guided project 1: Building a web-based quiz in JavaScript

The best way to learn a language is to actually use it.

In this guided project you will use JavaScript to create an online quiz that runs on any internet-enabled device. Your quiz will feature multiple choice questions, images, a countdown timer, a results screen, randomised questions and the ability to redo the quiz (Figure 16.17).

Try our online web version with four questions to show you where we will be heading (see the weblink opposite). A version also appears in the teacher's resources on Nelson MindTap.

You will use the web languages you learned in this chapter: HTML, CSS and JavaScript. In this project you will create separate files for each, although it is possible to combine all three as part of a single HTML file.

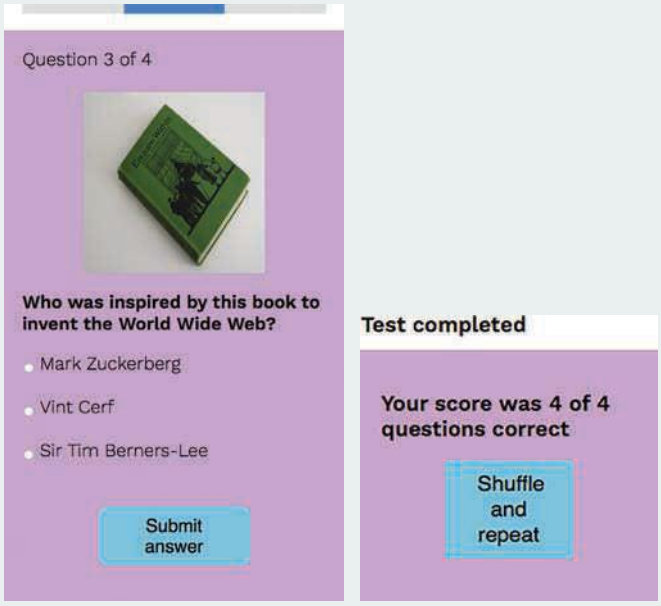


Figure 16.17 You will create this web-based Quiz app in this guided project.

Figure 16.18 shows the three separate files which work together to create your quiz web app.

Figure 16.18 HTML, JavaScript and CSS code will work with each other to create a quiz web app.

Weblink
Quiz Demo

Overview

We will break the project down into seven steps:

- Step 1: Select a web development tool.
- Step 2: Design the basic quiz.
- Step 3: Code the HTML page.
- Step 4: Code JavaScript to create quiz content.
- Step 5: Add CSS to improve user experience.
- Step 6: Make the quiz great: add quiz images, additional questions, random ordering and timer.
- Step 7: Add your own features and make further improvements.

Step 1: Select a web development tool

Earlier in this chapter you learned you could code a website using a regular text editor. Now it is time to make use of an IDE as your web development tool. These allow syntax checking as well as automatically setting up files for HTML, CSS and JavaScript.

We suggest the free online editor replit. Other recommended IDEs are *CodePen* or *jsFiddle* for this project. Advantages of *replit* and *CodePen* are:

- simple interface
- generates your own website and URL automatically
- images, files and folders can easily be dragged into the project
- web output can be seen live
- console for debugging
- project can be downloaded and saved as a zipped file.

Select one of these web development tools. If using *replit* first select the correct language choice as shown in Figure 16.19.

Weblinks
Replit
CodePen
JS Fiddle

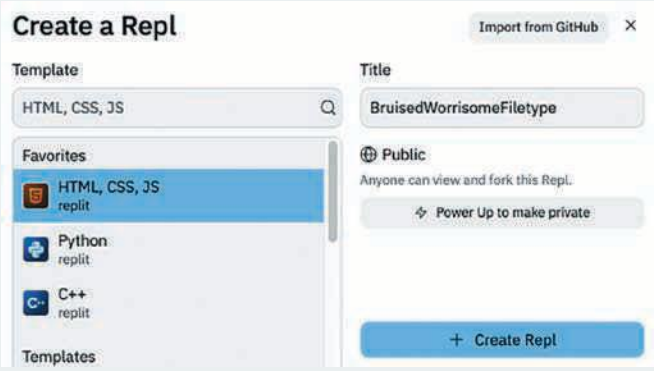


Figure 16.19 In *replit*, select the HTML, CSS, JS language choice from its drop-down menu.

Step 2: Design the quiz

Design the interface

Sketch the UI for your quiz, although this is likely to evolve while the quiz is coded. We have used our own design but you are free to customise the design by using your CSS to modify it later. See Figure 16.20.

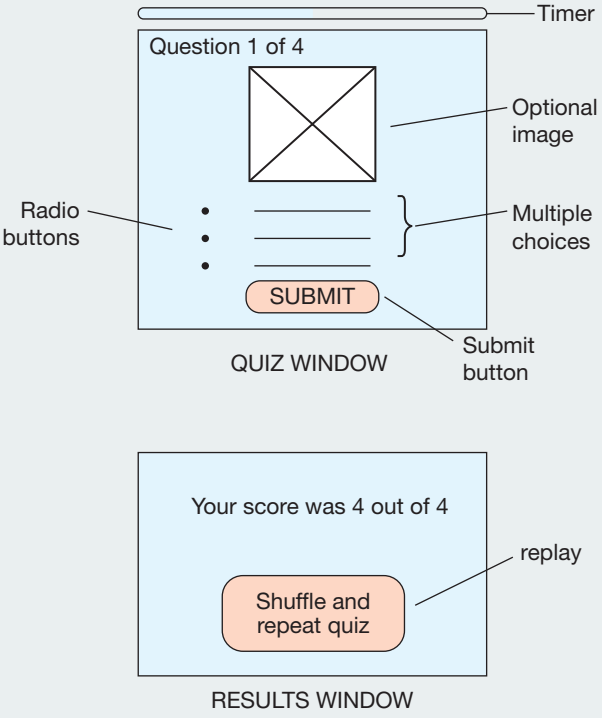


Figure 16.20 Designing the quiz interface.

Design the program

Solving a big problem is usually best done by *decomposition*, which means breaking the problem down into smaller, more manageable parts. Programmers do this using various tools. Here we will list the steps for developing the prototype version and draw a flowchart (Figure 16.21) to represent this.

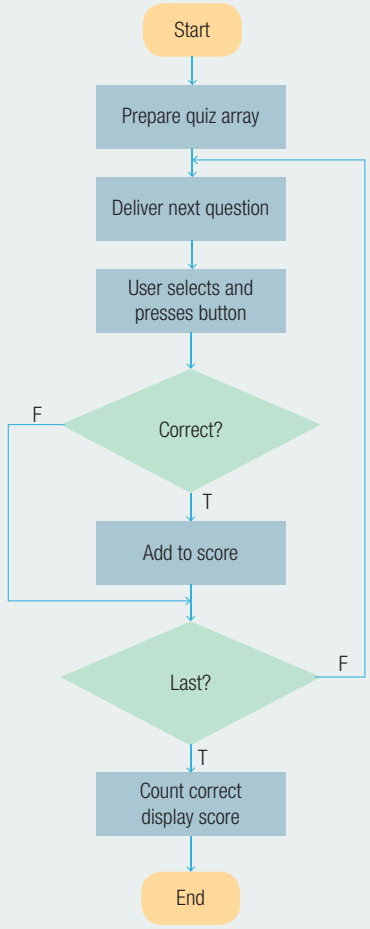


Figure 16.21 Using a flowchart to plan the program's logic.

- Set up an array data structure to hold questions and answers
- Deliver next question
  - Deliver first question when the web page is opened
  - Deliver next question when button is pressed
- User selects response
  - If correct, increment score
  - If incorrect or no response do not increment score
- Test if this is the last question and if so, go to next step; if not, repeat step 2.
- Count and display score

Step 3: Code the HTML

2 Here you create your HTML code. If you are using a web development tool such as *replit*, delete their default HTML file content and type the following code to form the content of your HTML file. See Figure 16.22 showing the replit interface with this HTML file added.

Even though we have not yet written any JavaScript or CSS, we provide links to both these external files so the HTML code can locate them. We have shown these here using HTML comments, which appear like this:

```
<!--this is a comment-->
```

```
<html>
<head>
    <title>Quiz</title>
    <link rel="stylesheet" href="style.css">    <!--link to CSS file -->
    <script src="script.js"> </script>    <!--link to JavaScript file -->
</head>
<body>
    <div id="quiz"></div>
</body>
</html>
```

You have now used HTML to create the single web page needed for your quiz project.

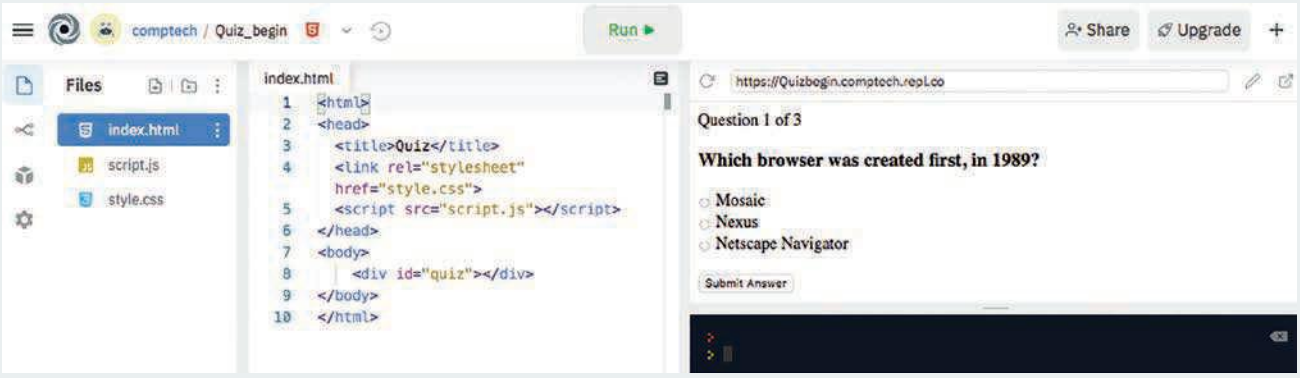


Figure 16.22 *Replit* web development environment, showing our Quiz prototype. Other code editors have similar layouts, such as *CodePen*.

KNOWLEDGE BUILDER

DOM, JavaScript and the HTML page

When a web page is first loaded, your browser creates a **Document Object Model (DOM)** to represent web pages (see Figure 16.23). The DOM is a W3C (World Wide Web Consortium) standard which creates a kind of tree diagram of all elements on the page. The DOM is an object-oriented representation of the HTML document.

JavaScript uses the DOM to interact with the elements of the HTML document. JavaScript can:

- add, change and remove the HTML elements
- change HTML attributes
- change CSS styles
- react to events such as mouse clicks on elements.

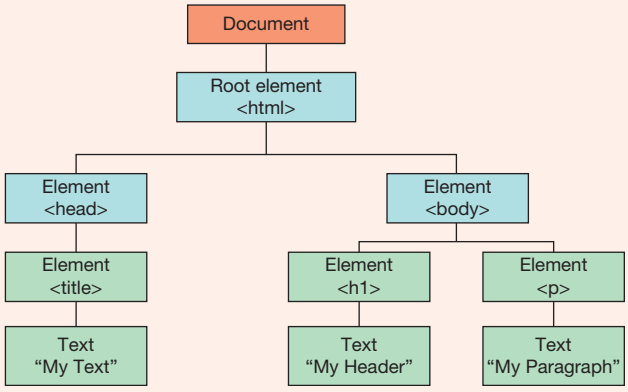


Figure 16.23 Browsers create this Document Object Model (DOM) of a web page when the page is loaded.

Step 4: Write JavaScript to create quiz

Here you create your JavaScript code. We will introduce the code in stages with each stage fully explained for you.

Plan the JavaScript code

The code will have these sections:

VARIABLES

- Global variables
- Array to hold questions

FUNCTIONS

- `function get(x)` will shorten lengthy addresses of HTML page elements
- `function renderQuestion()` will deliver questions to the HTML page
- `function checkAnswer()` will check user responses

Declare and initialise variables

Time to start coding your JavaScript! Into your editor's empty JavaScript script.js file type the first two lines of code for the quiz app:

```
let pos = 0;
let correct = 0;
```

A web designer may not know in advance what variables they will require but will add them as the code grows. We know we at least need variables for the current question and the number correct.

Design an array to hold questions

As we are designing a quiz, the most sensible data structure is an array of questions. Figure 16.24 shows the structure of this array. Each term is a collection of five elements that form that question. Note that numbering starts at zero and that Each of these questions consists of five objects like this:

```
{ Question wording, choice a, choice b,
  choice c, Correct answer}
```

To summarise, each term in the array represents a question which each contain five objects: the question, the three options and the correct answer. Now examine the code and understand how the array is structured.

Type the following js code immediately after the last code, being careful with punctuation! You can replace these with your own questions if you like but it would be wise to start with just three questions. Note there are two types of grouping symbols: square brackets defining the list data structure and curly braces defining separate questions. You can use either double or single quotes.

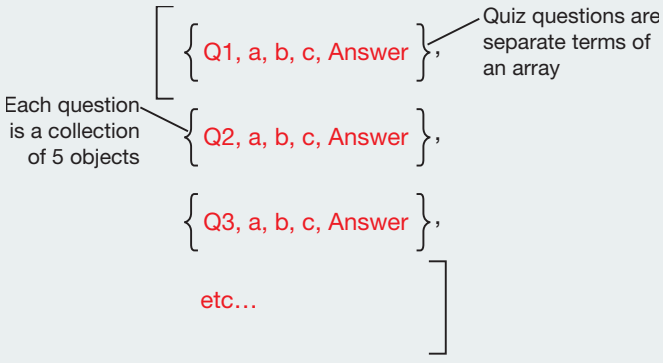


Figure 16.24 The array structure we will use to hold the quiz questions.

```
questions = [
    {
        question: "Which browser was created first, in 1989?",
        a: "Mosaic",
        b: "Nexus",
        c: "Netscape Navigator",
        answer: "B"
    },
    {
        question: "Who was inspired by this book to invent the World Wide Web?",
        a: "Mark Zuckerberg",
        b: "Vint Cerf",
        c: "Sir Tim Berners-Lee",
        answer: "C"
    },
    {
        question: "Where do files live on the Internet?",
        a: "On one massive computer managed by Google",
        b: "On your computer",
        c: "On many individual computers, often known as servers",
        answer: "C"
    },
    ];
```

The `get(x)` function

```
function get(x){
    return document.getElementById(x);
}
```

This first function eliminates the need to write `getElementById` each time. We could do without it but it makes our code much easier to read. Instead of typing `document.getElementById(ID)` every time we want to reference an element, we just call this function and it returns the long version.

Add this first function to your JavaScript code.

The `renderQuestion()` function

This function is the 'heavy-lifter' and is the one that actually builds our quiz. By breaking it into four parts it is easily understood. Continue reading as each stage is explained and code each stage as you progress. We now explain and build each part of the function.

A Check for end of quiz

```
// End of quiz check
function renderQuestion(){
    quiz = get("quiz");
    if(pos >= questions.length){
        quiz.innerHTML = "<h2>Your score was " + correct + " of " + questions.length + "
          questions correct</h2>";
        quiz.innerHTML += "Quiz completed!";
        pos = 0;
        correct = 0;
        return;
}
```

If you look at the HTML file you will see an empty `div` element called `quiz`. We use the `get` function coded in the last section to shorten its address, as it is required often.

The `if` statement checks if the user has reached the last quiz question:

```
if(pos >= questions.length). . .
```

If so, the JavaScript function `innerHTML` dynamically builds the `quiz` HTML content. Notice we are building *actual* HTML code, using tag pairs where separate strings are concatenated (joined) with the variable values by using the `'+'` symbol.

Examine the code to make sure you understand what each part does.

We also added a line saying "Quiz completed!"

Notice the use of the += symbol? This ensures the `innerHTML` is *added* to previous HTML, otherwise it would replace it.

As this is the end of the quiz we then reset the question number and the count of correct answers to 0. The `return` stops the function and the program returns to the line following where it was first called.

Add the above code as the first part of your `renderQuestion()` function. Note that this function is missing its final curly bracket, and this will be shown as an error by the IDE. We will add this after all the parts of this function have been written.

B Add question heading

```
// Question heading
quiz.innerHTML = "Question " + (pos+1) +
  " of " + questions.length;
```

This second part of the function is reached only if the quiz has not ended. It adds HTML to display the number of the current question.

Add the above code as the second part of your `renderQuestion()` function.

C Add questions

```
// Add question and choices
question = questions[pos].question;
choiceA = questions[pos].a;
choiceB = questions[pos].b;
choiceC = questions[pos].c;


quiz.innerHTML += "<h3>" + question + "</h3>";

quiz.innerHTML += "<label> <input type='radio' name='choices' value='A'> " + choiceA +
  "</label><br>";
quiz.innerHTML += "<label> <input type='radio' name='choices' value='B'> " + choiceB +
  "</label><br>";
quiz.innerHTML += "<label> <input type='radio' name='choices' value='C'> " + choiceC +
  "</label><br><br>";
```

This third part of the function (above) first fetches the correct question and its three possible answers (A, B and C) and assigns these to variables to make the code easier to follow.

As in the previous results section, it builds HTML content using JavaScript's `innerHTML`.

Notice `innerHTML` is used to add radio buttons to each of the three answers. HTML turns radio buttons into a group if they are given the same name (`choices`), which allows only one button to be chosen at any time.

Add the above code as the third part of your `renderQuestion()` function.

D Add a button

```
quiz.innerHTML += "<button onclick =
  'checkAnswer()'> Submit Answer </button>";
}
```

This final part of the function adds HTML to create a submit button which calls a function to submit the user's response, `checkAnswer()`.

Add the final line above to your `renderQuestion()` function. Carefully check the punctuation! Make sure you have not omitted any curly brackets, especially the final one to close the function!

The `checkAnswer()` function

```
function checkAnswer(){
    choice = null;
    choices = document.getElementsByName("choices");
    for(let i = 0; i < choices.length; i++){
        if(choices[i].checked){
          choice = choices[i].value;
        }
    }
    if(choice == questions[pos].answer){
        correct += 1;
    }
    pos += 1 ;
    renderQuestion();
}
```

The `checkAnswer()` function checks the user's choice against the correct one stored in the `questions` array.

Remember that `choices` was the name of the radio button group. It is an array, so each button is checked to see which was selected ('checked') and stores this in a variable `choice`. If it is correct, the score `correct` is incremented, the question number is incremented and the `renderQuestion()` function is called again to create the next question.

Add the above `checkAnswer()` function.

Code adapted from codemahal, https://www.codemahal.com/make-quiz-app-javascript

Start the program running on page load

```
window.addEventListener("load", renderQuestion);
```

If it were not for this very last line of code nothing would be happen! (Note that this is one single line of code.) The renderQuestion function has never been called. This line is called a 'listener'. Listeners are always waiting for their named event to occur. This one is waiting for a *page load* event. When it 'hears' that happen, it tells the program to run the `renderQuestion()` function.

Type the above final line of your code on a single line. Your code is now complete! Try running your quiz program. If you are using *replit*, click the green Run button.

Testing and evaluating your quiz program

Coding is an exact art! Every letter and punctuation mark, and often also spaces and tabs, have to be exact. It is often tedious to copy code but much more fun when you create your own, which we do later!

If the code does not run, follow these steps:

- Check that you have not broken a single line of code into two lines.
- Check for missing punctuation, especially curly brackets.
- Check for typos.
- Check your code line by line against ours seeing if you have missed a line.
- Add debugging output statements:
  - try adding `console.log(your_variable)` to check variable values at appropriate places
  - try adding `console.log("I reached here!")` to your code in appropriate places.
- If all else fails, check your version against the version provided in the Nelson MindTap resources: Quiz.basic. This version has comments explaining each section of the code.

When the program is working, add programmer's comments throughout explaining what each part of your code does, using the text explanations as your guide. Use your development tool to save this version locally (on your drive) your completed code as a zip file.

Step 5: Add CSS to improve the user experience

Now you have your quiz working we turn our attention to improving its interface. Remember that users are usually not interested in hearing how much work you did on the coding – they just want your app to work and be easy to use!

Identify improvements that could be made to this basic UI. Consider:

- fonts
- colours
- buttons.

You have already gained CSS skills. We will provide some starter CSS code you can try out. See Figure 16.25.

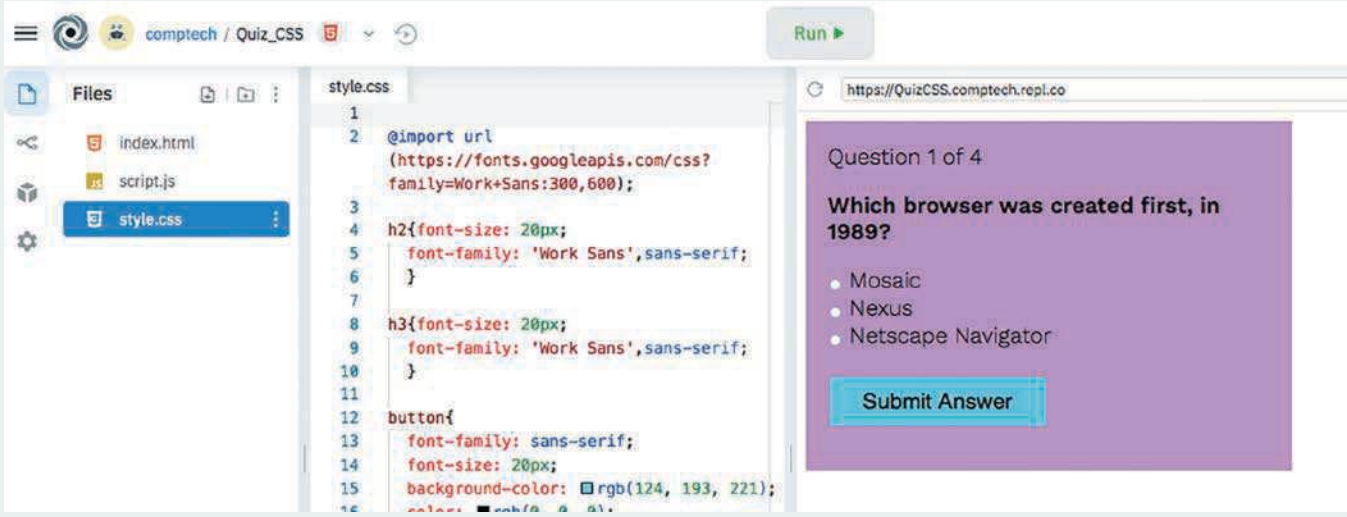


Figure 16.25 Our quiz with CSS added. Much prettier!

If using replit simply type the following into the empty *style.css* file provided. Otherwise create a text file, name it *style.css* and place it in the same project folder as your HTML and JavaScript files.

```
h2{font-size: 20px;
    font-family: sans-serif;
    }
h3{font-size: 20px;
    font-family: sans-serif;
    }
button{
    font-family: sans-serif;
    font-size: 20px;
    background-color: rgb(124, 193, 221);
    color: rgb(0, 0, 0);
    border: 0px;
    border-radius: 3px;
    padding: 10px 30px;
    cursor: pointer;
    margin-bottom: 20px;
}
#quiz{
    font-size: 20px;
    font-family: sans-serif;
    padding:20px 20px 20px 20px;
    background-color:rgb(201, 150, 216);
    position: absolute;
    margin: auto;
    width: 50%; }
}
```

Try running the code and see the change in the UI and UX!

Examine each CSS definition. We start by changing the font for headings `h2` and `h3`. We colour and style the submit button and add a background colour to the quiz section.

- Find out what margin: auto does.
- Try moving the quiz CSS section to after `h3`. Explain what happens. Change it back!
- Change values in the CSS and watch what happens. Don't forget to reload the app each time.
- The UI would be improved if it used a more stylish font. CSS allows you to import online fonts. We will import one called `'Work Sans'`. Add this to the beginning of your CSS:

```
@import url(https://fonts.googleapis.com/
  css?family=Work+Sans:300,600);
```

- Then replace sans-serif wherever it occurs in your CSS with `'Work Sans',sans-serif`
- Modify this CSS to create your own completely unique quiz UI!

Step 6: Add features to improve the Quiz

A great way to decide additional features in an app is to ask users. For the quiz you might survey both teachers and students as they may have different perspectives. In our survey the following features were suggested:

- responsive submit button
- add images to questions
- add a timer (advanced).

Responsive submit button

We would like to have the submit button change colour for a mouse over or a touch gesture.

This is easy with CSS! The *button:hover* and *button:active* selector allow us to modify a button's style attributes. We only need to supply the styles that we want to change. Predict what the following CSS will do. Note that in *replit* you can roll over the CSS colour patch and select any colour you wish!

Add the following to your CSS code immediately following the button CSS code and before the #quiz CSS.

```
button:hover{
      background-color: #279;
      color: #fff;
}
button:active {
      background-color: rgb(36, 10, 233);
      color: #fff;
}
```

Add images to questions

It would be great if images could be added to questions as shown in Figure 16.26.

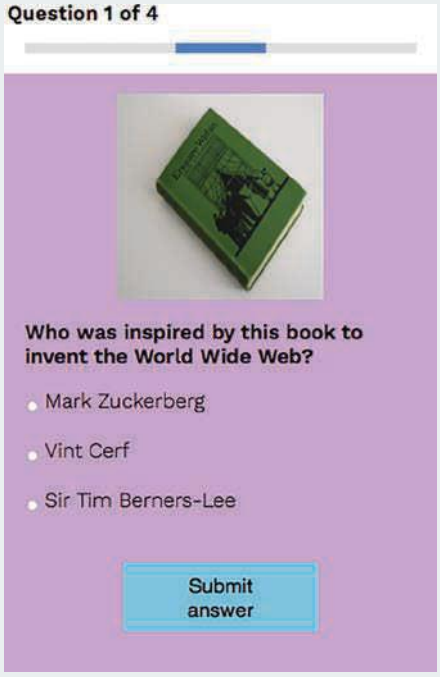


Figure 16.26 Add a images to your quiz questions.

We need to do two things:

- add an image URL object to each question in the `questions` array and set it to `null` if no image is used for that question
- add code to the `renderQuestion` function to place the image.

Here is the code for the first two questions in the JavaScript `questions` array. The first question has no image but the second one does. Add the bolded content shown (apologies for the long URL!). Commas are important! Remember we have not closed our list with a final square bracket yet.

```
var questions = [
    {
        imageURL: null,
        question: "Which browser was created first, in 1989?",
        a: "Mosaic",
        b: "Nexus",
        c: "Netscape Navigator",
        answer: "B"
},
{
    imageURL: " https://upload.wikimedia.org/wikipedia/commons/c/cf/Enquire_Within_Upon_
      Everything.jpg ",
    question: "Who was inspired by this book to invent the World Wide Web?",
    a: "Mark Zuckerberg",
    b: "Vint Cerf",
    c: "Sir Tim Berners-Lee",
    answer: "C"
},
```

Select suitable images from the Web for the other questions and use this approach to add their URLs to those questions, too.

After completing additional questions, add the square bracket to end the list structure for the set of questions.

15 Add the lines indicated in bold to your existing JavaScript code to the renderQuestion function in the positions shown. If parts of questions disappear suddenly mistyping this may be the cause!

```
quiz.innerHTML = "Question " + (pos + 1) + " of " + questions.length;
qImage = questions[pos].imageURL;
question = questions[pos].question;
chA = questions[pos].a;
chB = questions[pos].b;
chC = questions[pos].c;
quiz.innerHTML += "<h3>"+question + "</h3>";
    if (qImage) {quiz.innerHTML +=
      "<img src=" + qImage  + "width=200px" + "height=200px><br><br>";}
```

You have successfully built a quiz web-based app, which you can easily extend and modify. If using *replit*, use its URL (top right) to view it on a mobile device. The remainder of this guided project suggests further directions and improvements.

Add a timer progress bar (advanced)

We will use a progress bar to add a countdown timer to questions.

Add these two lines shown in bold to your HTML code. These add the progress bar element above your quiz question element:

```
<body>
    <div id="myProgress">
    <div id="myBar"></div></div>
    <div id="quiz"></div>
</body>
```

Add this CSS following the button descriptors:

```
#myProgress {
      width: 50%;
      background-color: #ddd;
}
#myBar {
      width: 50%;
      height: 10px;
      background-color: rgb(59, 130, 223);
}
```

Weblink
<video>: The Video Embed element

Chapter review quiz

Add this function to your JavaScript before the existing functions to control the progress bar:

```
function move() {
   width = 100;
   var elem = document.getElementById("myBar");
//the integer 100 can be changed
   id = setInterval(begin, 100);
   function begin() {
      if (width <=0) {
      clearInterval(id);
      checkAnswer()
      }else {
        width -=1;
        elem.style.width = width + "%";
        }
   }
}
```

Examine this function carefully to understand how the function works.

Place the call to this function immediately after the closing curly bracket of `renderQuestion` function's opening `if` statement as shown here in bold:

```
if (qImage) {quiz.innerHTML += "<img src=" + qImage  + "width=200px" +
  "height=200px><br><br>";}

move();
```

19 Add the following bolded line to the end of your `checkAnswer` function. This resets the timer after each question.

```
clearInterval(id);
renderQuestion();
```

Try running your program, fixing any errors.
A complete version is provided in the Nelson MindTap resources called *Quiz_improved*.

Extend this project

Here are some suggestions for further features:

- Using your new skills, build a Quiz for younger students.
- Get feedback from users and make improvements.
- Have the countdown timer reset only after the whole quiz is finished.
- Add a fourth choice to the three multiple choices.
- The timer appears on the results page. Fix this.
- Add the ability to auto-play short mp4 videos (see Mozilla).
- **Challenging:**
  - Enable users to replay the quiz. Hint: You need to reset some variables.
  - Shuffle questions without repeats.
  - Shuffle answers so they are not always in the same order.
  - Ask the user to set the time allowed for each question.
  - *Suggest further improvements – even if you don't know how to achieve them!*

We have provided a final version of this quiz with many of these features in the Nelson MindTap resources, named *Quiz_advanced*.

16.7 Project 2: A web-based retro snake game

The best way to learn to code is to start doing it! In this project, you will build the retro game *Snake* using HTML5 Canvas and JavaScript. Your game app will be playable on any desktop browser.

This is a quite challenging project but by completing it you will learn a great deal about web-based coding. If you get stuck, the Nelson MindTap resources contain working solutions for every stage of this project.

Play our desktop browser version of the game first to get a feel for it. Later we will build a mobile version for touch screens. Use arrow keys to play.

If you Google 'Snake game' you will find their version of this game!

Interface design is an important consideration if this game is to be played on a mobile device. Once you have the game working you will develop code to control it by touch and adapt its UI for playing on mobile devices.

Some advice: You will learn to code best by doing your own typing rather than copying and pasting ours. However, if at any time you need to inspect completed code you will find it inside the *Snake* folder in the Nelson MindTap resources.

Chapter 1 of this text provides an introduction to project management. Make sure you apply the important principles outlined in that chapter to your project, including addressing functional and non-functional requirements, considering social and ethical issues and documenting your project's development.

The game state

The values of variables in a game at any point in time are what is known as the *game state*. This is like a snapshot of the game. It is often helpful to list these states when designing a game. Our snake game could have any of the following states:

- snake is starting and is only a head
- the game is paused
- snake is moving right or left or up or down
- snake is colliding with an edge of the snake pit, with food or with itself
- the game has ended.

Identifying and defining

Weblinks
A basic snake game

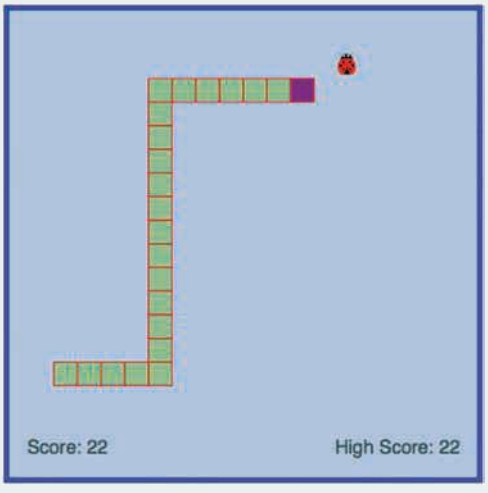
Google Maps snake game

Shutterstock.com/Peter Gudella

Figure 16.27 The Nokia mobile phone reignited interest in the game *Snake* when it was included in its software.

All snake games originate from the 1976 game *Blockade*. Search the web 'retro snake game' and you will find an original version. If you search for 'snake game' you will find an excellent version created by Google. Next, play the Google Maps version for Sydney. In 1997, the best selling Nokia mobile phone reignited interest in the game *Snake* (Figure 16.7).

Figure 16.28 shows how your snake game will look. The player controls the snake in one of four directions. If it crosses over a square containing a red ladybird the snake grows one segment longer. If it hits the edges of the snake pit or crosses over itself, it dies and the game ends.



Source: https://snakebasic.comptech.repl.co/

Figure 16.28 The final snake game keeps core, and also offers pause and replay functions.

Figure 16.29 Core mechanics for the snake game.

In the Games development topic you learn about the *Core mechanics* of a game which identifies the game world, its characters, its goal, challenges and rewards. Examine these characteristics for the snake game using Figure 16.29.

1 Develop lists for both functional and non-functional requirements of this snake game. If you need to revise these terms see Chapter 7.

Researching and planning

2 Sketch three possible designs for your UI including the elements shown in Figure 16.30.

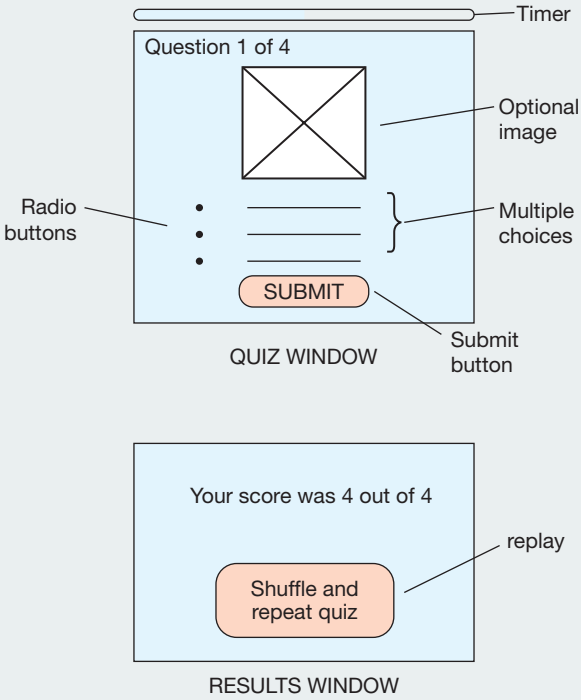


Figure 16.30 Sketch and label a few alternative layouts for your game including the elements shown here.

You will build the retro snake game in the form of an interactive web page using both HTML5 and JavaScript.

3 Open the *Snake_start* folder provided inside the *Snake* resources on Nelson MindTap. This contains four files: *index.html*, *script.js*, an *images* folder and an *audio* folder (see Figure 16.31).

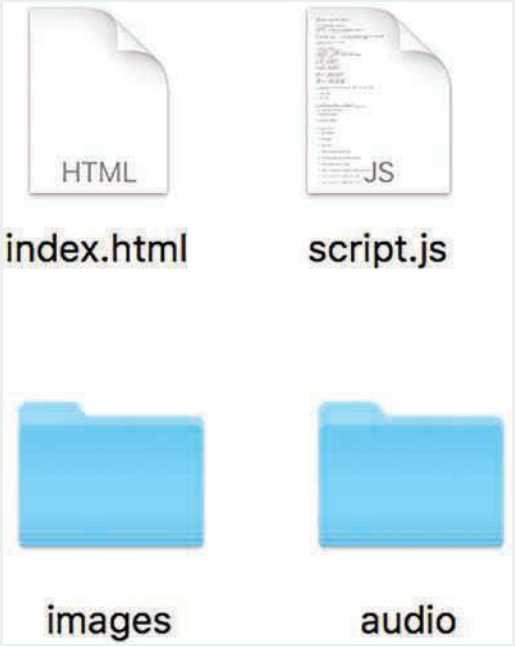


Figure 16.31 Folders and files for the snake guided project.

The HTML file is named *index.html* (the default name for the main homepage in a website). It holds basic content for this page. The JavaScript file is named *script.js*. Both contain pre-entered routine code. We also supply two starter folders: one with the images and one with audio files.

Choosing a tool for coding

We introduced the IDE *replit* earlier in this chapter. We use it here as our editor, although any of the editors introduced earlier can be used.

Developing an algorithm

Use the following outline of the logical flow of our game to draw a flowchart for the game.

DEFINITIONS

- Define variables

FUNCTIONS

- Set up functions

GAME LOOP

- Draw next animation frame
- Redraw snake pit
- Redraw snake
- Redraw food
- Store snake head position
- User selects direction
- Store new snake head position
- Check edge collisions:
  - yes: die and end
  - no: continue
- Check food collision:
  - yes: then don't remove tail (snake grows) + new food position
  - no: remove tail
- Check collision with self
- If no collisions insert new head in snake array
- Repeat game loop

Producing and implementing

Building the game

Here is where you begin creating your snake web app.

4 Open your IDE. If using *replit* when asked for language choice, select *HTML, CSS, JS* (see Figure 16.32).

Overwrite the existing three default files by dragging and dropping our start versions into the left Files panel.

- Drag *index.html* into the *replit* Files panel. Say OK to Replace. See Figure 16.33.

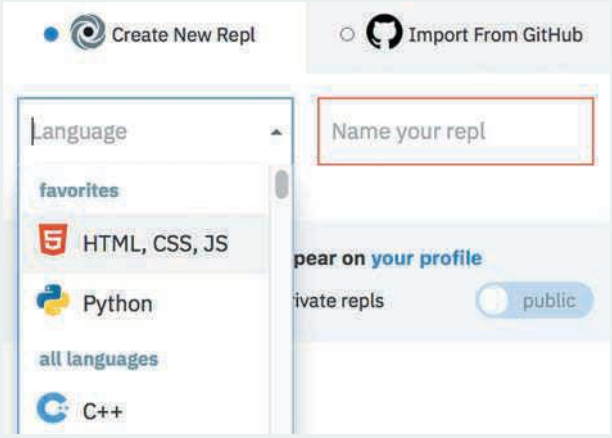


Figure 16.32 Select HTML, CSS, JS in *replit* for your language choice.

- Drag *script.js* into the *replit* Files panel. This file is where you will be doing your JavaScript coding.
- Drag both the *images* and *audio* folders into the *replit* Files panel.

The Files panel should resemble Figure 16.33. We do not need a style.css CSS file yet. Just leave this for now.

Figure 16.33 Drag your 4 files one at a time into the *replit* Files panel at the left along with audio and images folders.

The HTML file and canvas

5 The HTML is very basic for this game. We have pre-entered the code. You studied the canvas element earlier in this chapter. The HTML element `<canvas>` is like an artist's palette on which we draw the game. It uses two attributes: *width* and *height*. Inspect the HTML file to see that the canvas is already been defined in the HTML file and looks like this:

```
<canvas id="snake" width="550"
  height="550"></canvas>
```

Setting up the variables

Using replit, examine the JS file script.js in the left panel and try to interpret as much as you can. A detailed explanation follows.

```
//DEFINITIONS SECTION
const cvs = document.getElementById("snake");
const ctx = cvs.getContext("2d");
var nextTime = 0;
var duration = 400;   //animate every 400ms
// basic grid size for snake pit in pixels
const box = 25;
// path to ladybird
let foodImg = new Image();
// local image
foodImg.src = "images/ladybird.png";
// audio files
let dead = new Audio();
let eat = new Audio();
let up = new Audio();
let right = new Audio();
let left = new Audio();
let down = new Audio();
dead.src = "audio/dead.mp3";
eat.src = "audio/eat.mp3";
up.src = "audio/up.mp3";
right.src = "audio/right.mp3";
left.src = "audio/left.mp3";
down.src = "audio/down.mp3";
// a variable to hold directions
let d = " ";
```

For clarity we divided our JavaScript code into these four sections:

- DEFINITIONS
- FUNCTIONS
- GAME LOOP
- MAIN PROGRAM

We pre-entered some less important JavaScript code to define variables. Notice that there are lines commencing with `//`. These are comments and are not part of the program. Adding comments as you code makes it easier for others, and you, to interpret your code.

This pre-prepared code simply sets up the variables we will need.

We use JavaScript to draw the snake pit square, a snake and a ladybird (food) on the canvas.

We identify the canvas element using the `id "snake"` and define it as a constant called `cvs` (we could choose any name) and tell the browser to make the canvas 2D canvas methods available.

The snake pit board is a square 500 x 500. Imagine a grid of 25 pixels overlayed on it. We will need to refer to locations of snake and food using only multiples this grid size. We store the grid size as a constant called `"box"`. We are now ready to code the snake game using JavaScript!

Positioning the snake to start

6 The snake is an array of objects, each representing the coordinates of the snake segments. This array will lengthen as the snake eats ladybirds, as the snake grows longer. The snake starts with just one segment: a dark green head positioned at the centre of the snake pit. `This` first term `snake[0]` of the snake array uses two `key:value` terms, one for the `x` and one for the `y` coordinate. We could try to set x and y to the centre of the snake pit but we need the snake and food to always align with a grid space. Examine the following code:

```
let snake = [];
snake[0] = {
     x: Math.floor(250/box) * box,
     y: Math.floor(250/box) * box
};
```

By dividing the middle of the snake pit board by the grid size we get the number of 'boxes' to the centre but this may not be an integer. `Math.floor (250/box)` will give the nearest lower integer. We multiply this by the box size. This means all snake coordinates will now be multiples of the box dimension, and snake segments and food will always line up on the grid. (Clever, eh?!)

Copy the above code into the first empty section of the JS file under:

```
// snake code
```

Add listener function

7 To respond to button clicks, we need a JS handler to 'listen' for them. This listener will send the clicked button's value to a function. Examine the structure of a listener function, illustrated in Figure 16.34. This function waits for an event named `"keydown"` and when it occurs, calls its handler function, `direction()`.

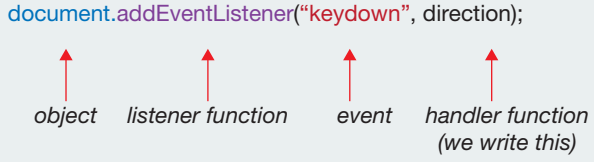


Figure 16.34 Structure of a JavaScript listener function.

Add the following listener as the last line of your Definitions under this heading `// listener to respond to keypresses`

```
document.addEventListener("keydown",
  direction);
```

Write the functions

We now set up the functions needed for the game.

Food function

8 We need to be able to randomly position food using an image of a ladybird using x and y coordinates that align with the grid. We need these two maths functions:

- `Math.random()` produces random decimal numbers in the range 0 up to but not including 1.
- `Math.floor()` returns the largest integer less than or equal to a given number.

Knowing how we positioned the snake in the last step, can you explain how the code below works?

```
function newFood() {
    food = {
        x: Math.floor(Math.random() * 500/box) * box,
        y: Math.floor(Math.random() * 500/box) * box
    };
    return food;
}
```

Copy this function into your JS file under the heading:

```
// draw food
```

We will keep food away from the snake pit edges. Can you explain how the code does this?

Snake pit function

9 This code draws our snake pit by drawing two rectangles, one `(fillRect)` is light blue and the other `(strokeRect)` is just a purple border.

```
function drawPit() {
    ctx.fillStyle = "LightSteelBlue";
    ctx.fillRect(0, 0, 500, 500);
    ctx.strokeStyle = "purple";
    ctx.lineWidth = 6;
    ctx.strokeRect(3, 3, 500, 500);
}
```

Copy this code into your JS file under the heading:

```
// draw snake pit
```

Keypresses direction function

10 Later in the project we control the snake's directions using touch on a mobile device. In this first version we use a keyboard. As soon as a key is pressed the earlier listener function captures it and sends it to this handler function, `direction()`. Each key on a keyboard has its own two digit code as shown in Figure 16.35 and the first line of the function converts it to its two-digit key code which JavaScript recognises.

The function stores this code in a variable "d" and plays an appropriate sound.

We must not allow the snake to reverse onto itself. We do this by making sure the current keypress is *not* opposite to the new one. Note that `!=` means "not equal to".

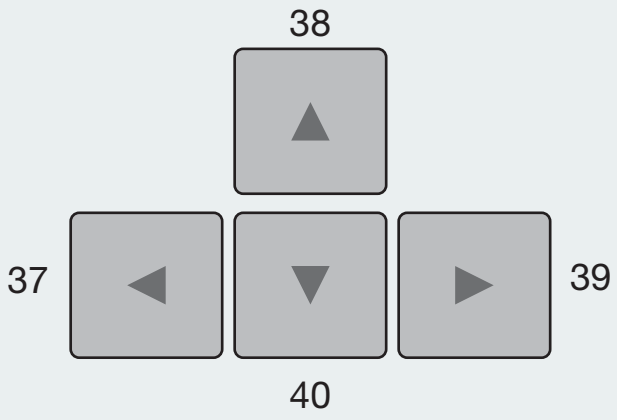


Figure 16.35 Keyboard key codes.

Using the codes for the other keys using Figure 16.35, complete code for the "left" key and remaining keys.

```
// keypresses direction handling
function direction(event) {
    let key = event.keyCode;
    if (key == 37 && d != "RIGHT") {
        d = "LEFT";
        left.play();
    } else if (...) {
        d = "UP";
        ...  }
        else if (...) {
        d = "RIGHT";
        ...
    } else if (...) {
        d = "DOWN";
        ...
    }
}
```

Snake collision function

11 We want collisions of the snake with itself to end the game. This happens when x and y values for the snake's head are the same as for a segment of its body. Our function is sent the snake's head position and the snake array. It checks through the array to see if any body coordinates match its head coordinates. Note that `&&` indicates conditions where both must be true. `i+=1` is just another way to write `i = i + 1`.

Having read the last explanation, can you explain how this code works?

Copy this as your next function at the location indicated in your JavaScript file.

```
// check snake collision with itself
function collision(head, array) {
   for (let i = 0; i < array.length; i+=1) {
      if (head.x == array[i].x && head.y == array[i].y) {
        return true;
      }
   }
   return false;
}
```

Why can't I see anything yet?

12 You may wonder why you can't see the game yet when you press "run". The reason is that objects must be drawn then animated for the snake and food to move. We haven't yet written this game loop.

The game loop function

Game loops keep repeating and drive the game. In our game this function will repeat every 400 ms and during each loop it will refresh three items: the snake pit, the snake position and the food position. Note that the game loop function, like all functions in JS must be enclosed in a matching pair of braces.

Copy the following code into your script. Note that this is just the first part of this `loop` function. Examine its logic.

```
//GAME LOOP
function loop(currentTime) {
   if (currentTime < nextTime) {
      requestAnimationFrame(loop);
      return;
   }
   // set nextTime
   nextTime = currentTime + duration;
   drawPit();
}
```

INFOBIT

You can change the value in the code for duration to speed up or slow down the game.

Running the game

13 Making something appear is the easy part! This code creates food then calls the `loop` function every 400 ms. Copy this code into your script at the very end, under this heading:

```
// MAIN PROGRAM
newFood();
requestAnimationFrame(loop);
```

Run the app and you should now (at last!) see a blue snake pit – but still no snake! Time to draw the snake…

Drawing the objects

You learned earlier how to draw on the HTML canvas so the following code should make sense to you.

14 Add the code below immediately before the final closing curly brace of the `loop` function in step 12. End this `loop` function with a curly brace.

```
//draw snake
for (let i = 0; i < snake.length; i+=1) {
   if (i == 0) {
      ctx.fillStyle = "green";
   } else {
      ctx.fillStyle = "DarkSeaGreen";
   }
   ctx.fillRect(snake[i].x, snake[i].y, box, box);
   ctx.lineWidth = 1;
   ctx.strokeStyle = "red";
   ctx.strokeRect(snake[i].x, snake[i].y, box, box);
}
```

15 Add the following code to insert the image of the food ladybird *to the very end* of the `loop` immediately before the final curly bracket that ends the loop.

```
// draw food
ctx.drawImage(foodImg, food.x, food.y);
```

16 Run the code and you should see the snake in the snake pit.

Handling snake movement

17 We do not yet have snake movement! Examine each line in the following to see if you can state what each line does. A detailed explanation follows.

```
// store existing head position
    let snakeX = snake[0].x;
    let snakeY = snake[0].y;
// determine new head position
    if (d == "LEFT") snakeX -= box;
    if (d == "UP") snakeY -= box;
    if (d == "RIGHT") snakeX += box;
    if (d == "DOWN") snakeY += box;
```

We are storing the current head position using `snakeX` and `snakeY` as the first term (at index 0) of the array. We increment these by the grid width and the head will move one grid square either up, down, left or right, giving us new coordinates `snakeX` and `snakeY` for the head. The snake won't move because we have not yet added this new head to the array. We need to handle collisions first.

Add the above *code to the very end* of the `loop` function immediately before its final curly brace.

Food and snake collision

18 A collision with food means the snake eats the food and will have a segment added and will become longer. Explain each line in the following.

```
// check collision create new food
if (snakeX == food.x && snakeY == food.y) {
// don't remove tail as new head added
    eat.play();
    newFood();
} else {
    // remove tail
    snake.pop();
}
// store new position of head
let newHead = {
    x: snakeX,
    y: snakeY
};
```

Add this code *to the very end* of the `loop` function immediately before its final curly brace.

Snake collides with edges or with itself

19 The game ends if the snake collides with the edges of the snake pit or with itself. Before adding the following code explain each line. An explanation follows. Note: `||` means OR.

```
// check collision with edges
if (
    snakeX < 0 ||
    snakeX >= 500||
    snakeY < 0 ||
    snakeY >= 500 ||
    collision(newHead, snake)
) {
    dead.play();
    return
}
// no collisions at all so insert new head
snake.unshift(newHead);
requestAnimationFrame(loop);
```

If the snake hasn't collided with either edges or itself we insert a new head at the front of snake array. JS arrays use `unshift` to push new items into the front of arrays. We keep the game loop going by telling it first to start up then we make it call itself every 400 ms!

Add the above code *at the very end* of the `loop` function but immediately *before* the final curly bracket that ends the loop. This loop's single closing curly brace should now appear immediately before the two lines which form the main program and also start the game running.

Your code is complete! Run your program.

Testing and evaluating

Test your game thoroughly. Use the functional test case table you prepared in step 3. Test all possible events. If the program is not working and you get stuck, compare your code with our completed version in the *Snake_basic* folder provided in the Nelson MindTap resources.

You can download your entire project from *replit* using the three vertical dots beside *Files* in the left panel (see Figure 16.36).

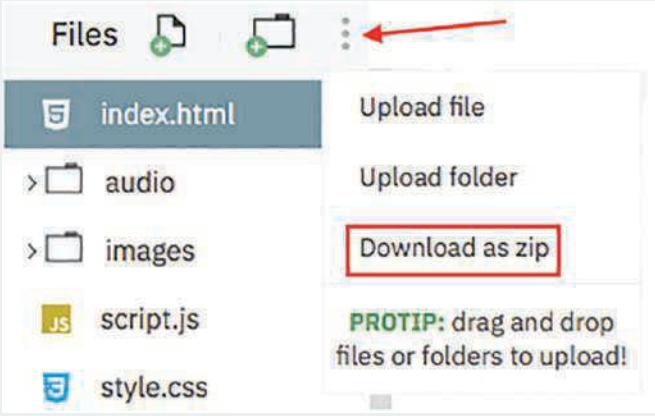


Figure 16.36 Downloading your project to your computer.

Debugging by checking variables as you code

20 *Replit* provides a "console" (available via the wrench icon top right) where you can view variable values and print messages using the function `console.log()`

Try this out by placing the following code wherever you would like to watch the changing value of a variable as your game code runs.

```
console.log(variables here)
```

Try:

```
console.log(d, snakeX, snakeY)
```

Using this expression you can check segments as they are added to the snake array.

Improving the user experience: Screen wrap

21 A big improvement to the UX would be to allow the snake to wrap around the board and eliminate it dying every time it hits the edges:

- remove the following edge collision code:

```
snakeX < 0 ||
snakeX >= 500 ||
snakeY < 0 ||
snakeY >= 500 ||
```

- complete and add the following immediately before this block of code for drawing the snake:

```
let newHead = {
    x: snakeX,
    y: snakeY
};
```

Note: We need to adjust it to one box width less than the board or it will travel off the edge. The infobit suggests an even better approach, however it needs a bit more maths.

```
//handle screen wrap
if (snakeX < 0) {
    snakeX = 500-box;
}
if (snakeX > 500-box) {
    complete this line
}
if (snakeY < 0) {
    complete this line
}
if (snakeY > 500-box) {
    complete this line
}
```

INFOBIT

For maths gurus who like a challenge!

You could replace all the above by two lines of code using the % (modulus) function giving the remainder when one number is divided by another. Can you explain what it is doing? Test it out.

```
snakeX = snakeX % 500;
snakeY = snakeY % 500
```

However, this works only when the snake leaves the screen to the right or down. The following version however works perfectly! Can you explain why?

```
snakeX = (snakeX + 500) % 500;
snakeY = (snakeY + 500) % 500;
```

What's next?

If you followed the steps accurately in this project, you will have a working snake game. You learned how important accuracy is when coding and also a lot about using JavaScript and the HTML canvas. In a number of places we left you on your own to try to figure out missing code.

In the next section we focus on UI and UX for this project. You will create buttons to play the game and then adapt it for a phone or tablet touchscreen.

Modifying UI for touch

22 Tablets and mobile phones typically use touchscreens for playing games; however, our current version of the snake game must be played using a keyboard. Consider the UI changes required to convert the snake game for a mobile phone.

One approach would be to create on-screen buttons for players to press to select changes in direction for the snake. Another could be for players to touch the edges of the snake pit to redirect the snake. We have provided both versions in the Nelson MindTap resources.

Here we will develop an on-screen button version (Figure 16.37).

Add on-screen buttons

In the code below we added a group of four buttons called U, D, R and L to the HTML code, however, we have completed code for the U button only.

Complete the code for the other three buttons. Note each button has these features:

- they belong to the class `'buttons'` which will be defined in your CSS file.
- each button has a different background colour using RGB for colour and a fourth value of 0.3 to set its transparency
- each button has a letter for its value to identify it and has that value as its label
- each button is instructed to respond to a click by sending its value to a JS `direction` function.

```
<link rel="stylesheet" href="style.css">
<div style="text-align:center;">
<button
   class ="buttons"
   value = "U"
   onclick ="direction(this.value)"
   button style = "background-color:rgb(0, 0, 255, 0.3)"
   >U
</button>
<br>
<button
   complete this code
   >L
</button>
<button
   complete this code
   >R
</button>
<br>
<button
   complete this code
   >D
</button>
<br>
```

Add this code to your HTML file immediately below the canvas element. Run the code to see the buttons but as we haven't written JavaScript code yet to handle the buttons they are inactive.

The first line of code above provides the link to the CSS stylesheet.

The second line centres the buttons on the page.

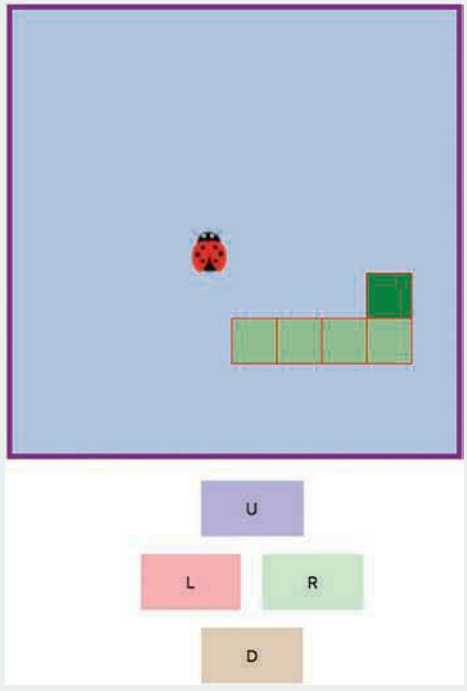
The rest of the code creates the buttons.

Further UI improvements

23 The CSS stylesheet is perfect for such adjustments of appearance. We will take this opportunity to centre our canvas for whatever sized screen the game is being played and resize our buttons.

Add this code to your separate *style.css* file:

```
canvas{
   margin:  auto;
   display: block;
   }
.buttons {
   border: none;
   padding: 20px 50px;
   font-size: 16px;
   margin:10px;
   }
```



Source: https://snakefinal.comptech.repl.co/

Figure 16.37 The snake game with touch and button control added.

Weblink
Snake game with scores and pause

Handling button clicks

24 Finally we need to write a handler function to tell the program what to do at each button press. It is sent the clicked button's value as its parameter, as instructed by the button's HTML code. This is very similar of course to our existing `direction` function which handled keyboard keys and now replaces it, however we no longer need to get a keycode to identify keys.

Replace the previous `direction()` function with this one and complete the missing lines.

```
function direction(whichbutton){
    if (whichbutton == "L" && d != "RIGHT")
      {
    left.play();
    d = "LEFT";
    }
complete the code for the other three
  buttons by inspecting your existing
  version of this function made for
  physical keys
}
```

Test the app on a mobile device

25 Using the automatically generated URL in *replit* (top right), open your own web page on a mobile device and run the program. If possible try different browsers. Can you suggest any further UI improvements?

Try changing the game speed and run it again.

Is the snake too small to see properly? Try changing the grid `box` size to `50` in the JS code.

The lady bird image is still small. This is easy to change! Images can be resized easily in JS by adding its dimensions.

In the following the first variable is the image, the next two are the coordinates for where it is drawn and the final two are its dimensions. Change your line of code for drawing food to:

```
// draw food
ctx.drawImage(foodImg, food.x, food.y,
  box, box);
```

Weblinks
snake_ onscreen buttons

snake_mobile

Game of Life

Completed mobile version

The early Nokia phone used physical buttons to play Snake. The keyboard keys were used for up, down, left and right. A modern mobile phone or tablet has no keyboard and uses touch.

Your game input can be modified to do this. The canvas border around the snake pit can be used to direct the snake by touching.

Can you work out how to do this using JavaScript?

You can play our button press version and touch version on your mobile device by visiting the two weblinks, *snake_onscreenbuttons* and *snake_mobile* listed on this page.

The teacher resources also contain solutions for both the button press and mobile touch versions.

These versions can also be played on your device at *snake_touch* and *snake_touchedge*

Further improvements

Replay game

Another improvement (which you may think would have been helpful at the start!) is to allow the game to be reset and replayed after the snake dies. We leave this to you, although this is demonstrated in the final version in the Nelson MindTap resources.

Other ideas

Can you think of ways to improve the game? There is a 'bug' at present involving a bug. The ladybird may be drawn over the top of the snake. Can you modify the code to avoid this?

Could you gain a faster response in the touch version by moving the code that responds to touch to another position in the main game loop?

Can you think of any additional features for the game? Ask your friends to try the game and suggest improvements or features.

Consider these:

- display a score
- add a pause button
- maintain a high score
- allow players to set speed
- have the snake speed increase with its length
- add more than one ladybird to the board
- add different food types worth more points
- add obstacles such as walls
- make the game your own by changing the 'world' or metaphor for the game, in the same way as *Breakout* was adapted to become *Space Invaders*.

Play the demo final version

You can play a final version featuring a pause feature (press 'p' key), screen wrap, game score, high score and faster speed using the weblink and inspect the code by having your teacher provide you with the *Snake_final* folder in the Nelson MindTap teacher resources.

An advanced challenge

1. What would happen if the snake became so long it completely filled the board? Could this be a bug?
2. Build simple AI into your game. If a player's snake gets to a given length, you could make the food appear near the snake's tail each time, or near a wall, to make it more challenging to reach.
3. You now know enough JavaScript to build the *Game of Life* (see weblink) or noughts-and-crosses. Up for a challenge?
4. Can you think of other ideas?

9780170442787