

O-Level COMPUTING

Third version

1
2
3
4
5
6
7
8
9
10



CURRICULUM PLANNING & DEVELOPMENT DIVISION
MINISTRY OF EDUCATION, SINGAPORE

© 2017 Curriculum Planning and Development Division

Ministry of Education, Singapore
MOE Building
1 North Buona Vista Drive
Singapore 138675

First published 2017
Second version 2018
Third version 2021

All rights reserved

Preface

This textbook is written in alignment with the Ministry of Education's O-Level Computing syllabus.

The concepts are presented through the use of real-life examples and authentic contexts with the aim of equipping students with the necessary knowledge in the subject, as well as opportunities for the development of computational thinking skills, and the ethical use and handling of data.

Chapter Opener

- Full-page lead-in to the chapter with a defined problem context
- Framed as a question and supported with characters or illustrations to stimulate thinking

1 Module 2: Systems and Communications

How Do Computers Work?

After visiting several computer retail stores and collects some flyers on computers for comparison. One of them is shown below. What do you think the specifications on the flyer mean?

VR7 Gaming PC

- High-end GPU (e.g. NVIDIA GeForce RTX 3080, 16 GB Cache, up to 4.8 GHz)
- High-end CPU (e.g. Core i9)
- 16 GB Dual Channel DDR4 XMP at 2933 MHz
- 1 TB NVMe SSD
- RJ-45 2.5 GbE Gigabit Ethernet NIC
- Wi-Fi 6 802.11ax Wireless and Bluetooth 5.1

CM3 Multimedia Laptop

- Intel Range CPU (6-MB Cache, 3.6 GHz)
- 1 TB 5400 RPM 2.5-inch SATA Hard Drive
- Integrated Graphics
- 14.1-inch FHD LED-Display
- 8 GB DDR4 2666 MHz RAM
- 802.11ac WiFi with Bluetooth 4.1
- Optical Disk Drive

More EMOD PCs

- 3.6 GHz Quad-Core low-end CPU (6-MB shared L3 Cache)
- 1 TB 5400 RPM 2.5-inch SATA Hard Drive
- 4 GB DDR4 2666 MHz SO-DIMM memory
- 256 GB PCIe-based SSD
- Integrated graphics
- 802.11ac WiFi with Bluetooth 5.0

Computers have changed the way we work, live and play. Besides desktops and laptops, today you can find computers in cars, mobile phones and even watches. To make the most of the computers around us, we need to understand how they are designed to function.

This chapter will introduce the major components of computer architecture and their functions. You will also learn about how instructions are executed during program operation and how the various components work together.

How Do I Use Flowcharts and Pseudo-Code?

By the end of this chapter, you should be able to:

- Produce an algorithm, in pseudo-code or diagrammatically, to solve a problem.
- Perform a dry run of a set of steps to determine its purpose and/or output.
- Produce trace tables to show values of variables at each stage in a set of steps.
- Locate logic errors in an algorithm, and correct or modify an algorithm to remove the errors or changes in task specification.

3.1 Variables

We often give names to values when defining inputs and outputs or generalising problems and solutions. For example, in the problem shown in Table 3.1, x and y are the named values.

Table 3.1 Defining the addition problem

Input	Output
<ul style="list-style-type: none">• x: a positive whole number• y: a positive whole number	<ul style="list-style-type: none">• The sum of x and y

In flowcharts and pseudo-code, these named values are called **variables**. A variable is a box that is labelled with its name. A variable can store different types of data just as a box can store different kinds of items (see Figure 3.1). Data can come in many forms such as letters, lists, numbers and words.

Variable (flowchart and pseudo-code)
Named storage space for a value that can be changed while the algorithm is running

Figure 3.1 Just like a box can store different kinds of items, a variable can store different types of data

In summary, the syntax for a replacement field is as follows:

```
Syntax 5.3 Replacement field
[1]
[index]
```

Did you know?

The `str.format()` method and replacement fields can also perform more complex tasks such as lengthening strings to a specified width and restricting the number of decimal places used when converting a float to a str. Use of the `str.format()` method to perform such tasks will not be covered in this textbook.

Quick Check 5.3

- Fill in the blank in the program below so that it correctly outputs whether the first character of name is an upper-case letter.

```
name = input("Enter name: ")
print([ ])
```

- Complete the program below by filling in the blank with one or more lines of code so that it correctly outputs whether title contains upper-case letters and spaces only (i.e., no lower-case letters, symbols or digits).

```
title = input("Enter title: ")
```

Figure 5.25 Syntax for replacement fields
Figure 5.26 Program template
Figure 5.27 Program template

173

Review Questions

- Provided at the end of the chapter to reinforce and consolidate the concepts learnt in the chapter through structured questions

Did you know?

- Provides interesting facts relevant to the topic

Quick Check

- Provided at the end of each section to reinforce concepts through short-answer questions

Review Questions

1. A program is needed to solve the following word reversal problem:

Table 6.34 Defining the word reversal problem

Input	Output
<ul style="list-style-type: none"> input_text: phrase to reverse, should contain only letters and spaces 	<ul style="list-style-type: none"> Gives phrase with the order of the letters of each word reversed but the words themselves still in order, separated by one space between words

The draft program in Figure 6.42 is written for this purpose.

a) The main algorithm for the program consists of three parts. Read through the code and state what each of the following parts of the program are intended to do:

- Lines 13 to 25
- Lines 27 to 35
- Lines 37 to 42

b) When this program is run on the following test case, the actual output does not match the expected output:

Table 6.35 Running word_reversal_draft.py on a test case

→ Input	Expected Output →
• input_text: "Meetings on etiquette"	sgnitteM no etiquette
Running word_reversal_draft.py on line 1 case →	Result
Enter input text: Meetings on etiquette	Failed
Measnigtae on etiquette	

c) State the kind of condition (normal, boundary or error) covered by this test case.

d) State the kind of error (syntax, run-time or logic) that has occurred and why.

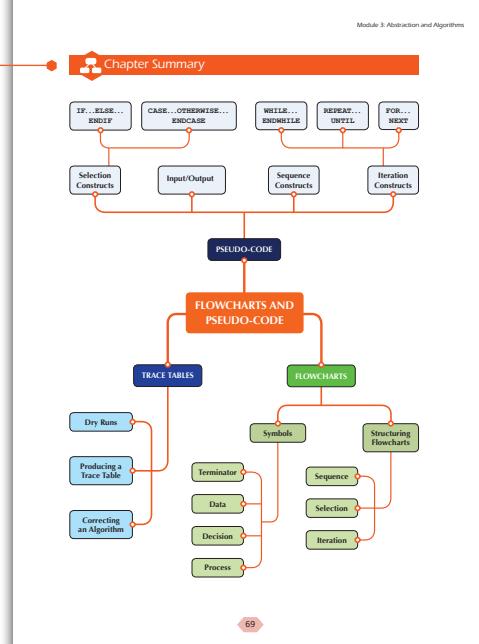
e) Find out where the error is located by disabling lines of code and testing the program in chunks or parts. State the line on which the error is located and explain why the line is incorrect.

f) Rewrite the program to fix the error so that it passes the test case.

243

Chapter Summary

- Provides an overview of the key concepts in the chapter as well as links between them using a concept map



ANSWERS

Chapter 1 How Do Computers Work?

Quick Check 1.1

- True. However, “kilobyte” is still often confused with “kilobit”, which is 1,024 bytes.
- a) $2,017 \text{ kB} = 2,017 \times (10^3 \text{ bytes}) = 2,017,000 \text{ bytes}$
 b) $19 \text{ GB} = 19 \times (10^9 \text{ bytes}) = 19,000,000,000 \text{ bytes}$
 c) $65 \text{ MB} = 65 \times (10^6 \text{ bytes}) = 65,000,000 \text{ bytes}$

Quick Check 1.3

- The main differences between RAM and ROM are summarised below:

RAM	ROM
Read and write: stored data can be easily changed	Read only: stored data cannot be easily changed
Volatile: loses data once power supply to the computer is interrupted	Non-volatile: retains data regardless of whether the power supply is switched on or off
Purpose: to store data and instructions temporarily so that they can be quickly accessed by the processor when needed	Purpose: to store data and instructions that would be needed for a computer to start up or before data can be loaded into RAM

Quick Check 1.4

- True. Memory addresses are just numbers so they can also be treated as data.

Quick Check 1.6

- a) A hard disk. It is the only device with storage capacity available in terabytes.
 b) A memory card. It is the only device small and flat enough to fit in a wallet. A memory card would also be less vulnerable to mechanical shocks that could occur in a wallet.

A-1

Answers

- Provides answers and explanations to Quick Check and Review Questions

Contents

PREFACE

Chapter 1	How Do Computers Work?	1
1.1	Computer Architecture	2
1.2	Processor	5
1.3	Memory	6
1.4	Data and Address Buses	8
1.5	Input and Output	10
1.6	Secondary Storage	11
Chapter 2	How Can Algorithms Be Used to Solve Problems?	17
2.1	Understanding Algorithms	18
2.2	Understanding Problems	22
2.3	Problem-Solving Techniques	25
Chapter 3	How Do I Use Flowcharts and Pseudo-Code?	37
3.1	Variables	38
3.2	Flowcharts	40
3.3	Trace Tables	51
3.4	Pseudo-Code	57
Chapter 4	How Do I Write Programs?	69
4.1	Flowcharts and Programs	70
4.2	Programming Languages	72
4.3	Compilers and Interpreters	74
4.4	Variables and Constants	80
4.5	Data Types	89
4.6	Functions and Operators	98
4.7	Control Flow Statements	124
Chapter 5	How Can Programs Be Used to Solve Problems?	147
5.1	Stages in Developing a Program	148
5.2	Mathematical Functions	158
5.3	String Functions	166
5.4	Common Problems and Solutions	175
Chapter 6	How Do I Ensure That a Program Works as Intended?	189
6.1	Validating Input Data	190
6.2	Designing Test Cases	201
6.3	Debugging Program Errors	218

Chapter 7	How Can I Be a Safe and Responsible Computer User?	239
7.1	Data Corruption and Loss	240
7.2	Unauthorised Access	244
7.3	Threats to Privacy and Security	254
7.4	Intellectual Property	262
7.5	Impact of Technology on Everyday Life	267
Chapter 8	How are Number Systems Applied in Real Life?	277
8.1	The Denary Number System	278
8.2	The Binary Number System	280
8.3	The Hexadecimal Number System	286
8.4	Applications of Binary and Hexadecimal Number Systems	297
Chapter 9	How Do Logic Circuits Make Decisions?	311
9.1	Boolean Logic	312
9.2	Truth Tables	313
9.3	Logic Gates	315
9.4	Logic Circuit Diagrams	319
9.5	Boolean Statements	322
9.6	Applications of Logic Circuits	331
Chapter 10	How are Spreadsheets Used to Process and Analyse Data?	337
10.1	Understanding Spreadsheets	338
10.2	Data Management	352
10.3	Functions	356
10.4	What-If Analysis	378
10.5	Conditional Formatting	382
Chapter 11	How Do I Create a Simple Network?	395
11.1	Computer Network	396
11.2	Types of Computer Networks	397
11.3	Factors Affecting Choice of Transmission Medium	404
11.4	Identifiers	405
11.5	Network Hardware and Their Functions	407
11.6	Network Topologies	412
11.7	Error-Checking Methods in Data Transmission	416

How Do Computers Work?

Alex visits several computer retail stores and collects some flyers on computers for comparison. One of them is shown below. What do you think the specifications on the flyer mean?

VR7 Gaming PC

- High-End CPU (8-Core, 16 MB Cache, up to 4.8 GHz)
- High-End GPU 8 GB GDDR6
- 16 GB Dual Channel DDR4 XMP at 2933 MHz
- 1 TB M.2 PCIe NVMe SSD
- RJ-45 2.5 Gigabit Ethernet NIC
- Wi-Fi 6 802.11ax Wireless and Bluetooth 5.1

CMD Multimedia Laptop

- Mid-Range CPU (6 MB Cache, 3.6 GHz)
- 1 TB 5400 RPM 2.5-inch SATA Hard Drive
- Integrated Graphics
- 15.6-inch FHD LED-Display
- 8 GB DDR4 2666 MHz RAM
- 802.11ac with Bluetooth 4.1
- Optical Disk Drive



Mini EMDX PC

- 3.6 GHz Quad-Core Low-End CPU 6 MB shared L3 Cache
- 8 GB of 2666 MHz DDR4 SO-DIMM memory
- 256 GB PCIe-based SSD
- Integrated graphics
- 802.11ac Wi-Fi with Bluetooth 5.0

Computers have changed the way we work, live and play. Besides desktops and laptops, today you can find computers in cars, mobile phones and even watches. To make the most of the computers around us, we need to understand how they are designed to function.

This chapter will introduce the major components of computer architecture and their functions. You will also learn about how instructions are executed during program operation and how the various components work together.



By the end of this chapter, you should be able to:

- Describe basic computer architecture with reference to:
 - Computer processors
 - Memory
 - Data and address buses
 - Input (e.g., data and instructions) and output (e.g., intermediate and final results of processing)
 - Storage media

1.1 Computer Architecture

A **computer** (or computer system) is a device that can receive **data**, process the data according to a set of instructions and produce the processed data as a result. The **computer architecture** used describes how a computer is designed and built to function, including how its parts are designed, organised and connected.

Did you know?

There are many ways to build a computer. For instance, while most computers today run on electricity, early computers did not use electricity at all. Instead, they worked based on mechanisms such as cranks, gears and levers. These mechanical computers thus used a completely different computer architecture from the computers we are familiar with today.

1.1.1 Units of Data

Since a computer has to receive, process and produce data, one of the most basic design choices for a computer is how the data is represented.

Modern computers evolved from machines designed for doing calculations, so for most computers today, data is represented as numbers. The smallest unit of data in a computer is a **bit**, or binary digit, which can take on the value of either 0 or 1. A computer stores and processes all data using binary numbers that consist of these digits.

Key Terms

Computer

Device that can receive data, process the data according to a set of instructions and produce the processed data as a result

Computer architecture

How a computer is designed and built to function, including how the parts of the computer are designed, organised and connected

Data

Information that is used in a computer program (singular: datum)

A single bit would be too simple to represent complex data, so we usually talk about data being represented as bytes instead. A **byte** is a unit of data made up of eight bits.

Table 1.1 summarises some units of measure of data in order of increasing size.

▼ **Table 1.1** Units of measure of data

Name of unit	Symbol	Size in bytes
kilobyte	kB	1,000
kibibyte	KiB	1,024
megabyte	MB	$1,000^2 = 1,000,000$
mebibyte	MiB	$1,024^2 = 1,048,576$
gigabyte	GB	$1,000^3 = 1,000,000,000$
gibibyte	GiB	$1,024^3 = 1,073,741,824$
terabyte	TB	$1,000^4 = 1,000,000,000,000$
tebibyte	TiB	$1,024^4 = 1,099,511,627,776$

Key Terms

Bit

Binary digit which can take on the value of either 0 or 1

Byte

Binary number made up of eight bits

Did you know?

In the past, units of data such as “kilobyte”, “megabyte” and “gigabyte” were based on powers of 1,024 (or 2^{10}) instead of the standard powers of 1,000 used in the International System of Units (SI). However, since 1998, multiple standards organisations have agreed that SI prefixes (i.e., “kilo”, “mega”, “giga”, etc.) should follow their standard meanings while new binary prefixes (i.e., “kibi”, “mebi”, “gibi”, etc.) would be used to represent powers of 1,024.

Despite this change, you may still encounter the use of “kilobyte” and “megabyte” to represent 1,024 bytes and $1,024^2$ bytes respectively, instead of the correct units “kibibyte” and “mebibyte”.

1.1.2 Inside the Computer

Most computer parts can be organised into the following roles:

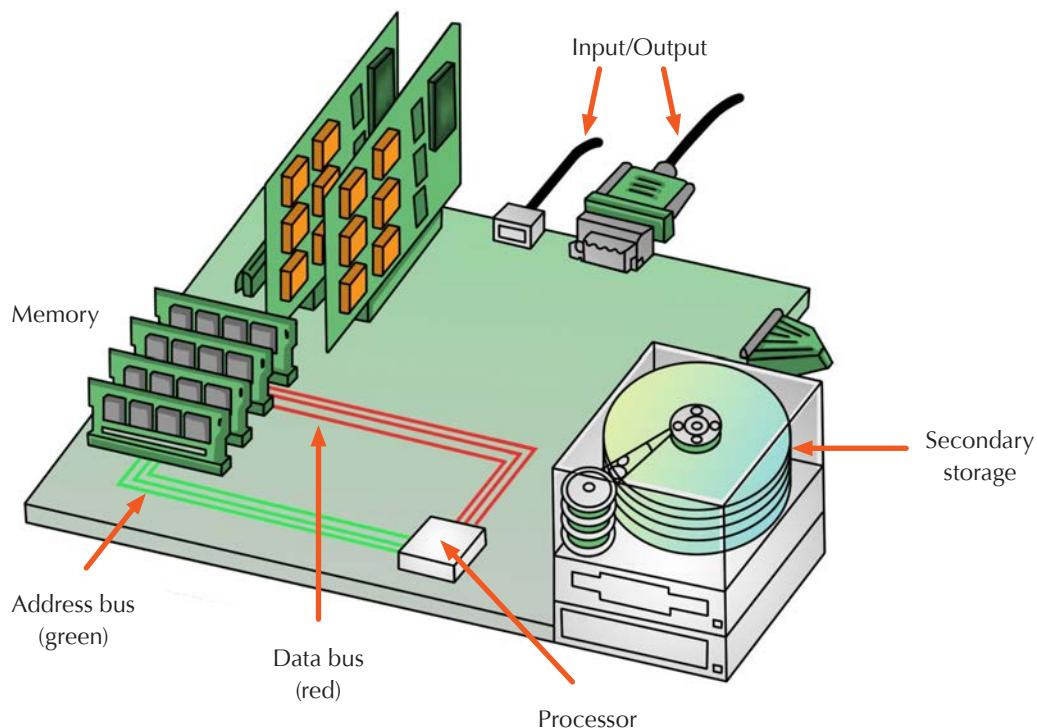
▼ **Table 1.2** Inside the computer

Role	Description
Processor	Processes data and follows instructions; consists of an arithmetic logic unit and a control unit
Memory	Stores data, instructions and the results of processing for immediate use
Secondary storage	Stores large amounts of data that will not be lost when power supply is interrupted

▼ Table 1.2 Inside the computer (continued)

Role	Description
Data bus	Tранспортирует данные между памятью и процессором; двунаправленный
Address bus	Транспортирует требуемую память местоположение от процессора к памяти; однодirectional
Input	Данные или инструкции, что компьютер получает
Output	Интермедиат или конечные результаты, производимые компьютером; обычно в форме обработанных данных

Figure 1.1 shows where the parts performing these roles may be found inside the case of a desktop computer.



▲ Figure 1.1 Components in a desktop computer case

The sections that follow will elaborate on each of these parts.



Quick Check 1.1

1. 1 kilobyte is the same size as 1,000 bytes. True or false?

2. Convert the following amounts of data into bytes:
 - a) 2,017 kB
 - b) 19 GB
 - c) 65 MB

1.2 Processor

You have already learnt that computers need to process data and follow instructions. The computer part that does this is called the **processor** or **central processing unit (CPU)**. It is usually a complex circuit made of many components that is compressed into a square or rectangular package. You will learn more about the components that make up this circuit in Chapter 9.

The two main functions of a processor are usually handled by separate parts:

1. The **arithmetic logic unit (ALU)** processes data by performing basic mathematical and logical operations such as addition, subtraction, multiplication and division. This unit performs the actual work of performing calculations and transforming data.
2. The **control unit** follows instructions and decides when data should be stored, received or transmitted by different parts of the computer (including the ALU). This unit makes sure that data is transported to where it needs to be, and that it is processed in the correct order.

Key Terms

Arithmetic logic unit (ALU)

Part of the processor that processes data by performing basic mathematical and logical operations

Central processing unit (or processor)

Part of the computer that processes data and follows instructions

Control unit

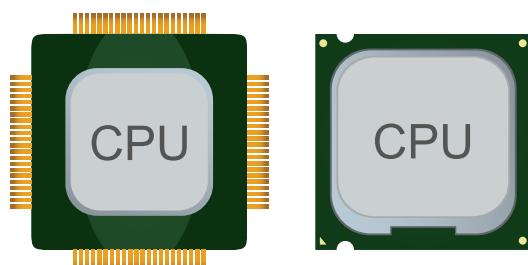
Part of the processor that follows instructions and decides when data should be stored, received or transmitted by different parts of the computer

Execute

To follow or perform an instruction

Software

Set of instructions to perform specific tasks on a computer



▲ **Figure 1.2** Some common processor packages

When we say that a processor runs or **executes** instructions, it means that the processor follows or performs instructions. A series of instructions is called a program, or simply **software**.

Often, a processor's speed is described by the number of instructions that the processor can perform in one second. For instance, a 1 MHz (megahertz) processor can perform one million instructions per second while a 1 GHz (gigahertz) processor can perform one billion instructions per second. In general, the larger the number and unit, the faster and more powerful the processor.

There are also “multi-core” processors that contain multiple processing units inside a single package. For instance, a “dual-core” package has two such processing units while a “quad-core” package has four. These “multi-core” processors can perform more than one instruction at the same time, and thus are more powerful than “single-core” processors.

1.3 Memory

Whenever a computer needs to store data, it uses **memory**. Physically, memory can exist in many forms, but in general, it is made up of many switches arranged in a fixed order. Each switch can store one bit of data based on whether it is ON or OFF (i.e., ON = 1, OFF = 0).

Since a byte consists of eight bits, we usually look at eight consecutive switches at a time, as shown in Figure 1.3.

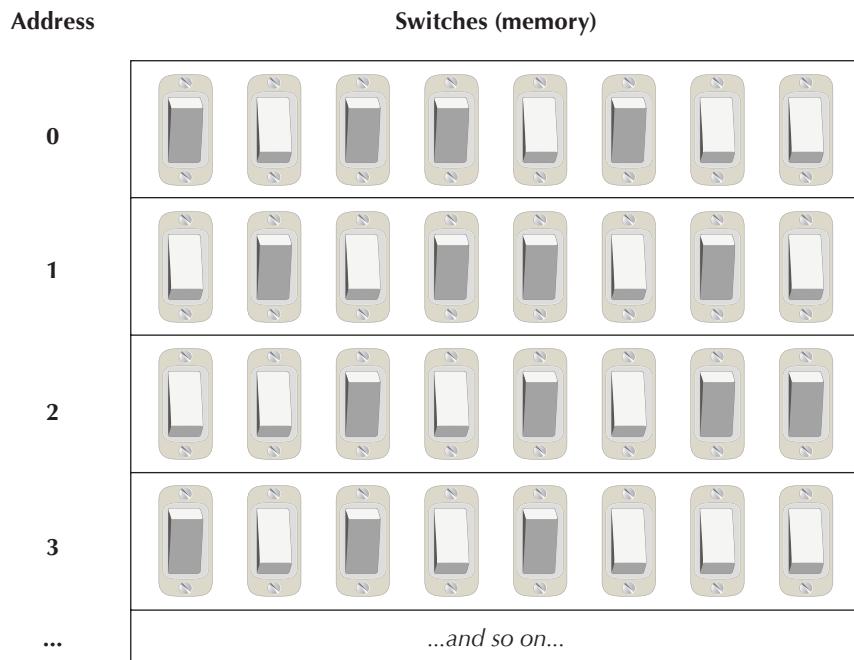
Key Terms

Address

Number that is used to locate a byte in memory

Memory

Device that is used to store data for immediate use in a computer



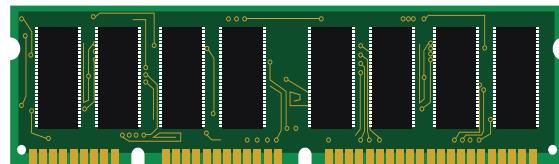
▲ **Figure 1.3** Memory as a collection of switches in a fixed order

In Figure 1.3, we used big physical switches to represent memory, but in most computers, memory is actually made up of very small electronic switches that take up much less space.

Usually, the position of each byte is represented by a number called an **address**. This is a number that allows a computer to quickly find those switches again if it needs to read or change the data that is stored. This is similar to how a unit number is used to identify the location of each residence in an HDB block.

The type of memory employed depends on how the data stored inside is meant to be used:

1. Processor registers are physically located inside the processor. These extremely fast but small data storage spaces are used directly by the ALU and control unit. The number and size of processor registers cannot be changed without changing the processor.
2. Random access memory (RAM) or main memory is where data and instructions are stored temporarily so that they can be quickly accessed by the processor when needed. For instance, when an application is started, its instructions may be loaded into RAM. Data stored on RAM can be easily changed and is also **volatile**, which means that it is lost once the power supply to the computer is interrupted.



▲ Figure 1.4 Example of a RAM card

When we use the word “memory”, we are usually referring to RAM and not the other types of memory. Most desktop computers have RAM located on removable cards so that the amount of RAM in the computer can be upgraded. Other kinds of computers, however, may have only a fixed amount of RAM that cannot be changed.

3. Read-only memory (ROM) is where data and instructions that rarely need to change or would be needed for a computer to start up are stored. Data stored on ROM cannot be easily changed and remains there regardless of whether the power supply is switched on or off. This makes it ideal for storing instructions that are needed before data can be loaded into RAM.
4. Secondary storage is where large amounts of data are stored, such as in a **hard disk** or **hard drive**. To avoid confusion, we typically use the word “storage” for this kind of storage space instead of the word “memory”. Different storage media are used for secondary storage. The different kinds of storage media will be discussed in section 1.6.



Quick Check 1.3

1. Describe the main differences between RAM and ROM.

Key Terms

Hard disk (or hard drive)

Secondary storage where data is stored on rigid rotating disks coated with a magnetic material

Volatile

Lost when the power supply is interrupted

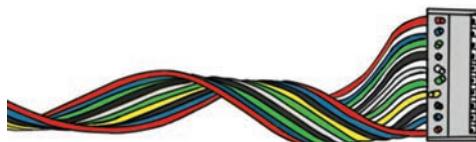
1.4 Data and Address Buses

Besides storing data, computers also use buses to transport data from one part of the computer to another. A **bus** is a collection of wires that serves as a “highway” for data to travel on. It can be made of either physical wires or conductive lines printed on a circuit board.

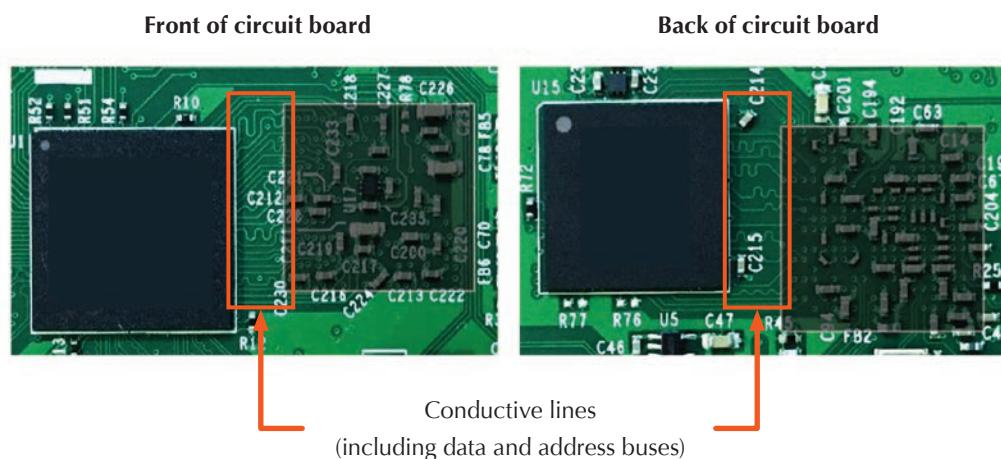
Key Term

Bus

Collection of wires for transporting data from one part of a computer to another



▲ Figure 1.5 Example of a computer bus made of physical wires



▲ Figure 1.6 Example of computer buses made of conductive lines

Two important buses that transport data between the processor and memory parts of a computer are the data bus and the address bus:

1. The **data bus** transports data that is going to be processed to the CPU, as well as data that has already been processed from the CPU. The data bus is **bi-directional** because data can be sent in both directions between the processor and memory.

Key Terms

Bi-directional

Able to work in two directions, to and fro

Data bus

Bus that is used to transport data between memory and the processor

2. The **address bus** specifies memory address information. When the processor reads from or writes to memory (RAM), the relevant address information is provided on the address bus. The address bus is **uni-directional** because address information is always sent in one direction only, that is, from the processor to memory.

Key Terms

Address bus

Bus that is used to specify memory address information

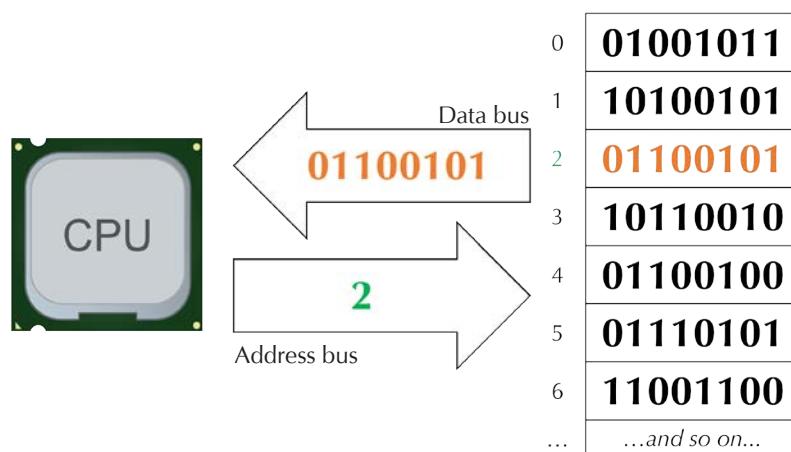
Uni-directional

Able to work in one direction only

Did you know?

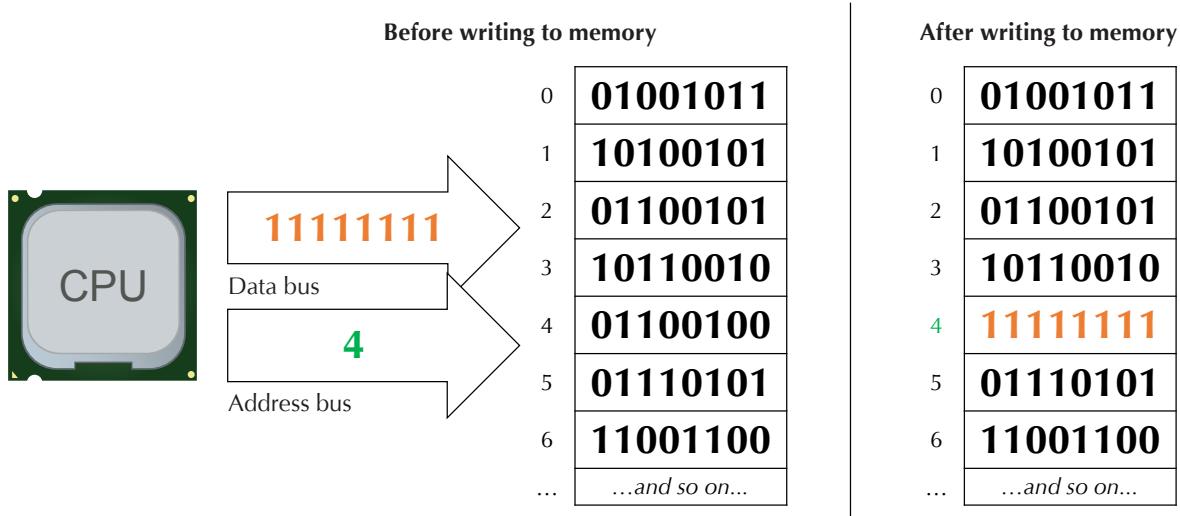
There are many other buses that transport data from one part of the computer to another. Some examples are the control bus that transports instructions from the processor's control unit, and the expansion buses such as those used for connecting external devices.

For instance, to read data from memory, the processor requests the data, and the address bus transports the requested data's address to RAM. A copy of the requested data is then sent from RAM back to the processor via the data bus. This is illustrated in Figure 1.7, which uses bits for data and simple numbers for addresses.



▲ **Figure 1.7** How the data bus and address bus are used to read from memory

When writing data to memory, the processor uses both the data bus and address bus at the same time to transport the data for writing as well as the destination address to RAM. RAM then sets the switches at the destination address according to the data received via the data bus, as illustrated in Figure 1.8.



▲ Figure 1.8 How the data bus and address bus are used to write to memory

As can be seen by the arrows in Figures 1.7 and 1.8, while the address bus is always used to transport information from the processor to memory, the data bus can transport information in either direction.



Quick Check 1.4

- Memory addresses can also be read from or written to memory as data. True or false?

1.5 Input and Output

In computer architecture, **input** refers to data or instructions that the computer receives for processing while **output** refers to any intermediate or final results produced by the computer in the form of processed data. Examples of input are words entered using a keyboard, pictures taken by a digital camera, and movement instructions entered using a mouse. Examples of output are images displayed on a screen, sounds played on a speaker and even sculptures printed using a 3D printer.

Key Terms

Input (computer architecture)

Data or instructions that the computer receives for processing

Output (computer architecture)

Intermediate or final results produced by the computer in the form of processed data

Often, input is received from and output is sent to **hardware** devices:

- An **input device** is a hardware device that allows users to enter data and instructions into a computer. Examples of input devices are keyboards, mice, scanners, touch screens and microphones.
- An **output device** is a hardware device used to display, project or print processed data from a computer so it can be used or understood by people using the computer. Examples of output devices are monitors, speakers and printers.

A computer usually has multiple input and output devices connected to it, as seen in Figure 1.9.

Key Terms

Hardware

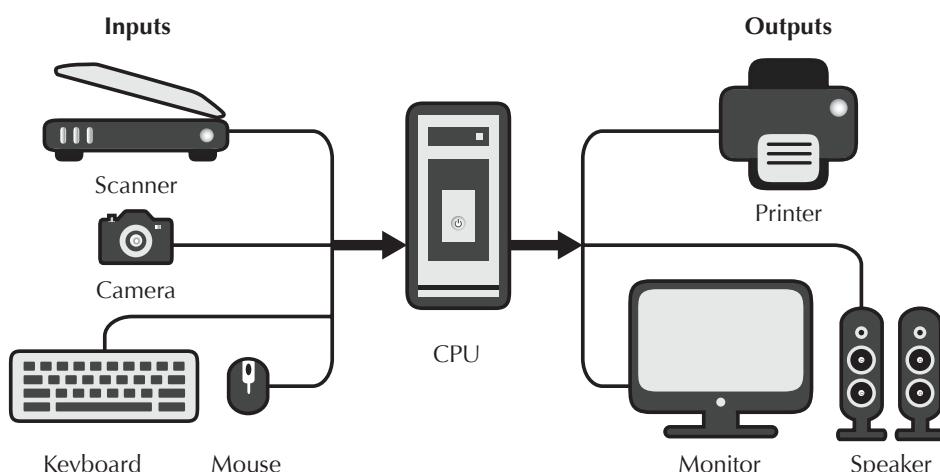
Physical part of a computer

Input device

Hardware device that allows users to enter data and instructions into a computer

Output device

Hardware device used to display, project or print processed data from a computer



▲ **Figure 1.9** Examples of connected input and output devices

1.6 Secondary Storage

Secondary storage is a way of storing large amounts of data that will not be lost when power supply is interrupted. Compared to RAM, secondary storage is usually cheaper and able to store much more data. It is also non-volatile, so the data that is stored remains there even without a power supply. This makes secondary storage ideal for physically transporting data from one computer to another. On the other hand, secondary storage is usually much slower in speed compared to RAM.

The processor usually does not access data in secondary storage directly. Instead, any data in secondary storage that the processor needs might be copied to RAM first.

Key Term

Secondary storage

Way of storing large amounts of data that will not be lost when power supply is interrupted

1.6.1 Choosing Storage Media for Secondary Storage

There are many types of storage media available. When deciding on the best type of storage media to use, these factors should be considered:

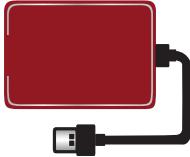
▼ **Table 1.3** Factors to consider when choosing storage media

Factor	Consideration
Capacity	What is the size of the data to be stored?
Speed	Does the data need to be accessed quickly?
Portability	Does the data need to be physically transported frequently?
Durability	How long is the storage device expected to last?
Cost	How much does the storage device cost?

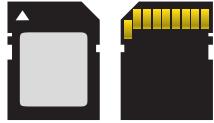
1.6.2 Types of Storage Media

Table 1.4 summarises three main types of storage media as well as their respective advantages and disadvantages.

▼ **Table 1.4** Three main types of storage media

Type	Description	Advantages	Disadvantages
Magnetic	<p>Data is stored on a magnetic material that can be read or written by a magnetic “head”</p> <p>Example: hard disk</p> 	<ul style="list-style-type: none"> Large storage capacity of up to terabytes (TB) of data Relatively cheaper than optical and solid-state storage media 	<ul style="list-style-type: none"> Easily damaged and will eventually break over time Much slower than solid-state storage media

▼ Table 1.4 Three main types of storage media (continued)

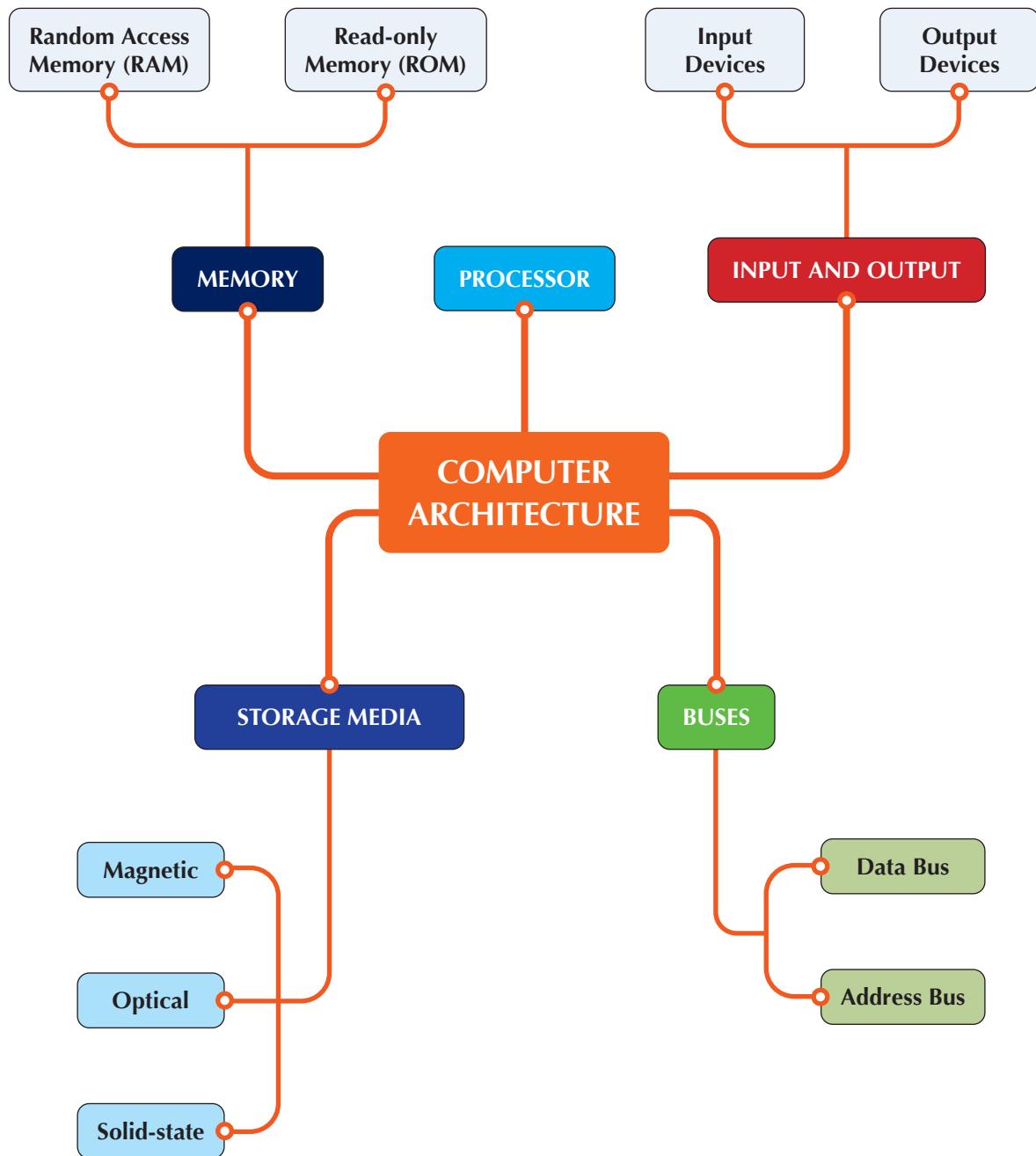
Type	Description	Advantages	Disadvantages
Optical	<p>Data is stored as very small pits or indentations that can be read or written by a laser</p> <p>Example: digital versatile disc (DVD)</p> 	<ul style="list-style-type: none"> Large storage capacity of up to gigabytes (GB) of data 	<ul style="list-style-type: none"> Data can only be written once for some non-rewritable formats Lower maximum storage capacity than magnetic storage media Vulnerable to scratches and fingerprints
Solid-state	<p>Data is stored in electronic circuits called “flash memory” that have no moving parts</p> <p>Example: memory cards</p> 	<ul style="list-style-type: none"> Much faster than magnetic or optical storage media Not as vulnerable to drops, mechanical shocks, scratches or fingerprints Smaller in size and lighter in weight than magnetic or optical storage media Uses very little power and produces no noise 	<ul style="list-style-type: none"> Much more expensive than magnetic or optical storage media



Quick Check 1.6

- For each of the following situations, suggest an appropriate secondary storage medium that should be used and give a reason for your choice:
 - Storing terabytes of video files for backup
 - Keeping an emergency copy of some important files in a wallet

Chapter Summary





Review Questions

1. Describe the function of each of the following components:
 - a) Control unit
 - b) Arithmetic logic unit (ALU)
 - c) Main memory
 - d) Secondary storage
2. Classify the following as either volatile or non-volatile data storage:
 - a) Processor register
 - b) RAM
 - c) ROM
 - d) Secondary storage
3. Arrange the following storage media in order of increasing typical storage capacity.

DVD

Processor register

Hard disk

4. Refer to the flyers that Alex collected earlier in the chapter. Tabulate the specifications of the following items for each computer stated in the flyers:
 - Processor
 - Memory
 - Storage media options (such as magnetic hard disks and optical disc writers)
5. Calculate the number of bytes in the following amounts of data:
 - a) 17 kB
 - b) 18 KiB
6. Calculate the number of bits (not bytes) in the following amounts of data:
 - a) 8 TB
 - b) 0.125 MB
 - c) 0.125 MiB

7. The position of each byte in memory is called its address.
 - a) Calculate the number of addresses needed for 64 KiB of memory.
 - b) The memory in a computer has 256 possible addresses. Calculate the total size of the computer's memory in bits (not bytes).
8. Suppose A and B are two memory addresses and R is a processor register. One way to copy data from address A to address B of memory is to perform the following two steps:

Step 1: Read the data at address A of memory and store it in register R.

Step 2: Write the data in register R to address B of memory.

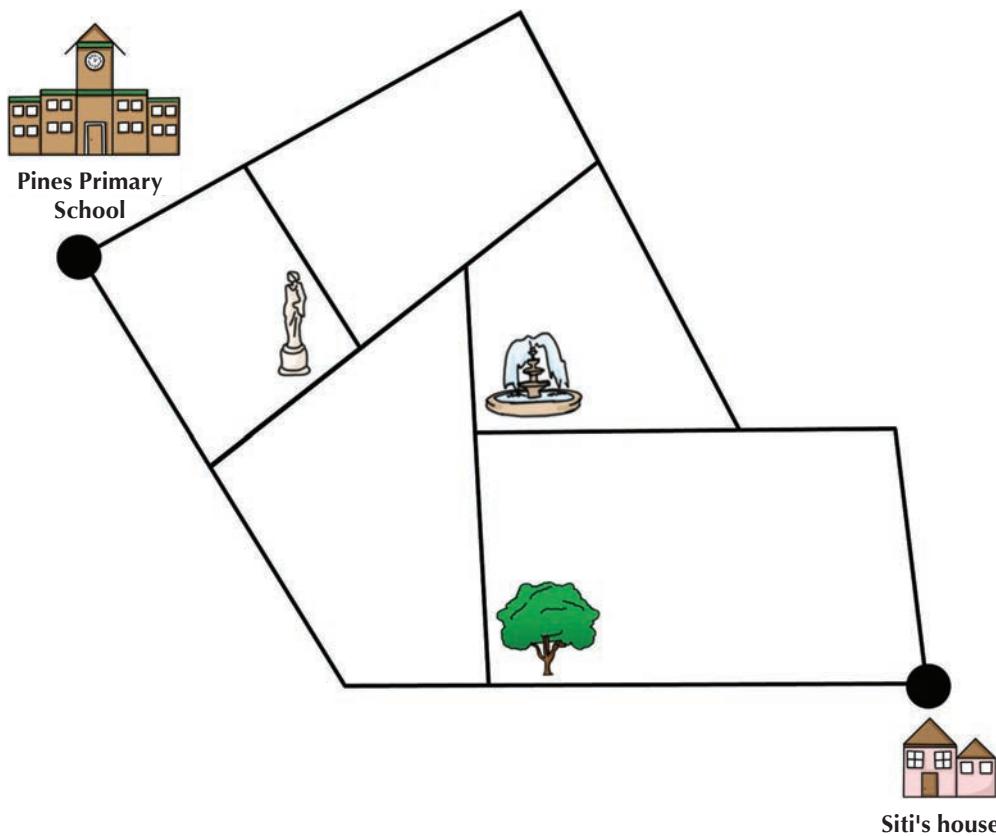
 - a) State where R is physically located.
 - b) Use the words "memory" and "processor" to complete the following paragraphs:

In Step 1, the address bus is used to transport A from the _____ to the _____, and the data bus is used to transport data from the _____ to the _____.

In Step 2, the address bus is used to transport B from the _____ to the _____, and the data bus is used to transport data from the _____ to the _____.
 - c) Suppose R and S are two processor registers. Explain why the data and address buses are not needed to copy data from R to S.
9. A hard disk is used as secondary storage for a laptop.
 - a) A data bus connects the laptop's processor directly to its hard disk. True or false?
 - b) The laptop's owner replaces the hard disk with a solid-state drive. Identify two improvements that the owner is likely to experience.

How Can Algorithms Be Used to Solve Problems?

Siti and her family have just moved to a new neighbourhood as her younger sister is attending Pines Primary School. Using the roads shown on the map below, which do you think is the shortest route from her home to her sister's school?



In the previous chapter, we learnt about the architecture of computers used today. One feature of this architecture is that computers can follow step-by-step instructions stored in memory. Whenever instructions to solve a problem can be represented as a number of steps, this set of steps is known as an algorithm.

In this chapter, you will learn about algorithms and what they can be used for. You will also learn how to specify inputs and outputs to problems so that algorithms can be used to solve them. Finally, you will apply the techniques of decomposition and generalisation to design your own algorithms.

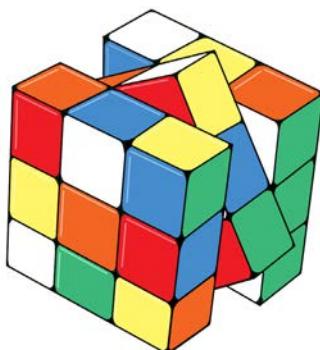


By the end of this chapter, you should be able to:

- For a given problem, specify the:
 - Inputs and the requirements for valid inputs
 - Outputs and the requirements for correct outputs
- Solve problems by decomposing them into smaller and manageable parts.
- Identify common elements across similar problems and state generalisations.

2.1 Understanding Algorithms

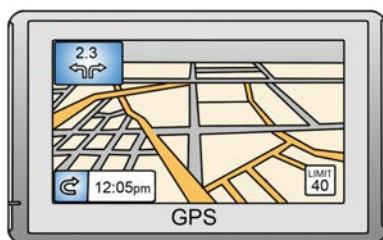
An **algorithm** is a solution that solves a problem through a set of clearly defined steps. It involves the development of a set of instructions or rules.



Rubik's cube

9	6		2			
1	8	7			2	
				3	7	1
4		9				2
6		2	4		5	3
7					8	4
2	5	4				
	6		8	3	5	
			6	4	2	

Sudoku puzzle



Navigation device

▲ **Figure 2.1** Examples of games and applications involving algorithms

For instance, the step-by-step process of solving a Rubik's cube can be considered an algorithm. The driving directions suggested by a navigation device or the steps needed to solve a Sudoku puzzle are also algorithms.

Key Term

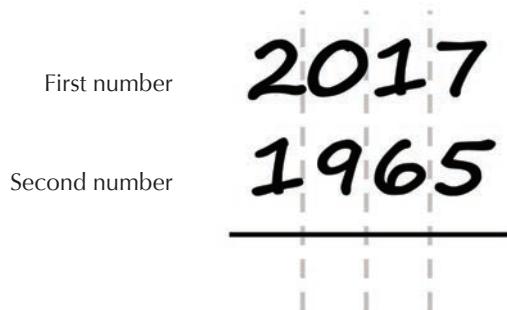
Algorithm

Set of step-by-step instructions for solving a problem

2.1.1 Example: Addition Algorithm

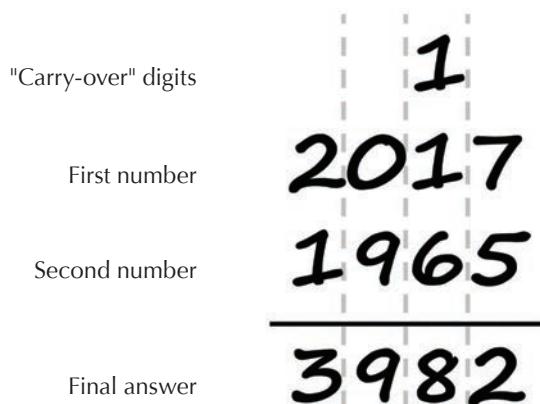
We use algorithms every day, although we may not see them explicitly written out. For example, the following steps can be used to add two positive whole numbers:

- Step 1:** Write the first number, then write the second number below it such that the digits in the ones place are aligned vertically. Draw a horizontal line below the second number, as in Figure 2.2.



▲ Figure 2.2 Step 1 of addition algorithm

- Step 2:** Let the current column be the right-most column of digits in the ones place.
- Step 3:** Add all the digits (including any “carry-over” digits) in the current column. If a digit is missing, treat it as 0. There is no “carry-over” digit for the right-most column.
- Step 4:** If the result is greater than 9, write “1” at the top of the column immediately to the left of the current column as a “carry-over” digit.
- Step 5:** Write the last digit of the result at the bottom of the current column, below the line.
- Step 6:** If there are still digits (including “carry-over” digits) to the left of the current column, redefine that column on the left as the new current column and go back to Step 3. Otherwise, the final answer is formed by the row of digits below the line.



▲ Figure 2.3 Example of completed addition algorithm

The advantage of writing out an algorithm step by step is that these steps can be followed precisely by anyone who can add digits to obtain the correct answer. This particular algorithm is also useful because it is general and works for adding any two positive whole numbers.

2.1.2 Example: Algorithm for Calculating the Mean Subject Grade for Seven Subjects

In some secondary schools, the Mean Subject Grade (MSG) is used to measure a student's performance in school. The MSG is a number between 1 (best) and 9 (worst) calculated from the student's rounded percentage scores in each subject.

The following steps provide an algorithm to calculate the MSG of a student taking seven subjects:

Step 1: Convert the rounded percentage scores into grade points using the table below.

▼ **Table 2.1** Converting rounded percentage scores to grade points

Rounded percentage score	Grade	Grade points
75–100	A1	1
70–74	A2	2
65–69	B3	3
60–64	B4	4
55–59	C5	5
50–54	C6	6
45–49	D7	7
40–44	E8	8
Less than 40	F9	9

Step 2: Add up all the grade points.

Step 3: Divide the result by 7.

Step 4: Round the result to two decimal places. This is the student's MSG.

For instance, if the rounded percentage scores for the student's seven subjects are as shown in Table 2.2, the corresponding MSG would be 3.14.

▼ **Table 2.2** Rounded percentage scores of a student taking seven subjects

Subject	Rounded percentage score
#1	63
#2	74
#3	73
#4	56
#5	80
#6	50
#7	72

This algorithm is useful as anyone can follow the steps precisely to obtain the MSG of any student (as long as the student takes exactly seven subjects).

2.1.3 Example: Algorithm for Baking a Cake

Cooking recipes are also examples of algorithms. The following example is a simplified recipe or algorithm for baking a cake:

Step 1: Pre-heat oven to 180 °C.

Step 2: Mix 250 g of butter, 1 cup of sugar, 4 eggs and 2 cups of flour in a bowl.

Step 3: Pour mixture into a baking pan.

Step 4: Place baking pan in pre-heated oven for 60 minutes.

Step 5: Take baking pan out of oven and cool for 10 minutes. The cake is complete.

Just like the addition and MSG calculation algorithms, this algorithm is useful because anyone can follow these steps precisely to bake a cake. However, this particular algorithm is not general as it only works for baking one type of cake and for a fixed serving size.

We usually want algorithms to be as general as possible so that they can be used for different problems. You will learn later about a technique for doing this, which is called generalisation.



Quick Check 2.1

1. Which of the following is not an algorithm?
 - A A map of your neighbourhood
 - B A recipe for pineapple tarts
 - C Instructions on how to use a phone
 - D Walking directions to get from home to school
2. Modify the addition algorithm described in section 2.1.1 so that it works for three positive whole numbers. Describe the changes that you made.
3. Write out the steps for an algorithm to find the top score of a class test.
(Hint: Make sure that the steps are written clearly so that anyone with just the steps and a list of scores can obtain the correct answer.)

2.2 Understanding Problems

2.2.1 Problems

Algorithms are used to solve problems. To understand a problem, we first need to specify its:

1. Inputs and requirements for valid inputs
2. Outputs and requirements for correct outputs

Here, **output** is the result that the algorithm produces while **input** is any detail that can affect what we require for the output. In computing, “input” and “output” usually refer to data or instructions.

An algorithm describes the process of transforming inputs into outputs. In particular, a **solution** is defined as an algorithm that solves the given problem by always giving correct outputs when some valid inputs are provided.

Key Terms

Input (algorithms)

Data used by an algorithm that can affect what is required for correct output

Output (algorithms)

Results produced by an algorithm

Solution

Algorithm that always gives correct outputs when provided with valid inputs



▲ Figure 2.4 Process flow in problem-solving

2.2.2 Specifying the Inputs

It is not always easy to specify the inputs for a problem correctly. Good input specifications must:

1. include only the important data that can affect what we require for the output and exclude any irrelevant details; and
2. state the range of valid or acceptable values for these inputs.

It is also a good practice to give each input a name. For instance, we can specify the inputs to the addition problem that can be solved using the algorithm in section 2.1.1 in this manner:

▼ **Table 2.3** Specifying the inputs to the addition problem for whole numbers (positive only)

Input
<ul style="list-style-type: none"> • x: a positive whole number • y: a positive whole number

Similarly, the inputs to the problem of calculating the MSG for a student taking seven subjects can be specified by treating each subject's rounded percentage score as a separate input:

▼ **Table 2.4** Specifying the inputs to the problem of calculating the MSG for seven subjects

Input
<ul style="list-style-type: none"> • <i>Score 1</i>: rounded percentage score for subject #1; must be a whole number between 0 and 100 inclusive • <i>Score 2</i>: rounded percentage score for subject #2; must be a whole number between 0 and 100 inclusive • <i>Score 3</i>: rounded percentage score for subject #3; must be a whole number between 0 and 100 inclusive • <i>Score 4</i>: rounded percentage score for subject #4; must be a whole number between 0 and 100 inclusive • <i>Score 5</i>: rounded percentage score for subject #5; must be a whole number between 0 and 100 inclusive • <i>Score 6</i>: rounded percentage score for subject #6; must be a whole number between 0 and 100 inclusive • <i>Score 7</i>: rounded percentage score for subject #7; must be a whole number between 0 and 100 inclusive

However, this way of specifying the inputs can be quite repetitive. Since the requirements for each rounded percentage score are the same, we can shorten the specification as follows:

▼ **Table 2.5** Simplified way of specifying the inputs to the problem of calculating the MSG for seven subjects

Input
<ul style="list-style-type: none"> Score: rounded percentage score for subject; must be a whole number between 0 and 100 inclusive (provided seven times, once for each subject)

2.2.3 Specifying the Outputs

To specify the outputs for a problem correctly, we need to include all the important features that the output is required to have. Any details that are not mentioned are assumed to be unimportant or irrelevant, and thus these details may differ from the output of one solution to another.

For instance, we can specify the output of the addition problem that can be solved using the algorithm in section 2.1.1 in this manner:

▼ **Table 2.6** Specifying the output of the addition problem for whole numbers (positive only)

Output
<ul style="list-style-type: none"> The sum of x and y

Similarly, we can specify the output of the problem of calculating the MSG for a student taking seven subjects in this manner:

▼ **Table 2.7** Specifying the output of the problem of calculating the MSG for seven subjects (original)

Output
<ul style="list-style-type: none"> MSG for the seven subjects

However, if we require the MSG values to be rounded to two decimal places, we can revise the problem of calculating the MSG for seven subjects accordingly:

▼ **Table 2.8** Specifying the output of the problem of calculating the MSG for seven subjects (rounded)

Output
<ul style="list-style-type: none"> MSG for the seven subjects, rounded to two decimal places



Quick Check 2.2

1. Modify the input specifications for the addition problem in Table 2.3 so that both positive and negative whole numbers are valid inputs. Determine whether the original addition algorithm is a solution to this modified problem, and if not, explain why.
2. Alex wants to find the weight of the heaviest apple on display at the market. Specify the input and output requirements for Alex's problem.
3. Table 2.9 shows a possible set of input and output requirements for calculating the average score of a class test. Suggest how it can be improved and why.

▼ **Table 2.9** Input and output requirements for the average-score problem

Input	Output
<ul style="list-style-type: none"> • <i>Title</i>: name of the class test • <i>Scores</i>: list of class test scores with each student's score on a separate row 	<ul style="list-style-type: none"> • The average score of the class test

2.3 Problem-Solving Techniques

2.3.1 Decomposition

Decomposition is a technique of breaking down a complex problem or process into smaller parts (known as **sub-problems**) such that each part is more manageable and easier to understand. The parts are evaluated separately and the solutions to these parts are then combined to solve the original problem.

Decomposing a problem is used not just for computing, but for any problem that may appear too big or complex to understand or solve. For instance, to lift and install the 7,000-tonne Marina Bay Sands SkyPark, the structure was divided into 14 different pieces which were built and assembled separately at ground level. Breaking down the construction into 14 pieces made each piece small enough to be lifted up, piece by piece, onto the 57th floor. This would not have been possible if this problem had not been decomposed beforehand.

Key Terms

Decomposition

Breaking down a complex problem or process into smaller and more manageable parts

Sub-problem

A problem whose solution contributes to the solution of a bigger problem

Two common approaches to decomposing a problem:

1. Incremental approach:

Identify quantitative features of the input or output that are causing the problem to be too large to handle. Sometimes, the solution to a small version of the problem with one or more of these features reduced can be found and gradually extended to larger versions of the problem. Each gradual extension of the solution is a separate sub-problem.

2. Modular approach:

Solve simple examples of the problem manually and identify tasks that are of different natures. Usually, these tasks can be separated from each other to become distinct (and sometimes unrelated) sub-problems. This usually results in sub-problems that are different from each other.

2.3.1.1 Example: Decomposing the Addition Problem

Let us look at the addition problem shown earlier:

▼ **Table 2.10** Input and output requirements for the addition problem for whole numbers (positive only)

Input	Output
<ul style="list-style-type: none">• x: a positive whole number• y: a positive whole number	<ul style="list-style-type: none">• The sum of x and y

We know how to add single digits. The reason why this problem is more difficult to solve is that both the inputs and the output can have multiple digits.

Using the incremental approach, we can identify the number of digits for the inputs as one of the features that cause the problem to be too large to handle.

Incremental approach:

Identify quantitative features of the input or output that are causing the problem to be too large to handle. Sometimes, the solution to a small version of the problem with one or more of these features reduced can be found and gradually extended to larger versions of the problem. Each gradual extension of the solution is a separate sub-problem.

Suppose the inputs to the problem have multiple digits; we can then gradually extend the solution for single digits to larger versions of the problem via the following sub-problems:

▼ **Table 2.11** Decomposing the addition problem for whole numbers (positive only)

Sub-problem	Input	Output
#1	<ul style="list-style-type: none"> x_1: the last digit of x y_1: the last digit of y 	<ul style="list-style-type: none"> Result #1: The sum of x_1 and y_1
#2	<ul style="list-style-type: none"> x_2: the last two digits of x y_2: the last two digits of y Result #1: The result of solving sub-problem #1 	<ul style="list-style-type: none"> Result #2: The sum of x_2 and y_2
#3	<ul style="list-style-type: none"> x_3: the last three digits of x y_3: the last three digits of y Result #2: The result of solving sub-problem #2 	<ul style="list-style-type: none"> Result #3: The sum of x_3 and y_3
...and so on...	...and so on...	...and so on...

This is how the addition algorithm in section 2.1.1 was developed.

2.3.1.2 Example: Decomposing the Calculation of MSG for Seven Subjects

Let us look at the revised problem of calculating the MSG for seven subjects:

▼ **Table 2.12** Input and output requirements for the problem of calculating the MSG for seven subjects

Input	Output
<ul style="list-style-type: none"> Score: rounded percentage score for subject; must be a whole number between 0 and 100 inclusive (provided seven times, once for each subject) 	<ul style="list-style-type: none"> MSG for the seven subjects, rounded to two decimal places

If we try to calculate the MSG for the example provided in Table 2.2, we find that the task of converting rounded percentage scores to grade points is quite separate from the task of summing up the grade points and dividing the result by 7. This motivates us to use the modular approach.

Modular approach:

Solve simple examples of the problem manually and identify tasks that are of different natures. Usually, these tasks can be separated from each other to become distinct (and sometimes unrelated) sub-problems. This usually results in sub-problems that are different from each other.

We can thus decompose the original problem into the following two sub-problems:

▼ **Table 2.13** Decomposing the problem of calculating the MSG for seven subjects

Sub-problem	Input	Output
#1	<ul style="list-style-type: none"> <i>Score</i>: rounded percentage score for subject; must be a whole number between 0 and 100 inclusive (provided seven times, once for each subject) 	<ul style="list-style-type: none"> <i>Grade points</i>: corresponding grade points for subject (output seven times, once for each subject)
#2	<ul style="list-style-type: none"> <i>Grade points</i>: grade points for subject; must be a whole number between 1 and 9 inclusive (provided seven times, once for each subject) 	<ul style="list-style-type: none"> MSG for the seven subjects, rounded to two decimal places

Each sub-problem is simpler and easier to solve on its own compared to the original problem.

2.3.2 Pattern Recognition

Pattern recognition is the technique of identifying similarities among two or more items. In problem-solving and algorithm design, it is usually used in three ways:

- Identifying patterns among two or more problems. These problems may come from different situations or they may be sub-problems obtained from decomposition. These patterns are clues that the solutions to these problems will most likely be similar. Thus, if we already know the solution for one problem, we can probably modify that solution to solve other problems that follow a similar pattern.

Key Term

Pattern recognition
Identifying similarities among two or more items

2. Identifying patterns among two or more solutions. These solutions may come from solving the sub-problems obtained from decomposition, or from solving examples of a more general problem. These patterns are clues that these solutions can be generalised to produce a single solution that can solve multiple problems.

3. Identifying patterns among two or more steps in a solution. These patterns indicate that the solution can be simplified or shortened using iteration structures that you will learn about in the next chapter.

2.3.2.1 Example: Solving the Sub-Problem of Calculating the MSG from the Grade Points of Seven Subjects

Let us look at sub-problem #2 we obtained from decomposing the calculation of MSG for seven subjects:

▼ **Table 2.14** Input and output requirements for the problem of calculating MSG from the grade points of seven subjects

Sub-problem	Input	Output
#2	<ul style="list-style-type: none"> • <i>Grade points</i>: grade points for subject; must be a whole number between 1 and 9 inclusive (provided seven times, once for each subject) 	<ul style="list-style-type: none"> • MSG for the seven subjects, rounded to two decimal places

Compare this with the problem of calculating the average of seven numbers:

▼ **Table 2.15** Input and output requirements for the problem of calculating the average of seven numbers

Input	Output
<ul style="list-style-type: none"> • <i>Number</i>: a number (provided seven times) 	<ul style="list-style-type: none"> • <i>Average</i>: average of the seven numbers

A simple solution to this problem is as follows:

Step 1: Add up all the numbers.

Step 2: Divide the result by 7.

The MSG is just an average of the grade points, so by using pattern recognition, we can see that both problems are similar, with the only differences being that:

1. The numbers to be averaged are called “grade points” and must be whole numbers between 1 and 9 inclusive.
2. The average is called “MSG” and must be rounded to two decimal places.

With this information as a guide, we can easily modify the algorithm to calculate the average of seven numbers so that it becomes a solution to the problem of calculating MSG from the grade points of seven subjects:

- Step 1:** Add up all the *grade points*.
Step 2: Divide the result by 7.
Step 3: Round the result to two decimal places.

In this case, the only significant modifications needed are the renaming of “numbers” to “grade points” and the addition of one more step.

2.3.3 Generalisation

Generalisation is a technique of replacing two or more similar items with a single, more general item. This can be done with both problems and solutions. In particular, a general solution is extremely useful because it allows us to solve a large number of problems with just a single solution. The addition algorithm we learnt earlier is an example of a general solution.

In most cases, generalisation can be accomplished by following a basic algorithm similar to how factorisation is done in algebra. For example, the expression “ $x^2y + xy^2$ ” can be simplified by identifying “ xy ” as the common factor in every term. The common factor can then be “pulled out” so that only the factors that are different remain.

Key Term

Generalisation

Technique of replacing two or more similar problems or solutions with a single, more general problem or solution

$$\begin{aligned}x^2y + xy^2 &= xxy + xyy \\&= x(xy) + (xy)y \\&= (xy)(x + y)\end{aligned}$$

▲ Figure 2.5 Factorising $x^2y + xy^2$

The same idea can be used to “pull out” common elements or steps from multiple problems or solutions into a more general problem or solution.

2.3.3.1 Example: Generalising Similar Problems

Suppose we want to generalise the following three problems:

▼ **Table 2.16** Input and output requirements for the problem of calculating the MSG for seven subjects

Input	Output
<ul style="list-style-type: none"> Score: rounded percentage score for subject; must be a whole number between 0 and 100 inclusive (provided seven times, once for each subject) 	<ul style="list-style-type: none"> MSG for the seven subjects, rounded to two decimal places

▼ **Table 2.17** Input and output requirements for the problem of calculating the MSG for eight subjects

Input	Output
<ul style="list-style-type: none"> Score: rounded percentage score for subject; must be a whole number between 0 and 100 inclusive (provided eight times, once for each subject) 	<ul style="list-style-type: none"> MSG for the eight subjects, rounded to two decimal places

▼ **Table 2.18** Input and output requirements for the problem of calculating the MSG for nine subjects

Input	Output
<ul style="list-style-type: none"> Score: rounded percentage score for subject; must be a whole number between 0 and 100 inclusive (provided nine times, once for each subject) 	<ul style="list-style-type: none"> MSG for the nine subjects, rounded to two decimal places

The only difference between the three problems is the number of subjects. Let us call this value *number* and come up with a more general problem with this name:

▼ **Table 2.19** Input and output requirements for the problem of calculating the MSG for a number of subjects

Input	Output
<ul style="list-style-type: none"> Number: number of subjects; must be a positive whole number Score: rounded percentage score for subject; must be a whole number between 0 and 100 inclusive (provided <i>number</i> times, once for each subject) 	<ul style="list-style-type: none"> MSG for the <i>number</i> subjects, rounded to two decimal places

Table 2.19 shows the generalised version of the three problems because it can recreate those problems by just providing different values for *number*.

2.3.3.2 Example: Generalising Similar Solutions

The solutions to the three problems can also be generalised in this way. Suppose that the three solutions are identical except for Step 3 in the algorithm:

- For the original problem of calculating the MSG for seven subjects:
Step 3: Divide the result by 7.
- For the problem of calculating the MSG for eight subjects:
Step 3: Divide the result by 8.
- For the problem of calculating the MSG for nine subjects:
Step 3: Divide the result by 9.

From this, we can see that the sum of grade points must be divided by the number of subjects to obtain the correct MSG. If we use *number* to represent the number of subjects that the student is offering, we have the following general solution:

Step 1: Convert the rounded percentage scores into grade points using the table below.

▼ **Table 2.20** Converting rounded percentage scores to grade points

Rounded percentage score	Grade	Grade points
75–100	A1	1
70–74	A2	2
65–69	B3	3
60–64	B4	4
55–59	C5	5
50–54	C6	6
45–49	D7	7
40–44	E8	8
Less than 40	F9	9

Step 2: Add up all the converted grade points.

Step 3: Divide the result by *number*.

Step 4: Round the result to two decimal places. The final answer is the student's MSG.

This general solution solves the corresponding general problem:

▼ **Table 2.21** Input and output requirements for the problem of calculating the MSG for a number of subjects

Input	Output
<ul style="list-style-type: none"> • <i>Number</i>: number of subjects; must be a positive whole number • <i>Score</i>: rounded percentage score for subject; must be a whole number between 0 and 100 inclusive (provided <i>number</i> times, once for each subject) 	<ul style="list-style-type: none"> • MSG for the <i>number</i> subjects, rounded to two decimal places

2.3.3.3 Basic Algorithm for Generalising Problems and Solutions

Based on the earlier examples, a basic algorithm to generalise two or more problems would be:

- Step 1:** Identify the parts that are different among the problems being considered and give each part a name (e.g., *number* for the different number of subjects).
- Step 2:** Copy the identical parts into a new problem, replacing the parts that are different using the names chosen in Step 1.
- Step 3:** Add the names chosen in Step 1 as required inputs to the problem. Think through and define the range of valid or acceptable values for these new inputs (e.g., *number* should be a positive whole number). The result is a generalised version of the problems being considered.

We can use a similar algorithm to generalise the corresponding solutions:

- Step 1:** Identify the parts that are identical among the solutions being considered and copy them to a new solution.
- Step 2:** Generalise the corresponding problems using the previous algorithm.
- Step 3:** Fill in the parts that are different by expressing them in terms of required inputs to the general problem (e.g., dividing the result by *number* instead of a fixed value). The result is a generalised version of the solutions being considered.



Quick Check 2.3

1. Choose the correct words for the passage below from the given choices:

If a complex problem is not broken down into parts, the time taken to solve the problem may be too (*short/long*). The technique of breaking down the problem into parts, called (*decomposition/generalisation*), allows the parts to be evaluated separately. By making the parts (*less/more*) manageable, the solutions to these parts can then be (*combined/separated*), and the original problem can be solved.

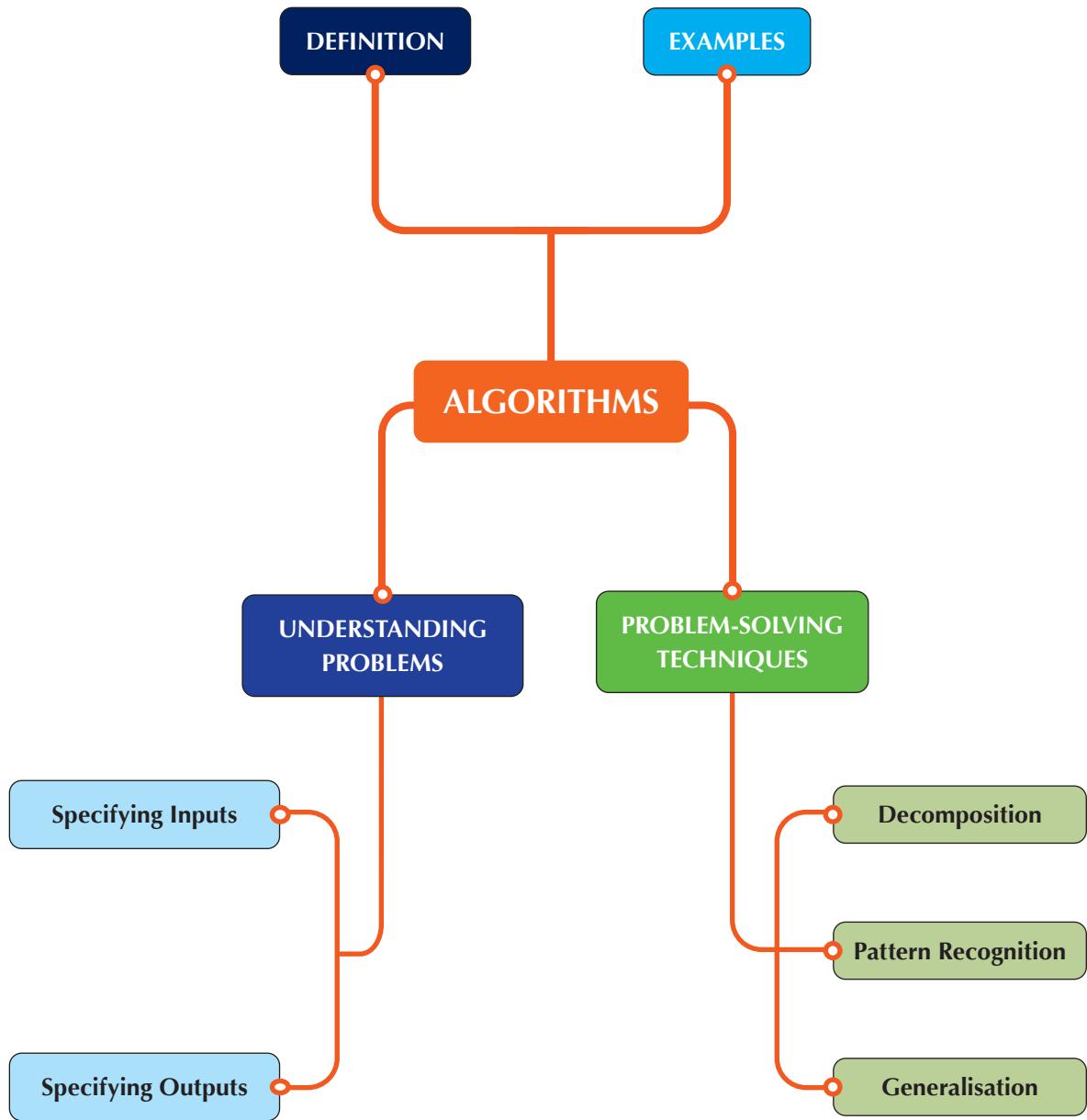
2. Why do we look for patterns in problems?

- A It helps in identifying common solutions that can be applied to similar problems.
- B It becomes more difficult to solve complex problems using patterns.
- C The patterns would be pulled out from the problems as they are unnecessary.
- D The patterns help us to specify and state a problem clearly.

3. Siti wants to find a general solution to calculate the absolute or positive difference between any two positive whole numbers. For example, the absolute difference between 17 and 20 is 3 (not -3) and the absolute difference between 20 and 17 is also 3.

- a) Specify the input and output requirements for Siti's problem.
- b) Calculate the absolute difference between 2017 and 1965. Write out the algorithm that you used. You may assume that the person following the algorithm can subtract and compare numbers without help.
- c) Calculate the absolute difference between 2017 and 2020. Write out the algorithm that you used. You may assume that the person following the algorithm can subtract and compare numbers without help.
- d) Generalise the solutions used for the two problems in b) and c) above and write out a general solution for Siti's problem.

Chapter Summary



Review Questions

- The following are the steps involved in making and drinking a cup of tea. Arrange the steps in the appropriate order to describe the correct algorithm for this process.
 - Drink the contents of the cup.
 - Add tap water to the kettle.
 - Wait for a few minutes to let the tea steep.
 - Place a teabag in a cup and pour some water from the kettle into the cup.
 - Boil the contents of the kettle.
- Consider Siti's problem of finding the shortest route from her home to her sister's school, which was shown at the beginning of this chapter.

▼ **Table 2.22** Input and output requirements for Siti's problem

Input	Output
<ul style="list-style-type: none"> • <i>Map</i>: a map of roads in Siti's neighbourhood • <i>Home</i>: the position of Siti's home on <i>map</i> • <i>School</i>: the position of Pines Primary School on <i>map</i> 	<ul style="list-style-type: none"> • The shortest route from <i>home</i> to <i>school</i> using the roads shown on <i>map</i>

The following is a set of input and output requirements for sub-problem #2 after Siti's problem is decomposed into two sub-problems.

▼ **Table 2.23** Sub-problem #2 obtained by decomposing Siti's problem

Sub-problem	Input	Output
#2	<ul style="list-style-type: none"> • <i>Routes</i>: list of routes 	<ul style="list-style-type: none"> • Shortest route in <i>routes</i>

Suggest a possible set of input and output requirements for sub-problem #1.

How Do I Use Flowcharts and Pseudo-Code?

Recently, there has been a significant increase in the number of latecomers at Alex's school. Alex has never been late and wants to share how he has achieved this with his schoolmates. What methods can Alex use to represent the algorithm that he uses to reach school on time?



In the previous chapter, we learnt how to write out algorithms as a series of steps. Sometimes, this may not be the best way to represent an algorithm.

In this chapter, you will learn how to give names to values and represent algorithms in flowcharts and pseudo-code, as well as how to analyse and correct algorithms using trace tables.



By the end of this chapter, you should be able to:

- Produce an algorithm to solve a problem, either as a flowchart or in pseudo-code. The following pseudo-code keywords and constructs should be used:
 - INPUT/OUTPUT
 - IF...THEN...ELSEIF...ELSE...ENDIF
 - WHILE...ENDWHILE
 - FOR...NEXT
- Perform a dry run of a set of steps to determine its purpose and/or output.
- Produce trace tables to show values of variables at each stage in a set of steps.
- Locate logic errors in an algorithm, and correct or modify an algorithm to remove the errors or for changes in task specification.

3.1 Variables

We often give names to values when specifying inputs and outputs or generalising problems and solutions. For example, in the problem shown in Table 3.1, x and y are the named values.

▼ **Table 3.1** Input and output requirements for the addition problem

Input	Output
<ul style="list-style-type: none"> • x: a positive whole number • y: a positive whole number 	<ul style="list-style-type: none"> • The sum of x and y

In flowcharts and pseudo-code, these named values are called **variables**. Imagine a variable as a box that is labelled with its name. A variable can store different types of data just as a box can store different kinds of items (see Figure 3.1). Data can come in many forms such as letters, lists, numbers and words.



Key Term

Variable (flowchart and pseudo-code)
Named storage space for a value that can be changed while the algorithm is running

▲ **Figure 3.1** Just like a box can store different kinds of items, a variable can store different types of data

3.1.1 Storing Values

We use the equals sign ($=$) when we want to store a value in a variable. The value to be stored is placed to the right of the equals sign, while the variable to store it in is placed on the left. For example, to store the value 5 in a variable named x , we can say $x = 5$.

Did you know?

The left arrow symbol (\leftarrow) is sometimes used instead of the equals sign ($=$) to store values in variables. For example, $x \leftarrow 5$ has the same meaning as $x = 5$.

A variable is **initialised** when a value is first stored in it. It is usually a good practice to initialise a variable as soon as there is a value to be assigned to it.

In flowcharts and pseudo-code, a variable can only store one value at a time. Assigning new data to a variable that already contains data will overwrite the old data. In this way, the stored value of a variable can be changed.

Variable names should be meaningful, short and consist of a single word only. They need to be meaningful as variable names are often the only descriptions readers have to help them determine what each variable represents.

At the same time, they need to be short as variable names are likely to be used multiple times in an algorithm. Variable names that are too long can result in algorithms that are tedious to both write and read. For example, “name” or “age” would be good variable names. If a variable is meant to store just a number or numerical value, then single-letter names like “ x ” and “ y ” are also acceptable.

Key Term

Initialise

To store or assign a value to a variable for the first time

3.1.2 Reading Values

To use the value stored in a variable, we should just use the name of the variable. For example, we can say $x + 1$ to take a copy of the value stored in x and add 1 to it.

3.1.3 List Variables

Some variables store a list of multiple values. Each value is identified by a position number starting from 0 and we can access individual values by using the variable name followed by its position number in square brackets. Each numbered position in the list serves as a separate variable.

For example, if a variable named y is a list, then we can use the variables $y[0]$, $y[1]$ and $y[2]$ to access the first three values stored in y . To store the value 5 in the first position of the list, we can say $y[0] = 5$.

3.2 Flowcharts

The algorithm below shows a part of Alex's routine each morning:

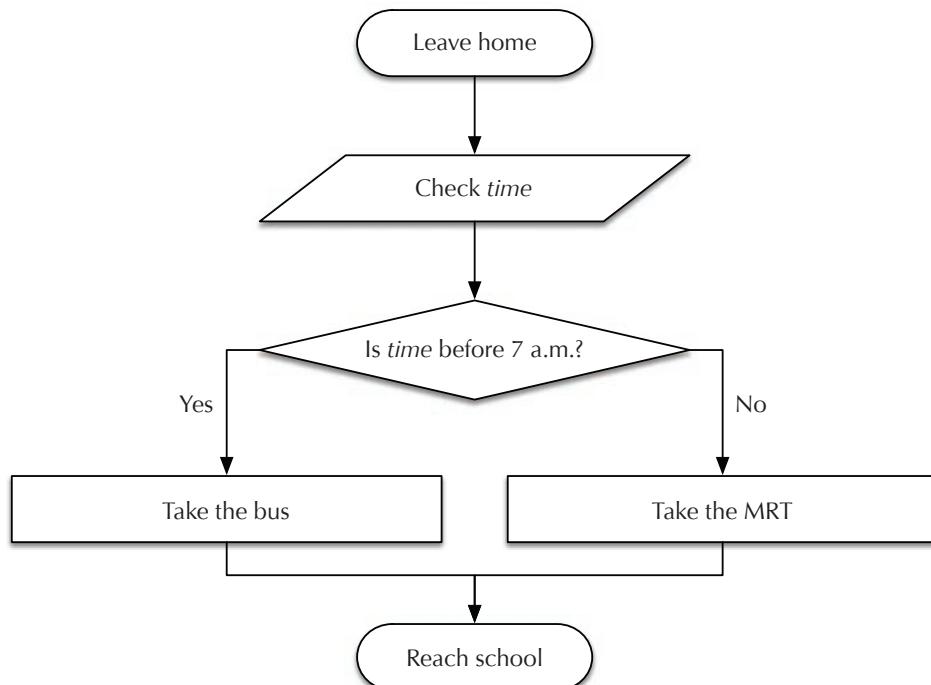
- Step 1:** Leave home.
- Step 2:** Check the time.
- Step 3:** If it is before 7 a.m., proceed to Step 4.
Otherwise, proceed to Step 5.
- Step 4:** Take the bus, then proceed to Step 6.
- Step 5:** Take the MRT, then proceed to Step 6.
- Step 6:** Reach school.

Key Term

Flowchart

Visual presentation of an algorithm using symbols to show the flow of a sequence of steps

This same algorithm can also be represented using the **flowchart** as shown in Figure 3.2.



▲ **Figure 3.2** Alex's algorithm represented in a flowchart

Did you know?

The earliest forms of flowcharts were introduced by engineer Frank Gilbreth in 1921 to visualise processes used in industrial production, sales and finance.

As can be seen, a flowchart is a more visual way of presenting an algorithm and may be easier to understand than a long series of steps. Each step is represented by a **symbol** and the order in which steps are followed is shown using **flow lines** or arrows. To create our own flowcharts, we need to learn about the symbols that are used and how to connect them in a structured manner.

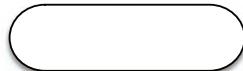
3.2.1 Standard Flowchart Symbols

There are four standard symbols used in flowcharts:

1. Terminator
2. Data
3. Decision
4. Process

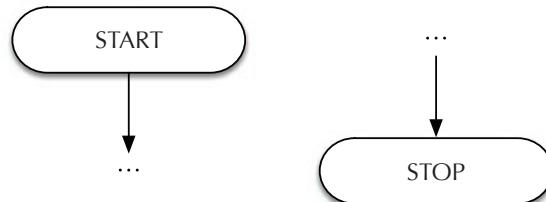
3.2.1.1 Terminator Symbol

The terminator symbol is a rectangle with rounded corners, or a “pill” shape.



▲ Figure 3.3 The terminator symbol

It represents the beginning or end of a set of steps, and usually contains either the START or STOP command as shown in Figure 3.4.



▲ Figure 3.4 Examples of how the terminator symbol is used

Did you know?

The END command is sometimes used instead of the STOP command.

3.2.1.2 Data Symbol

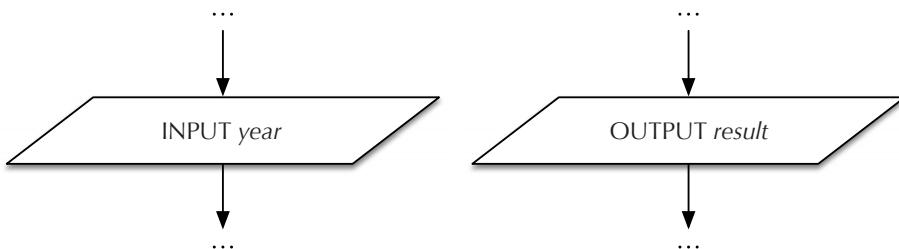
The data symbol is a parallelogram.



▲ **Figure 3.5** The data symbol

It represents the step of either receiving input data from outside the algorithm using the INPUT command or producing output from within the algorithm using the OUTPUT command.

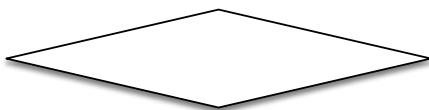
An example of each type of command is shown in Figure 3.6.



▲ **Figure 3.6** Examples of how the data symbol is used

3.2.1.3 Decision Symbol

The decision symbol is a diamond.

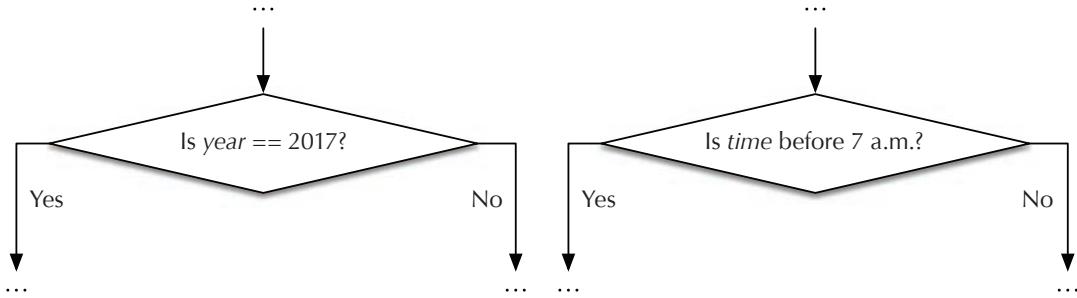


▲ **Figure 3.7** The decision symbol

It represents a step involving a question. The outgoing arrows represent the possible outcomes to the question and are usually labelled “Yes” and “No”. There may be two or three outgoing arrows depending on the number of possible outcomes. Only one of these outgoing arrows should be followed when performing the algorithm.

Note that the question used in a decision symbol may involve checking if two values are equal or not equal. We typically use the special symbols “`==`” and “`!=`” to represent such checks. For example, to check if the variables `x` and `y` are equal, we would ask “Is `x == y`?”. Conversely, to check if the variables `x` and `y` are not equal, we would ask “Is `x != y`?“

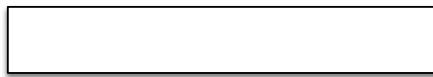
Some examples of the decision symbol are shown in Figure 3.8.



▲ **Figure 3.8** Examples of how the decision symbol is used

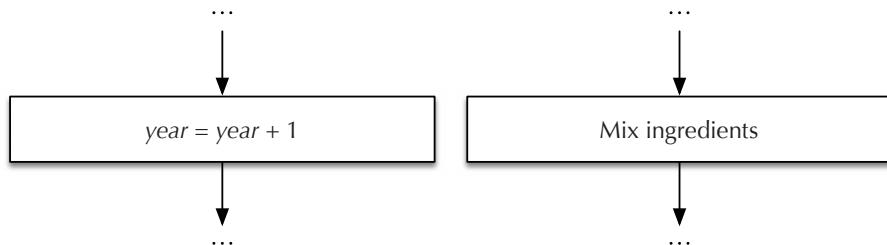
3.2.1.4 Process Symbol

The process symbol is a rectangle.



▲ **Figure 3.9** The process symbol

It represents a step involving an action or operation. This usually involves changing the value of a variable or performing more complex actions, as shown in Figure 3.10.

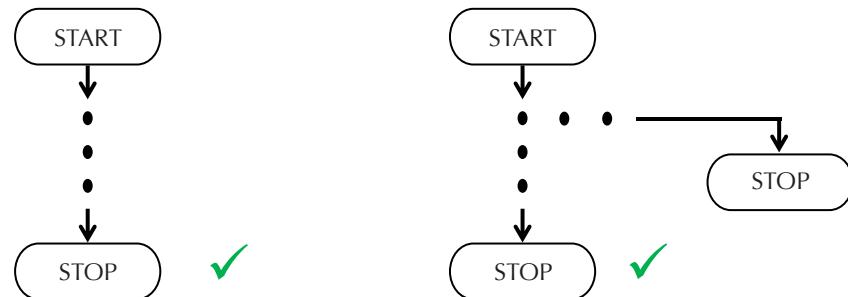


▲ **Figure 3.10** Examples of how the process symbol is used

3.2.2 Structuring Flowcharts

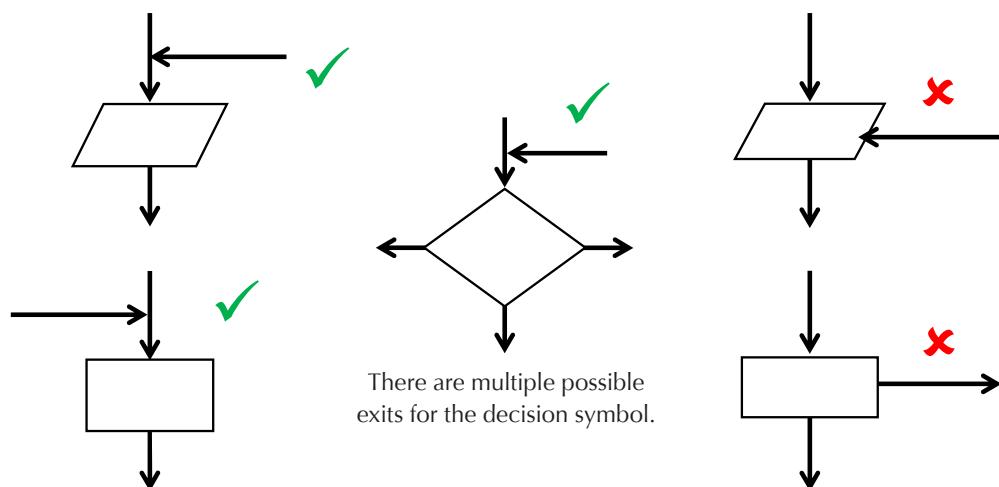
Flowchart symbols can be connected using flow lines or arrows to indicate the order in which they should be followed. Flowcharts and flow lines must obey the following rules:

1. A flowchart must start with exactly *one* terminator symbol and may stop with one or more terminator symbols.



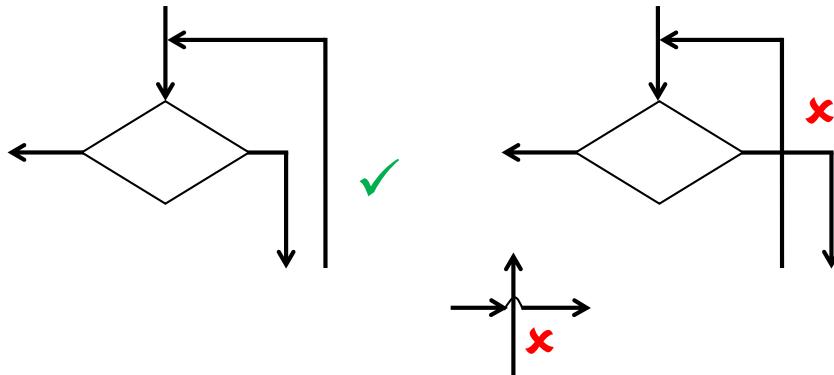
▲ **Figure 3.11** A flowchart must only have one start and one or more end terminator symbols.

2. Each flowchart symbol (except the terminator and decision symbols) must have exactly *one* entry point and *one* exit point.



▲ **Figure 3.12** Each flowchart symbol must have only one entry and one exit point.

3. Flow lines should not cross one another.



▲ Figure 3.13 Flow lines should not cross

However, these rules are not enough. The flow lines should also follow the sequence, selection and iteration constructs (which will be discussed in the following sections) to avoid ending up with overly complicated flowcharts that are difficult to read or understand. Like symbols, each of these **constructs** has exactly one entry point and one exit point.

3.2.2.1 Sequence

The most straightforward way to connect multiple symbols or constructs is to link them into a chain with flow lines pointed in the same direction so that there is only one entry point and one exit point. This is called a **sequence construct** and it is used to perform multiple instructions in a fixed order.

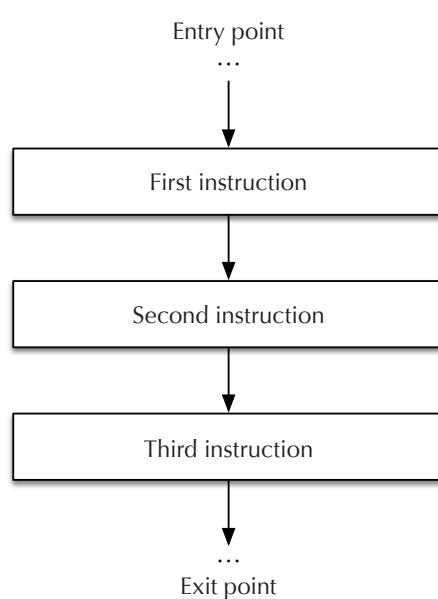
Key Terms

Construct

Algorithm structure with exactly one entry point and one exit point

Sequence construct

Construct for performing multiple instructions in a fixed order



▲ Figure 3.14 Example of a sequence construct

3.2.2.2 Selection

Whenever a decision symbol is used, it always has exactly one incoming flow line that serves as an entry point but may have two or more outgoing flow lines exiting from it. These outgoing flow lines lead to different **branches** or sequences of symbols and constructs. In some flowcharts, these separate sequences merge back into a single flow line. This forms a **selection construct** with one entry point at the decision symbol and one exit point where the multiple branches merge.

We call this a selection construct as the person or computer running the flowchart has to *select only one* of the different branches from the decision symbol to follow. For instance, Figure 3.15 shows a selection construct with two branches (i.e., a “Yes” branch and a “No” branch).

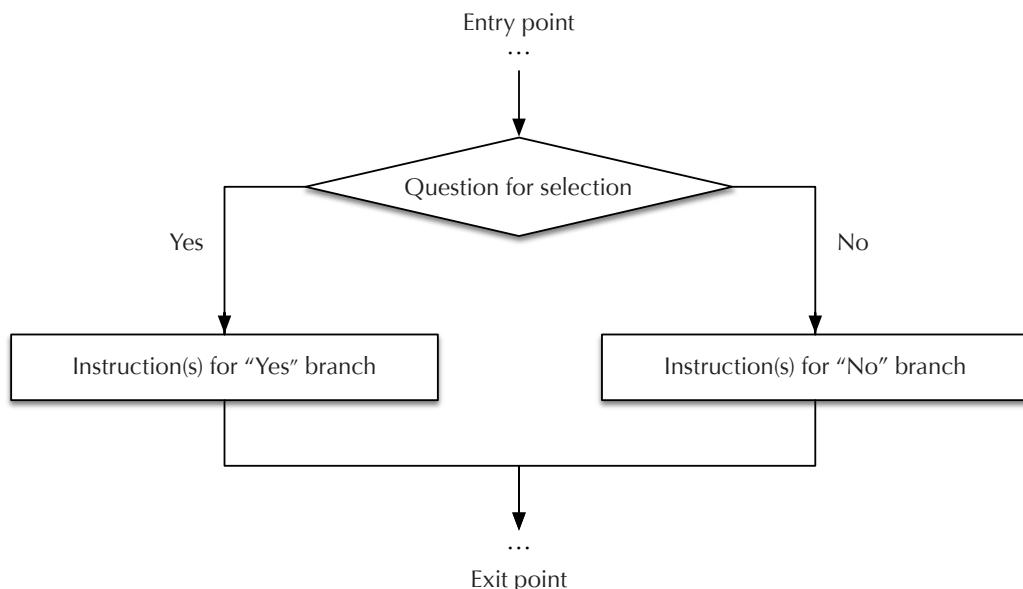
Key Terms

Branch

Sequence of instructions that serves as one option out of two or more choices

Selection construct

Construct for choosing between two or more branches based on a particular condition



▲ Figure 3.15 Example of a selection construct

3.2.2.3 Iteration

Sometimes, instead of having all the branches merge together, one or more of the branches may lead back to the original decision symbol, forming a closed loop. In the flowchart in Figure 3.16, there is one entry point at the decision symbol and one exit point at the branch that does not form a loop. This is called an **iteration construct** and it is used to repeat instructions while a particular condition is true. This construct is often used to **iterate** or repeat the instructions.

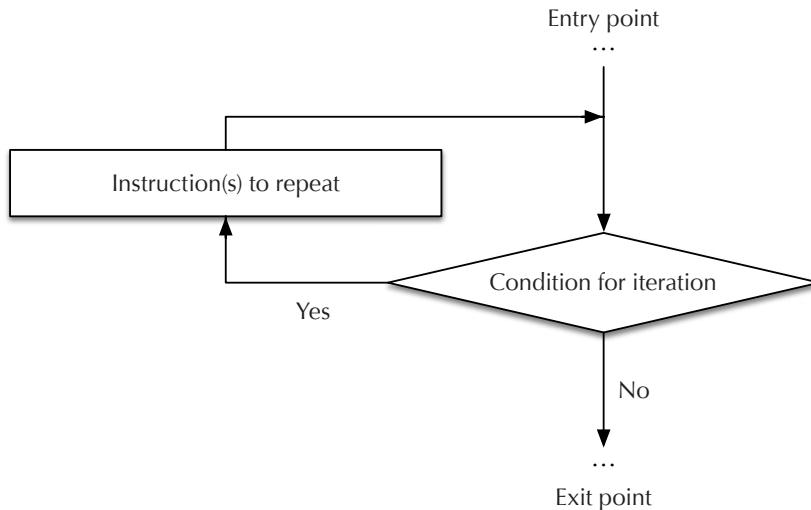
Key Terms

Iterate

Repeat

Iteration construct

Construct for repeating instructions while a particular condition is true



▲ Figure 3.16 Example of an iteration construct

3.2.3 Example: Adding Five Numbers

Now that you are familiar with flowchart symbols and constructs, let us try to solve a problem and represent its solution using a flowchart. Suppose we wish to find the sum of five numbers but we can only add two numbers at a time. One way to specify this problem is as follows:

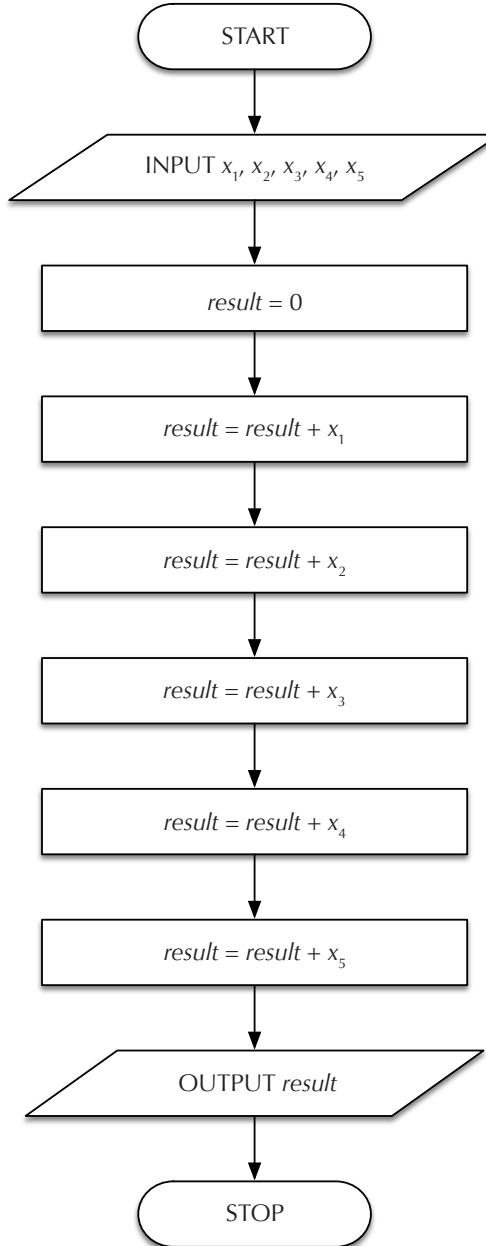
▼ Table 3.2 Input and output requirements for the problem of adding five numbers

Input	Output
<ul style="list-style-type: none"> • x_1: a number • x_2: a number • x_3: a number • x_4: a number • x_5: a number 	<ul style="list-style-type: none"> • The sum of x_1, x_2, x_3, x_4 and x_5, calculated by adding only two numbers at a time

Using this definition, a possible solution is as follows:

- Step 1:** Let $result$ be a variable. Initialise it to 0.
- Step 2:** Add x_1 to $result$ and store the sum back in $result$.
- Step 3:** Add x_2 to $result$ and store the sum back in $result$.
- Step 4:** Add x_3 to $result$ and store the sum back in $result$.
- Step 5:** Add x_4 to $result$ and store the sum back in $result$.
- Step 6:** Add x_5 to $result$ and store the sum back in $result$.
- Step 7:** The final answer is read from $result$.

We can represent this algorithm as a flowchart by converting Steps 1 to 6 into process symbols and Step 7 into a data symbol for output (see Figure 3.17). The symbols can then be arranged into a sequence construct so that they are performed in order. Terminator symbols to start and stop the flowchart, and a data symbol to receive input are then added accordingly.



▲ **Figure 3.17** Solution to the problem of adding five numbers

Looking at the flowchart in Figure 3.17, we can see a pattern that repeats – many steps of the algorithm are similar. This usually means that we can shorten the algorithm by generalising the steps and using an iteration construct instead. To do this, instead of having all five inputs being provided all at once, the inputs can be provided as a sequence that can be checked up to five times for the next input. We can change how the problem is specified as shown in Table 3.3.

▼ **Table 3.3** Input and output requirements for the problem of adding five numbers

Input	Output
<ul style="list-style-type: none"> • x: a number (provided five times) 	<ul style="list-style-type: none"> • The sum of all x values provided, calculated by adding only two numbers at a time

The solution can then be shortened to:

Step 1: Let $result$ be a variable. Initialise it to 0.

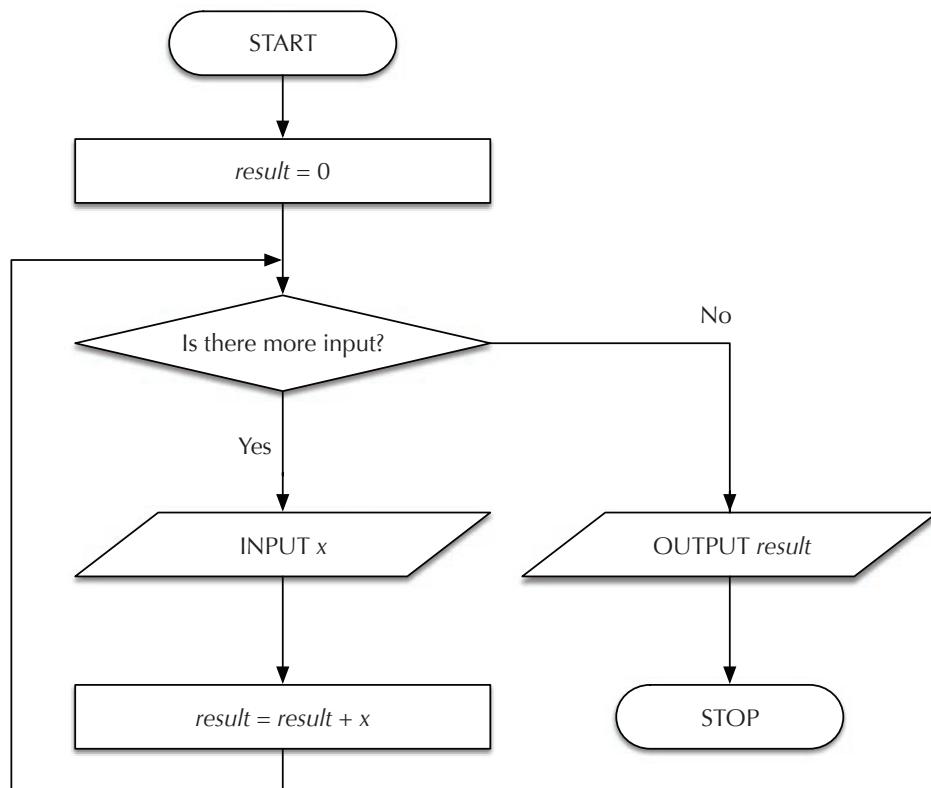
Step 2: If there is more input, proceed to Step 3. Otherwise, proceed to Step 5.

Step 3: Get the next input x .

Step 4: Add x to $result$ and store the answer back in $result$. Go back to Step 2.

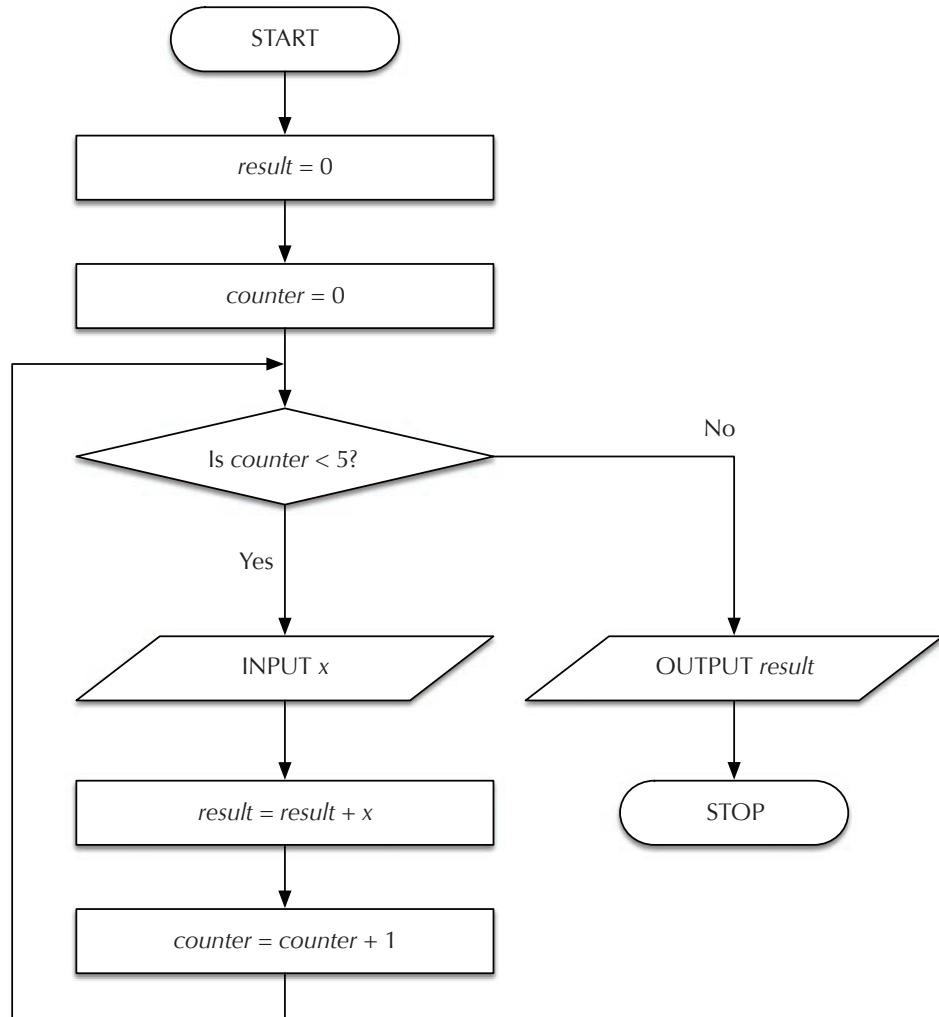
Step 5: The final answer is read from $result$.

This can be directly translated to a flowchart by having each step converted into a symbol, as shown in Figure 3.18.



▲ **Figure 3.18** Solution to the problem of adding five numbers that uses iteration

This solution shown in the flowchart is shorter than the previous solution as the repeated steps have been replaced by an iteration construct. However, this flowchart does not tell us how to check for any more input. To avoid leaving such details out, we can explicitly keep track of how many inputs have been processed so far by using a variable named *counter* (see Figure 3.19).



▲ **Figure 3.19** Solution to the problem of adding five numbers that uses iteration and a counter variable

The level of detail that is needed in a flowchart depends on the amount of guidance that the reader will need. However, as you will learn in the next chapter, computers can only understand very simple instructions, so more guidance (such as including the additional *counter* variable here) is usually needed.



Quick Check 3.2

- The algorithm of another student's journey to school is described as follows:

Step 1: Leave home

Step 2: Check time

Step 3: If it is before 8 a.m., take the bus

Step 4: If it is after 8 a.m., take the train

Step 5: Reach school

a) Specify the input and output requirements of the algorithm.

b) Draw a flowchart to represent the algorithm.

3.3 Trace Tables

3.3.1 Dry Runs

It is not always easy to determine what an algorithm or flowchart does just from looking at it. This is especially true if we do not know the original problem that the algorithm was meant to solve or if the flowchart has many variables and branches.

To figure out what an algorithm does or why it is not producing the correct output, we can perform a **dry run** of the algorithm.

A dry run is a process where the steps of the algorithm are followed one by one manually. If any input is needed, some **test data** must also be prepared beforehand.

The result of a dry run is a **trace table** which records the change in variables and the output of the algorithm during the run.

Key Terms

Dry run

Process of running through a set of steps manually

Test data

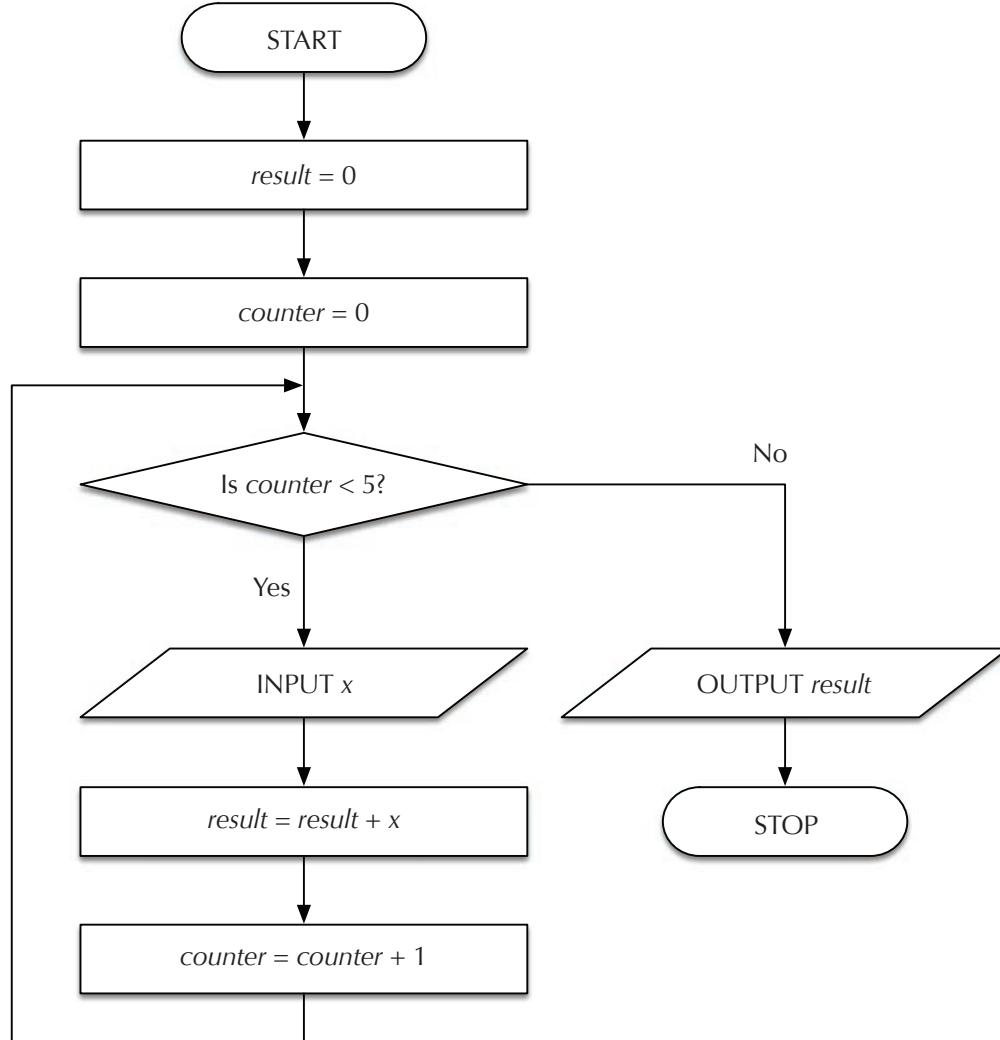
Input data that is used to perform a dry run

Trace table

Tabulation of the change in variables as well as the output of an algorithm when performing a dry run

3.3.2 Producing a Trace Table

Suppose we perform a dry run on the flowchart below for the addition of five numbers using test data of 10, 20, 30, 40 and 50:



▲ **Figure 3.20** Solution to the problem of adding five numbers that uses iteration and a counter variable

Table 3.4 shows the completed trace table from the dry run.

▼ **Table 3.4** Completed trace table from the dry run

result	counter	x	OUTPUT
0			
	0		
		10	
10			
	1		
		20	
30			
	2		
		30	
60			
	3		
		40	
100			
	4		
		50	
150			
	5		
			150

A trace table always has one column for every variable used in the flowchart and an additional OUTPUT column on the right. A new row is added to the trace table each time output is produced or if the value in a variable is changed during the dry run. The corresponding output or new variable value is then recorded under the appropriate column.

3.3.3 Determining an Algorithm's Purpose

A trace table can help us determine the purpose of an algorithm or what the variables in a flowchart mean by using some test data.

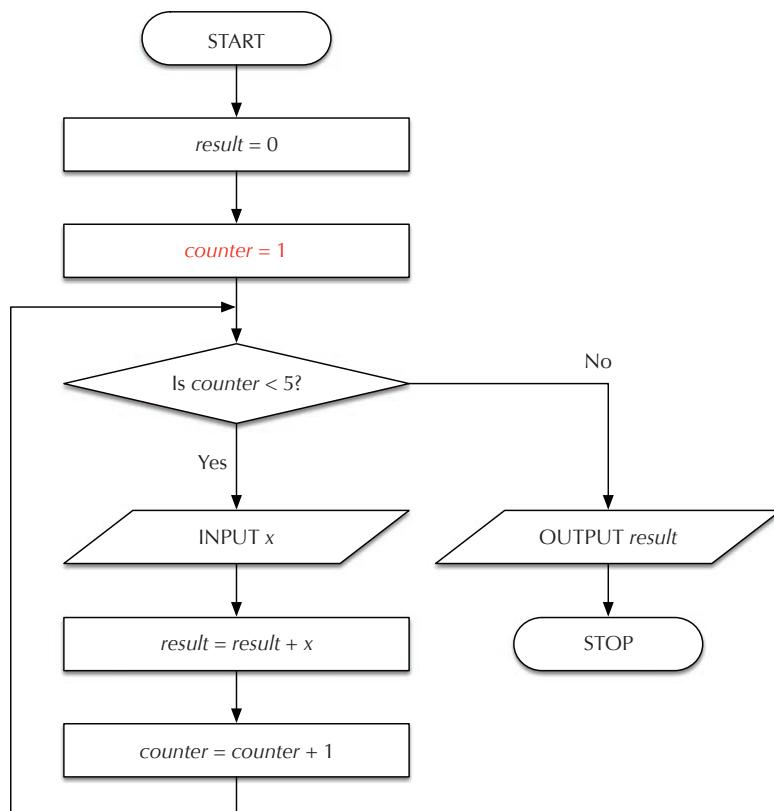
For instance, by studying the completed trace table in the previous example, we can see that:

1. The values of x follow the sequential order of the test data. Thus, we can conclude that the purpose of x is to store the current input for processing.
2. The value of $result$ starts from 0 and each subsequent iteration increases the value of $result$ by the value of x . Thus, we can conclude that the purpose of $result$ is to store the cumulative sum of inputs.
3. The value of $counter$ starts from 0 and increases by 1 each time $result$ is updated. Thus, we can conclude that the purpose of $counter$ is to track the number of processed inputs.
4. The output is given by the last value of $result$. Thus, we can conclude that the overall purpose of the algorithm is to calculate the sum of all the inputs.

3.3.4 Correcting an Algorithm

A trace table can also help us determine why an algorithm is not working as expected or does not give the correct output for certain test data.

For instance, suppose that the addition algorithm incorrectly initialises $counter$ to 1 instead of 0:



▲ **Figure 3.21** Incorrect solution for the problem of adding five numbers

When we perform a dry run with test data of 10, 20, 30, 40 and 50, we get the trace table as shown in Table 3.5.

▼ **Table 3.5** Trace table using an incorrect algorithm for the problem of adding five numbers

result	counter	x	OUTPUT
0			
	1		
		10	
10			
	2		
		20	
30			
	3		
		30	
60			
	4		
		40	
100			
	5		
			100

From the trace table, we can see that only four of the five inputs are actually used and that the algorithm ends earlier than expected to give the wrong output of 100 instead of 150.

Recall that *counter* is supposed to keep track of how many inputs have been processed so far. Thus, we can easily determine that initialising *counter* to 1 is incorrect since no inputs have been processed at the start of the algorithm.

You will learn more about other methods to correct algorithm errors in Chapter 6.

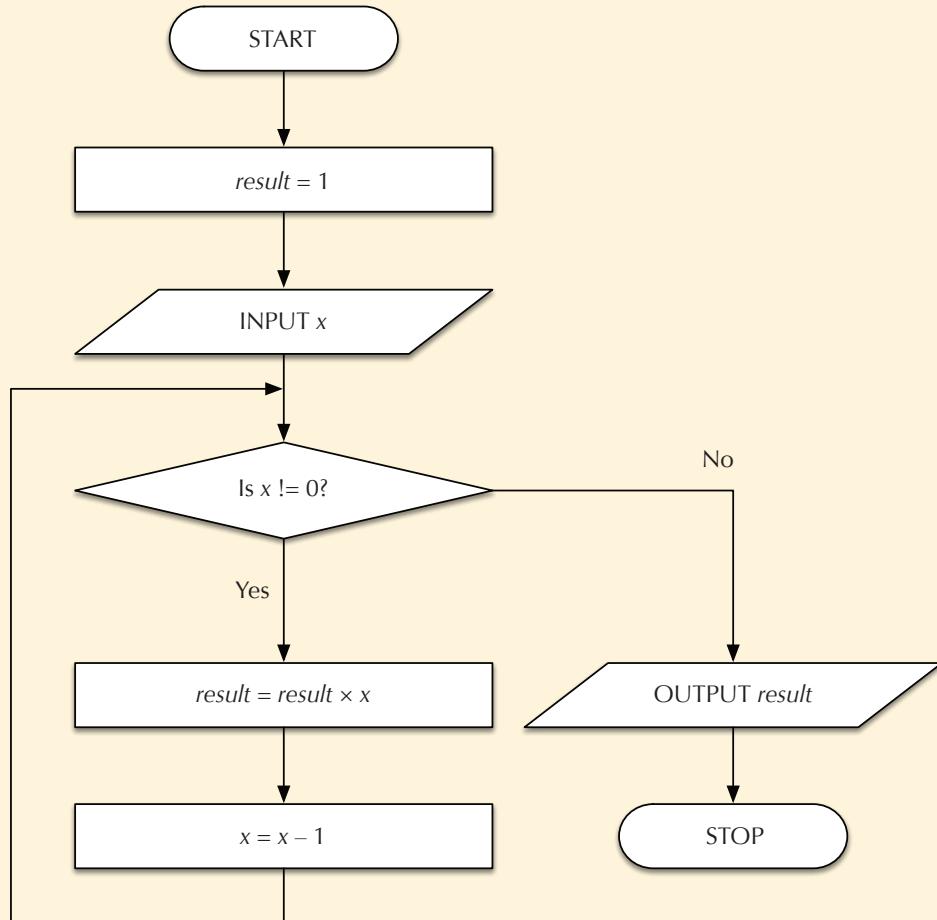
Did you know?

It is also possible to change the instruction in the decision symbol from “Is *counter* < 5?” to “Is *counter* < 6?” in order to correct the algorithm. However, this is not a good programming practice as *counter* is supposed to keep count of the number of times the loop has been repeated. Changing the instruction causes *counter* to be incremented from 1 to 6 instead of from 0 to 5. This does not follow the original meaning of *counter* and makes it harder to describe in plain English what the variable is keeping track of.



Quick Check 3.3

- The following algorithm takes in a positive whole number as input:



▲ Figure 3.22 Algorithm that takes in a positive whole number

- Draw and complete a trace table for this algorithm using test data of 3.
- Draw and complete a trace table for this algorithm using test data of 5.
- Describe the purpose of this algorithm.

3.4 Pseudo-Code

Besides flowcharts, algorithms can also be represented using **pseudo-code**. For instance, the pseudo-code below is another way of representing our solution to the problem of adding five numbers:

```

result = 0
counter = 0
WHILE counter < 5
    INPUT x
    result = result + x
    counter = counter + 1
ENDWHILE
OUTPUT result
```

▲ **Figure 3.23** Representing the solution to the problem of adding five numbers using pseudo-code

Did you know?

There are many ways to write pseudo-code. The pseudo-code presented in this textbook is one way of writing it. In other books, the pseudo-code may appear differently.

Note that in pseudo-code, each step is written on a separate line, just like how each step is represented by a separate symbol in a flowchart. However, unlike flowcharts, there are no flow lines to indicate the order in which the steps are followed. Instead, the order is based on sequence, selection and iteration constructs similar to the ones you have learnt for flowcharts.

Key Term

Pseudo-code

Description of an algorithm using natural language (such as English) mixed with sequence, selection and iteration constructs

3.4.1 Input/Output

You have learnt that the input and output requirements are critical factors in deriving an algorithm. When using pseudo-code, the INPUT command is used to receive data from outside the algorithm and the OUTPUT command is used to produce a result from within the algorithm. These statements serve a similar purpose as the data symbol in a flowchart.

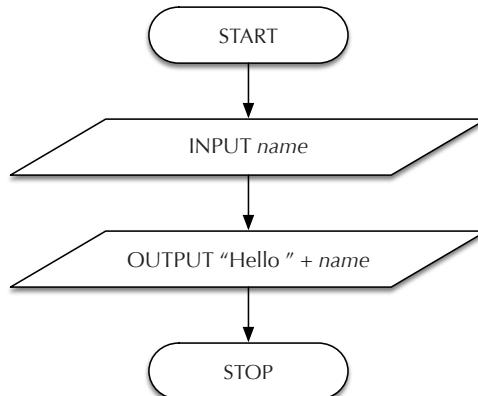
For example, a user is prompted to enter his or her name before a greeting with the user's name is produced:

```
INPUT name
OUTPUT "Hello " + name
```

▲ **Figure 3.24** Example of sequence pseudo-code

3.4.2 Sequence Constructs

It can be seen from Figure 3.24 that if no special commands are used, consecutive lines of pseudo-code are followed as if they were already arranged in a sequence construct. For instance, Figure 3.25 presents the same algorithm used in Figure 3.24 as a flowchart:



▲ **Figure 3.25** Example of sequence pseudo-code presented as a flowchart

3.4.3 Selection Constructs

Selection constructs are used when a choice between two or more actions has to be made based on a condition or the answer to a question.

Similar to the decision symbol in a flowchart, the **IF...THEN...ELSE...ENDIF** commands in pseudo-code are used to determine which steps are followed based on a condition or the answer to a question. The condition or question is provided between the words **IF** and **THEN**. If the answer is “Yes”, the first set of steps after the **IF** line are followed. If the answer is “No”, the second set of steps after the **ELSE** line are followed. After one of these two sets of steps is completed, the steps after the **ENDIF** line (if any) are followed.

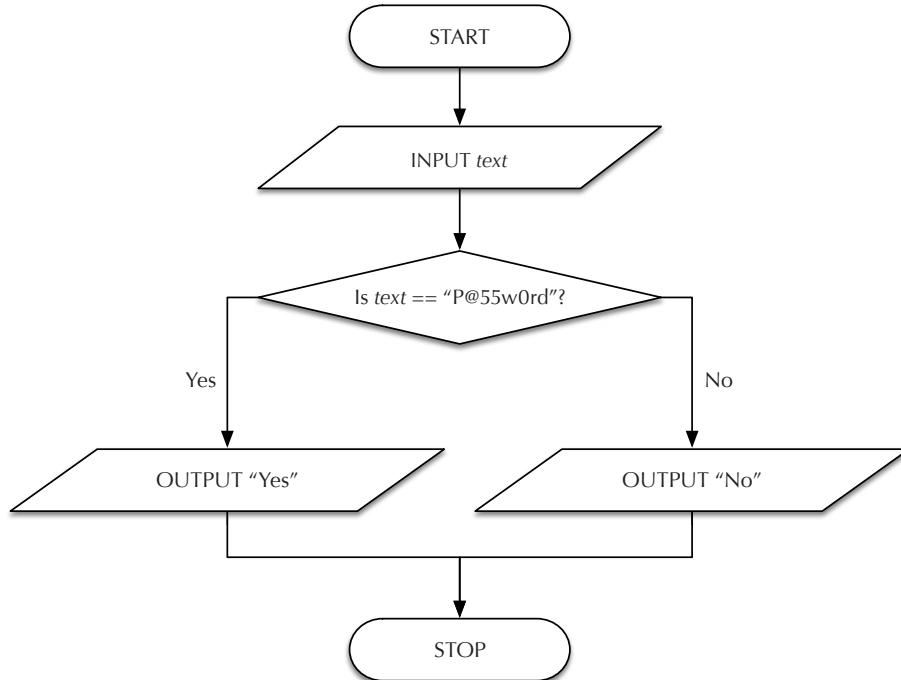
For example, Figures 3.26 and 3.27 represent the same algorithm. In this case, the person or computer following the algorithm chooses whether to output "Yes" or "No" depending on whether text matches the special phrase "P@55w0rd".

```

INPUT text
IF text == "P@55w0rd" THEN
    OUTPUT "Yes"
ELSE
    OUTPUT "No"
ENDIF

```

▲ **Figure 3.26** Example of IF...THEN...ELSE...ENDIF pseudo-code



▲ **Figure 3.27** Example of IF...THEN...ELSE...ENDIF pseudo-code presented as a flowchart

In pseudo-code, certain steps are indented to show that they are part of the selection construct.

Note that the word `ELSE` and the second set of steps are optional. If they are left out and the answer to the question is "No", then the steps after the `ENDIF` line (if any) are followed immediately.

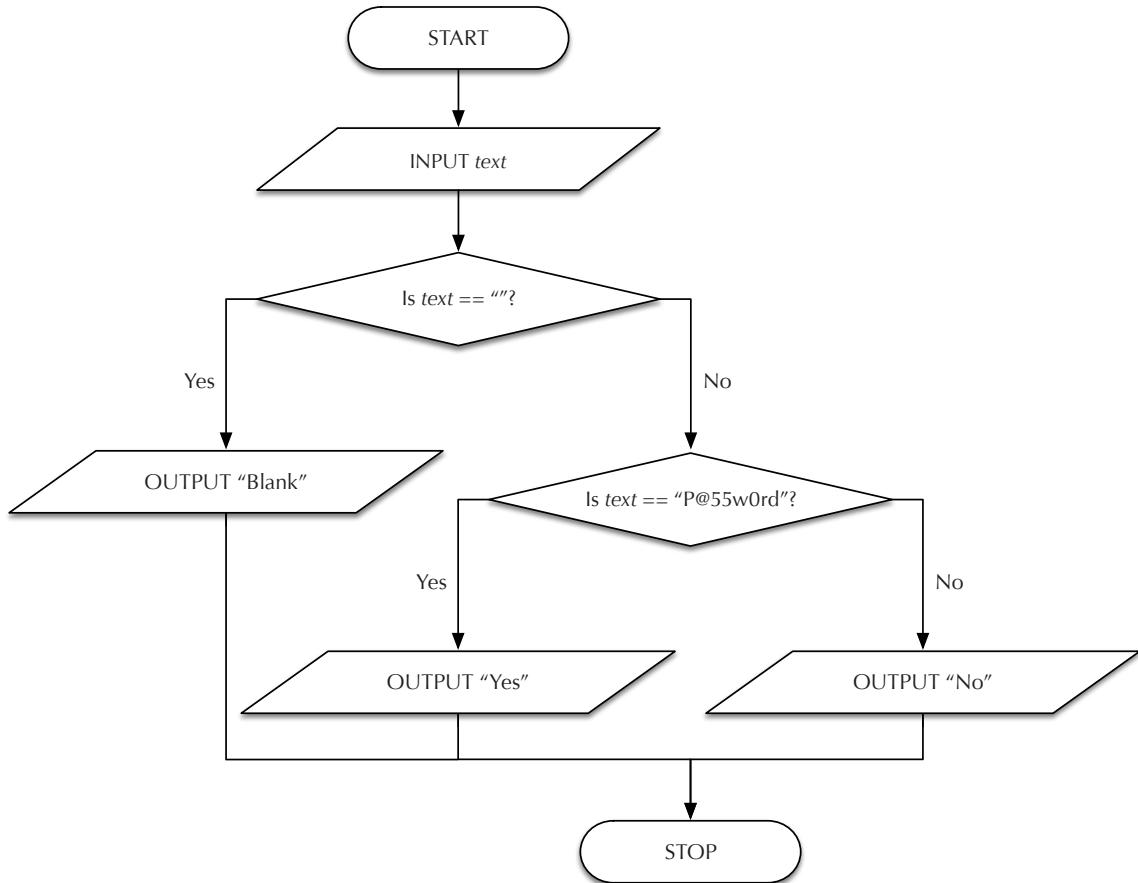
Sometimes the set of steps after the `ELSE` line is another `IF...THEN...ELSE...ENDIF` command. In such cases, the two `IF...THEN...ELSE...ENDIF` commands can be combined into a single `IF...THEN...ELSEIF...ELSE...ENDIF` command. For instance, Figures 3.28, 3.29 and 3.30 all represent the same algorithm where the output is "Blank" if *text* happens to be empty.

```
INPUT text
IF text == "" THEN
    OUTPUT "Blank"
ELSE
    IF text == "P@55w0rd" THEN
        OUTPUT "Yes"
    ELSE
        OUTPUT "No"
    ENDIF
ENDIF
```

▲ **Figure 3.28** Example where the set of steps after `ELSE` is just another `IF...THEN...ELSE...ENDIF` command

```
INPUT text
IF text == "" THEN
    OUTPUT "Blank"
ELSEIF text == "P@55w0rd" THEN
    OUTPUT "Yes"
ELSE
    OUTPUT "No"
ENDIF
```

▲ **Figure 3.29** Example of equivalent `IF...THEN...ELSEIF...ELSE...ENDIF` pseudo-code



▲ **Figure 3.30** Example of `IF...THEN...ELSEIF...ELSE...ENDIF` pseudo-code presented as a flowchart

3.4.4 Iteration Constructs

Iteration constructs are used to repeat instructions. In pseudo-code, there are two sets of commands that can be used to create iteration constructs:

1. WHILE...ENDWHILE
2. FOR...NEXT

3.4.4.1 WHILE...ENDWHILE

The `WHILE...ENDWHILE` commands are used to repeat a set of steps as long as a condition that is provided on the same line as the word `WHILE` is true. The condition is checked the first time the `WHILE` command is encountered and each time the set of steps is completed.

The lines between `WHILE` and `ENDWHILE` are the set of steps to repeat, and if the condition is found to be false, the steps after the `ENDWHILE` line (if any) are followed instead.

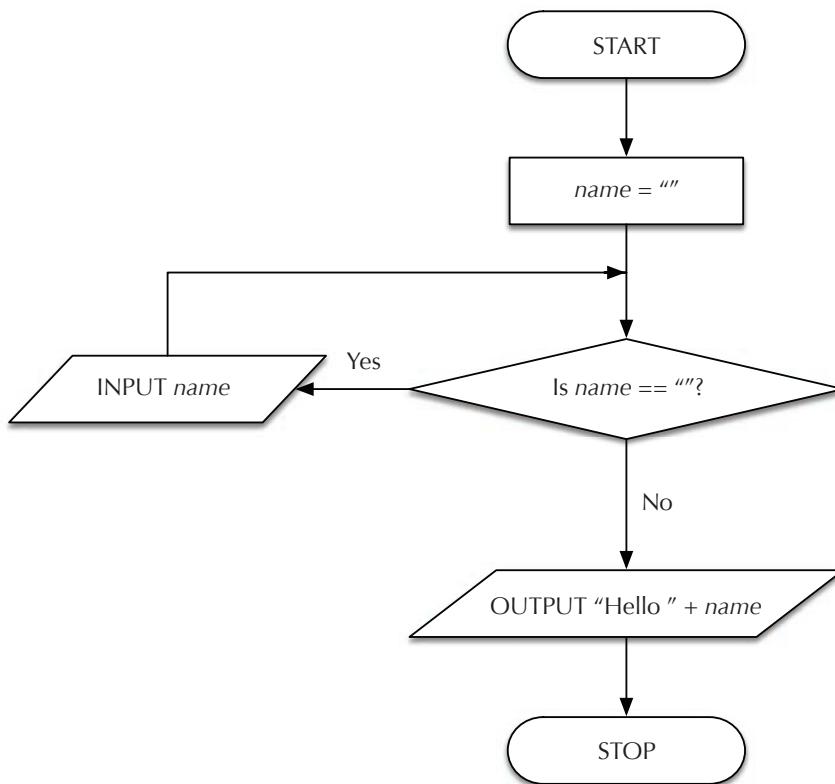
For example, Figures 3.31 and 3.32 represent the same algorithm. In this case, we wish to keep performing the INPUT command as long as *name* is a blank phrase to ensure that a non-blank *name* is provided. The INPUT command is thus placed between the WHILE and ENDWHILE lines.

```

name = ""
WHILE name == ""
    INPUT name
ENDWHILE
OUTPUT "Hello " + name

```

▲ **Figure 3.31** Example of WHILE...ENDWHILE pseudo-code



▲ **Figure 3.32** Example of WHILE...ENDWHILE pseudo-code presented as a flowchart

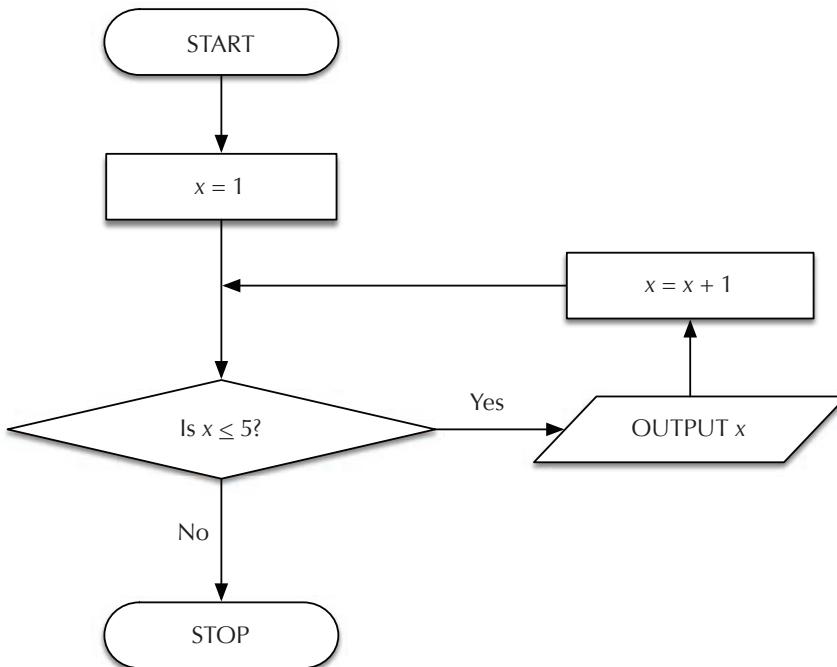
3.4.4.2 FOR...NEXT

The FOR...NEXT commands are used as a shortcut for repeating a set of steps using a counter or when the number of repetitions is known beforehand. The FOR command goes through the values in a sequence using a variable and repeats the set of steps between FOR and NEXT for each value. After the set of steps has been repeated for each value in the sequence, the algorithm continues with the steps that follow after the NEXT command (if any).

For example, Figures 3.33 and 3.34 represent the same algorithm. In this case, we use the variable x to repeat an **OUTPUT** command while x is incremented in whole numbers from 1 to 5. Each time the **NEXT** command is encountered, the person or computer following this algorithm checks whether there is a next value for x , and if so, updates x to the next value before repeating the **OUTPUT** command again. Once there are no further values for x , the **OUTPUT** command is no longer repeated.

```
FOR x = 1 to 5
    OUTPUT x
NEXT x
```

▲ Figure 3.33 Example of FOR...NEXT pseudo-code

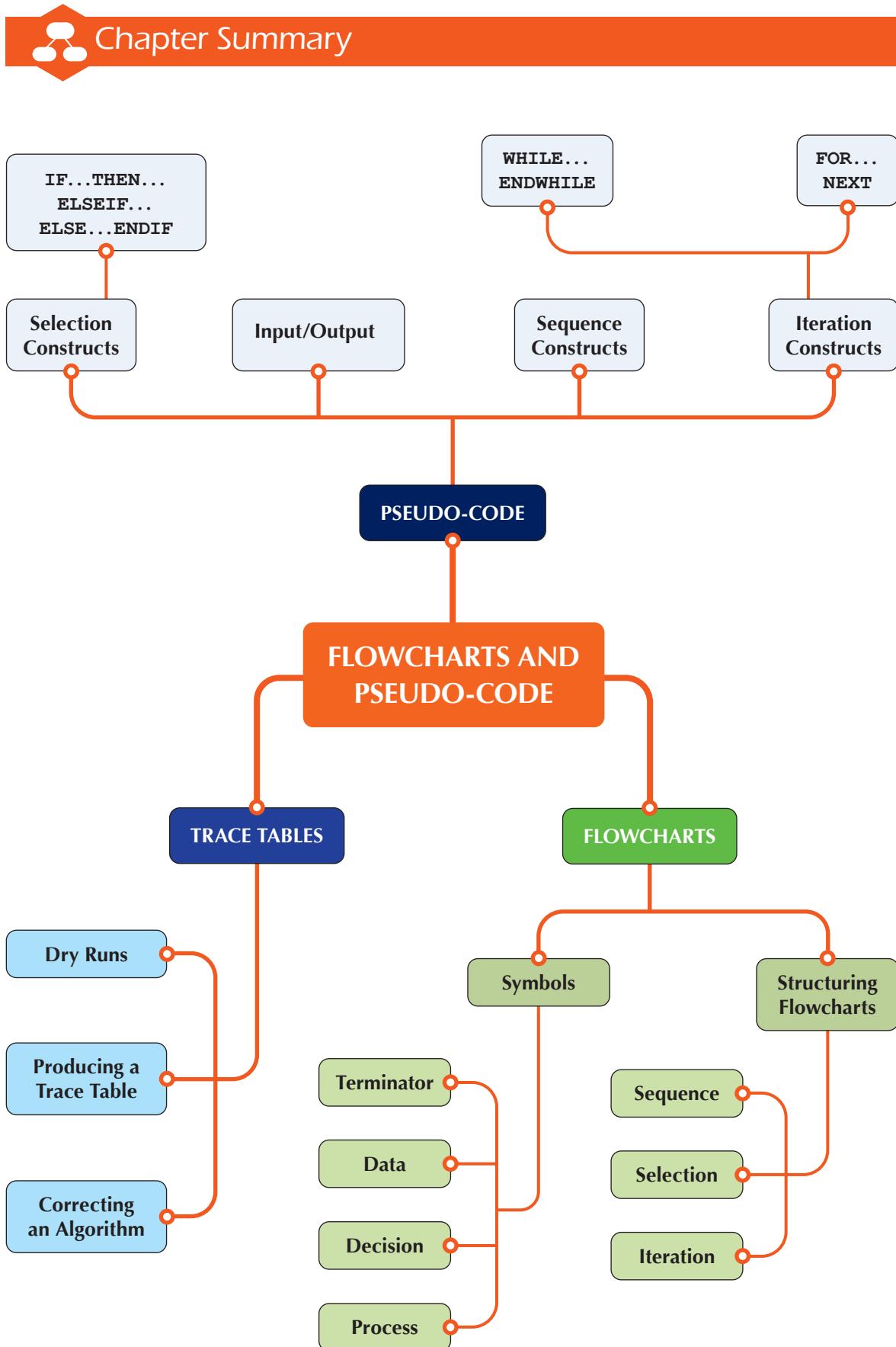


▲ Figure 3.34 Example of FOR...NEXT pseudo-code presented as a flowchart



Quick Check 3.4

- Choose an appropriate construct and write the pseudo-code using this construct to represent an algorithm that takes in a whole number from 1 to 7 inclusive and produces the corresponding number in words (for example, the input 5 produces the output “five”). Specify the input and output requirements for the problem that this algorithm solves.
- Choose an appropriate iteration construct and write the pseudo-code to represent an algorithm that takes in 10 numbers and outputs the average of the 10 numbers. Specify input and output requirements for the problem that this algorithm solves and complete a trace table using the following test data: 20, 17, 19, 65, 20, 18, 19, 66, 20 and 19.





Review Questions

1. A simple algorithm is provided below:

Step 1: Get the input value x .

Step 2: Let $counter$ be a variable. Initialise it to 0.

Step 3: If x is 1, proceed to Step 6.

Step 4: If x is even, divide it by two and store the answer back in x . Otherwise, multiply x by 3 and add 1, then store the answer back in x .

Step 5: Add 1 to $counter$ and store the answer back in $counter$. Go back to Step 3.

Step 6: Output the value of $counter$.

a) Express this algorithm using a flowchart.

b) Produce a trace table for this algorithm with test data of 5.

c) Produce a trace table for this algorithm with test data of 6.

d) Express this algorithm in pseudo-code.

2. The temperature in a server room should be kept between 18 °C and 20 °C for the servers to function optimally. This is done by a computer continually checking the temperature. When the temperature exceeds 20 °C, the air conditioner is switched on. When the temperature falls below 18 °C, the air conditioner is switched off. Otherwise, the air conditioner's state is unchanged.

Express the algorithm as a flowchart. (Note that unlike normal flowcharts, this flowchart will not have an end terminator symbol.)

3. Table 3.6 shows the exchange rate between Singapore Dollars (SGD) and Malaysian Ringgit (MYR) as well as between SGD and Chinese Yuan (CNY).

▼ **Table 3.6** Exchange rate between SGD, MYR and CNY

SGD	MYR	CNY
1	3	4.66

Using pseudo-code, express an algorithm which:

- takes in the choice of either “Ringgit” for conversion from SGD to MYR or “Yuan” for conversion from SGD to CNY;
- takes in the amount to convert in SGD; and
- returns the converted amount in either MYR or CNY as the output, depending on the option chosen.

You may assume that the provided input will always be valid.

4. The pseudo-code in Figure 3.35 takes in the type of co-curricular activity ("Club/Society", "Sport" or "Uniformed Group") for five students, and then outputs the number of students in each type of co-curricular activity.

```

1 Count = 0
2 ClubSociety_Count = 0
3 Sport_Count = 0
4 UniformedGroup_Count = 0
5
6 WHILE Count <= 5
7     INPUT Type
8     IF Type == "Club/Society" THEN
9         ClubSociety_Count = ClubSociety_Count + 1
10    Count = Count + 1
11    ELSEIF Type == "Sport" THEN
12        Sport_Count = Sport_Count + 2
13        Count = Count + 1
14    ELSEIF Type == "Uniformed Group" THEN
15        UniformedGroup_Count = UniformedGroup_Count + 1
16        Count = Count + 1
17    ELSE
18        OUTPUT "Invalid Option"
19    ENDIF
20 ENDWHILE
21
22 OUTPUT "Clubs and Societies: " + ClubSociety_Count
23 OUTPUT "Sports: " + Sport_Count
24 OUTPUT "Uniformed Groups: " + Sport_Count

```

▲ **Figure 3.35** Pseudo-code with three logic errors

Locate and correct three errors in the pseudo-code. Line numbers have been provided for your reference. You may also wish to create some test data and use trace tables to find out where these errors are located.

5. The following pseudo-code in Figure 3.36 takes in seven numbers and calculates the sum and average of these numbers as the outputs.

```
Count = 0
Sum = 0

WHILE Count != 7
    INPUT Num
    Sum = Sum + Num
    Count = Count + 1
ENDWHILE

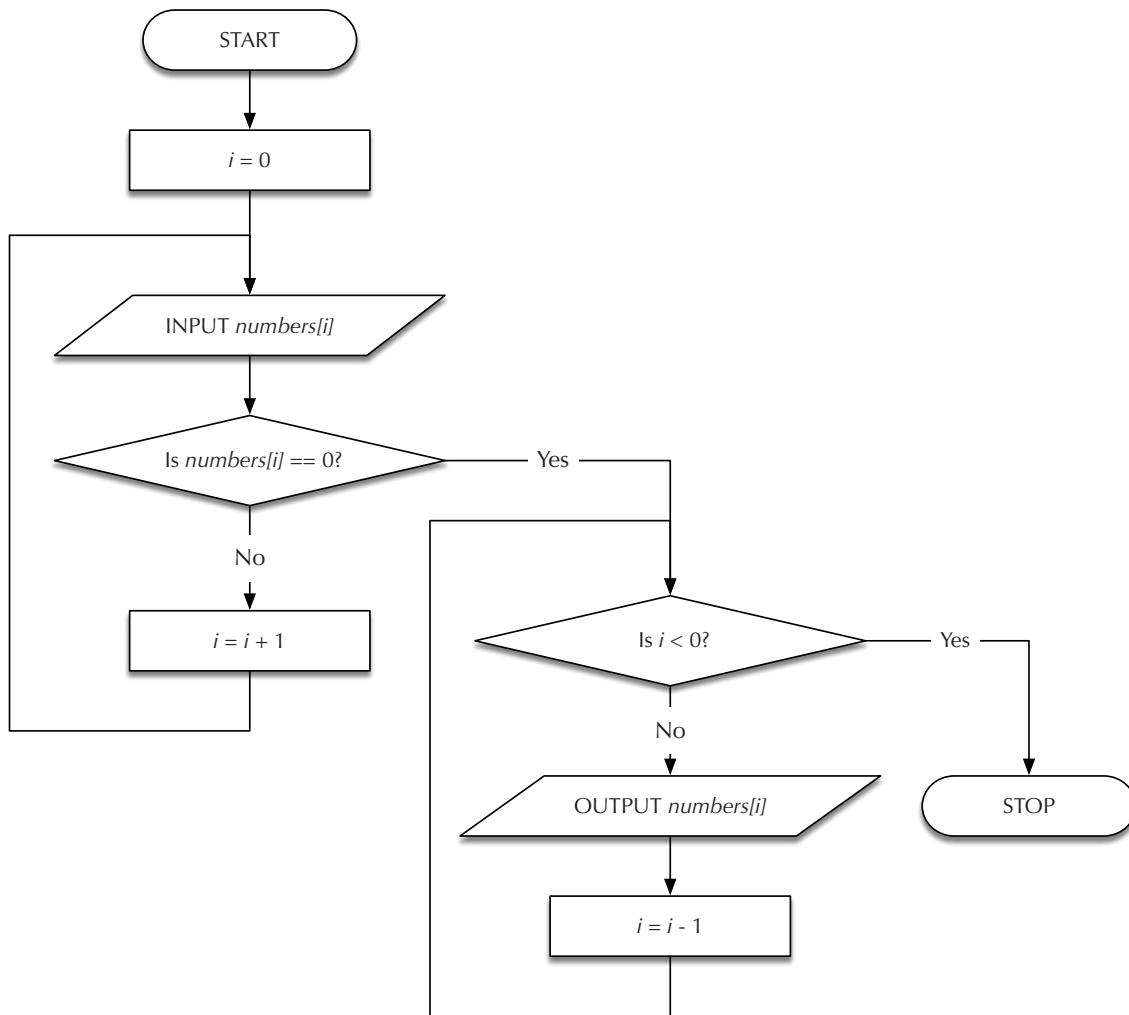
Average = Sum / Count
OUTPUT "The sum is " + Sum
OUTPUT "The average is " + Average
```

▲ **Figure 3.36** Pseudo-code to calculate the sum and average of seven numbers

Given the following test data, draw and complete a trace table. (Provide values of up to 2 decimal places where necessary.)

Test data: 7, 15, 11, 18, 5, 17, 1

6. The following flowchart takes in a sequence of non-zero integers as input. The input ends when a zero is detected.



▲ Figure 3.37 Flowchart that takes in a sequence of non-zero integers as input

- a) Express the flowchart in pseudo-code.
- b) Given the following test data, draw and complete a trace table. (Note that the value of a list variable containing multiple values can be written as its individual values separated by commas, e.g., 14, 3, 21, 32)

Test data: 20, 17, -19, 65, 0

- c) State the purpose of this flowchart.

How Do I Write Programs?

Alex helps out at his school library and uses a simple algorithm to find out which loaned books are overdue. He finds it very tedious because there are too many loaned books. How can Alex use a computer to make his work easier?



In the previous chapters, we learnt about how to express algorithms as flowcharts. Well-designed flowcharts allow algorithms to be written down in a standard format that people can follow.

However, we often want to automate the implementation of algorithms because some algorithms can be slow, tedious or prone to errors when done manually, especially when the number of inputs becomes very large.

When writing algorithms for computers, we need to consider that computers require algorithms to be expressed more precisely than humans would. This chapter will introduce the Python programming language and how we can implement algorithms using a computer.



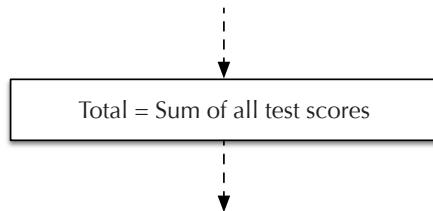
By the end of this chapter, you should be able to:

- Explain how translators are used to detect syntax errors and state the difference between interpreter and compiler translators.
- Use and justify the use of variables, constants and simple lists in different problem contexts.
- Understand and use different data types (integers, floating-point numbers, strings, Booleans, lists) and built-in functions in programs.
- Write and use user-defined functions.
- Understand and use sequence, selection and iteration constructs to create a program.

4.1 Flowcharts and Programs

Just as flowcharts present a set of activities and decisions for a person to understand, programs list a set of instructions for a computer to follow. However, computers do not have the same common sense and knowledge of natural languages (such as the English language) that humans have, so without additional information, they can only understand a very limited set of instructions.

For instance, a flowchart meant for people to calculate the total score of a test may have the following instruction:

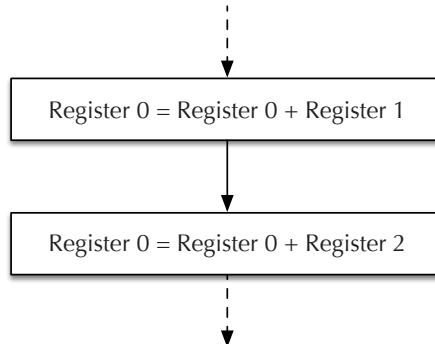


▲ **Figure 4.1** Example of a flowchart instruction

Without more information, a computer would not be able to understand how to carry out this instruction. Some examples of additional information needed would be:

- The meaning of “test scores”: What type of data is “test scores” and how are they stored in (or received by) the computer?
- The meaning of “all”: How many test scores are there, or how do we know when the sum is complete?

For a computer to follow the instruction, it must be decomposed and re-written into steps that the computer can understand. For instance, if the test scores are stored in processor registers 0, 1 and 2, and we want the computer to store the sum of the test scores back in processor register 0, we can write a set of instructions as shown in Figure 4.2.



▲ Figure 4.2 Decomposing the flowchart instruction

Even though we have decomposed the instruction into steps, it is still not in a format that a computer's processor can use directly. Since all data in a computer is represented as bytes, we have to write the steps using bytes as well.

We do this by using the processor's **instruction set**, which specifies how the instructions that a processor understands directly can be represented as bytes.

When instructions are written in a form that can be run directly on a computer's processor, we call these instructions **machine code**. A set of such instructions is called a **program**.

The steps in Figure 4.2 can be re-written as:

Key Terms

Instruction set

Set of basic commands or instructions that a processor can understand directly, represented in byte form

Machine code

Instructions that can run directly on a computer's processor

Program

Set of instructions written to perform specific tasks on a computer

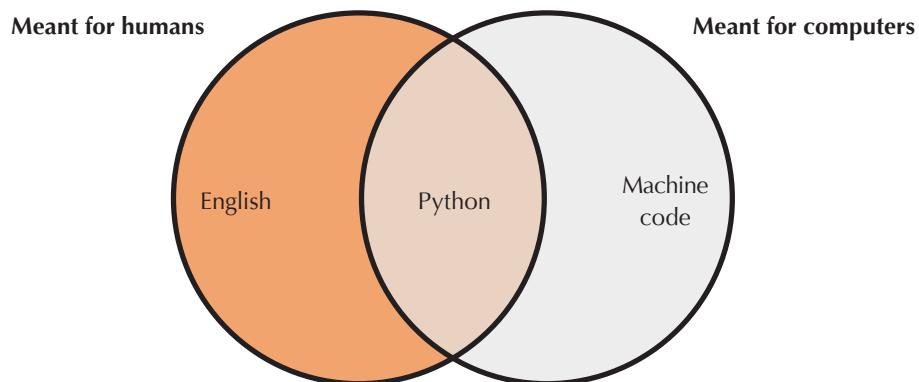
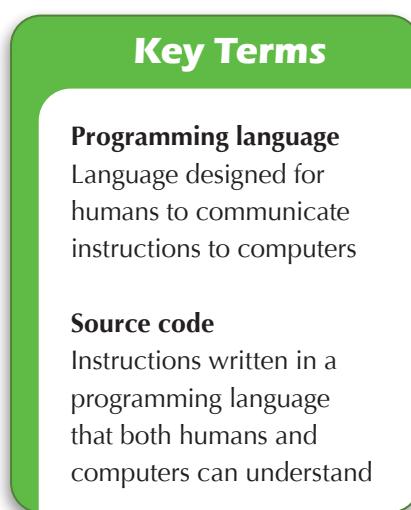
00000000 11001000 00000000 11010000

▲ Figure 4.3 Representing the decomposed instructions as bytes

4.2 Programming Languages

Machine code is not easy for humans to read or write. Most people do not write machine code manually. However, an algorithm's instructions must be provided in machine code in order for them to be carried out by a computer.

Thankfully, we do not have to write machine code manually. Instead, programs are usually written with a **programming language** such as Python. Just as how a common language such as English allows people from different cultures to communicate with each other, programming languages can serve as a common language between humans and computers because they are both simple enough for humans to write and specific enough for computers to follow. Figure 4.4 illustrates this.



▲ **Figure 4.4** Comparing languages for humans and computers

Instructions written in a programming language are called **source code**, source, or even just code for short. The following is an example of Python source code:

```
total = test_score_1 + test_score_2
```

▲ **Figure 4.5** Example of Python source code

Source code is easy for humans to understand. For instance, you can infer from Figure 4.5 that `total` is the sum of `test_score_1` and `test_score_2`.

Source code is translated into machine code by special translator programs. However, in order to successfully translate source code into machine code, the source code must follow particular rules that dictate how the words and symbols (such as punctuation marks) in each instruction must be arranged. These rules are called **syntax**.

When source code does not follow the rules of the language, it results in a **syntax error**. This type of mistake is detected by the translator program and the incorrect source cannot be translated into machine code. Therefore syntax rules must be followed when writing source code.

Besides Python, there are a variety of other programming languages. Table 4.1 shows some examples of source code in other programming languages such as C, Pascal, Scratch and Lisp.

▼ **Table 4.1** Examples of source code in other programming languages

Programming language	Source code
C	total = test_score_1 + test_score_2;
Pascal	total := test_score_1 + test_score_2;
Scratch	
Lisp	(defvar *total* (+ *test_score_1* *test_score_2*))

In this example, the C and Pascal source codes look very similar to Python source code but Scratch and Lisp source codes are quite different. It is sometimes useful to recognise that the same algorithm can be represented in a variety of ways depending on the programming language used.

Did you know?

Unlike most other programming languages, the source code in Scratch is not typed out but assembled by dragging and dropping graphical blocks. Scratch is a project of the Lifelong Kindergarten Group at the MIT Media Lab.

Key Terms

Syntax

Rules that determine how the words and symbols in a valid instruction are arranged

Syntax error

Result of incorrect source code that does not follow the rules of the language

4.3 Compilers and Interpreters

We have learnt that only machine code can be run directly on a computer's processor. When we use a programming language, the source code must be translated into machine code before it can be run.



▲ **Figure 4.6** A translator translates the source code into machine code

There are two ways by which source code is translated:

1. Compiler

A code translator program, called a **compiler**, reads the source code, performs the translation once, and then stores the translated machine code in the computer. When the program is run, the machine code that was compiled previously is reused and the compiler is no longer needed for the program to function.

2. Interpreter

A code translator program, called an **interpreter**, runs every time the program starts. The interpreter reads the source code while the program is running and either performs on-the-spot translation or determines which part of its own machine code to run. In general, any translated machine code is discarded after the program is stopped and the interpreter is needed every time the program is run.

Key Terms

Compiler

Code translator program that translates source code into machine code completely before running the compiled program

Interpreter

Code translator program that translates source code into machine code while the interpreted program is running

C and Pascal are examples of programming languages that typically use a compiler. Scratch and Python are examples of programming languages that typically use an interpreter. Lisp is an example of a programming language where both compilers and interpreters are commonly used.

Each translation method has its own advantages and disadvantages, which are summarised in Table 4.2.

▼ **Table 4.2** Comparing compilers and interpreters

	Compilers	Interpreters
Advantages	<ul style="list-style-type: none"> The resulting program runs at a faster speed because all the translation has been done beforehand. The compiler is not needed to run the program after compilation is complete. Syntax errors are detected before the program is even run. 	<ul style="list-style-type: none"> Changes to the source code take effect immediately. Interpreters usually offer an interactive mode, which facilitates learning and experimentation.
Disadvantages	<ul style="list-style-type: none"> Any changes to the source code require recompilation before taking effect. Compilers usually do not offer an interactive mode. 	<ul style="list-style-type: none"> The resulting program runs at a slower speed because translation occurs while the program is running. The interpreter needs to be run every time the program is started. Syntax errors may interrupt the running of a program.

4.3.1 Using the Python Interpreter

The Python programming language that you will learn in this textbook uses an interpreter. This interpreter can be downloaded for free online.

Note that there are two slightly incompatible versions of Python that are commonly used today, namely Python 2 and Python 3. In this textbook, we will only use Python 3.

There are two ways to use the Python interpreter:

1. **Graphical user interface**

The Python interpreter comes with a program named IDLE with menus and **syntax highlighting** to facilitate the writing and running of Python source code. IDLE is most useful if the programmer wishes to use an “all-in-one” program to edit, run and test his or her source code. Many alternatives to IDLE are also available for download online.

2. **Command line interface**

The Python interpreter can also be run using a command line interface where commands are given as lines of text. This interface is most useful for combining the use of Python with other programs or for automating the use of the Python interpreter.

Key Terms

Command line interface

Means of interacting with a program such that commands are given as lines of text

Graphical user interface

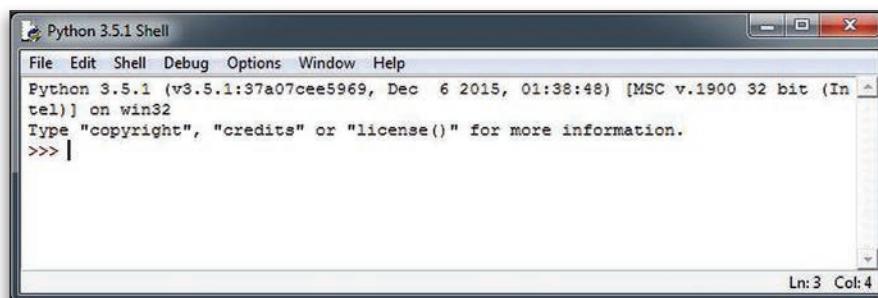
Means of interacting with a program such that commands are given using visual elements such as windows, icons, menus and mouse pointers

Syntax highlighting

Displaying different parts of source code differently (such as using different colours and fonts) in a text editor based on the rules of the programming language to make reading easier

4.3.2 Graphical User Interface (IDLE)

To start IDLE on Windows, press the Start key, type “idle” and press Enter. Figure 4.7 shows what IDLE looks like when it is initially run on Windows:



▲ Figure 4.7 IDLE on initial start-up

This main window that appears when IDLE starts is called the Python Shell window or shell window for short. This is where the output appears and where the input is provided via the keyboard. If there is no program running, the shell window will have the Python interpreter running in **interactive mode** so you can interact with data and try out short examples of Python code quickly.

Key Term

Interactive mode

Feature of Python that allows source code to be entered, line by line, for the interpreter to run immediately and show the results

Did you know?

IDLE is an example of an “integrated development environment” (IDE) that makes it easier to use Python by providing an “all-in-one” program for editing, running and testing source code. The name IDLE officially stands for “Integrated Development and Learning Environment”, but it is also meant to honour Eric Idle, a member of the British comedy group Monty Python from which Python got its name.

Python’s interactive mode is where you can enter source code, line by line, for the interpreter to run immediately. This mode is useful for learning and experimenting with Python source code. For instance, try entering the following line of Python source code in the shell window and pressing Enter:

```
>>> print("Hello, World!")
```

▲ Figure 4.8 Entering source code in interactive mode

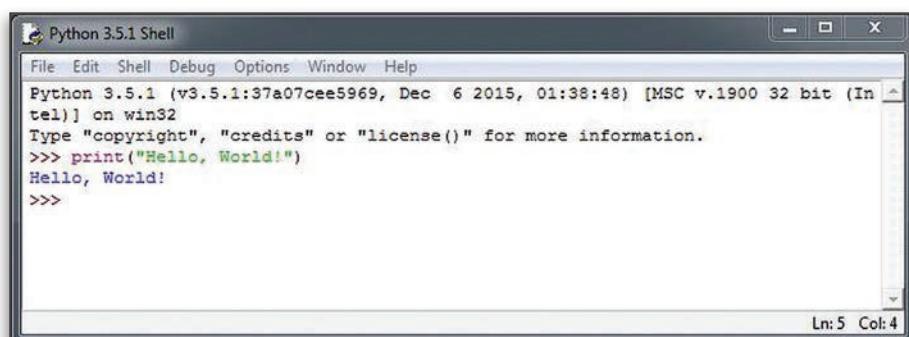
Note that for the interactive mode examples given in this textbook, anything that you may need to type will be shown in **bold**. Any text that is not in bold is output from either your program or the interpreter and you will not need to type them in, such as the >>> prompt displayed by Python. In addition, you will usually need to press Enter after typing each line of code.

As Python uses an interpreter, no compilation step is needed and the source code is run immediately. If there are no errors, you should see the phrase “Hello, World!” displayed on the screen:

```
>>> print("Hello, World!")
Hello, World!
```

▲ Figure 4.9 Output in interactive mode

This is what the result looks like on Windows:



▲ Figure 4.10 Screenshot of output in IDLE’s shell window

If there is an error, Python will display an error message, such as the example in Figure 4.11.

```
>>> Print("Hello, World!")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    Print("Hello, World!")
NameError: name 'Print' is not defined
```

▲ **Figure 4.11** Example of an error message being displayed in interactive mode

Note that the Python programming language is case-sensitive and that “Print” (with an upper-case “P”) is not the same as “print” (with a lower-case “p”), as shown by the error in Figure 4.11. It is typical to encounter such errors when learning a new programming language. You will learn more in Chapter 6 on what to do if a program does not work as intended.

Besides using the interactive mode, you can also save your Python programs as plain text files with the .py file extension. You can write your Python programs using Notepad or any other text editor.

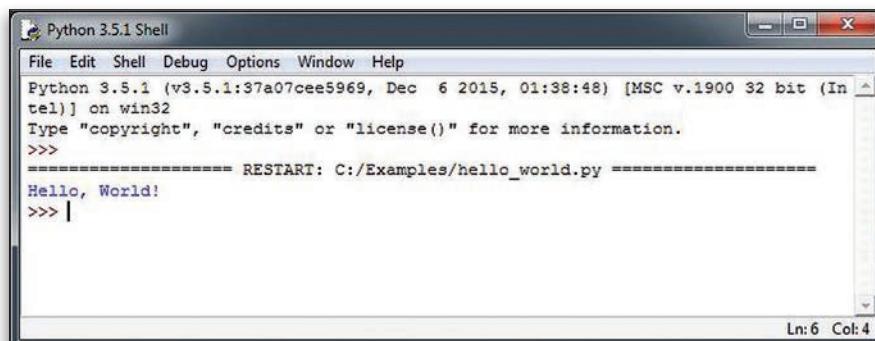
To use IDLE as a text editor, go to the shell window, open the “File” menu, and select “New File”. A new editor window should open for you to enter your Python program. To save your Python program as a file, go to the editor window, open the “File” menu, and select “Save”. If you are working on a school computer, your teacher will inform you of where your programs should be saved. For this textbook, we will save our programs in a folder named c:\Examples.

The following is an example of a Python program that can be saved as a file:

#	Program 4.1 hello_world.py
1	print("Hello, World!")

▲ **Figure 4.12** “Hello, World!” program in Python

To run the program, save the file by pressing Ctrl+S, then press the F5 key while the file is open in IDLE. The phrase “Hello, World!” will be displayed in the shell window:



▲ **Figure 4.13** Output of hello_world.py in IDLE’s shell window

Note that, unlike in interactive mode, the result for each line of source code is not automatically displayed when running a Python program that is saved in a file. To display text on the screen when running a Python program that is saved in a file, you must use the `print()` function (see section 4.6.1 on functions).

When you save a Python program in a file, you should also add **comments** to your program. Comments make the source code easier for other people to understand but will be ignored by the Python interpreter. In Python, comments start with the hash symbol (#) and finish when the line ends. Any text from the hash symbol until the end of the line will be ignored by the interpreter. For example, Figure 4.14 shows `hello_world.py` with some additional comments:

Key Term

Comment

Piece of source code that explains the program but is ignored by the Python interpreter

#	Program 4.2 hello_world_with_comments.py
1	<code># This is a comment at the start of a program. Usually, comments at</code>
2	<code># the start of a program explain the purpose of a program or</code>
3	<code># describe its expected inputs and outputs.</code>
4	
5	<code>print("Hello, World!") # This is also a comment.</code>

▲ **Figure 4.14** `hello_world.py` with added comments

Even with the addition of comments, this program still behaves exactly like the original program, as shown in Figure 4.15:

```
===== RESTART: C:/Examples/hello_world.py =====
Hello, World!
>>>
===== RESTART: C:/Examples/hello_world_with_comments.py =====
Hello, World!
>>>
```

▲ **Figure 4.15** Output in IDLE's shell window

4.3.3 Command Line Interface

While IDLE can be convenient, a graphical user interface usually requires the use of a mouse, making tasks difficult to automate without human intervention. A command line interface, on the other hand, can be automated easily as commands are given using only lines of text. Use of Python's command line interface, however, is not covered in this textbook.

The rest of this chapter covers the various types of instructions in Python that computers can use to perform algorithms.

4.4 Variables and Constants

Unlike humans, computers are designed to store and read values quickly and reliably. To take advantage of this, we will need to learn two types of Python instructions: one for *assigning values* and one for *reading values*.

4.4.1 Assigning Values

To assign or keep a value in Python, we need a name as well as a corresponding value to be assigned. The format of the Python instruction for assigning values is: the name we intend to use followed by the equals sign (=), followed by the value to be assigned. This syntax is summarised in Figure 4.16.

Key Terms

Identifier

Sequence of characters that follows certain rules and can be used as a variable name

Keyword (programming)

Word that has a special meaning in a programming language and cannot be used as an identifier

Syntax 4.1 Assignment Statement

```
variable_name = value
```

▲ Figure 4.16 Syntax for assignment statements

Variable names (known as **identifiers**) must also follow certain rules:

- Can contain the upper-case and lower-case letters A through Z and the underscore (_)
- Can contain the digits 0 through 9 but not as the first character
- Cannot contain spaces or special symbols (e.g., "!", "@", "#", "\$")
- Cannot be any of the reserved words (or keywords) in Figure 4.17 that have special meanings in Python

False	None	True	and	as
assert	break	class	continue	def
del	elif	else	except	finally
for	from	global	if	import
in	is	lambda	nonlocal	not
or	pass	raise	return	try
while	with	yield		

▲ Figure 4.17 Reserved words with special meanings in Python

Table 4.3 shows some examples of valid and invalid identifiers.

▼ **Table 4.3** Examples of valid and invalid identifiers

Valid identifiers	Invalid identifiers	Explanation
sum_of_scores	sum-of-scores	Identifiers cannot contain the hyphen character “-”
my_class	class	class is a reserved word
BOX_SIZE	size_correct?	Identifiers cannot contain the question mark character “?”
test_score_3	3rd_test_score	The first character of an identifier cannot be any of the digits “0” to “9”
oneword	two words	Identifiers cannot contain any spaces

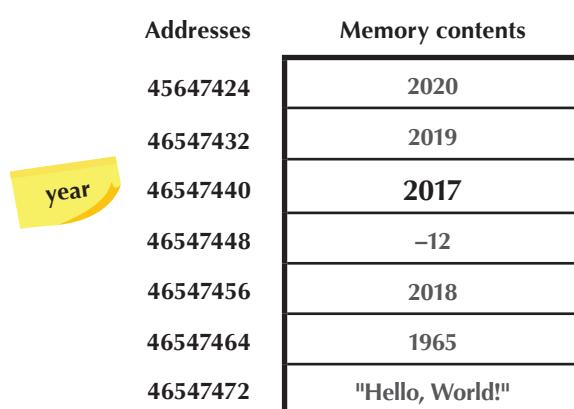
Typically, identifiers in Python are written in lower case with words separated by underscores. To make our source code easier to read, we should follow the same naming style.

Unlike flowcharts and pseudo-code which are meant for humans, syntax rules for Python instructions must be followed strictly or else the computer will not be able to perform the instruction. For example, the code in Figure 4.18 assigns the value 2017 to a variable with the name `year`:

```
>>> year = 2017
```

▲ **Figure 4.18** Example of an assignment statement

We can visualise what this does by imagining a sticky note with label “year” pasted on or attached to the location or address where 2017 is located in memory:



▲ **Figure 4.19** Assigning 2017 to the variable `year`

Did you know?

Unlike variables in flowcharts and pseudo-code that act like boxes, variables in Python act like sticky notes attached to locations in memory. For most of the tasks in this textbook, you need not take note of this difference. However, this will be important to understand when using advanced features of Python not covered in this textbook. In general, you should be aware that different programming languages will have slightly different rules for how variables work.

In most versions of Python, you can get the memory address that a variable is attached to by using the `id()` function, as shown in Figure 4.20:

```
>>> year = 2017
>>> print(year)
2017
>>> id(year)
46547440
```

▲ **Figure 4.20** Example of using the `id()` function

Of course, the memory address that is printed out will vary from computer to computer depending on which memory addresses are available to Python. Do not be alarmed if your computer does not produce the same output as your friend's!

We call the name `year` a **variable** because we can change (i.e., vary) the value that is assigned to it while the program is running.

```
>>> year = 2017
>>> year = 2018
>>> year = 2019
>>> year = 2020
```

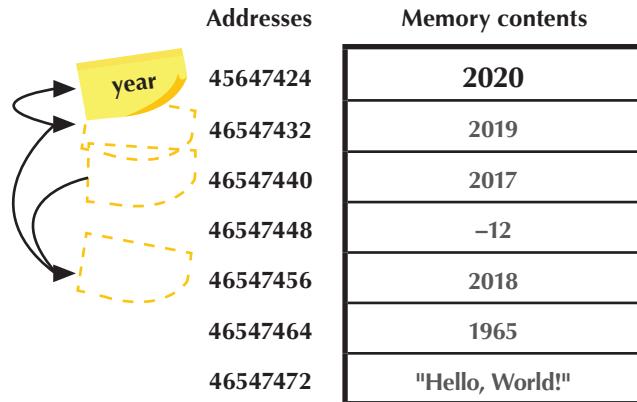
▲ **Figure 4.21** Changing the value assigned to the variable `year`

After Python runs the source code in Figure 4.21, the variable `year` is now assigned the value of 2020. The previous values of 2017, 2018 and 2019 are no longer associated with `year`. Only the last value that is assigned to a variable is kept.

Key Term

Variable (Python)

A name with an associated value that can be changed while the program is running



▲ **Figure 4.22** Changing the assigned value of the variable `year` from 2017 to 2020

In assigning values, what comes after the equals sign is not limited to a number. It can include calculations or even other variables. For example, the following source code creates a new value (the sum of 1 and 2) and attaches a variable named `total` to its address:

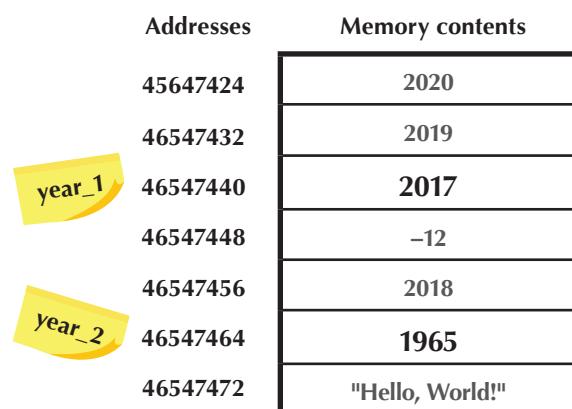
```
>>> total = 1 + 2
```

▲ **Figure 4.23** Assigning the result of a calculation to the variable `total`

Note that when one variable is assigned to another variable, both variables become attached to the same address. You can imagine this as moving the sticky note for one variable onto the same location as another variable. For instance, let us assign different values to two variables:

```
>>> year_1 = 2017
>>> year_2 = 1965
```

▲ **Figure 4.24** Assigning values to `year_1` and `year_2`

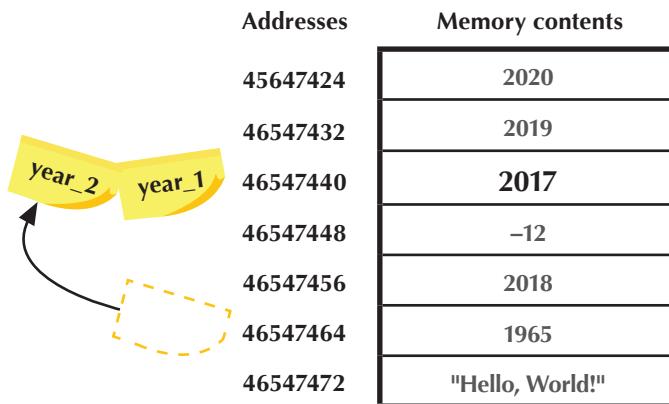


▲ **Figure 4.25** Assigning values to `year_1` and `year_2`

If we take `year_1` and assign it to `year_2`, we can imagine the sticky note for `year_2` being moved over to the location of the sticky note for `year_1`, so both `year_1` and `year_2` now refer to the same value. This is illustrated in Figures 4.26 and 4.27:

```
>>> year_2 = year_1
```

▲ **Figure 4.26** Assigning values to `year_1` and `year_2`



▲ **Figure 4.27** `year_1` and `year_2` now refer to the same value

4.4.2 Reading Values

We can retrieve or “read” the value assigned to a variable by using the variable’s name. If a value was previously assigned to a variable with the given name, Python will use the value that was assigned in place of the variable name.

To display the value of a variable on the screen, use the `print()` function as follows:

```
>>> print(variable)
```

▲ **Figure 4.28** Displaying the value assigned to a variable

For instance, in interactive mode, entering `print(year)` would print out the value assigned to the variable named `year`:

```
>>> year = 2020
>>> print(year)
2020
```

▲ **Figure 4.29** Displaying the value assigned to the variable `year`

However, if the name `year` does not correspond to any variable that was used or defined previously, Python will produce an error message instead:

```
>>> print(year)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(year)
NameError: name 'year' is not defined
```

▲ **Figure 4.30** Displaying the value of an undefined variable

These instructions are translated by the Python interpreter into machine code. This machine code takes care of finding where values are located in memory or if additional space in memory is needed to store a value. The Python interpreter handles these tasks so that programmers can focus on the algorithm.

4.4.3 Assigning and Reading Values Together

As mentioned previously, we can take the value from one variable, manipulate it, and then assign it to another variable. For example, the variable `next_year` will be assigned a value of 2018 after the source code in Figure 4.31 is run:

```
>>> year = 2017
>>> next_year = year + 1
>>> print(year)
2017
>>> print(next_year)
2018
```

▲ **Figure 4.31** Using the value of `year` to assign a value to `next_year`

We can also take the value from a variable, manipulate it, and then assign it back to the same variable. This is because Python will always fully calculate the value of the right-hand side before assigning it to the variable on the left-hand side:

```
>>> year = 2017
>>> year = year + 1
>>> print(year)
2018
```

▲ **Figure 4.32** Using the value of `year` to assign a new value back to `year`

In the second line of the source code in Figure 4.32, Python calculates the value of `year + 1` using the original value of `year` before assigning the final result (2018) back to `year`. The old value of 2017 will no longer be referred to. This technique is often used to update the value assigned to a variable without having to create additional variables to store the interim results.

4.4.4 None Value

Sometimes we need to keep track of the fact that a value is missing or that there is no suitable value to be assigned to a variable. For instance, some problems may have optional inputs. Since Python will produce an error message if we try to retrieve a variable that has not been defined, we may still wish to create the variable but assign it a value that indicates the value is missing. For such cases, Python provides a special `None` value that can be used instead:

```
>>> optional_input = None
>>> print(optional_input)
None
```

▲ Figure 4.33 Using `None` to indicate a missing value

4.4.5 Constants

Programs often use one or more fixed values over and over again. For instance, let us consider a school in which the maximum score for all tests is always 30 and the number of students in each class is always 40. Suppose we want to find out the percentage score of one student and the overall percentage score of a whole class. We can use the following source code to calculate these scores:

```
student_percentage = score / 30 * 100
class_percentage = total_score / (40 * 30) * 100
```

▲ Figure 4.34 Example of calculations that use repeated fixed values

The above source is valid, but using the raw values of 30 and 40 in this manner has two disadvantages:

1. It is not clear what the values 30 or 40 mean from just the source code.
2. There is no clear indication that the value of 30 in both lines of source refers to the same thing (that is, the maximum test score). If the maximum test score changes, someone updating the code may change one line correctly but forget about the other line.

To avoid this, we should use **constants**. Constants are similar to variables in that they both have names and assigned values. However, once the value of a constant is set, it cannot be changed while the program is running.

Key Term

Constant (programming)
A name with an associated value that usually cannot be changed while the program is running

For example, we can use constants to provide more readable names for the raw values of 30 and 40:

```
MAX_SCORE = 30
CLASS_SIZE = 40
```

▲ **Figure 4.35** Declaring constants

Using these constants, the source code becomes more readable and easier to maintain should there be any changes required in the future:

```
student_percentage = score / MAX_SCORE * 100
class_percentage = total_score / (CLASS_SIZE * MAX_SCORE) * 100
```

▲ **Figure 4.36** Example of calculations rewritten using constants

Some programming languages such as C have special instructions for constants to make sure that their values cannot be changed. However, Python does not. Whenever we need a constant in Python, we have to use a variable instead. To indicate that a variable is actually a constant and that its value should not be changed after it is set, we follow the **naming convention** of using upper-case letters for its name.

Key Term

Naming convention

A set of generally accepted rules for how identifiers or names should be chosen

Note that, even though these variables are treated as constants, Python does not prevent a programmer from changing the value stored in them while the program is running. However, doing so would go against convention and might lead to unexpected errors.

Did you know?

Many systems work because most people follow the same norms or “unwritten laws”. These are called conventions. For instance, in Singapore, it is conventional to keep left on the escalator so that there is a clear path on the right for those in a hurry.

In the same way, Python programs have a set of conventions that we should try to follow. An example of these conventions is using upper-case letters to name constants.



Quick Check 4.4

1. For each assignment statement in Figure 4.37, identify:
 - a) the name of the variable that is assigned a value, and
 - b) the final value which is being assigned.

```
1 ratio = 0.75
2 greeting = "Hi!"
3 to_hit = 17 + 3
```

▲ Figure 4.37 Assignment statements

2. The source code in Figure 4.38 can be entered line by line, in the order given, using Python's interactive mode. For each line of source code, predict what will be displayed on the screen (if any) or if an error will occur. Check your answers by entering the same source code, line by line, in IDLE's shell window and observing the results.

```
1 name= "Computing"
2 name name
3 Name = "Science"
4 print(name)
5 print(Name)
6 Real_Name = name
7 Real_Name
8 print(Real_Name)
9 Fake_Name =
```

▲ Figure 4.38 Source code for entry in interactive mode

3. Write a program that assigns the value of 99 to a variable named `original_value`, and then assigns `original_value` to two variables named `copy_1` and `copy_2`. The program should only need three lines of source code.

4. Identify three different values in the program in Figure 4.39 that should each be represented by a constant. Suggest suitable names for each constant, and then modify the program to use the constants with the names you have chosen. The modified program should behave identically to the original program.

#	Program 4.3 greetings_with_title.py
1	<code># Prints out a series of greetings, with title based on gender.</code>
2	<code>print("Hello, " + "Ms. " + "Aishah")</code>
3	<code>print("Hello, " + "Mr. " + "Paul")</code>
4	<code>print("Hello, " + "Mr. " + "Xiaoming")</code>
5	<code>print("Hello, " + "Ms. " + "Jessica")</code>

▲ **Figure 4.39** Program that uses repeated fixed values

4.5 Data Types

Data in a computer is stored as a sequence of bytes. The same sequence of bytes can represent different types of data such as numbers or text, depending on how the sequence is interpreted.

However, we do not need to worry about how the bytes are arranged when assigning values to or reading values from variables. Instead, Python manages the arrangement and interpretation of the bytes in each variable. This is done by associating each value and variable with a data type. Some built-in data types in Python are shown in Table 4.4:

▼ **Table 4.4** Built-in data types

Data type	Python	Valid values	Example
Integer	int	Whole numbers	1234
Floating point	float	Real numbers	12.34
String	str	Text (can include digits)	"ABC123"
Boolean	bool	True, False	True
List	list	Multiple values	[1, 2, 3]

4.5.1 Integers

An **integer** (`int` for short) is a whole number. To represent an `int` value directly in Python source, use its **literal** format, that is, the digits of the `int` written out:

```
>>> 2017
2017
```

▲ **Figure 4.40** Example of valid integer (`int`) literal

Since many algorithms involve counting and manipulating whole numbers, `ints` are handled as a special data type in Python to ensure accuracy. Python `ints` can handle almost any whole number, positive or negative, large or small, without losing accuracy. This is different from some programming languages where there is a limit to the most positive or most negative integer that can be represented by an integer type. For instance, some programming languages would not be able to perform the following calculation of 2 to the power of 2^{10} correctly:

```
>>> 2 ** 2 ** 10
1797693134862315907729305190789024733617976978942306572734300811577
3267580550096313270847732240753602112011387987139335765878976881441
6622492847430639474124377767893424865485276302219601246094119453082
9520850057688381506823424628814739131105408272371633505106845862982
39947245938479716304835356329624224137216
```

▲ **Figure 4.41** Demonstration of using `ints` to perform a large calculation

The `**` symbol here represents exponentiation. This and other mathematical operators will be discussed in section 4.6.2.2.

4.5.2 Floating-Point Numbers

To represent values that are not whole numbers, Python uses the **floating-point** (`float` for short) data type. Unlike an `int`, a `float` is not especially accurate, but it is usually accurate enough for most situations. We can write `float` literals in two formats:

1. Writing out the digits of the `float` with decimal point included explicitly. If no decimal point is included, Python will treat the number as an `int` instead.
Example: `12345.6789`
2. Expressing the `float` in the form AeB or AEB to represent $(A \times 10^B)$.
Example: `6.02e23` represents 6.02×10^{23}

Key Terms

Floating-point number (`float`)

Data type to represent real numbers that contain a fractional part

Integer (`int`)

Data type representing whole numbers (positive and negative)

Literal

Format for representing a value directly in source code

Figure 4.42 shows some examples of valid `float` literals:

```
>>> 2017.0
2017.0
>>> 2e3
2000.0
```

▲ **Figure 4.42** Examples of valid `float` literals

Due to the format in which floating-point values are stored, `floats` are not especially accurate, as shown in Figure 4.43:

```
>>> 3.0 * 0.1
0.3000000000000004
```

▲ **Figure 4.43** Example of an inaccurate floating-point calculation

There is also a limit to the most positive and most negative number that can be represented using a `float`. In most versions of Python, the most positive number that can be represented is approximately 1.79×10^{308} while the most negative number that can be represented is approximately -1.79×10^{308} . Beyond this limit, the `float` gets converted into the special value of infinity (represented by `inf` or `-inf`):

```
>>> 1.79e308
1.79e+308
>>> 1.80e308
inf
>>> -1.79e308
-1.79e+308
>>> -1.80e308
-inf
```

▲ **Figure 4.44** Demonstration of `float` limits

4.5.3 Strings

Python uses the **string** (`str` for short) data type to represent text. The name “string” is shorthand for the phrase “string of characters/letters”. String literals consist of the text contents enclosed by *matching* single-quotes (`'`) or double-quotes (`"`). In addition, the enclosing quotes must be entered as straight quotes (like `'` or `"`) and not curly quotes (like `’` or `”`); otherwise a syntax error will occur.

Key Term

String (`str`)

Data type to represent text as a sequence of characters or symbols

Some examples of valid `str` literals are provided in Figure 4.45:

```
>>> "Hello, World!"
'Hello, World!'
>>> '123.456'
'123.456'
>>> 'Say "Hello" with me!'
'Say "Hello" with me!'
```

▲ **Figure 4.45** Examples of valid string (`str`) literals

Note that in Python's interactive mode, if a `str` is entered directly as a literal, it is printed with additional single-quotes around the content. To display only the content of the `str` without the quotes, use the `print()` function instead, as shown in Figure 4.46:

```
>>> print("Hello, World!")
Hello, World!
>>> print('123.456')
123.456
>>> print('Say "Hello" with me!')
Say "Hello" with me!
```

▲ **Figure 4.46** Using the `print()` function to display `str`s without additional single quotes

Often, `str`s may need to include characters that are either difficult to type out as part of a literal or would cause a syntax error if not treated specially. To include such characters, we need to input them using the backslash (\) key and special **escape codes**. Some common escape codes for string literals are shown in Table 4.5:

▼ **Table 4.5** Common escape codes

Escape code	Meaning
\\	Backslash (\)
\'	Single-quote ('')
\"	Double-quote (")
\n	Newline character
\t	Tab character
\	Ignore end of line

Key Term

Escape code

Sequence of characters used to input characters that are either difficult to type out as part of a literal or would cause a syntax error if not treated specially

Note that the escape code "\\" represents a backslash that is the last character on a line of source code. This trailing backslash means to ignore the line break and treat the following line as if it continues on the current line.

Figure 4.47 shows some examples of how escape codes can be used:

```
>>> print('Line 1\nLine 2')
Line 1
Line 2
The \n escape code breaks
the string up into two lines.

>>> print('abc\
def')
abcdef
Python ignores the line break as the first
line ends with a backslash.

>>> print('It\'s a small world.')
It's a small world.
The single quote in "It's" is escaped and thus
does not end the string prematurely.
```

▲ Figure 4.47 Examples of how escape codes are used

Programmers often need to convert `str`s into `int`s or `float`s and vice versa. The process of intentionally converting a value from one data type to another is called **type casting**. To do this, use the name of the desired data type like a function. For instance, the `int()` function can convert `str`s into `int`s. Figure 4.48 shows that while `"123"` and `int("123")` both appear as 123 when printed, `"123"` is text and cannot be used to perform calculations while `int("123")` is the actual number 123 and *can* be used to perform calculations. We can check the data type of these two values using the `type()` function, which will be described in section 4.5.6.

Key Term

Type casting

Process of
converting a
value from one
data type to
another

```
>>> print("123")
123
>>> print("123" + 1)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print("123" + 1)
TypeError: Can't convert 'int' object to str implicitly
>>> print(type("123"))
<class 'str'>
>>> print(int("123"))
123
>>> print(int("123") + 1)
124
>>> print(type(int("123")))
<class 'int'>
```

▲ Figure 4.48 Using the `int()` function to convert a `str` into an `int`

Similarly, the `str()` function can convert ints and floats to strs. In the following example, an error occurs as the “+” operator expects both the value on its left and the value on its right to be of the same type. To fix this error, we use the `str()` function to convert number into a str so that both values on the left and right are strs.

```
>>> number = 2017
>>> print("My number is " + number)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print("My number is " + number)
TypeError: Can't convert 'int' object to str implicitly
>>> print("My number is " + str(number))
My number is 2017
```

▲ **Figure 4.49** Using the `str()` function to convert an int into a str

The use of the “+” operator to join strings together will be discussed in section 4.6.2.3.

4.5.4 Booleans

Boolean (`bool` for short) is a special data type used by Python to represent logic-related data. Unlike other data types, there are only two valid values that a `bool` variable can have assigned to it: `True` or `False`.

```
>>> print(1 < 3)
True
>>> print(3 < 1)
False
>>> example_bool_1 = True
>>> example_bool_2 = False
```

▲ **Figure 4.50** Example of `bool` variables with two valid values

In the example above, Python uses the `bool` values of `True` and `False` to report whether 1 is less than 3, and vice versa. You will learn how to manipulate `bools` using logic operators in section 4.6.2.4 on Boolean operators.

Key Term

Boolean (`bool`)
Data type
representing either
`True` or `False`

Did you know?

Booleans are named after George Boole, an English mathematician, philosopher and logician, who introduced the use of Boolean logic, a form of algebra where values or variables are either “true” or “false”, in 1847.

4.5.5 Lists

Programs often need to keep and manipulate multiple values without prior knowledge of the total number of values. The `list` data type can be used to store a sequence of values of any length.

The format for `list` literals is a left square bracket (`[]`), followed by the items of the `list` (if any) separated by commas, and ending with a right square bracket (`]`). The items in the `list` can be of any type, although mixing types in a list is not a good practice.

Figure 4.51 shows some examples of valid `list` literals, including an empty list.

Key Terms

Array

Sequence where all the items have the same size and are arranged consecutively in memory; informally also used as a synonym for list

List (programming)

Data type for storing multiple values in a sequence

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ["Mixing types in a list is bad.", 2017, False]
['Mixing types in a list is bad.', 2017, False]
>>> []
    This is an empty list.
[]
```

▲ Figure 4.51 Examples of valid `list` literals

A simple `list` is sometimes called a one-dimensional **array**. However, a `list` can only be called an array if all the items in the `list` have the same size and are arranged consecutively in memory.

When we use flowcharts or pseudo-code to work with multiple values, we need to either input the total number of values at the start, or wait for a special “end value” to be provided so that we know when to stop accepting values. While both these methods still work in Python, `lists` allow for another way to process multiple values by having a single variable represent a collection of values.

You will learn how to manipulate items in a `list` using the index and slice operators in section 4.6.2.3.

4.5.6 Variable Types

In Python, the data type associated with a variable depends on the value currently assigned to it. It is possible, though strongly discouraged, to reuse the same variable name by assigning different data types to it.

Being inconsistent with the data types can lead to programming mistakes because it becomes difficult to predict the type of the value for each variable. It is thus good programming practice to avoid changing the data type of a variable once it has been initialised with a value.

Did you know?

You can find out the data type of any value by using the `type()` function. Python will then display the data type of any value that is placed between the parentheses. For example, the following output means that the data type of 2017 is an `int`:

```
>>> print(type(2017))  
<class 'int'>
```

▲ Figure 4.52 Using the `type()` function to display the data type



Quick Check 4.5

1. State the data type for each of the following literals:

- a) -2017
- b) ''''
- c) 'Computing'
- d) 3.14159
- e) "\\""
- f) "\\"[]\\""
- g) True
- h) 6.02e23
- i) '\\\'
- j) "2017"
- k) ''''
- l) ["""]
- m) 'False'

2. For each line of source code in Figure 4.53, predict what will be displayed on the screen (if any) or if an error will occur. Check your answers by entering the same source code, line by line, in IDLE's shell window and observing the results.

```

1 print(1e100)
2 print(103E1)
3 print(1EE1)
4 print(3.141)
5 print(3.1.3)
6 print("Hello,\nWorld")
7 print("Hello,\n\nWorld")
8 print("Hello,\tWorld")
9 print('Hello,\n\tWorld')
10 print('Hello,\t\nWorld')
11 print("\\"\\\"")
12 print("\\\\")
13 print(True)
14 print(true)
15 print(False)
16 print(false)
17 print(None)
18 print(none)
19 print([])
20 print([2, 3, 4])
21 print([[ ]])

```

▲ **Figure 4.53** Source code for entry in interactive mode

3. For each line of source code in Figure 4.54, predict what will be displayed on the screen (if any) or if an error will occur. Check your answers by entering the same source code, line by line, in IDLE's shell window and observing the results.

```

1 print(int("-3"))
2 print(float('-3'))
3 print(float("minus three"))
4 print(str(-3))
5 print(str(-3.0))
6 print(str(-3e3))

```

▲ **Figure 4.54** Source code for entry in interactive mode

4.6 Functions and Operators

Now that we know how to assign and read values of different data types, we can perform more complex tasks that involve manipulating data. Most programming languages allow programmers to manipulate values and variables in two main ways: using functions and operators.

4.6.1 Functions

Just as variables can store values such as numbers and lists, **functions** can be thought of as a way of storing instructions to be used again later.

You have already seen functions used in some previous examples. For example, in Figure 4.55, the `print()` function takes in a `str` value, called an **argument**, and displays it on the screen without quotes around the content:

```
>>> print("Hello, World!")
Hello, World!
```

Key Terms

Argument

Additional value that can be supplied as input to a function call

Function

Set of instructions assigned to a name that can be used again later

Function call

Process of assigning arguments to new variables and running the instructions assigned to a function

When using a function, type its name, followed by parentheses that can either be empty or contain comma-separated arguments (as shown in Figure 4.56). When a function is used, the Python interpreter will assign any arguments to new variables before running the instructions that are stored in the function. This process is also known as a **function call**.

Syntax 4.2 Function Call

<code>function_name()</code>
<code>function_name(argument_1)</code>
<code>function_name(argument_1, argument_2)</code>

▲ Figure 4.56 Syntax for function calls

Figure 4.57 shows some examples of function calls using the `print()` function.

```
>>> print()
>>> print(1)
1
>>> print(1, 2)
1 2
```

▲ **Figure 4.57** Using the `print()` function

This syntax is why a set of parentheses are used after function names to distinguish them from variable names.

In addition to the `print()` function, Python comes with many other built-in functions that you can use straight away. Some functions can even **return values** for your program to assign or use in other ways. For instance, the `len()` function accepts one argument of type `list` or `str` and returns its length as an `int`:

Key Terms

Nested function call

Using the return value of one functional call as an argument for another function call

Return value

Output of a function, or the value that a function call is treated as, after the instructions assigned to the function are completed

```
>>> test_scores = [72, 68, 85]
>>> print(len(test_scores))
3
>>> number_of_scores = len(test_scores)
>>> x = number_of_scores + 1
>>> y = len(["A", "B"])
>>> z = len([])
>>> print([x, y, z])
[4, 2, 0]
```

▲ **Figure 4.58** Using the `len()` function

The return value (or result) of one function call can be used as an argument for another function call. This is called a **nested function call**.

```
>>> print("Length is", len("Hello, World!"))
Length is 13
```

▲ **Figure 4.59** Using a nested function call

The code can get difficult to read if there are too many nested function calls. For such cases, it is recommended to break the instruction into smaller pieces by assigning return values to intermediate variables before performing the outermost function call. For instance, the code in Figure 4.60 gives the same result as the nested function call in Figure 4.59.

```
>>> length_of_greeting = len("Hello, World!")
>>> print("Length is", length_of_greeting)
Length is 13
```

▲ **Figure 4.60** Breaking an instruction into smaller pieces instead of using a nested function call

4.6.1.1 Built-In Functions

Some common built-in functions in Python and their typical arguments are described in Table 4.6. Note that some functions such as `range()` will behave differently depending on the types of arguments they receive. Each valid combination of arguments is described in a separate row.

▼ **Table 4.6** Common built-in functions

Function	Argument(s)	Return value	Description/Examples
<code>float()</code>	Usually an <code>int</code> or a <code>str</code>	A <code>float</code> equal to the argument converted to a floating-point number (if possible)	Used to convert a value into a <code>float</code> <pre>>>> float(2) 2.0 >>> float('2.017') 2.017</pre>
<code>input()</code>	A <code>str</code> to use as a prompt	A <code>str</code> containing text that the user enters in response to the prompt	Used to get input from the user <pre>>>> name = input("Enter name: ") Enter name: Samantha >>> print(name) Samantha</pre>
<code>int()</code>	Usually a <code>float</code> or <code>str</code>	An <code>int</code> equal to the argument with digits after the decimal point cut off (if possible)	Used to convert a value into an <code>int</code> <pre>>>> int(2.017) 2 >>> int('2017') 2017 >>> int(3.9) 3 >>> int(-3.9) -3</pre>

▼ Table 4.6 Common built-in functions (continued)

Function	Argument(s)	Return value	Description/Examples
<code>len()</code>	A sequence, such as a list or str	An int equal to the length of the argument	Used to get the length of a sequence, such as a list or str <pre>>>> test_scores = [72, 68, 85] >>> print(len(test_scores)) 3</pre>
<code>print()</code>	Any number of printable values	None	Used to display output <pre>>>> print("Hello, World!") Hello, World!</pre>
<code>range()</code>	One int	A sequence of ints starting from 0 up to but not including the argument	Used to make a sequence of ints for iterating through lists; use the <code>list()</code> function to convert it into a list <pre>>>> print(list(range(3))) [0, 1, 2]</pre>
<code>range()</code>	Two ints	A sequence of ints starting from the first argument up to but not including the second argument	Used to make a sequence of ints for iterating through lists; use the <code>list()</code> function to convert it into a list <pre>>>> print(list(range(4, 8))) [4, 5, 6, 7]</pre>
<code>range()</code>	Three ints	A sequence of ints starting from the first argument up to but not including the second argument, in increments of the third argument	Used to make a sequence of ints for iterating through lists; use the <code>list()</code> function to convert it into a list <pre>>>> print(list(range(3, 20, 4))) [3, 7, 11, 15, 19]</pre>
<code>str()</code>	Usually a number (int or float)	A str equal to the argument converted to a string (if possible)	Used to convert a value into a str <pre>>>> str(2.017) '2.017' >>> str(2017 + 2017) '4034' >>> str(2017) + str(2017) '20172017'</pre>

Did you know?

If you ever need help with a built-in function, you can use the `help()` function to print out its documentation. For instance, to learn more about the `len()` function, enter the following in Python's interactive mode, taking care to enter `len` *without* the trailing parentheses:

```
>>> help(len)
```

▲ Figure 4.61 Example of using the `help()` function

Note that some built-in functions may use advanced features that are not covered in this textbook.

4.6.1.2 User-Defined Functions

Besides Python's built-in functions, you can also write your own functions. These are called **user-defined functions (UDFs)**.

To define a UDF, we first need to choose its name. The name must be a valid Python identifier that follows the same rules described previously for variable names (see section 4.4.1). The UDF's name should also be different from any existing variable or function names. Otherwise, the UDF will overwrite the existing value or function associated with that name.

For example, the program `define_hello.py` in Figure 4.62 defines a function named `hello()` that prints out the phrase "Hello, World!". Lines 2 and 3 form the UDF's **body** which is run whenever the UDF is called. A UDF's body is always indented by starting each line with a level of **indentation** (usually four spaces). This indentation is used to determine the start and end of each UDF. You will see more examples of how Python uses indentation in section 4.7.

If we run `define_hello.py`, nothing appears on the screen because `hello()` is never actually called. In order to use a UDF, you need to call it. This process is known as a function call (see section 4.6.1) and can only be performed after the UDF is defined.

Key Terms

Body (functions)

Sequence of statements in a UDF

Indentation

Number of spaces at the start of each line of source code

User-defined function (UDF)

Function provided by the user of a programming language

Program 4.4 define_hello.py

<pre>1 def hello(): 2 # This code does not get run at all. 3 print('Hello, World!')</pre>	<p>Line 1 specifies the name.</p> <p>Lines 2 and 3 form the body.</p>
---	---

▲ Figure 4.62 Defining a UDF without calling it

The program `say_hello.py` in Figure 4.63 defines the same function `hello()`, then calls the `hello()` function three times. In this program, the indentation after line 1 indicates that lines 2 and 3 belong to the `hello()` UDF's body while lines 5 to 7 do not. Also, if you place lines 5 to 7 before lines 1 to 3, running the program will produce an error.

#	Program 4.5 <code>say_hello.py</code>
1	<code>def hello():</code>
2	<code># Say hello</code>
3	<code>print('Hello, World!')</code>
4	
5	<code>hello()</code>
6	<code>hello()</code>
7	<code>hello()</code>

▲ **Figure 4.63** Defining and calling a UDF

When you run this program, the phrase “Hello, World!” should appear on the screen three times, as in Figure 4.64, once for each call of `hello()` from lines 5 to 7:

```
Hello, World!
Hello, World!
Hello, World!
```

▲ **Figure 4.64** Output after running `say_hello.py`

Line 1 of the program `say_hello.py` is the UDF's **signature** that specifies the UDF's name and its **parameters**. Parameters are special variables that arguments are assigned to when the UDF is called. In this case, `hello()` function has no parameters. For a UDF to accept arguments, its signature must specify a parameter for every argument it expects to receive.

For instance, the program `quiz.py` in Figure 4.65 has a UDF `get_answer()` that expects to receive two arguments using the parameters `prompt` and `reply`. This means that when the function is called (but before the function body is run), the first argument will be assigned to a new variable named `prompt` and the second argument is assigned to a new variable named `reply`.

Key Terms

Parameter

Special variable in a UDF that an argument is assigned to when the UDF is called

Signature (programming)

First line of a UDF that specifies the UDF's name and its parameters

#	Program 4.6 quiz.py
	<pre> 1 def get_answer(prompt, reply): 2 answer = input(prompt) 3 print(reply) 4 return answer 5 6 answer1 = get_answer('What is your name?', 'Thanks!') 7 answer2 = get_answer('What is your age?', 'Thanks again!') 8 answer3 = get_answer('What is your hobby?', 'Thank you!') 9 10 print('Your answers were...') 11 print(answer1, answer2, answer3) </pre>

▲ Figure 4.65 Defining a UDF that accepts arguments and returns a value

Line 4 of `quiz.py` also shows that a UDF can provide a return value using the `return` keyword. When Python encounters a **return instruction**, the function immediately ends and the original function call is treated as the provided return value. If no return instruction is encountered before the function body ends, then the default return value is `None` and the original function call is also treated as `None`. An example of this is the `hello()` function in Figure 4.63.

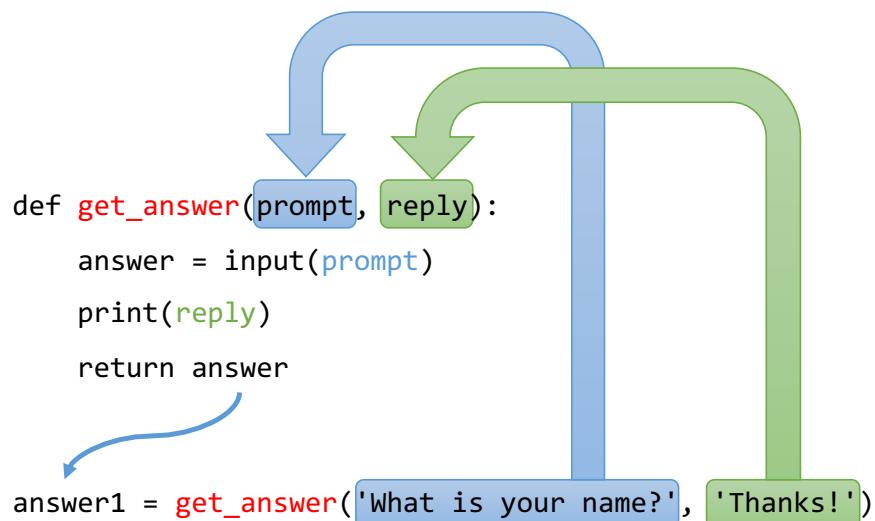
Key Term

Return instruction

Instruction that ends the function immediately and treats the original function call as the provided return value

From Lines 6 to 8 of `quiz.py`, `get_answer()` is called three times and the return value of each call is assigned to variables `answer1`, `answer2` and `answer3` respectively.

Figure 4.66 illustrates the process of how the arguments of a function call are assigned to new variables with names given by the function's signature and how the return value is assigned to `answer1`.



▲ Figure 4.66 How arguments are assigned to new variables during a function call

Figure 4.67 summarises the syntax rules for a UDF that accepts either no arguments, one argument or two arguments. This pattern can be extended to write UDFs that accept even more arguments.

Syntax 4.3 User-Defined Function

```
def function_name():
    commands to run when function_name is called
```



```
def function_name(parameter_1):
    commands to run when function_name is called (needs 1 argument)
```



```
def function_name(parameter_1, parameter_2):
    commands to run when function_name is called (needs 2 arguments)
```

▲ Figure 4.67 Syntax for defining a UDF

The syntax rules for `return` are summarised in Figure 4.68. The return value after the `return` keyword is optional. If it is provided, the return value can be of any data type. If it is omitted, the return value is treated as `None`.

Syntax 4.4 return Statement

```
return
```



```
return return_value
```

▲ Figure 4.68 Syntax for returning from a UDF

Previously, we explained that arguments are assigned to new variables based on the parameters given in the function's signature. These new variables are special because they are valid only within the function. This is also true for any variables that are created inside the function. Such variables are called **local variables** because they can only be used inside the function's body.

Key Term

Local variable

Variable that is created inside a UDF and can only be used in the UDF's body

For instance, the program `area.py` in Figure 4.69 has a UDF named `area_of_circle()` that, when called, will assign its argument to a variable named `radius`. Inside the function, a new variable named `area` is also created:

#	Program 4.7 <code>area.py</code>
1	<code>PI = 3.14159</code>
2	
3	<code>def area_of_circle(radius):</code>
4	<code>area = PI * radius ** 2</code>
5	<code># Both radius and area are valid variables here</code>
6	<code>return area</code>
7	
8	<code>print(area_of_circle(2))</code>
9	<code># However, radius and area are NOT valid variables here</code>

▲ **Figure 4.69** Defining a UDF that uses local variables

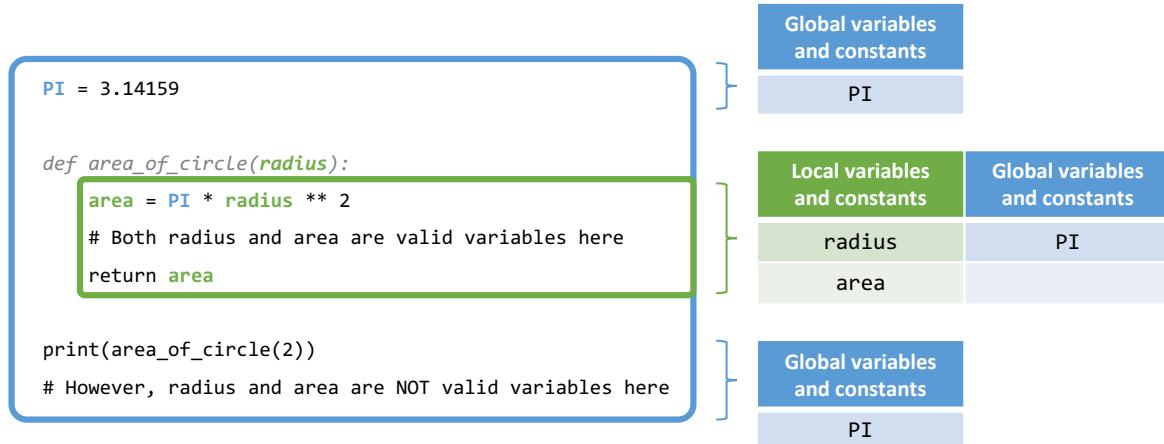
In this case, both `radius` and `area` are local variables for `area_of_circle()`. This means that outside the function body of `area_of_circle()`, `radius` and `area` are not defined. For example, commands like `print(radius)` and `print(area)` would run correctly inside the function body (e.g., line 5), but the same code would produce an error message outside the function body (e.g., line 9).

Notice that `area_of_circle()` uses the constant `PI` that is defined outside of the function body. Besides local variables, the code inside a function body also has *read* access to the **global variables**, constants and functions that are defined outside of the function. The different local as well as global variables and constants found in `area.py` are illustrated in Figure 4.70:

Key Term

Global variable

Variable that is created outside of a UDF and is readable from the UDF's body if its name is not hidden by a local variable



▲ **Figure 4.70** Local and global variables and constants in `area.py`

In a typical program, constants and functions are defined once and do not change. Hence, it is usually predictable to use global constants and call global functions such as `input()`, `print()` or other UDFs inside a function body.

Variables are different from constants and functions as they can change while a program is running. A function that depends on global variables may produce completely different behaviour each time it is called. To avoid this, UDFs should request for all the inputs that they need as arguments instead of using global variables. For instance, the program `add_function.py` in Figure 4.71 shows two ways of writing a function to add two numbers. In general, the `good_example()` approach is better than the `bad_example()` approach as all the inputs that can affect the return value are clearly specified in the function's signature and the function body does not use any global variables.

#	Program 4.8 add_function.py
1	<code>def good_example(num1, num2): # num1 and num2 are local</code>
2	<code> return num1 + num2 # OK</code>
3	
4	<code>def bad_example(): # num1 and num2 are global</code>
5	<code> return num1 + num2 # Don't do this!</code>
6	
7	<code>num1 = 19</code>
8	<code>num2 = 65</code>
9	<code>print(good_example(20, 17))</code>
10	<code>print(bad_example())</code>

▲ **Figure 4.71** Defining a UDF that uses local variables

For `add_function.py`, the variable names `num1` and `num2` are used in two different contexts:

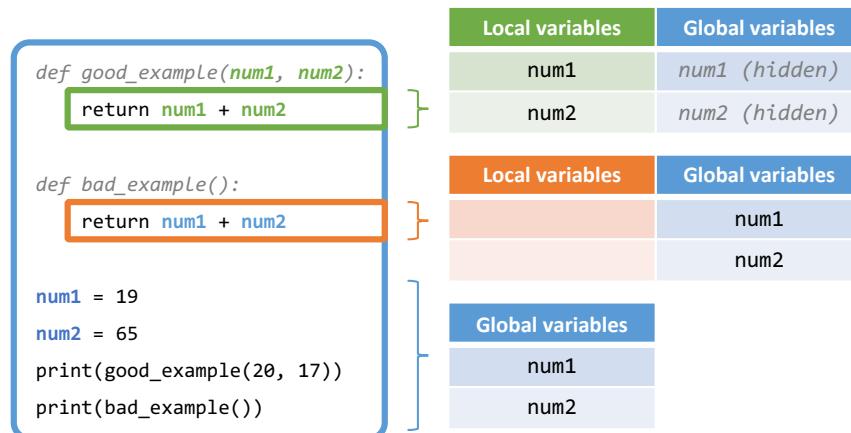
1. On lines 1 and 2, they are local variables for the `good_example()` function.
2. On lines 5, 7 and 8, they are global variables.

While they have the same names, it is important to understand that the local variables named `num1` and `num2` on lines 1 and 2 are completely separate variables from the global variables named `num1` and `num2` on lines 5, 7 and 8.

When `good_example()` is called on line 9, two new local variables are created and given the names `num1` and `num2` just before the function body on line 2 is run. This temporarily hides the global variables that are also named `num1` and `num2` and prevents them from being accessed inside the body of `good_example()`.

However, both global variables are merely hidden and not lost. After the call to `good_example()` ends, the local versions of `num1` and `num2` are removed and the global variables that are named `num1` and `num2` become accessible again with the same values that were assigned to them previously.

Figure 4.72 illustrates the local and global variables in `add_function.py` and how having a local variable can hide access to global variables with the same name.



▲ **Figure 4.72** Local and global variables and constants in `add_function.py`

Notice how the global variables named `num1` and `num2` are hidden for `good_example()` but are not hidden for `bad_example()`.

UDFs have several advantages:

- Code that is repeated in different locations of the program can be put in UDFs. The repeated code can then be replaced with shorter function calls.
- The solution to each sub-problem can be placed in a separate UDF using the modular method of decomposition (see Chapter 2). Each UDF will be smaller than the main program and thus can be more easily understood.
- When working in a team, different programmers can be given responsibility for different UDFs so they can all work concurrently.
- Each UDF can be tested separately and more thoroughly with different inputs compared to a single large program with no UDFs.

4.6.2 Operators

You have already seen **operators** used in many of our previous examples. For instance, the addition operator (+) takes two values, one to the left and one to the right, then returns the sum of the two values.

```
>>> 1 + 2
3
```

▲ Figure 4.73 Using the addition operator

Similarly, whenever we assign a value to a variable, we are actually using the assignment operator (=), which takes the value calculated on the right and stores it in the variable on the left:

```
>>> year = 2017
```

▲ Figure 4.74 Using the assignment operator

Most of the operators we will learn about take in exactly two values, one to the left and one to the right. These are called **binary operators**.

Operators and functions are similar in that they take in values, manipulate them, and usually return a value. While functions have names and use a standard calling syntax, operators usually look like symbols and the syntax for using them varies. However, because they are short and quick to type, operators are ideal for representing instructions that are frequently used.

Key Terms

Binary operator

Operator that takes in exactly two values

Operator

Symbol for performing instructions like a function but using a different syntax

4.6.2.1 Multi-Type Operators

Some operators work with values of almost any type in Python. For instance, it makes sense to ask whether two variables have same or different values, regardless of their data types. Python can make such comparisons using the equivalence and non-equivalence operators.

▼ Table 4.7 Relational operators that work with multiple data types

Operator	Name	Description	Examples
<code>==</code>	Equivalence	Returns the <code>bool</code> value <code>True</code> if the two values are equivalent and <code>False</code> if they are not	<code>>>> 2017 == 2017</code> <code>True</code> <code>>>> 2017 == 2018</code> <code>False</code>
<code>!=</code>	Non-equivalence	Returns the <code>bool</code> value <code>False</code> if the two values are equivalent and <code>True</code> if they are not	<code>>>> 2017 != 2017</code> <code>False</code> <code>>>> 2017 != 2018</code> <code>True</code>

Did you know?

Note that the single equals sign is for assignment and double equals sign, or equivalence, is for testing whether two values are equal. The two operators are *not* the same. For instance, the equals sign (`=`) does not have a return value while the equivalence operator (`==`) *always* has a `bool` return value of either `True` or `False`.

The non-equivalence operator is a combination of the exclamation mark and equals sign. To better understand why it is written this way, read the exclamation mark as “not”. In some programming languages, the exclamation mark is often used to represent the logical NOT operation.

When using the equivalence and non-equivalence operators, the two values must usually be the same in both content and data type to be considered equivalent.

```
>>> '1' == "1"
True
>>> 1 == "1"
False
>>> 1 != [1]
True
```

▲ **Figure 4.75** Using the equivalence and non-equivalence operators

However, numbers are treated a little differently. If an `int` and a `float` are compared with each other, the `int` is converted to a `float` before the comparison is made. If the conversion fails, the two values are treated as non-equivalent. Furthermore, since `floats` are not particularly accurate, testing for equivalence or non-equivalence using `floats` may give surprising results as seen in Figure 4.76. To avoid such issues, if values can be easily represented using `ints` instead of `floats` (such as by storing prices in cents instead of dollars), it is generally a good practice to do so.

```
>>> 3.0 * 0.1 == 0.3
False
>>> 1e16 + 1 == 1e16
True
```

▲ **Figure 4.76** Using the equivalence operator with `floats` may give unexpected results

Another operator that works with almost any data type is the assignment operator.

▼ **Table 4.8** Assignment operator that works with multiple data types

Operator	Name	Description	Examples
=	Assignment	Calculates the value on the right and assigns it to a variable on the left	>>> <code>year = 2017</code> >>> <code>print(year)</code> 2017

4.6.2.2 Mathematical Operators

Mathematical operators are commonly used to operate on `int` and `float` data types. The most familiar of these operators are those we use for performing calculations or arithmetic.

▼ **Table 4.9** Arithmetic operators that work with `int` and `float` data types

Operator	Name	Description	Examples
+	Addition	Returns the sum of two values	>>> <code>2017 + 1e3</code> 3017.0
-	Subtraction	Returns the difference of two values	>>> <code>2017 - 1e3</code> 1017.0
*	Multiplication	Returns the product of two values	>>> <code>2017 * 1e3</code> 2017000.0
/	Division	Returns the value on the left divided by the value on the right	>>> <code>2017 / 1e3</code> 2.017
//	Floor division	Returns the value on the left divided by the value on the right, rounded down to the nearest integer (Note that the data type of the result may not necessarily be an <code>int</code> .)	>>> <code>2017 // 1e3</code> 2.0
%	Modulus (remainder)	Returns the remainder when the value on the left is divided by the value on the right	>>> <code>2017 % 1e3</code> 17.0
**	Exponentiation (power)	Returns the value on the left raised to the power of the value on the right	>>> <code>2 ** 1e3</code> 1.07150860719e+301

Note that, because `float` values are not as accurate as `int` values, any mathematical operation that mixes the two types will force the calculation to use the one with lower accuracy and thus produce a `float` result. For instance, all the examples in Table 4.9 return `float` values because `1e3` is a `float` value.

The division operator (`/`) is an exception to this rule as the division of whole numbers often does not result in a whole number. In Python, we can divide an `int` with another `int` and get back a `float`. If we want to perform a division that rounds down to the nearest integer, we would use **floor division** (`//`) instead.

Key Term

Floor division

Division operation that rounds down to the nearest integer

```
>>> 2017 / 1000
2.017
>>> 2017 // 1000
2
```

▲ Figure 4.77 Using the floor division operator

While the result of floor division is always a whole number, if any of the values used in the calculation is a `float`, then the result will still be a `float`. In such cases, you can safely use type casting to convert the result to an `int` if needed.

```
>>> 2017 // 1000.0
2.0
>>> int(2017 // 1000.0)
2
```

▲ Figure 4.78 The floor division operator may return a `float` value

A common programming task is to use the value in a variable for a calculation and then assign the result back into the variable. For convenience, Python provides additional assignment operators to help save some typing when performing this task.

▼ Table 4.10 Assignment operators that work with `int` and `float` data types

Operator	Example	Equivalent
<code>+=</code>	<code>x += a</code>	<code>x = x + a</code>
<code>-=</code>	<code>x -= a</code>	<code>x = x - a</code>
<code>*=</code>	<code>x *= a</code>	<code>x = x * a</code>
<code>/=</code>	<code>x /= a</code>	<code>x = x / a</code>
<code>//=</code>	<code>x //= a</code>	<code>x = x // a</code>
<code>%=</code>	<code>x %= a</code>	<code>x = x % a</code>
<code>**=</code>	<code>x **= a</code>	<code>x = x ** a</code>

Since numbers are naturally ordered, `int`s and `float`s can also be used with the following comparison operators (also known as relational operators) that return a `bool` depending on whether the value on the left is larger, smaller or equal to the value on the right:

▼ **Table 4.11** Relational operators that work with `int` and `float` data types

Operator	Name	Description	Examples
<code><</code>	Less than	Returns the <code>bool</code> value <code>True</code> if the value on the left is less than the value on the right and <code>False</code> if it is not	<code>>>> 2017 < 2017</code> <code>False</code> <code>>>> 2017 < 2018</code> <code>True</code> <code>>>> 2017 < 20.17</code> <code>False</code>
<code><=</code>	Less than or equal to	Returns the <code>bool</code> value <code>True</code> if the value on the left is less than or equal to the value on the right and <code>False</code> if it is not	<code>>>> 2017 <= 2017</code> <code>True</code> <code>>>> 2017 <= 2018</code> <code>True</code> <code>>>> 2017 <= 20.17</code> <code>False</code>
<code>></code>	Greater than	Returns the <code>bool</code> value <code>True</code> if the value on the left is greater than the value on the right and <code>False</code> if it is not	<code>>>> 2017 > 2017</code> <code>False</code> <code>>>> 2018 > 2017</code> <code>True</code> <code>>>> 2017 > 20.17</code> <code>True</code>
<code>>=</code>	Greater than or equal to	Returns the <code>bool</code> value <code>True</code> if the value on the left is greater than or equal to the value on the right and <code>False</code> if it is not	<code>>>> 2017 >= 2017</code> <code>True</code> <code>>>> 2018 >= 2017</code> <code>True</code> <code>>>> 2017 >= 20.17</code> <code>True</code>

4.6.2.3 String and List Operators

Just as `int` and `float` are similar in that they represent numbers, `str` and `list` are similar in that they represent a **sequence** of values. However, an important difference between the two types is that a `list` can contain items of any and different types but a `str` can only contain characters. Examples of characters include the upper-case and lower-case letters from A to Z, the digits 0 to 9, as well as punctuation symbols such as commas and exclamation points. You will learn more about characters in Chapter 5.

Key Term

Sequence

Any data type for storing multiple values in order; usually a `list`, `str` or `range`

When solving problems, we often need to join two sequences together (called **concatenation**) or repeat the contents of a sequence multiple times. To perform these tasks, Python reuses some of the mathematical operators you have just learnt.

Key Term

Concatenation

Joining two sequences to make a longer sequence

▼ **Table 4.12** Mathematical operators that work with `str` and `list` data types

Operator	Name	Description	Examples
+	Concatenation	Joins the sequence on the right onto the end of the sequence on the left	<code>>>> print("Computing" + "2017")</code> Computing2017
*	Repetition	Repeats the contents of a sequence a number of times	<code>>>> print("Computing" * 3)</code> ComputingComputingComputing <code>>>> print(3 * "2017")</code> 201720172017

Note that the addition (+) and multiplication (*) operators are exceptions. In general, `strs` and `lists` do not work with mathematical operators. For instance, trying to subtract one `str` from another `str` results in an error:

```
>>> "computer" - "er"
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    "computer" - "er"
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

▲ **Figure 4.79** Not all arithmetic operators work with `str` and `list`

In the previous example, Python outputs that the subtraction operator (-) does not know what to do with values of the provided data type `str` (that is, it cannot support the task given).

To perform more complex operations such as accessing or rearranging the values stored inside `strs` and `lists`, we will need to use the index operator and the slice operator.

We use the **index operator** in the form `sequence_name[i]`, where `i` is an `int` value. This `int` value is called the **index**. Taking the first value in a sequence as having a position number of 0, the index operator will return the value with the same position number as the index.

For example, in Figures 4.80 and 4.81, using the index operator with an index of 5 on the `str` `subject_name` will return the value of `t`, which holds the position number of 5 in the `str`.

```
>>> subject_name = "Computing"
>>> print(subject_name[5])
t
```

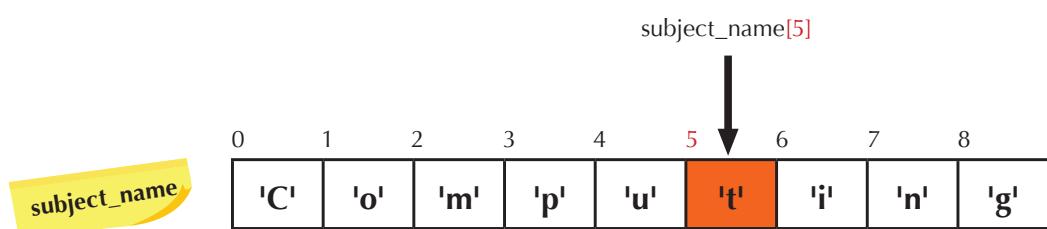
Key Terms

Index

An `int` value for the position of a value in a sequence

Index operator

Operator for returning a value in a sequence



▲ Figure 4.80 Using the index operator

If we want to extract a subset of values from a `str` or `list` (instead of just a single value), we can use the **slice operator**. We use this operator in the form `sequence_name[a:b]`, where `a` and `b` are `int` values indicating the start and stop indices respectively. This extracts the sequence of values or characters positioned from the start index up to but not including the stop index.

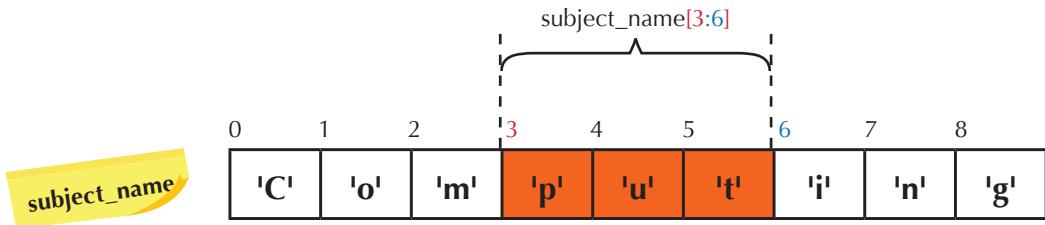
```
>>> subject_name = "Computing"
>>> print(subject_name[3:6])
put
```

Key Term

Slice operator

Operator for returning a sequence (not a value) by extracting values from a sequence

▲ Figure 4.82 Using the slice operator



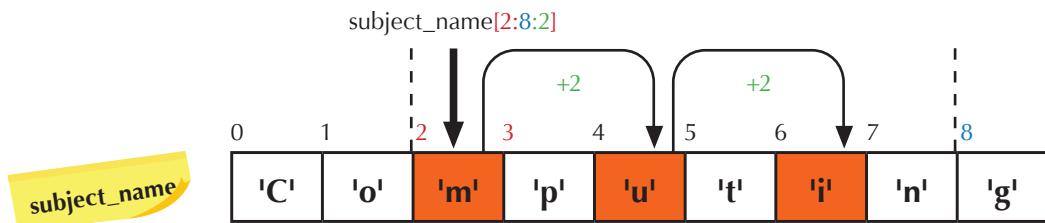
▲ Figure 4.83 Using the slice operator

If the start index is left out, it is treated as 0. If the stop index is left out, it is treated as the length of the `str` or `list`. Leaving out the stop index usually has the same meaning as making sure all remaining values of the `str` or `list` are included in the return value.

Although not often used, the slice operator can also accept an optional third `int` value in the form `sequence_name[a:b:c]`, where `a` and `b` are the start and stop indices while `c` is called the step. This extracts the sequence of values or characters positioned from the start index up to but not including the stop index in increments of the step.

```
>>> print("Computing"[2:8:2])
mui
```

▲ Figure 4.84 Using the slice operator with a step



▲ Figure 4.85 Using the slice operator with a step

As before, if the start index is left out, it is treated as 0. If the stop index is left out, it is treated as the length of the `str` or `list`. If the step is left out, it is treated as 1.

```
>>> print("Computing"[:8:2])
Cmui
>>> print("Computing"[2::2])
muig
>>> print("Computing"[2:8:])
mputin
```

▲ Figure 4.86 Using the slice operator with the start index, stop index or step left out

▼ Table 4.13 Index and slice operator syntax

Operator	Name	Description	Examples
[i]	Index	Returns the value in the <i>i</i> -th position of the sequence	>>> print("Computing"[0]) C >>> print("Computing"[3]) p >>> print("Computing)[-1]) g
[a:b]	Slice	Returns a sequence of values starting from the value at index <i>a</i> up to but not including the value at index <i>b</i> ; <i>a</i> is treated as 0 if omitted and <i>b</i> is treated as the length of the sequence if omitted	>>> print("Computing"[3:6]) put >>> print("Computing":7)) Computi >>> print("Computing":3:)) uting
[a:b:c]	Slice with step	Returns a sequence of values starting from the value at index <i>a</i> up to but not including the value at index <i>b</i> , in increments of <i>c</i> ; <i>a</i> is treated as 0 if omitted, <i>b</i> is treated as the length of the sequence if omitted and <i>c</i> is treated as 1 if omitted.	>>> print("Computing":2:8:2)) mui >>> print("Computing":8:2)) Cmu >>> print("Computing":2::2)) muig >>> print("Computing":2:8:)) mputin

By using the index operator on the left-hand side of an assignment, we can also change the value that is stored in a `list` at a particular index.

```
>>> scores = [85.0, 88.1, 72.9, 63.4]
>>> scores[2] = 50.0
>>> print(scores)
[85.0, 88.1, 50.0, 63.4]
```

▲ Figure 4.87 Changing a value in a list using the index operator

However, trying to change the characters in a `str` will result in an error. This is because `str` is an **immutable** type, meaning that once a `str` is created, its contents cannot be changed. For instance, suppose we wish to change the first character of “Computing” to a K so that the string becomes “Komputing”:

```
>>> subject_name = "Computing"
>>> subject_name[0] = 'K'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

▲ **Figure 4.88** Failing to change a character in a `str`

Unlike the example in Figure 4.87 where we could successfully change the contents of a `list`, for the example in Figure 4.88, Python outputs that `str`s are not capable of (does not support) changing their contents.

The best we can do is to create a new `str` by concatenating pieces from the original `str`:

```
>>> subject_name = "Computing"
>>> subject_name = 'K' + subject_name[1:]
>>> print(subject_name)
Komputing
```

Key Term

Immutable
Cannot be changed once created

▲ **Figure 4.89** Creating a new `str` to change the first letter of `subject_name`

The second line in Figure 4.89 creates a new `str` value of “Komputing” and assigns it back to `subject_name`. The previous `str` value of “Computing” is no longer referred to. Note that a new `str` has been created. This is not the same as changing characters in an existing `str`.

Another common task in solving problems is accessing the last value in a `str` or `list`. This can be done with some simple index calculation by using the `len()` function, as in Figure 4.90.

```
>>> subject_name = "Computing"
>>> letter = subject_name[len(subject_name)-1]
>>> print(letter)
g
```

▲ **Figure 4.90** Accessing the last value by using the `len()` function

If the index for an index operator is negative, Python treats it as a position counting from the opposite end of the sequence, treating the last value as position `-1`, the second-to-last value as position `-2`, and so on. This provides us with a shorter way to access the last value in a `str` or `list`.

```
>>> subject_name = "Computing"
>>> letter = subject_name[-1]
>>> print(letter)
g
```

▲ **Figure 4.91** Accessing the last value by using a negative index

Negative indices also work with the slice operator:

```
>>> subject_name = "Computing"
>>> letters = subject_name[4:-2]
>>> print(letters)
uti
```

▲ **Figure 4.92** Using a negative index with the slice operator

Finally, Python also provides a membership operator to test if a value is included in a `str` or `list`. Unlike the other operators you have learnt so far, the membership operator looks like a word (`in`) instead of a symbol, but it otherwise behaves the same way as other binary operators. The membership operator returns a `True` result if the value on the left exists in the `str` or `list` on the right, otherwise the result is `False`.

▼ **Table 4.14** Membership operator

Operator	Name	Description	Examples
<code>in</code>	Membership	Returns the <code>bool</code> value <code>True</code> if the value on the left can be found inside the sequence on the right and <code>False</code> if it cannot be found	<pre>>>> "C" in "Computing" True >>> "c" in "Computing" False</pre>

```
>>> 15 in [11, 13, 15, 17, 19]
True
>>> 16 in [11, 13, 15, 17, 19]
False
```

▲ **Figure 4.93** Using the membership operator

4.6.2.4 Boolean Operators

Many operators in Python inform you of their results by returning a `bool`. Table 4.15 shows the logical operators used for making decisions using this information.

▼ Table 4.15 Logical operators

Operator	Name	Description	Examples
not	Negation	Takes a <code>bool</code> and returns its opposite value	<code>>>> not True</code> False <code>>>> not False</code> True
and	Conjunction	Returns the <code>bool</code> value True if both values are True and False if either one or both values are False	<code>>>> True and True</code> True <code>>>> True and False</code> False
or	Disjunction	Returns the <code>bool</code> value True if either one or both values are True and False if both values are False	<code>>>> True or True</code> True <code>>>> True or False</code> True

These operators allow multiple pieces of information to be combined to make a conclusion or decision. For instance, we can use these operators to decide whether we can leave home based on the weather outside. Suppose `is_raining` is a `bool` that represents whether or not it is currently raining, and `have_umbrella` is a `bool` that represents whether or not there is an umbrella available. The decision to leave home may then be written as a logic statement:

```
>>> leave_home = (not is_raining) or (is_raining and have_umbrella)
```

▲ Figure 4.94 General logic statement to decide whether we can leave home based on the weather outside and whether an umbrella is available

This statement translates very naturally into English as “we can leave home if it is not raining or if it is raining and we have an umbrella”. Suppose it is raining and there is no umbrella available. In this case, `is_raining` is `True` and `have_umbrella` is `False`:

```
>>> is_raining = True
>>> have_umbrella = False
>>> leave_home = (not is_raining) or (is_raining and have_umbrella)
>>> print(leave_home)
False
```

▲ Figure 4.95 Specific example of using a logic statement

Python gives a result of `False` for `leave_home`, which leads us to the conclusion that we cannot leave home yet. This is consistent with our English translation of the statement.



Quick Check 4.6

- For each line of source code in Figure 4.96, predict what will be displayed on the screen (if any) or if an error will occur. Check your answers by entering the same source code, line by line, in IDLE's shell window and observing the results.

```

1 print(len(range(3)))
2 print(len(3))
3 phrase = "Hello, World!"
4 print(len(phrase))
5 print(len([phrase]))
6 print(list(range(7)))
7 print(list(range(len("2017"))))
8 x = len(range(-2, 3))
9 print(x, x)

```

▲ **Figure 4.96** Source code for entry in interactive mode

- Write a Python program that asks the user to input a `str` phrase and outputs the number of characters in the `str` that was entered.
- For each line of source code in Figure 4.97, predict what will be displayed on the screen (if any) or if an error will occur. Check your answers by entering the same source code, line by line, in IDLE's shell window and observing the results.

```

1 x = 2
2 x **= 3
3 x / 10
4 print(x)
5 print(1 + 1 <= "2")
6 print(3 * "2")
7 quotient = 55 // 7
8 remainder = 55 % 7
9 print(quotient * 7 + remainder == 55)
10 print(remainder + quotient * 7)
11 print((remainder + quotient) * 7)
12 55 = x
13 2.0 ** 3 == x
14 print(-55 // 10)

```

▲ **Figure 4.97** Source code for entry in interactive mode

4. Write a UDF with the function name `area_of_triangle` that takes in the base and height of a triangle as input and gives the triangle's area as its return value.
5. Suggest what should be written in the box in the program in Figure 4.98 so that it correctly outputs whether the number of characters in `phrase` is a multiple of 3.

```
phrase = input("Enter phrase: ")  
print()
```

▲ Figure 4.98 Program template

6. Suggest what should be written in the box in the program in Figure 4.99 so that it outputs `True` when both the following conditions are met (and `False` if either is not met):
 - The first character of `name` is the same as its last character.
 - `name` does not contain a space character.

```
name = input("Enter name: ")  
print()
```

▲ Figure 4.99 Program template

7. For each line of source code in Figure 4.100, predict what will be displayed on the screen (if any) or if an error will occur. Check your answers by entering the same source code, line by line, in IDLE's shell window and observing the results.

```
1  data = [7, 9, 3, 5]  
2  print(data[3:])  
3  print(data[:1])  
4  print(data + [2])  
5  print(data + 2)  
6  print(data[-3:-2])  
7  print(data[len(data)- 1])  
8  print(data[len(data)])  
9  print(['A', 'B', 'C'][1])  
10 ['F', 'G'][0] = 'A'  
11 "FG"[0] = 'A'
```

▲ Figure 4.100 Source code for entry in interactive mode

8. For each line of source code in Figure 4.101, predict what will be displayed on the screen (if any) or if an error will occur. Check your answers by entering the same source code, line by line, in IDLE's shell window and observing the results.

```

1  x = 5
2  print(x > -5 and x < 5)
3  print(x >= -5 and x <= 5)
4  print(x > -5 or x < 5)
5  print(not not True)
6  print(not True and False)
7  print(not (True and False))

```

▲ **Figure 4.101** Source code for entry in interactive mode

9. The following program uses UDFs to a prompt for a message, then prints out the message in a box made up of a repeated symbol:

```

1  SYMBOL = '*'
2
3  def print_border(length):
4      border = SYMBOL * length
5      print(border)
6
7  def print_box(message):
8      left_side = SYMBOL + ' '
9      right_side = ' ' + SYMBOL
10     middle_line = left_side + message + right_side
11     print_border(len(middle_line))
12     print(middle_line)
13     print_border(len(middle_line))
14
15 message = input("Enter a message: ")
16 print()
17 print_box(message)

```

▲ **Figure 4.102** Program that prints out a message in a box

For each of the following appearances of variables and constants in the program, identify whether the variable or constant being used is local or global:

- a) SYMBOL on line 4
- b) length on line 4

- c) border on line 5
- d) SYMBOL on line 8
- e) left_side on line 10
- f) message on line 10
- g) right_side on line 10
- h) middle_line on line 12
- i) message on line 17

4.7 Control Flow Statements

So far, we have covered all the basic Python commands for storing, reading and manipulating values of different data types. With these tools, we can use Python to perform some interesting tasks. For instance, use a text editor (or Python's built-in IDLE editor) to enter the program in Figure 4.103 into a file and save it as `message_box.py`:

#	Program 4.9 message_box.py
1	<code>message = input("Enter a message: ")</code>
2	<code>print()</code>
3	<code>print("*" * (len(message) + 4))</code>
4	<code>print("* " + message + " *)</code>
5	<code>print("*" * (len(message) + 4))</code>

▲ **Figure 4.103** Program for printing a message box

If you are using Python's built-in IDLE editor, press the F5 key while the file is open. The F5 key is used to run the Python file that is currently open.

When the program runs, it should prompt you to enter a message. Type in a short message and press Enter. The program will then print out your message in a box made of asterisks:

```
Enter a message: Hello, World!

*****
* Hello, World! *
*****
```

▲ **Figure 4.104** Result of running message_box.py

The Python commands we have learnt so far only allow us to use sequence constructs that arrange instructions in a fixed order. We still have not learnt how to produce selection and iteration constructs similar to those used in Chapter 3.

Recall that the decision symbols and flow lines of a flowchart allow instructions to be skipped or repeated as necessary. The rest of this section will focus on how to similarly skip or repeat instructions in Python using **control flow statements**.

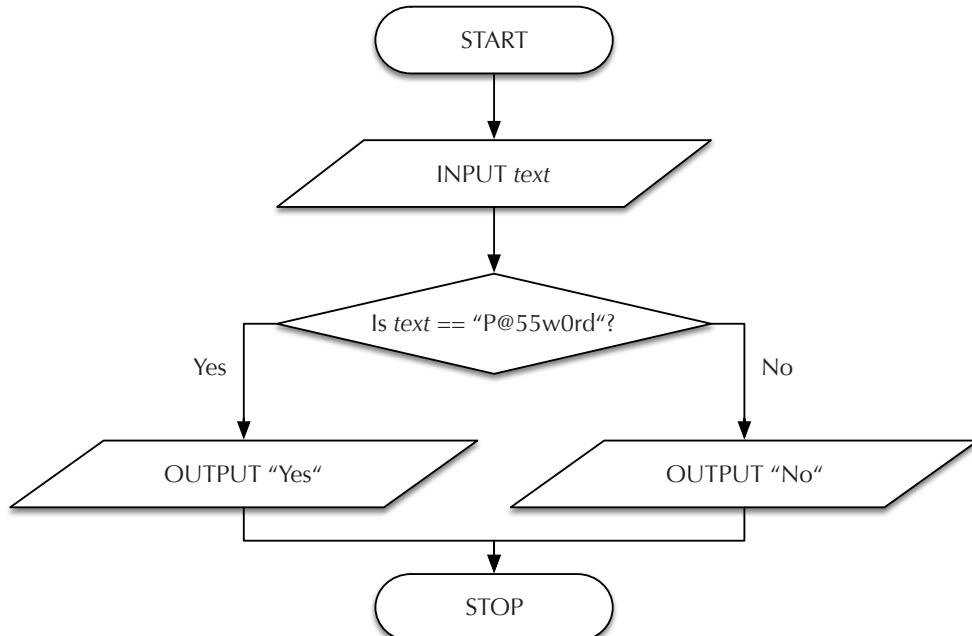
Key Term

Control flow statement

Programming instruction that changes the order in which other instructions are run

4.7.1 if-elif-else Statements

The most basic way to determine which flowchart instructions are followed is by using a decision symbol to split the program flow into two:



▲ Figure 4.105 Password checker flowchart

The flowchart in Figure 4.105 will output “Yes” if the input text is the same as the special phrase “P@55w0rd”, otherwise it will output “No”. We can get the same result using the `if-else` statement in Python:

#	Program 4.10 password.py
1	<code>text = input("Enter text: ")</code>
2	<code>if text == "P@55w0rd":</code>
3	<code> print("Yes")</code>
4	<code>else:</code>
5	<code> print("No")</code>

▲ **Figure 4.106** Using the `if-else` statement

If a program is like an essay, a **statement** is like a sentence. The `if-else` statement in `password.py` is an example of a **compound statement** that can contain multiple statements. You can run the program to confirm that it follows the flowchart in Figure 4.105.

Note that some of the lines in the program `password.py` are indented. Python uses indentation (the number of spaces at the start of each line) to determine which lines of code belong to the same branch. Besides being a good programming practice, in Python it is a *requirement* to indent lines of code correctly.

In this textbook, we will use four spaces to represent each level of indentation. This is the recommended and typical amount of indentation used in Python.

Suppose that now we want to perform more commands in each branch of `password.py` and for the program to output “Goodbye” just before it ends (whether or not the input text matches the password).

Key Terms

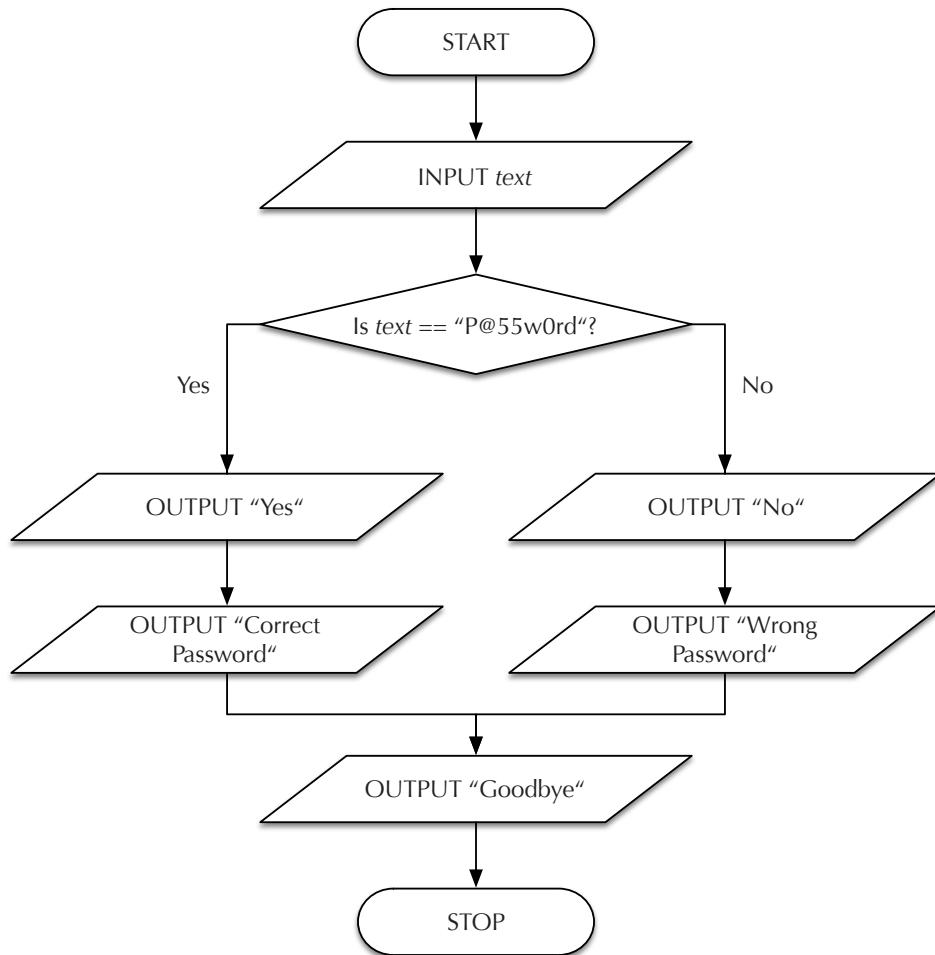
Compound statement

Statement that can contain multiple statements as its components

Statement

Element of source code that represents a complete instruction in a programming language

The amended flowchart and Python source may look like this:



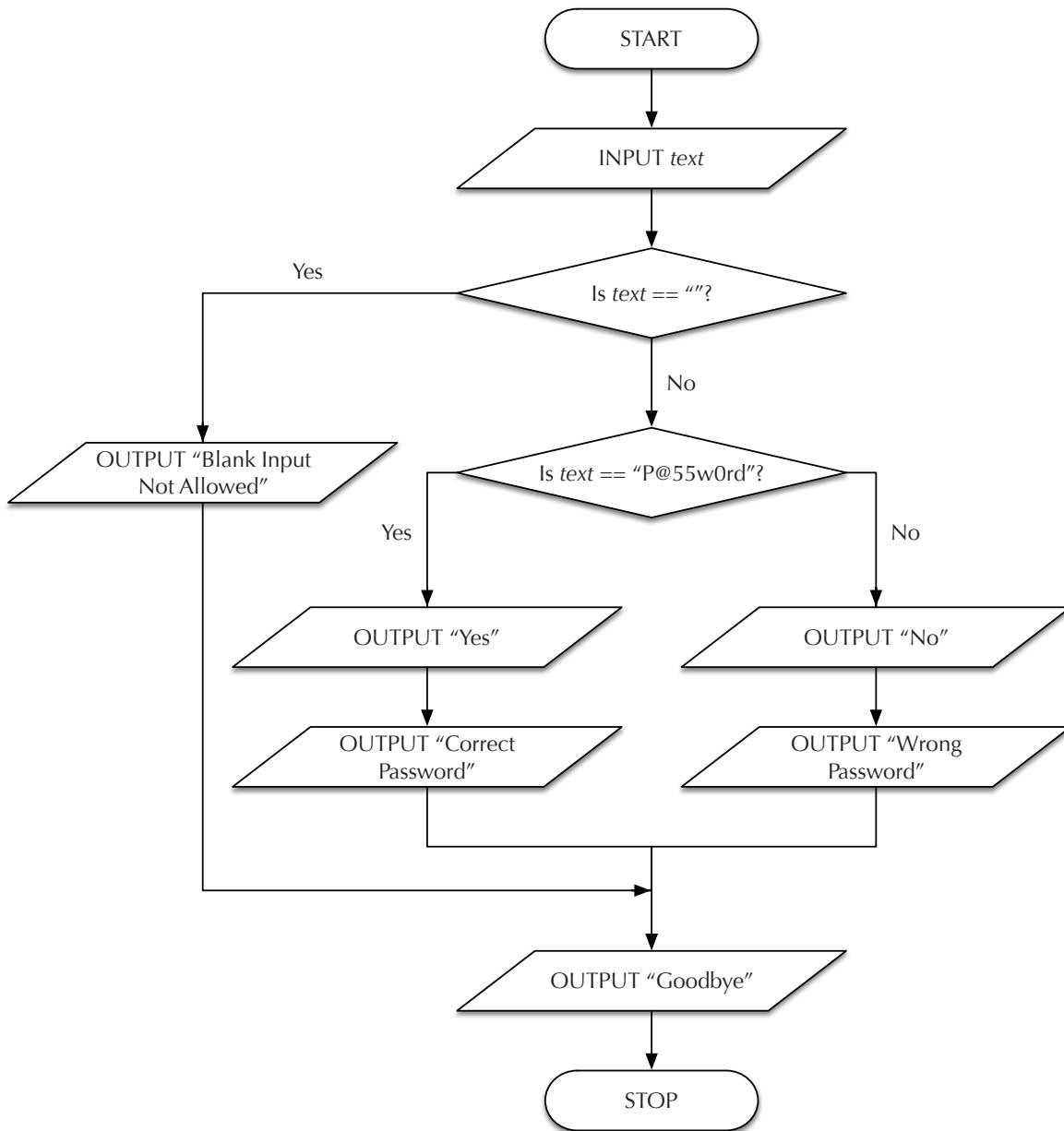
▲ **Figure 4.107** Password checker flowchart amended to include “Goodbye” output

#	Program 4.11 password_2.py
1	<code>text = input("Enter text: ")</code>
2	<code>if text == "P@55w0rd":</code>
3	<code> print("Yes")</code>
4	<code> print("Correct Password")</code>
5	<code>else:</code>
6	<code> print("No")</code>
7	<code> print("Wrong Password")</code>
8	<code>print("Goodbye")</code>

▲ **Figure 4.108** Password checker program amended to include “Goodbye” output

In Figure 4.108, we see that the line that outputs “Goodbye” is not indented. This is how Python knows that this line is not part of the `if-else` statement and that it should be run after the `if-else` statement is completed, no matter which branch was previously followed.

It is also possible to nest an `if-else` statement inside another `if-else` statement. For instance, we might want to provide a separate error message if the text entered is blank. One possible solution would be for the flowchart and Python source to look like this:



▲ **Figure 4.109** Password checker with detection of blank input added

#	Program 4.12 password_3.py
1	<code>text = input("Enter text: ")</code>
2	<code>if text == "":</code>
3	<code> print("Blank Input Not Allowed")</code>
4	<code>else:</code>
5	<code> if text == "P@55w0rd":</code>
6	<code> print("Yes")</code>
7	<code> print("Correct Password")</code>
8	<code> else:</code>
9	<code> print("No")</code>
10	<code> print("Wrong Password")</code>
11	<code>print("Goodbye")</code>

▲ **Figure 4.110** Password checker program with detection of blank input added

Note that the `if-else` statements from line 5 to line 10 are indented by one additional level compared to the rest of the program. Hence, line 5 and line 8 have one level of indentation, while lines 6, 7, 9 and 10 have two levels of indentation. In Python, it is important to indicate which branch each line of code belongs to.

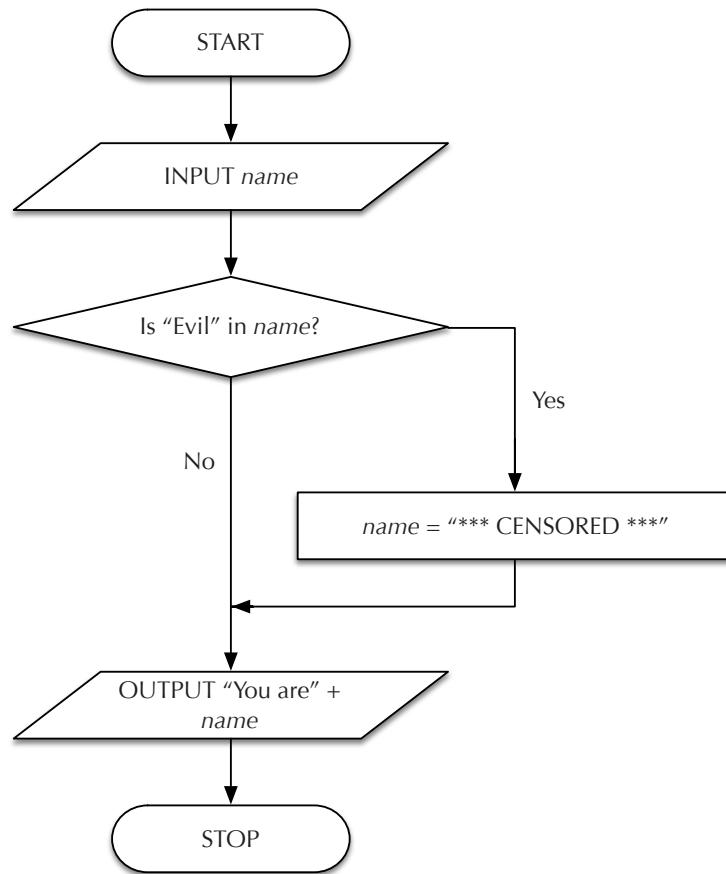
Nested `if-else` statements are common. However, excessive indentation can make the code difficult to read. To avoid this, Python provides a way to combine nested `if-else` statements into a single statement so that there is no need to increase the level of indentation. For example, `password_3.py` can be re-written as:

#	Program 4.13 password_3_elif.py
1	<code>text = input("Enter text: ")</code>
2	<code>if text == "":</code>
3	<code> print("Blank Input Not Allowed")</code>
4	<code>elif text == "P@55w0rd":</code>
5	<code> print("Yes")</code>
6	<code> print("Correct Password")</code>
7	<code>else:</code>
8	<code> print("No")</code>
9	<code> print("Wrong Password")</code>
10	<code>print("Goodbye")</code>

▲ **Figure 4.111** Password checker with additional indentation avoided by using `elif`

The `elif` keyword in line 4 of `password_3 elif.py` replaces the `else` and `if` in lines 4 and 5 of the original `password_3.py` and avoids additional indentation of the remaining code. When reading the source code, treat `elif` as an abbreviation for “else if”. Besides these cosmetic changes, this program otherwise behaves exactly the same as the original `password_3.py`.

Some algorithms do not require any instructions to be followed for the `else` portion of an `if-else` statement. The `else` keyword should then be omitted entirely. For instance, the following program tries to censor the word “Evil” if it is entered as part of a name:



▲ **Figure 4.112** Name censor flowchart

In Figure 4.113, name has its contents replaced by a censorship notice if “Evil” is found in the name entered by the user. Otherwise, the program continues normally by outputting the name with no modification to its contents.

#	Program 4.14 censor.py
1	<code>name = input("Enter name: ")</code>
2	<code>if "Evil" in name:</code>
3	<code> name = "*** CENSORED ***"</code>
4	<code>print("You are " + name)</code>

▲ **Figure 4.113** Name censor program

The syntax rules in Figure 4.114 summarise how the words and indentation for an `if-elif-else` statement can be arranged:

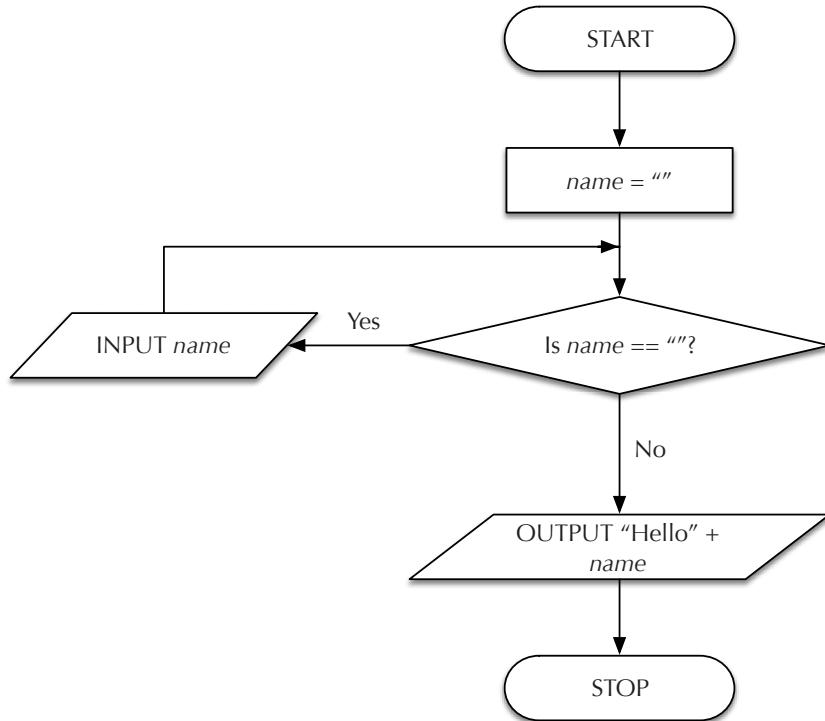
Syntax 4.5 if-elif-else Statement
<pre>if condition: commands when condition is True</pre>
<pre>if condition: commands when condition is True else: commands when condition is False</pre>
<pre>if condition_1: commands when condition_1 is True elif condition_2: commands when condition_1 is False and condition_2 is True else: commands when condition_1 is False and condition_2 is False</pre>

▲ **Figure 4.114** Summary of if-elif-else statement

4.7.2 while Loops

Flowcharts can be made to repeat commands multiple times by using flow lines that return to a previous decision symbol.

In Figure 4.115, the algorithm will keep asking for a new value of name as long as name remains blank.



▲ Figure 4.115 Personal greeting flowchart

We can achieve a similar effect in Python using the `while` statement:

#	Program 4.15 greeting.py
1	<code>name = ""</code>
2	<code>while name == "":</code>
3	<code> name = input("Enter name: ")</code>
4	<code> print("Hello " + name)</code>

▲ Figure 4.116 Personal greeting program

Once `name` is correctly entered, the program proceeds to run line 4 and `name` is printed with the greeting “Hello”. Since the flowchart for repeated commands will always have a loop, the `while` statement is also called a **while loop**.

Like the `if-elif-else` statement, Python uses indentation to determine which lines of code belong to the `while` statement. In the case of `greeting.py`, only line 3 belongs to the `while` statement and it is run repeatedly as long as nothing is entered into `name` (in other words, as long as `name == ""` is True).

Note that, exactly like the flowchart, Python checks the condition after the `while` keyword at least once while the program is running *and* every time after the commands in the loop are complete. This means that if we set `name` to anything other than an empty string, Python will skip the contents of the loop.

Key Term

Loop

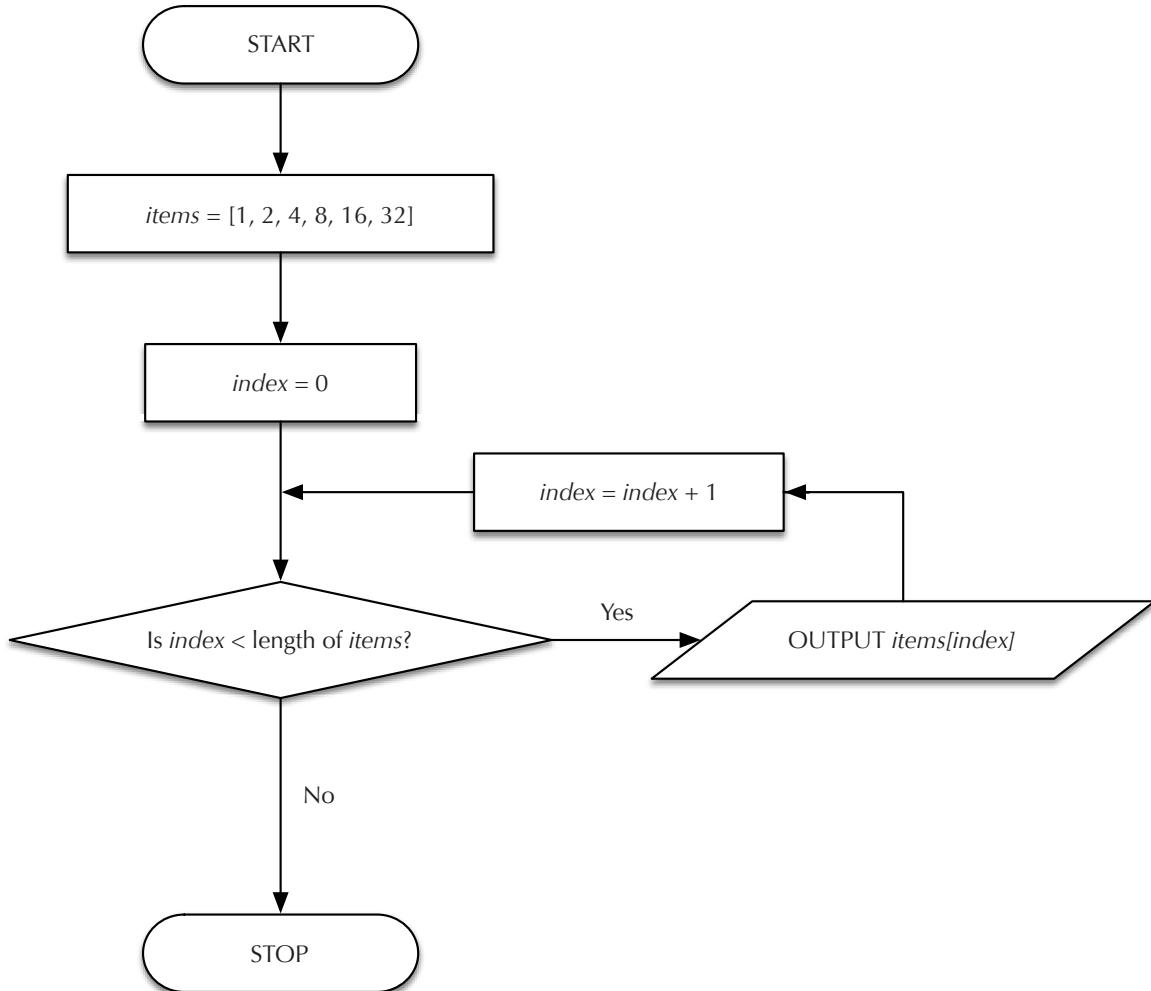
Set of instructions or commands that are repeated until a condition is met

#	Program 4.16 greeting_skiploop.py
1	<code>name = "Computing"</code>
2	<code>while name == "":</code>
3	<code>name = input("Enter name: ")</code>
4	<code>print("Hello " + name)</code>

```
===== RESTART: C:/Examples/greeting.py =====
Enter name: Nadiah
Hello Nadiah
>>>
===== RESTART: C:/Examples/greeting_skiploop.py =====
Hello Computing
>>>
```

▲ **Figure 4.117** Demonstration that `while` condition is always checked at least once

Additional lines with the same indentation belong to the same while loop. For instance, the flowchart and program in Figures 4.118 and 4.119 go through (or iterate) the contents of a list and output each item on a separate line.



▲ **Figure 4.118** Flowchart for printing a list

#	Program 4.17 printlist_while.py
1	<code>items = [1, 2, 4, 8, 16, 32]</code>
2	<code>index = 0</code>
3	<code>while index < len(items):</code>
4	<code> print(items[index])</code>
5	<code> index = index + 1</code>

▲ **Figure 4.119** Program for printing a list

The syntax for `while` statements is summarised in Figure 4.120.

Syntax 4.6 while Statement

```
while condition:  
    commands to repeat while condition is True
```

▲ **Figure 4.120** Syntax for `while` statement

Note that, by itself, the `while` loop cannot represent some of the more complex flowcharts we have encountered. This is because Python is a **structured programming language** that does not allow programs to jump from one command to another easily like you can using flow lines in a flowchart. Instead, Python programs are encouraged to use the same sequence, selection and iteration constructs that we learnt in Chapter 3.

Nonetheless, sometimes an algorithm requires the program to go back to the start of the loop or to exit the loop early. For such cases, Python provides the `continue` and `break` keywords.

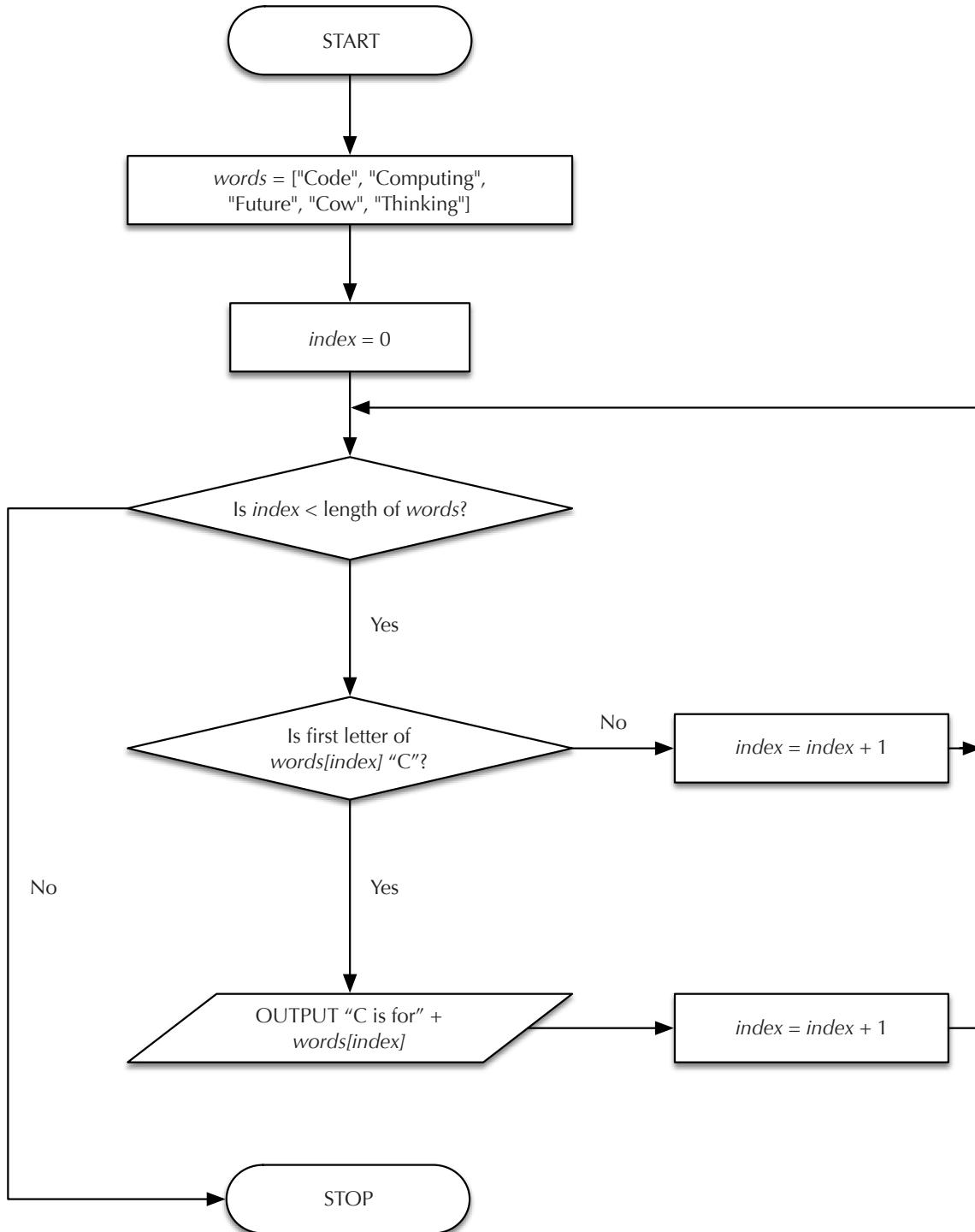
The `continue` keyword causes Python to skip the remaining commands in the loop and go straight to deciding whether the loop should run again by testing the condition after the `while` keyword. For example, suppose we have a loop that (by default) outputs “C is for word” for every word in a list. However, if a word does not start with C, we can decide that the default behaviour does not apply and that the program should advance to the next word instead.

Key Term

Structured programming language

Programming language that either encourages or is limited to the use of sequence, selection and iteration constructs that have exactly one entry point and one exit point

The flowchart and source code in Figures 4.121 and 4.122 demonstrate how such an algorithm can be implemented using the `continue` keyword:



▲ **Figure 4.121** Flowchart demonstrating `continue`

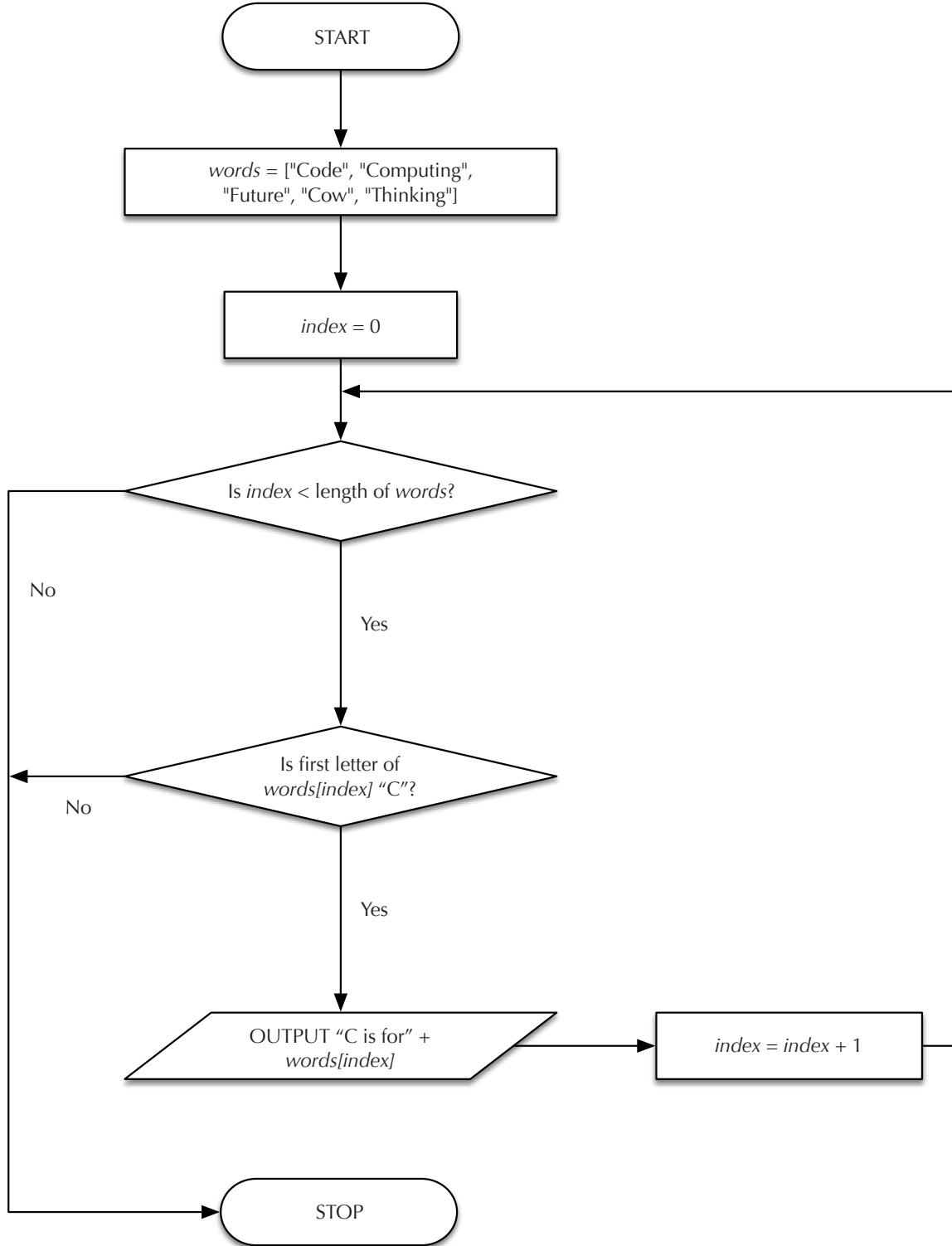
#	Program 4.18 firstletter_continue.py
1	<code>words = ["Code", "Computing", "Future", "Cow", "Thinking"]</code>
2	<code>index = 0</code>
3	<code>while index < len(words):</code>
4	<code> if words[index][0] != 'C':</code>
5	<code> index = index + 1</code>
6	<code> continue</code>
7	<code> print("C is for " + words[index])</code>
8	<code> index = index + 1</code>

```
===== RESTART: C:/Examples/firstletter_continue.py =====
C is for Code
C is for Computing
C is for Cow
>>>
```

▲ **Figure 4.122** Demonstration of using `continue` keyword to skip iterations

The `break` keyword, on the other hand, causes Python to immediately skip the remaining commands and exit the loop completely. For example, suppose we once again have a loop that (by default) outputs “C is for word” for every word in a list. However, this time we want the loop to end once a word that does not start with C is detected.

The flowchart and source code in Figures 4.123 and 4.124 demonstrate how such an algorithm can be implemented using the `break` keyword:



▲ **Figure 4.123** Flowchart demonstrating break

#	Program 4.19 firstletter_break.py
1	<code>words = ["Code", "Computing", "Future", "Cow", "Thinking"]</code>
2	<code>index = 0</code>
3	<code>while index < len(words):</code>
4	<code> if words[index][0] != 'C':</code>
5	<code> break</code>
6	<code> print("C is for " + words[index])</code>
7	<code> index = index + 1</code>

```
===== RESTART: C:/Examples/firstletter_break.py =====
C is for Code
C is for Computing
>>>
```

▲ **Figure 4.124** Demonstration of using break keyword to end a loop early

The program in Figure 4.125 demonstrates how the `continue` and `break` keywords work by playing a simple “Simon Says” game.

#	Program 4.20 simonsays.py
1	<code># Constants</code>
2	<code>MAGIC_WORDS = "Simon says "</code>
3	<code>QUIT_COMMAND = "Quit"</code>
4	
5	<code>print("Enter \"\" + QUIT_COMMAND + "\" to end game.")</code>
6	<code>while True:</code>
7	<code> command = input("Enter command: ")</code>
8	<code> if command == QUIT_COMMAND:</code>
9	<code> break</code>
10	<code> elif command[:len(MAGIC_WORDS)] != MAGIC_WORDS:</code>
11	<code> print("We ignore the command")</code>
12	<code> continue</code>
13	<code> print("We " + command[len(MAGIC_WORDS):])</code>
14	<code>print("Thanks for playing!")</code>

▲ **Figure 4.125** Program for playing Simon Says

In this game, the computer will repeat what the user enters as long as the input starts with the words “Simon says”. This is the default behaviour that is repeated using a `while` loop that seems to repeat forever as its condition is always `True` (line 9). However, if the input matches the special word “Quit”, the loop ends immediately using the `break` keyword and the game ends with the thank-you message. On the other hand, if the input does not start with “Simon says”, the default behaviour is skipped and we proceed to ask for the next input instead. Try playing the game a few times and see if the new keywords behave the way you expect them to.

4.7.3 for-in Loops

Previously, we saw an example of iterating through the items of a `list` by using an index variable:

#	Program 4.21 printlist_while.py
1	<code>items = [1, 2, 4, 8, 16, 32]</code>
2	<code>index = 0</code>
3	<code>while index < len(items):</code>
4	<code>print(items[index])</code>
5	<code>index = index + 1</code>

▲ **Figure 4.126** Iterating through a list using a counter

Iterating through the items of a sequence is such a common task that Python provides a shortcut for doing so: the `for-in` statement. When we use the `for-in` statement, each item in the sequence is assigned to a variable automatically before the loop is run. For instance, we can simplify the code shown in Figure 4.126 to become the code in Figure 4.127:

#	Program 4.22 printlist_for.py
1	<code>items = [1, 2, 4, 8, 16, 32]</code>
2	<code>for item in items:</code>
3	<code>print(item)</code>

▲ **Figure 4.127** Iterating through a list without a counter using a `for-in` loop

Note that this new code no longer has an `int` counter (`index`). Instead it has a variable `item` that is assigned a new item from the list each time the loop repeats.

The syntax for `for-in` statements is shown in Figure 4.128:

Syntax 4.7 for-in Statement

```
for name in sequence:  
    commands to repeat for each item in the sequence
```

▲ **Figure 4.128** Syntax for `for-in` statement

The `for-in` loop also works with the `range()` function that you learnt earlier. As with lists, the loop is run for each item in the range. By using the length of a list as the argument for `range()`, we have an efficient way to iterate through a list with a counter variable.

#	Program 4.23 printlist_range.py
1	<code>items = [1, 2, 4, 8, 16, 32]</code>
2	<code>for index in range(len(items)):</code>
3	<code> print(items[index])</code>

▲ **Figure 4.129** Iterating through a list with a counter using a `for-in` loop and `range()`

Notice that there is no longer a need to manually increase `index` by 1 as the `for-in` loop will automatically update `index` with the next item in the provided sequence.

The `continue` and `break` keywords work with `for-in` loops as well.



Quick Check 4.7

1. Write a program that lets the user input a list of `strs` by asking for each item of the list in order. Receiving an empty `str` from the user would indicate the end of the list. The program would then output the list on the screen.

▼ **Table 4.16** Input and output requirements for the list input problem

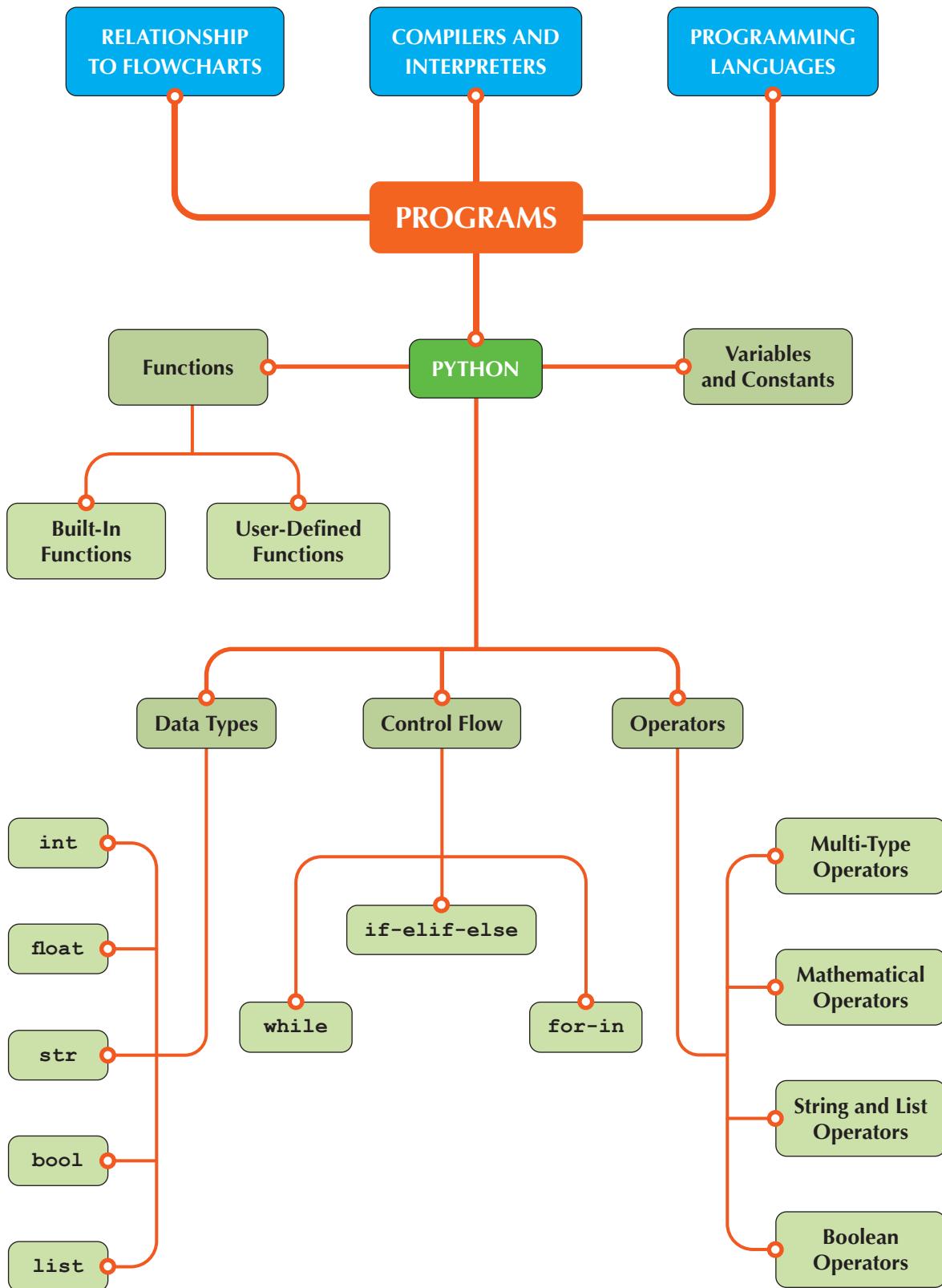
Input	Output
<ul style="list-style-type: none"> • A sequence of non-empty <code>strs</code>, in order, ending with an empty <code>str</code> 	<ul style="list-style-type: none"> • Printed list of all input <code>strs</code> arranged in the correct order

2. Write a program that lets the user input a password. To make sure that the password has been entered correctly, the program should ask for it twice. If the two entered passwords are different, the program should output “Invalid” and repeat the input process all over again. The program should output “Valid” and end when the two entered passwords match.

▼ **Table 4.17** Input and output requirements for the password input problem

Input	Output
<ul style="list-style-type: none"> • <code>password</code>: password entered by user • <code>password_again</code>: password entered again by user 	<ul style="list-style-type: none"> • “Invalid” whenever <code>password</code> and <code>password_again</code> do not match, repeated as many times as needed; “Valid” when the two inputs match

Chapter Summary



Review Questions

1. Let us revisit the situation at the beginning of this chapter where Alex is responsible for finding out which loaned books are overdue in his school library
 - a) Alex studies the problem and decides to use `int` values with the digits arranged as YYYYMMDD to represent day DD, month MM and year YYYY. Table 4.18 shows some examples of how dates would be represented in Alex's program.

▼ **Table 4.18** Representation of dates in Alex's program

Date	<code>int</code> value
1 January 2017	20170101
22 October 2017	20171022
11 November 2017	20171111
1 January 2018	20180101

Explain why Alex's method of representation will simplify his program later when he needs to compare dates.

- b) The input and output requirements for Alex's problem are provided in Table 4.19. Write a program that processes the provided inputs and correctly prints out the titles of all the overdue books. In your program, write and use a UDF that takes in the due date of a book and today's date as input arguments, then returns `True` if the due date has passed and `False` otherwise. Assume that the input data will always be valid.

▼ **Table 4.19** Input and output requirements for the problem of overdue books using lists

Input	Output
<ul style="list-style-type: none"> • <code>today</code>: <code>int</code> representation of today's date • <code>titles</code>: titles of loaned book (via text file) • <code>duedates</code>: corresponding <code>int</code> representation of due dates (via text file) 	<ul style="list-style-type: none"> • Titles of all loaned books with due dates earlier than today

A program template (Figure 4.130) is provided as follows:

#	Program 4.24 overdue_template.py
1	<code># This program outputs the titles of all overdue books.</code>
2	<code></code>
3	<code># Input</code>
4	<code>today = int(input("Enter today's date in YYYYMMDD: "))</code>
5	<code>titles = [</code>
6	<code> # The following data is copied from the given text file.</code>
7	<code> "How to Solve a Mystery", "Vacant Memories",</code>
8	<code> "The Cybersnake Chronicles", "Music History",</code>
9	<code> "Like Tears in Rain", "Out of the Abyss"</code>
10	<code>]</code>
11	<code>duedates = [</code>
12	<code> # The following data is copied from the given text file.</code>
13	<code> 20170727, 20170704, 20170711, 20170630, 20170703, 20170707</code>
14	<code>]</code>
15	<code></code>
16	<code># Process and Output</code>
17	<code>...Fill in this section...</code>

▲ **Figure 4.130** Program template

- Write a program to extract the hour, minute and second values (as `ints`) from a time string that is provided in the format “`HH:MM:SS`”. Assume that the input data will always be valid.

▼ **Table 4.20** Input and output requirements for the problem of reading a time string

Input	Output
<ul style="list-style-type: none"> <code>time_str</code>: time string in the format “<code>HH:MM:SS</code>”, where the hour value must be from 0 to 23 inclusive, minute value must be from 0 to 59 inclusive, and second value must be from 0 to 59 inclusive 	<ul style="list-style-type: none"> Hour value, minute value and second value, each on a separate line in the above order

3. Write a program to calculate the number of seconds between a start time and an end time, which are both provided in the format “HH:MM:SS”. Assume that the input data will always be valid, the end time is after the start time, and both times occur on the same day.

▼ **Table 4.21** Input and output requirements for the time interval problem

Input	Output
<ul style="list-style-type: none">• <code>start_time</code>: start time in the format “HH:MM:SS”, where the hour value must be from 0 to 23 inclusive, minute value must be from 0 to 59 inclusive, and second value must be from 0 to 59 inclusive• <code>end_time</code>: end time in the format “HH:MM:SS”, where the hour value must be from 0 to 23 inclusive, minute value must be from 0 to 59 inclusive, and second value must be from 0 to 59 inclusive	<ul style="list-style-type: none">• Number of seconds between the start time and end time

How Can Programs Be Used to Solve Problems?

Siti is a school librarian. She has been asked to develop a program to manage the loan records and check the availability of books at the library. Can you help her figure out how she can do this?



In the previous chapters, we learnt how to specify the required inputs and outputs for a problem, how to develop a possible solution in the form of an algorithm, and how to implement such algorithms using a computer by writing programs in Python.

For some problems and programs, it may not be easy to specify the inputs and outputs or the correct algorithm to use. Hence, it can be useful to follow a more structured approach.

In this chapter, you will learn the stages in developing a program, some built-in functions for manipulating numbers and strings, as well as a catalogue of common problems and solutions.



By the end of this chapter, you should be able to:

- Understand and describe the stages in developing a program:
 - Gather requirements
 - Plan solutions
 - Write code
 - Test and refine code
 - Deploy code
- Understand, use and justify the use of built-in mathematical and string functions in programs.
- Produce a programming solution for a given problem to:
 - Find the minimum/maximum value in a list
 - Find the average of a list of numeric values
 - Search for an item in a list and report the result of the search
 - Find check digits
 - Find the length of a string of characters
 - Extract required characters from a string of characters

5.1 Stages in Developing a Program

In general, there are five stages in developing a program.



▲ Figure 5.1 Stages in developing a program

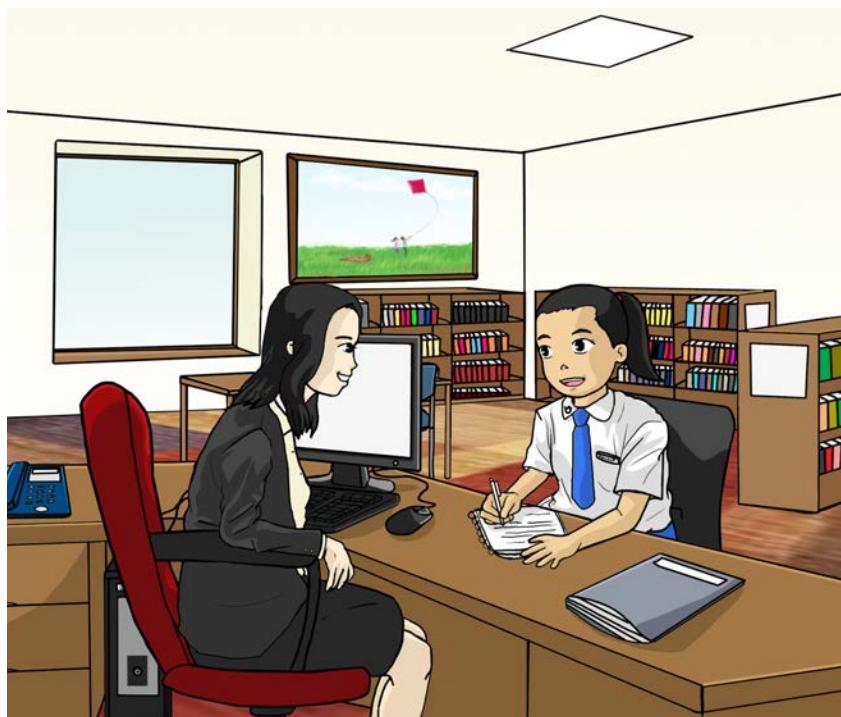
5.1.1 Gather Requirements

The goal of this initial stage is to determine the nature of the problem, why a program is needed and what the program is expected to do. If this stage is not performed properly, it is likely that any program that is developed will not actually solve the problem. Some of the tasks that can be done during this stage:

- Interviewing the intended audience of the program to understand the nature of the problem or their expectations
- Specifying the complete set of inputs that is necessary for the problem and how the inputs can be supplied to the program being developed
- Specifying the complete set of outputs that is necessary for the problem and the format for the output

Consider the situation in which Siti is required to develop a program to manage the loan records and check the availability of books at the school library. She interviews the librarians in her school to understand why a program is needed and what the program is expected to do. They give the following responses:

- “When someone asks me if a book is available, I have to go check the bookshelf. This can take a long time. It would be faster if I could find out the answer from the computer.”
- “This is a very small library so no two books have the same title.”
- “We only have one computer connected to a keyboard and screen. The computer has a list of book titles and their availability in separate text files.”



▲ Figure 5.2 Siti interviewing a librarian

From the information that she has gathered, Siti concludes that the program would need the following inputs and output shown in Table 5.1. She has also given each input a descriptive name to work with.

▼ Table 5.1 Input and output requirements for the book availability problem

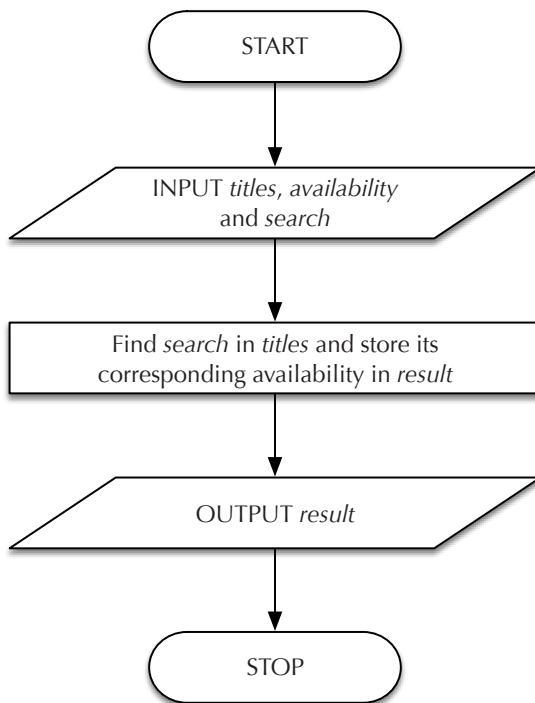
Input	Output
<ul style="list-style-type: none"> • titles: list of book titles (via text file) • availability: list of availability of each corresponding book in titles (via text file) • search: title of book to check availability 	<ul style="list-style-type: none"> • Whether the book in search is available

5.1.2 Plan Solutions

After gathering and refining the required inputs and outputs of the program, the next stage is where the skills of decomposition, pattern recognition and generalisation are used to plan and lay out a solution. The goal of this stage is to consider the options available before any code is written, and to choose an algorithm based on the resources available (such as manpower and time). Some of the tasks that can be done at this stage:

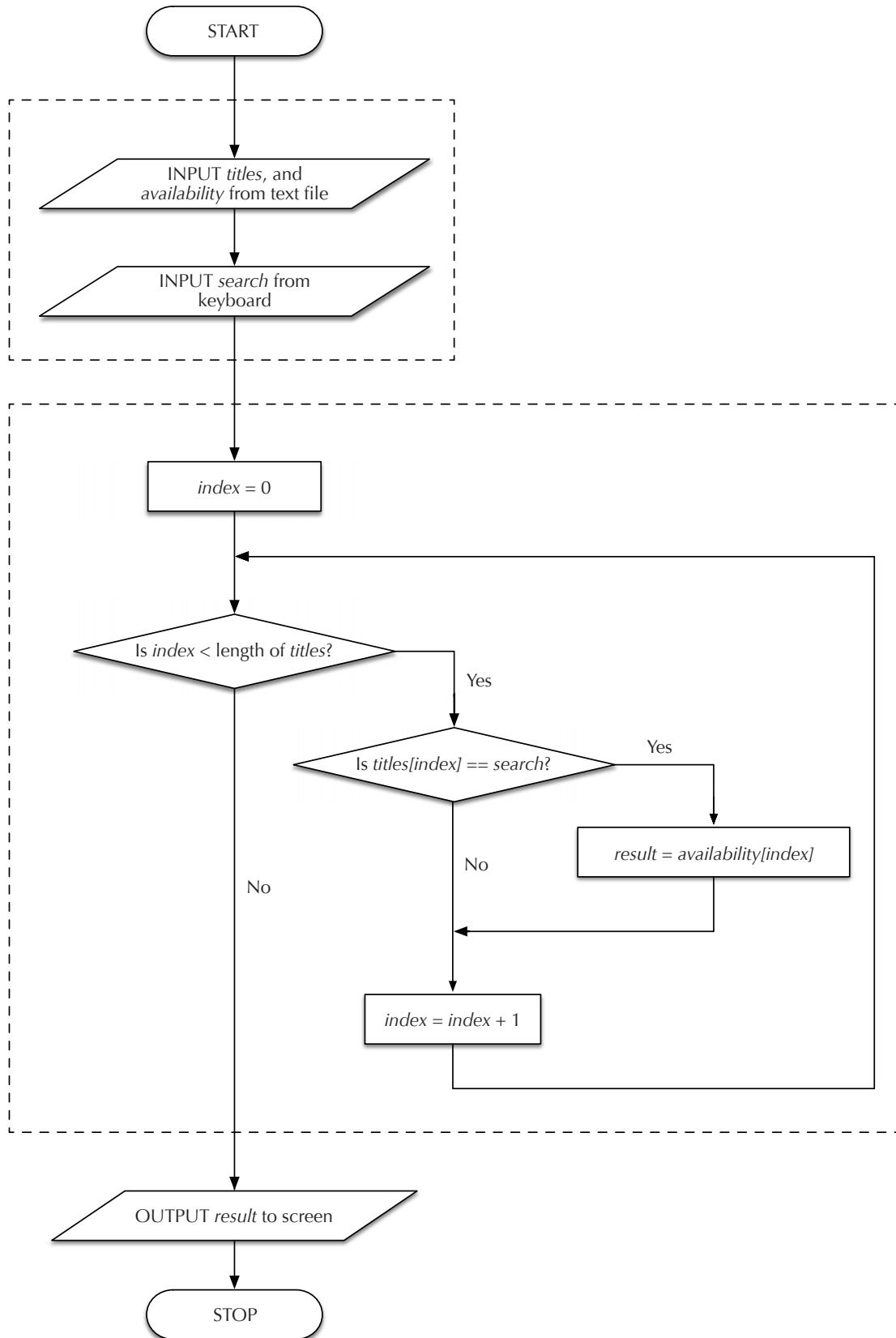
- Manually solving different simplified examples of the problem and generalising the steps needed to produce the required output
- Trying different ways to break down the problem into smaller parts such that the intended output of each part gets closer and closer to what is needed to solve the problem
- Comparing the problem (or its smaller parts) to other problems that have been solved before and identifying which algorithms can be used
- Estimating the amount of effort needed to write the code or the time needed to complete the algorithm before making a definite choice
- Writing possible algorithms using either flowcharts or pseudo-code

Siti notices that she can decompose the problem into three distinct tasks of obtaining input, searching through the list of book titles and, finally, generating output of whether the requested book is available. She illustrates this using a simple flowchart shown in Figure 5.3:



▲ **Figure 5.3** Planned solution to Siti's problem

She notices that the task of searching for the book title is similar to the task of iterating through the items in a list (see Figure 4.118 in Chapter 4). From there, she can adapt the algorithm that she already knows to complete this new task.



▲ **Figure 5.4** Expanding the planned solution to Siti's problem

Siti decides that this planned solution is feasible and moves on to the next stage.

5.1.3 Write Code

In traditional program development, the code for the final program is not actually written until the first two stages of gathering requirements and planning solutions have been completed. This minimises the possibility of producing code based on incomplete requirements or inefficient algorithms that would eventually need to be discarded. The goal of this stage is to write code that performs the algorithm as planned in the previous stage as efficiently as possible.

Did you know?

Some non-traditional approaches to program development do not have five distinct stages. For instance, in “agile” software development, code may be continuously written and refined while the gathering of requirements is still taking place.

One possible way for Siti to write her code would be:

#	Program 5.1 availability_draft.py
1	<code># Input</code>
2	<code>titles = [</code>
3	<code> # The following data is copied from the given text file.</code>
4	<code> "How to Solve a Mystery", "Vacant Memories",</code>
5	<code> "The Cybersnake Chronicles", "Music History",</code>
6	<code> "Like Tears in Rain", "Out of the Abyss"</code>
7	<code>]</code>
8	<code>availability = [</code>
9	<code> # The following data is copied from the given text file.</code>
10	<code> "Not Available", "Available", "Available", "Not Available",</code>
11	<code> "Not Available", "Available"</code>
12	<code>]</code>
13	<code>search = input("Enter title of book to check: ")</code>
14	
15	<code># Process</code>
16	<code>index = 0</code>
17	<code>while index < len(titles):</code>
18	<code> if titles[index] == search:</code>
19	<code> result = availability[index]</code>
20	<code> index = index + 1</code>
21	
22	<code># Output</code>
23	<code>print(result)</code>

▲ Figure 5.5 Draft of code to solve Siti’s problem

5.1.4 Test and Refine Code

After the initial code is written, the resulting program is likely to require further refinement. Some possible reasons for this:

- The programmer may have made mistakes in translating the planned algorithm into code or may have forgotten to consider exceptional cases where the input would need to be treated specially. For example, the programmer may have made a syntax error or forgotten to check for invalid input. These are relatively minor errors that usually would not require a major rewriting of code as simply correcting the syntax error or adding an `if` statement would usually be sufficient to correct the program.
- The solution-planning stage may not have been performed properly, resulting in an unsuitable or incomplete algorithm. Depending on how serious the mismatch is, it may be possible to keep most of the written code and simply make refinements. Otherwise, it may be necessary to discard the written code and redo the evaluation of algorithms.
- The requirement-gathering stage may have been incomplete, resulting in code that does not actually solve the problem. Depending on how serious the mismatch is, it may or may not be possible to reuse most of the written code.

Testing is the process by which these and other imperfections are uncovered. This is done by running the code through one or more test cases to evaluate whether the written code adequately satisfies the gathered requirements and is ready to be used. A **test case** usually consists of a set of inputs and the corresponding set of expected outputs. By feeding these inputs into a program and comparing its actual output with the expected output, we can evaluate whether the program is working as intended.

The effectiveness of testing depends entirely on how comprehensive and well-designed the test cases are. The process of designing good test cases is discussed further in Chapter 6.

Key Terms

Test case

A set of inputs and the corresponding set of expected outputs for a particular program used to verify whether the program works as intended

Testing

Process by which defects in a program can be detected

Did you know?

Testing is so important in making sure code is written correctly that, instead of five distinct stages, some programmers use an alternative approach to program development – test cases are written first and code is written incrementally to pass each test. This alternative process is known as Test-Driven Development (TDD).

Siti has developed a number of test cases to make sure that her code is functioning as intended. She runs the program through the test cases one by one, comparing the program's output with the expected output:

▼ **Table 5.2** Book availability problem – Test Case 1

→ Input	Expected Output →	
<ul style="list-style-type: none"> titles: provided in program availability: provided in program search: "Vacant Memories" 	Available	
Running availability_draft.py →		Result
Enter title of book to check: Vacant Memories		Passed
Available		

▼ **Table 5.3** Book availability problem – Test Case 2

→ Input	Expected Output →	
<ul style="list-style-type: none"> titles: provided in program availability: provided in program search: "Music History" 	Not Available	
Running availability_draft.py →		Result
Enter title of book to check: Music History		Passed
Not Available		

So far, the output of Siti's program has matched the expected output, so the program seems to be functioning as intended. However, she decides to include a test case that searches for the name of a book which is not in the library collection.

▼ **Table 5.4** Book availability problem – Test Case 3

→ Input	Expected Output →
<ul style="list-style-type: none"> titles: provided in program availability: provided in program search: "Resilience" 	Book Not Found
Running <code>availability_draft.py</code> →	Result
<pre>Enter title of book to check: Resilience Traceback (most recent call last): File "C:/Examples/availability_draft.py", line 23, in <module> print(result) NameError: name 'result' is not defined</pre>	Failed

Now, Siti notices an imperfection in her program. When the user enters the name of a book that does not exist in the library, her program generates an error instead of producing the expected output "Book Not Found".

Siti studies her program carefully and realises that she did not consider this special case. Fortunately, she can resolve this by initialising the value of `result` to "Book Not Found". Her refined code is shown in Figure 5.6:

#	Program 5.2 availability_refined.py
	<pre> 1 # Input 2 titles = [3 # The following data is copied from the given text file. 4 "How to Solve a Mystery", "Vacant Memories", 5 "The Cybersnake Chronicles", "Music History", 6 "Like Tears in Rain", "Out of the Abyss" 7] 8 availability = [9 # The following data is copied from the given text file. 10 "Not Available", "Available", "Available", "Not Available", 11 "Not Available", "Available" 12] 13 search = input("Enter title of book to check: ") 14 15 # Process 16 result = "Book Not Found" 17 index = 0 18 while index < len(titles): 19 if titles[index] == search: 20 result = availability[index] 21 index = index + 1 22 23 # Output 24 print(result) </pre>

▲ **Figure 5.6** Refined draft of code to solve Siti's problem

With this modification, her program passes Test Case 3:

▼ **Table 5.5** Book availability problem – Test Case 3 (modified program)

→ Input	Expected Output →	
<ul style="list-style-type: none"> titles: from titles.txt availability: from availability.txt search: "Resilience" 	Book Not Found	
Running availability_refined.py →		Result
Enter title of book to check: Resilience		Passed
Book Not Found		

5.1.5 Deploy Code

With the code tested and refined, this is the stage where the program is actually “rolled out” and used by its intended audience. Some of the tasks that can be performed under this stage:

- Training users to use the program
- Transitioning from an old program or system to the new program
- Evaluating the effectiveness of the program in solving the problem and considering any changes that might increase its usability or effectiveness

Siti deploys her code by installing Python in the library’s computer and copying the program over. She organises a training session to teach the librarians how to use the program and gathers their feedback after the first day of use. She receives the following comments:

- *“Your program makes it much easier to check if a book is available. Thank you!”*
- *“Copying data from text files into the program is very tiresome. Can this be automated?”*
- *“The program is good, but can it be faster?”*
- *“Some book titles take very long to type. However, it’s still faster than checking the bookshelves!”*

It seems that while Siti’s program meets the requirements, it can still be improved in terms of usability and speed. To make some of these improvements, however, she may need to perform more complex processing of numbers and strings. The next section introduces some of the more common built-in functions that Python already provides for such tasks.

5.2 Mathematical Functions

5.2.1 Built-In Mathematical Functions

Computers are essentially powerful calculators, so it is not surprising that Python provides a wide variety of built-in functions to perform various mathematical operations. Table 5.6 summarises the most common mathematical functions in Python.

▼ **Table 5.6** Summary of built-in mathematical functions

Function	Argument	Returns	Examples
<code>abs()</code>	A number (<code>int</code> or <code>float</code>)	An <code>int</code> or a <code>float</code> equal to the absolute value of the argument	<code>>>> abs(2017)</code> 2017 <code>>>> abs(-2017)</code> 2017
<code>max()</code>	Multiple numbers (<code>int</code> or <code>float</code>)	The maximum value out of the given arguments	<code>>>> max(2, -1, 7)</code> 7
<code>max()</code>	A list of numbers (<code>int</code> or <code>float</code>)	The maximum value out of the values in the given list	<code>>>> max([2, -1, 7])</code> 7
<code>min()</code>	Multiple numbers (<code>int</code> or <code>float</code>)	The minimum value out of the given arguments	<code>>>> min(2, -1, 7)</code> -1
<code>min()</code>	A list of numbers (<code>int</code> or <code>float</code>)	The minimum value out of the values in the given list	<code>>>> min([2, -1, 7])</code> -1
<code>pow()</code>	Two numbers (each of which can be <code>int</code> or <code>float</code>)	A <code>float</code> equal to the first argument raised to the power of the second argument	<code>>>> pow(2017, 2)</code> 4068289 <code>>>> pow(2, 2.017)</code> 4.047412804107715
<code>round()</code>	A <code>float</code>	An <code>int</code> equal to the argument rounded to the nearest integer	<code>>>> round(2.017)</code> 2 <code>>>> round(-2.017)</code> -2
<code>round()</code>	A <code>float</code> and an <code>int</code>	A <code>float</code> equal to the first argument rounded to the number of decimal places indicated by the second argument	<code>>>> round(2.017, 2)</code> 2.02 <code>>>> round(-2.017, 2)</code> -2.02

5.2.2 Using the `math` Module

While convenient, Python does not make its entire collection of mathematical functions available by default. This is because not every program needs these additional functions, and making them available all the time has two disadvantages:

1. Loading functions and making them available takes up time and computer memory.
2. It becomes harder to choose names for variables and functions as the names chosen may overwrite or clash with the ones provided by Python.

For these reasons, Python provides some of its built-in functions through **modules**. A module is a collection of variables and functions that need to be loaded before they can be used.

Some of the less commonly used mathematical functions are provided using a module named `math`. To load the `math` module so that its functions are available, we use an `import` statement:

Syntax 5.1 import Statement

```
import name_of_module
```

Key Term

Module

Collection of variables and functions that need to be loaded before they can be used

▲ Figure 5.7 Syntax for `import` statement

After importing the `math` module, the following functions will become available:

▼ Table 5.7 Summary of mathematical functions in the `math` module

Function	Argument	Returns	Examples
<code>math.ceil()</code>	Usually a float	An int equal to the smallest integer greater than or equal to the argument	<pre>>>> math.ceil(2.017) 3 >>> math.ceil(-2.017) -2</pre>
<code>math.floor()</code>	Usually a float	An int equal to the largest integer smaller than or equal to the argument	<pre>>>> math.floor(2.017) 2 >>> math.floor(-2.017) -3</pre>

▼ Table 5.7 Summary of mathematical functions in the `math` module (continued)

Function	Argument	Returns	Examples
<code>math.sqrt()</code>	A number (int or float)	A float equal to the square root of the argument	>>> <code>math.sqrt(2017)</code> 44.91102314577124
<code>math.trunc()</code>	Usually a float	An int equal to the argument with digits after the decimal point cut off (i.e., truncated)	>>> <code>math.trunc(2.017)</code> 2 >>> <code>math.trunc(-2.017)</code> -2

Using any of these functions without importing the `math` module will result in an error. In the example below, Python does not understand what the name “`math`” means as it does not correspond to an existing variable or loaded module:

```
>>> math.ceil(20.17)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    math.ceil(20.17)
NameError: name 'math' is not defined
```

▲ Figure 5.8 Using the `math` module before it is imported will result in an error

To start using a module’s contents, we need to import it once each time we start a Python session in interactive mode or create a Python program:

```
>>> import math
>>> math.ceil(20.17)
21
```

▲ Figure 5.9 The `math` module can be used after it is imported

Did you know?

The period (.) in `math.ceil()` actually has the meaning of searching inside the value on its left and looking for a function with the name given on its right. As you will see later, besides searching inside modules, this operator is also used to access functions that are stored inside `str` values (called methods).

5.2.3 Rounding Functions

A number of functions such as `math.ceil()`, `math.floor()`, `round()` and `math.trunc()` are similar in that they can convert floats to ints. This conversion is performed by rounding floats such that the resulting ints are close in value to the original floats. However, each function uses a different rounding method. Understanding the difference between these functions is important, because using the correct function can greatly reduce the number of steps needed for a solution.

For `round()`, the argument is always rounded to the nearest whole number. However, an argument that is exactly halfway between two integers (such as 0.5) may be rounded either up or down. Figure 5.10 demonstrates how `round()` can appear to give unpredictable results – notice that 0.5 is rounded down while 1.5 is rounded up.

▼ **Table 5.8** Behaviour of the `round()` function for numbers around zero

x	round(x)	Explanation
-1.5	-2	Unpredictable since -1.5 is halfway between -2 and -1
-1.0	-1	Exact
-0.75	-1	Rounded down since -0.75 is nearer to -1 than 0
-0.5	0	Unpredictable since -0.5 is halfway between -1 and 0
-0.25	0	Rounded up since -0.25 is nearer to 0 than -1
0.0	0	Exact
0.25	0	Rounded down since 0.25 is nearer to 0 than 1
0.5	0	Unpredictable since 0.5 is halfway between 0 and 1
0.75	1	Rounded up since 0.75 is nearer to 1 than 0
1.0	1	Exact
1.5	2	Unpredictable since 1.5 is halfway between 1 and 2

```
>>> round(-1.0)
-1
>>> round(0.5)
0
>>> round(0.75)
1
>>> round(1.5)
2
```

▲ **Figure 5.10** Using the `round()` function

Did you know?

The `round()` function only *appears* to give unpredictable results. In reality, `round()` will always return the even integer if its argument is exactly halfway between two integers. For example, `round(0.5)` will return 0 and `round(1.5)` will return 2.

On the other hand, `math.ceil()` always rounds up. Take note that this means a negative non-integer argument will end up becoming *less* negative.

▼ **Table 5.9** Behaviour of the `math.ceil()` function for numbers around zero

x	math.ceil(x)	Explanation
-1.0	-1	Exact
-0.75	0	Rounded up
-0.5	0	Rounded up
-0.25	0	Rounded up
0.0	0	Exact
0.25	1	Rounded up
0.5	1	Rounded up
0.75	1	Rounded up
1.0	1	Exact

```
>>> import math
>>> math.ceil(-1.0)
-1
>>> math.ceil(0.5)
1
>>> math.ceil(0.75)
1
```

▲ **Figure 5.11** Using the `math.ceil()` function

Conversely, `math.floor()` always rounds down. Take note that this means a negative non-integer argument will end up becoming *more* negative.

▼ **Table 5.10** Behaviour of the `math.floor()` function for numbers around zero

x	math.floor(x)	Explanation
-1.0	-1	Exact
-0.75	-1	Rounded down
-0.5	-1	Rounded down
-0.25	-1	Rounded down
0.0	0	Exact
0.25	0	Rounded down
0.5	0	Rounded down
0.75	0	Rounded down
1.0	1	Exact

```
>>> import math
>>> math.floor(-1.0)
-1
>>> math.floor(0.5)
0
>>> math.floor(0.75)
0
```

▲ **Figure 5.12** Using the `math.floor()` function

Finally, `math.trunc()` always rounds towards zero. This is equivalent to dropping the digits of the argument after the decimal point (i.e., truncating the argument).

▼ **Table 5.11** Behaviour of the `math.trunc()` function for numbers around zero

x	math.trunc(x)	Explanation
-1.0	-1	Exact
-0.75	0	Rounded up towards 0
-0.5	0	Rounded up towards 0
-0.25	0	Rounded up towards 0
0.0	0	Exact
0.25	0	Rounded down towards 0
0.5	0	Rounded down towards 0
0.75	0	Rounded down towards 0
1.0	1	Exact

```
>>> import math
>>> math.trunc(-1.0)
-1
>>> math.trunc(-0.75)
0
>>> math.trunc(0.75)
0
>>> math.trunc(1.0)
1
```

▲ **Figure 5.13** Using the `math.trunc()` function

5.2.4 Using the random Module for randint()

Many computer games often require the computer to make a random choice or to generate a random number. To do this, Python provides a module named `random`. For example, to generate a random integer, we will use the `randint()` function. After importing the `random` module, the `random.randint()` function will become available.

▼ **Table 5.12** The `random.randint()` function in the `random` module

Function	Argument	Returns	Examples
<code>random.randint()</code>	Two ints (the first argument should be less than or equal to the second argument)	A random integer between the two arguments (inclusive)	<pre>>>> random.randint(2, 7) 6 >>> random.randint(2, 7) 3 >>> random.randint(2, 7) 2</pre>



Quick Check 5.2

- For each value of `x` below, predict the result of the respective rounding functions. Check your answers by printing the actual return values in Python.

▼ **Table 5.13** Results of various rounding functions

x	-20.17	-19.65	-0.1	0.0	0.1	19.65	20.17
<code>math.ceil(x)</code>							
<code>math.floor(x)</code>							
<code>round(x)</code>							
<code>math.trunc(x)</code>							

- When two six-sided dice are rolled, the resulting sum can range from 2 to 12 (inclusive). Write a program to simulate rolling two six-sided dice 1000 times and output how often the resulting sum is 2, 3, 4 and so on up to 12.

The output of your program should look similar to the following:

```
2: 29
3: 59
4: 110
5: 102
6: 146
7: 160
8: 122
9: 90
10: 85
11: 72
12: 25
```

▲ **Figure 5.14** Sample output for dice simulation

5.3 String Functions

5.3.1 Built-In String Functions

As computers are essentially calculators, everything in a computer must be represented using numbers. This is true even for `strs`. Each symbol or letter in a `str` is generally called a **character** and is represented in the computer by a number.

The numbers that represent common characters such as upper-case and lower-case letters come from a standard called the American Standard Code for Information Interchange (ASCII), which has been in use since the 1960s. Using ASCII, only 128 distinct characters (256 for Extended ASCII) can be represented. Table 5.14 shows some ASCII characters and the numbers used to represent them.

Key Term

Character

A unit of information for `strs`, usually a symbol, letter or digit

▼ **Table 5.14** Some ASCII characters and the numbers used to represent them

Number	Character	Number	Character	Number	Character
32	(space)	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	\$	68	D	100	d

▼ Table 5.14 Some ASCII characters and the numbers used to represent them (continued)

Number	Character	Number	Character	Number	Character
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(72	H	104	h
41)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[123	{
60	<	92	\	124	
61	=	93]	125	}
62	>	94	^	126	~
63	?	95	-		

To convert between characters and the numbers that represent them, Python provides the following two built-in functions:

▼ **Table 5.15** Built-in string functions

Function	Argument	Returns	Examples
<code>chr()</code>	An <code>int</code>	A <code>str</code> of length 1 containing the character that the argument represents	<code>>>> chr(65)</code> <code>'A'</code>
<code>ord()</code>	A <code>str</code> (must be of length 1)	An <code>int</code> that represents the argument	<code>>>> ord('A')</code> <code>65</code>

To represent characters in other written languages such as Tamil and Mandarin, Python uses an expanded standard called Unicode. Like ASCII, the Unicode standard uses numbers to represent characters. The Unicode standard is compatible with ASCII as the first 128 numbers represent the same characters in both ASCII and Unicode. However, unlike ASCII, up to 1,114,112 distinct characters can be represented in Unicode. Some examples using non-English characters are shown in Figure 5.15:

```
>>> ord('ஹ')
2949
>>> chr(2949)
'ஹ'
>>> ord('中')
20013
>>> chr(20013)
'中'
```

▲ **Figure 5.15** Using the `chr()` and the `ord()` functions

5.3.2 Built-In String Methods

Python provides some additional string functions through **methods**. A method is a function that is associated with a value. To use a method, type a value that is associated with the method, followed by a period (.), followed by the name of the method, and followed by zero or more comma-separated arguments enclosed within parentheses (see Figure 5.16). Just like in calling functions, arguments are assigned to new variables so that the original variables will not be affected.

Key Term

Method

Function that is associated with a value

Syntax 5.2 Method Call

```
value.name_of_method()
```

```
value.name_of_method(argument_1)
```

```
value.name_of_method(argument_1, argument_2)
```

▲ **Figure 5.16** Syntax for method calls

Table 5.16 shows some common string functions that can be used as methods.

▼ **Table 5.16** Built-in string methods

Method	Argument	Returns	Examples
<code>str.isalnum()</code>	None	A bool that indicates whether every character in the <code>str</code> is alphanumeric (either a letter or a digit)	<pre>>>> 'CS2017'.isalnum() True >>> 'CS 2017'.isalnum() False >>> '2017'.isalnum() True</pre>
<code>str.isalpha()</code>	None	A bool that indicates whether every character in the <code>str</code> is a letter	<pre>>>> 'Computing'.isalpha() True >>> 'Com pu ting'.isalpha() False</pre>
<code>str.isdigit()</code>	None	A bool that indicates whether every character in the <code>str</code> is a digit	<pre>>>> 'Computing'.isdigit() False >>> '2017'.isdigit() True</pre>
<code>str.isspace()</code>	None	A bool that indicates whether every character in the <code>str</code> is a space	<pre>>>> ' '.isspace() True >>> ''.isspace() False</pre>

▼ Table 5.16 Built-in string methods (continued)

Method	Argument	Returns	Examples
<code>str.islower()</code>	None	A bool that indicates whether every letter in the str is lower case and there is at least one letter	<pre>>>> 'computing'.islower() True >>> 'com pu ting'.islower() True >>> 'Computing'.islower() False >>> '2017'.islower() False</pre>
<code>str.isupper()</code>	None	A bool that indicates whether every letter in the str is upper case and there is at least one letter	<pre>>>> 'COMPUTING'.isupper() True >>> 'COM PU TING'.isupper() True >>> 'Computing'.isupper() False >>> '2017'.isupper() False</pre>
<code>str.lower()</code>	None	A str containing every character of the original str converted to lower case	<pre>>>> 'Computing'.lower() 'computing' >>> 'Com pu ting'.lower() 'com pu ting'</pre>
<code>str.upper()</code>	None	A str containing every character of the original str converted to upper case	<pre>>>> 'Computing'.upper() 'COMPUTING' >>> 'Com pu ting'.upper() 'COM PU TING'</pre>
<code>str.endswith()</code>	A str	A bool that indicates whether the str ends with the given argument	<pre>>>> 'CS2017'.endswith('CS') False >>> 'CS2017'.endswith('17') True</pre>
<code>str.startswith()</code>	A str	A bool that indicates whether the str starts with the given argument	<pre>>>> 'CS2017'.startswith('CS') True >>> 'CS2017'.startswith('17') False</pre>

▼ Table 5.16 Built-in string methods (continued)

Method	Argument	Returns	Examples
<code>str.find()</code>	A <code>str</code>	A int index where the given argument first appears in the <code>str</code> ; -1 if not found	<pre>>>> 'banana'.find('na') 2 >>> 'banana'.find('cs') -1</pre>
<code>str.find()</code>	A <code>str</code> and an <code>int</code>	A int index where the first argument first appears in the <code>str</code> starting from the second argument index; -1 if not found	<pre>>>> 'banana'.find('na', 1) 2 >>> 'banana'.find('na', 3) 4 >>> 'banana'.find('na', 5) -1</pre>
<code>str.find()</code>	A <code>str</code> , an <code>int</code> and an <code>int</code>	A int index where the first argument first appears in the <code>str</code> starting from the second argument index up to but not including the third argument index; -1 if not found	<pre>>>> 'SCSC'.find('CS', 0, 3) 1 >>> 'SCSC'.find('CS', 1, 3) 1 >>> 'SCSC'.find('CS', 0, 2) -1</pre>
<code>str.split()</code>	None	A list of <code>strs</code> that are separated by one or more spaces in the original <code>str</code>	<pre>>>> 'Hi, World!'.split() ['Hi,', 'World!'] >>> 'C S'.split() ['C', 'S']</pre>
<code>str.split()</code>	A <code>str</code>	A list of <code>strs</code> that are separated by the argument in the original <code>str</code>	<pre>>>> 'Hi, World'.split(',') ['Hi', ' World!'] >>> 'C S'.split(' ') ['C', '', 'S']</pre>
<code>str.format()</code>	Any number of values	A new <code>str</code> with each field (such as '{ }' or '{0}') in the <code>str</code> replaced by a corresponding argument	<pre>>>> '{} 2017'.format('CS') 'CS 2017' >>> '{0} {1}'.format(6, 5) '6 5' >>> '{1} {0}'.format(6, 5) '5 6' (see section 5.3.3 on formatting strings)</pre>

5.3.3 Formatting Strings

In Python, we often use type casting and the concatenation operator to produce `strs` or output that follow a desired format. For instance, we can report the values assigned to variables named `student` and `year` together using short descriptive text like this:

```
>>> student = "Bala"
>>> year = 2017
>>> print("student is " + student + ", year is " + str(year) + ".")
student is Bala, year is 2017.
```

▲ Figure 5.17 Using type casting and the concatenation operator to format `strs`

However, Python can also produce the same output using the `str.format()` method. Using the `str.format()` avoids concatenating `strs` manually and results in code that is easier to read.

```
>>> student = "Bala"
>>> year = 2017
>>> print("student is {}, year is {}".format(student, year))
student is Bala, year is 2017.
```

▲ Figure 5.18 Using the `str.format()` method to format `strs`

Notice that the output of `str.format()` replaces each set of curly braces `{}` in the original `str` with one of the provided arguments. Each set of curly braces in the original `str` is called a **replacement field**, or just field for short. By default, fields are replaced with arguments in order from left to right. For instance, in Figure 5.18, the first `{}` is replaced with the first argument `student` while the second `{}` is replaced with the second argument `year`.

To change the order of arguments used to replace each field, we can specify an index for each field as follows:

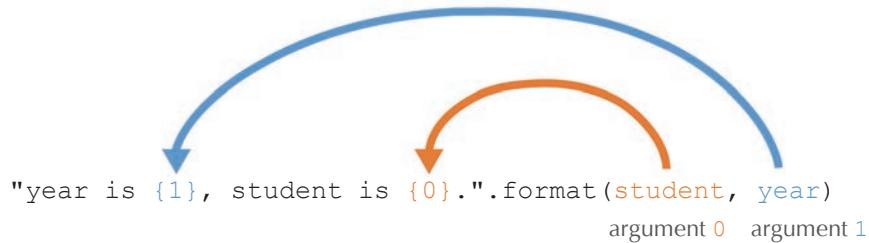
```
>>> student = "Bala"
>>> year = 2017
>>> print("year is {1}, student is {0}.".format(student, year))
year is 2017, student is Bala.
```

Key Term

Replacement field
Placeholder denoted by a set of curly braces that will be replaced with an argument using the `str.format()` method

▲ Figure 5.19 Specifying an index for each field

Just like indices for `lists` and `strs`, indices for arguments start from 0 and not 1. Hence, the field of `{0}` specifies an index of 0 and is replaced by the *first* argument `student`. The field of `{1}` specifies an index of 1 and is replaced by the *second* argument `year`.



▲ **Figure 5.20** Arguments are matched to fields based on index

Note that multiple fields may specify the same index and that it is not compulsory for all arguments to be included in the output.

```
>>> student = "Bala"
>>> year = 2017
>>> print("{0}, {0}, {0}".format(student, year))
Bala, Bala, Bala
>>> print("{1}, {0}, {1}".format(student, year))
2017, Bala, 2017
```

▲ **Figure 5.21** Examples of using fields with specified indices

Also note that fields with specified indices and fields without specified indices cannot be used together.

```
>>> print("{}, {}, {}".format(student, year))
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print("{}", {}, {}).format(student, year))
ValueError: cannot switch from automatic field numbering to manual field
specification
```

▲ **Figure 5.22** Fields with and without specified indices cannot be used together

To “escape” the curly braces so that they are not treated as a field, use double braces so that it looks like “{{ “ or ”}}”. The `str.format()` method will convert the double curly braces back into single curly braces and will not treat the curly braces as a field.

```
>>> student = "Bala"
>>> year = 2017
>>> print("student is {{}}, ignore {{}}, year is {{}}".format(student, year))
student is Bala, ignore {}, year is 2017
```

▲ **Figure 5.23** Using double curly braces to prevent them from being treated as a field

In summary, the syntax for a replacement field is as follows:

Syntax 5.3 Replacement field

{ }

{index}

▲ **Figure 5.24** Syntax for replacement fields

Did you know?

The `str.format()` method and replacement fields can also perform more complex tasks such as lengthening `strs` to a specified width and restricting the number of decimal places used when converting a `float` to a `str`. Use of the `str.format()` method to perform such tasks will not be covered in this textbook.



Quick Check 5.3

1. Fill in the blank in the program below so that it correctly outputs whether the first character of `name` is an upper-case letter.

```
name = input("Enter name: ")  
print()
```

▲ **Figure 5.25** Program template

2. Complete the program below by filling in the blank with one or more lines of code so that it correctly outputs whether `title` contains upper-case letters and spaces only (i.e., no lower-case letters, symbols or digits).

```
title = input("Enter title: ")  

```

▲ **Figure 5.26** Program template

5.4 Common Problems and Solutions

It is useful to have a catalogue or “toolbox” of common solutions which can be adapted to solve problems that cannot be solved easily by a built-in function. Through pattern recognition and generalisation, we can identify a previously used solution that can be applied to a new or seemingly unfamiliar problem. In this section, we will look at some common problems and their solutions.

5.4.1 Finding the Minimum Value in a List

Sometimes we need to find the minimum value in a list of values. Suppose we have a list of heights of a class of students. Finding the height of the shortest student is thus equivalent to finding the minimum value in this list.

A common extension to this problem is to find out where this minimum value is located in the list. Suppose the list of heights is for a class of students and that the heights are arranged in order of registration number. Finding the registration number of the shortest student in the class is thus equivalent to determining the index of the minimum value in the list.

Depending on the situation, we may also need to check if a minimum value even exists. Suppose the list of heights is for students who sign up for a basketball game. If it turns out that no students sign up for the game, the list of heights would be empty and no minimum value would exist. The solution would need to be able to handle this case as well.

We can generalise these problems as shown in Table 5.17:

▼ **Table 5.17** Input and output requirements for the minimum value problem

Input	Output
<ul style="list-style-type: none"> • <code>values</code>: list to find smallest value from (already provided) 	<ul style="list-style-type: none"> • <code>minimum_value</code>: smallest value in <code>values</code>; <code>None</code> if <code>values</code> is empty • <code>minimum_index</code>: index of <code>minimum_value</code> in <code>values</code>; <code>None</code> if <code>values</code> is empty

A solution to this problem is as follows:

#	Program 5.3 minimum.py
1	<code># Input</code>
2	<code>values = [16, 64, 4, 128, 8, 2, 1, 32]</code>
3	
4	<code># Process</code>
5	<code>if len(values) > 0:</code>
6	<code> minimum_value = values[0]</code>
7	<code> minimum_index = 0</code>
8	<code> for index in range(len(values)):</code>
9	<code> if values[index] < minimum_value:</code>
10	<code> minimum_value = values[index]</code>
11	<code> minimum_index = index</code>
12	<code>else:</code>
13	<code> minimum_value = None</code>
14	<code> minimum_index = None</code>
15	
16	<code># Output</code>
17	<code>print(minimum_value)</code>
18	<code>print(minimum_index)</code>

▲ **Figure 5.27** Finding the minimum value in a list and its index

The solution starts by checking whether `values` has at least one value (line 5). If not, we conclude that `values` is empty and simply set the outputs to `None` (lines 13 and 14).

Otherwise, we decompose the problem using an incremental approach. We start by considering this sub-problem: what is the minimum value if `values` contains one value only? The solution to this sub-problem is straightforward: `minimum_value` is equal to the first value in `values` and `minimum_index` is equal to 0 (lines 6 and 7).

We then gradually extend this sub-problem by considering the rest of the values in `values`, one by one (line 8). The solution is extended accordingly by checking if the new value being considered is less than `minimum_value` (line 9). If so, we update `minimum_value` and `minimum_index` with this new value (lines 10 and 11). This ensures that `minimum_value` and `minimum_index` are always keeping track of the minimum value encountered so far.

When the loop ends, we would have considered all the values in `values` and thus can conclude that the resulting `minimum_value` and `minimum_index` are correct for the entire list.

On the other hand, if it is not necessary to find the index of the minimum value, we can simplify the program by removing the variables `index` and `minimum_index` completely.

#	Program 5.4 <code>minimum_without_index.py</code>
1	<pre> # Input values = [16, 64, 4, 128, 8, 2, 1, 32] # Process if len(values) > 0: minimum_value = values[0] for value in values: if value < minimum_value: minimum_value = value else: minimum_value = None # Output print(minimum_value) </pre>

▲ **Figure 5.28** Finding the minimum value of a list (no index)

Python also provides a built-in `min()` function that can further simplify the solution if finding the index is not necessary:

#	Program 5.5 <code>minimum_using_function.py</code>
1	<pre> # Input values = [16, 64, 4, 128, 8, 2, 1, 32] # Process if len(values) > 0: minimum_value = min(values) else: minimum_value = None # Output print(minimum_value) </pre>

▲ **Figure 5.29** Finding the minimum value of a list using the built-in `min()` function

Even though this solution is much simpler than the previous ones, it is still useful to understand the algorithm in Figure 5.27 as the `min()` function does not provide the index of the minimum value and cannot be customised for more complex situations, such as finding the second or third smallest value in a list.

5.4.2 Finding the Maximum Value in a List

Besides finding the minimum value, we can also find the maximum value in a list. The corresponding general problem is as follows:

▼ **Table 5.18** Input and output requirements for the maximum value problem

Input	Output
<ul style="list-style-type: none"> • <code>values</code>: list to find biggest value from (already provided) 	<ul style="list-style-type: none"> • <code>maximum_value</code>: biggest value in <code>values</code>; <code>None</code> if <code>values</code> is empty • <code>maximum_index</code>: index of <code>maximum_value</code> in <code>values</code>; <code>None</code> if <code>values</code> is empty

By using pattern recognition, we can easily modify the program in Figure 5.27 so that it finds the maximum value in a list instead:

#	Program 5.6 maximum.py
<pre> 1 # Input 2 values = [16, 64, 4, 128, 8, 2, 1, 32] 3 4 # Process 5 if len(values) > 0: 6 maximum_value = values[0] 7 maximum_index = 0 8 for index in range(len(values)): 9 if values[index] > maximum_value: 10 maximum_value = values[index] 11 maximum_index = index 12 else: 13 maximum_value = None 14 maximum_index = None 15 16 # Output 17 print(maximum_value) 18 print(maximum_index) </pre>	

▲ **Figure 5.30** Finding the maximum value in a list and its index

The resulting program is largely the same, except that the greater-than operator (`>`) is used on line 9 instead of the less-than operator (`<`), and the output variables are renamed to `maximum_value` and `maximum_index`.

Just as with the previous problem, if it is not necessary to find the index of the maximum value, we can simplify the program by removing the variables `index` and `maximum_index` completely.

#	Program 5.7 maximum_without_index.py
1	<code># Input</code>
2	<code>values = [16, 64, 4, 128, 8, 2, 1, 32]</code>
3	
4	<code># Process</code>
5	<code>if len(values) > 0:</code>
6	<code>maximum_value = values[0]</code>
7	<code>for value in values:</code>
8	<code>if value > maximum_value:</code>
9	<code>maximum_value = value</code>
10	<code>else:</code>
11	<code>maximum_value = None</code>
12	
13	<code># Output</code>
14	<code>print(maximum_value)</code>

▲ **Figure 5.31** Finding the maximum value in a list (no index)

Similarly, Python provides a built-in `max()` function that can further simplify the solution if finding the index is not necessary:

#	Program 5.8 maximum_using_function.py
1	<code># Input</code>
2	<code>values = [16, 64, 4, 128, 8, 2, 1, 32]</code>
3	
4	<code># Process</code>
5	<code>if len(values) > 0:</code>
6	<code>maximum_value = max(values)</code>
7	<code>else:</code>
8	<code>maximum_value = None</code>
9	
10	<code># Output</code>
11	<code>print(maximum_value)</code>

▲ **Figure 5.32** Finding the maximum value in a list using the built-in `max()` function

5.4.3 Finding the Average Value in a List

Many problems require us to find the average value in a list. Suppose we have a list of scores for a class test. Finding the average score for the class is thus equivalent to finding the average value in this list.

Depending on the situation, we may also need to check if an average value even exists. Suppose the list of scores is for an optional test. If it turns out that no students choose to take the test, the list of scores would be empty and no average value would exist.

The general problem to the situation above can be described as follows:

▼ **Table 5.19** Input and output requirements for the average problem

Input	Output
<ul style="list-style-type: none"> values: list to calculate the average value from (already provided) 	<ul style="list-style-type: none"> average: average value in values; None if values is empty

A solution to this problem is as follows:

#	Program 5.9 average.py
<pre> 1 # Input 2 values = [16, 64, 4, 128, 8, 2, 1, 32] 3 4 # Process 5 if len(values) > 0: 6 sum_values = 0 7 for value in values: 8 sum_values += value 9 average = sum_values / len(values) 10 else: 11 average = None 12 13 # Output 14 print(average) </pre>	

▲ **Figure 5.33** Finding the average value in a list

The solution starts by checking whether `values` has at least one value (line 5). If not, we conclude that `values` is empty and simply set the output to `None` (line 14).

Otherwise, we calculate the sum of the values in `values` using a `for-in` loop (lines 7 to 8) and divide this sum by the length of `values` to obtain the required average (line 15).

Python also provides a built-in `sum()` function that simplifies the task of calculating the sum of the values in `values` without the need for a loop:

#	Program 5.10 average_using_sum.py
1	<code># Input</code>
2	<code>values = [16, 64, 4, 128, 8, 2, 1, 32]</code>
3	
4	<code># Process</code>
5	<code>if len(values) > 0:</code>
6	<code>sum_values = sum(values)</code>
7	<code>average = sum_values / len(values)</code>
8	<code>else:</code>
9	<code>average = None</code>
10	
11	<code># Output</code>
12	<code>print(average)</code>

▲ **Figure 5.34** Finding the average value in a list using the built-in `sum()` function

While both the programs in Figures 5.33 and 5.34 are correct, the program in Figure 5.34 is preferred as it is simpler.

5.4.4 Searching for a Value in a List

As we have learnt in section 4.6.2.3, the `in` operator can tell us whether a value is in a list, but it does not tell us where the value is located. Hence, the `in` operator is insufficient for problems that require us to search for a value in a list and retrieve its index. Suppose we have a list of students' names such that the index of each value is the corresponding student's registration number. Finding the registration number of a particular student is thus equivalent to searching for the student's name in this list and retrieving its index.

Depending on the situation, the search may not be able to find the value. Suppose the name that we input for searching is misspelled. The program would not be able to find this name in the list and must be able to handle this situation as well.

We can generalise this problem as follows:

▼ **Table 5.20** Input and output requirements for the searching problem

Input	Output
<ul style="list-style-type: none"> • <code>values</code>: list to search from (already provided) • <code>search</code>: value to search for (already provided) 	<ul style="list-style-type: none"> • <code>search_index</code>: index of <code>search</code> in <code>values</code>; <code>None</code> if not found or <code>values</code> is empty

A solution to this problem is as follows:

#	Program 5.11 search.py
<pre> 1 # Input 2 values = ["Alex", "Bala", "Siti", "Zee"] 3 search = input("Enter name: ") 4 5 # Process 6 search_index = None 7 for index in range(len(values)): 8 if values[index] == search: 9 search_index = index 10 break 11 12 # Output 13 print(search_index) </pre>	

▲ **Figure 5.35** Searching for an item in a list

The solution first initialises the output `search_index` to `None` (line 6). This ensures that if no match is found or if `values` is empty, `search_index` correctly remains as `None`.

Using an incremental approach, we then check each value in the list in order (line 7) to see if a match can be found (line 8). Once a match is found, we set `search_index` as the index of the matching value and stop the search immediately using a `break` command (lines 9 to 10).

5.4.5 Finding Check Digits

A **check digit** is usually an additional digit or letter added to the end of a sequence of digits that is intended to be read by or entered into a computer manually. The check digit is mathematically related to the original sequence of digits so that simple input errors, such as accidentally swapping two digits or wrongly entering a digit, would break this mathematical relationship and hence be detected. If the check digit is a letter, it would usually need to be converted to a number so that it can be used in the algorithm to check the sequence.

Key Term

Check digit

Digit or letter that is added to a sequence of digits to detect errors in manual input

After the digits have been manually entered, the computer will calculate whether the expected mathematical relationship is true. If the relationship is still true, it is likely that the numbers were entered correctly. Otherwise, the computer can ask for the numbers to be re-entered and checked again.

For instance, most products sold in shops are labelled with a barcode that also contains a number. This number is also known as a product code, which is used to identify the product being purchased. A product code is obtained when the barcode is decoded by a barcode reader or when the number is entered manually at the cashier. The last digit of this product code is actually a check digit that helps to prevent errors in the decoding or entry process.

One popular standard for check digits that are used in product barcodes is the Universal Product Code (UPC-A) standard. Figure 5.36 shows an example of a 12-digit UPC-A product code. The 12th digit is the check digit.



▲ Figure 5.36 Example of a UPC-A barcode

To calculate the check digit, the first 11 digits need to be processed using the following algorithm:

1. Add the digits in the odd-numbered positions (i.e., 1st, 3rd ... 11th).
2. Multiply the result by three.
3. Add the digits in the even-numbered positions (i.e., 2nd, 4th ... 10th) to the result.
4. If the last digit of the result is 0, the check digit should be 0. Otherwise, subtract the last digit of the result from 10. The check digit should be the same as the resulting answer.

The program in Figure 5.37 demonstrates how to calculate the check digit from the first 11 digits of a UPC-A code. There are also many other possible check digit algorithms.

#	Program 5.12 upc.py
1	<code># This program calculates the check digit of a Universal Product Code</code>
2	<code># (UPC-A) based on the first 11 digits of the code.</code>
3	<code></code>
4	<code># Input</code>
5	<code>upc = input("Enter first 11 digits of UPC-A: ")</code>
6	<code></code>
7	<code># Process</code>
8	<code>odd_sum = 0</code>
9	<code>even_sum = 0</code>
10	<code>for index in range(11):</code>
11	<code> digit = int(upc[index])</code>
12	<code> if (index + 1) % 2 == 0:</code>
13	<code> even_sum += digit</code>
14	<code> else:</code>
15	<code> odd_sum += digit</code>
16	<code>check_digit = (odd_sum * 3 + even_sum) % 10</code>
17	<code>if check_digit != 0:</code>
18	<code> check_digit = 10 - check_digit</code>
19	<code></code>
20	<code># Output</code>
21	<code>print(check_digit)</code>

▲ Figure 5.37 Finding the check digit of a Universal Product Code

5.4.6 Extracting Required Characters from a String

Many string problems require extracting a subset of the characters in a string. If the required subset is based solely on the characters' positions in the string, the slice and slice with a step operators will usually be sufficient. For instance, `my_string[4:6]` easily extracts the 5th and 6th characters of the string named `my_string`. (Recall the use of the slice operator in section 4.6.2.3.)

However, more complex string problems will usually need string functions or methods such as those found in section 5.3 to determine whether each character should be extracted. For instance, extracting all the letters and numbers from a string is equivalent to extracting all the characters for which the `str.isalnum()` method returns `True`.

We can specify such problems as follows:

▼ **Table 5.21** Input and output requirements for the extracting problem

Input	Output
<ul style="list-style-type: none"> • original: str to extract characters from (already provided) • test(): function or method to determine whether character should be extracted (chosen or written by programmer) 	<ul style="list-style-type: none"> • extract: the extracted characters

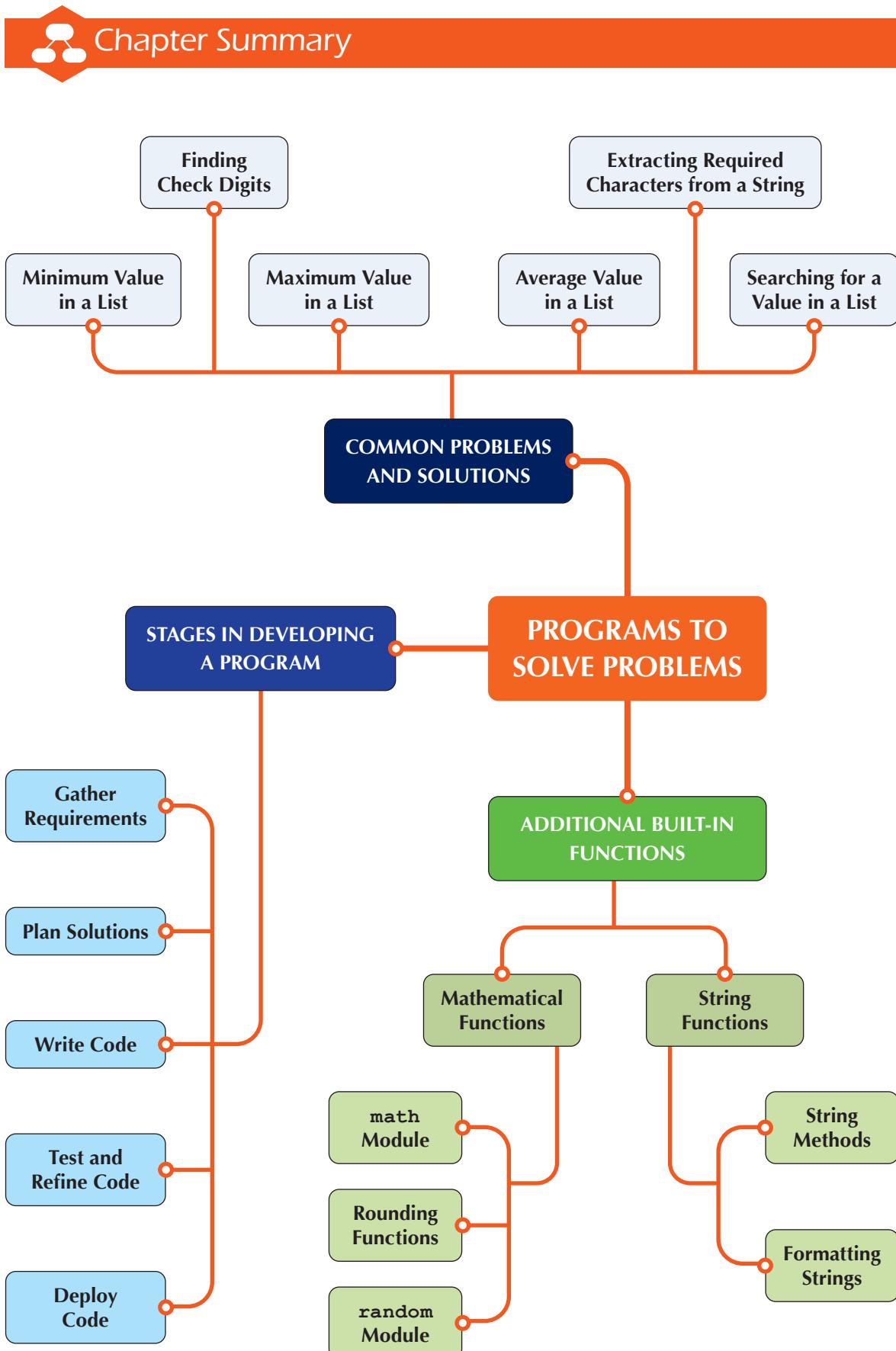
A solution for this problem, using the `str.isalnum()` method as an example input for `test()`, is as follows:

#	Program 5.13 extract.py
<pre> 1 # Input 2 original = input("Enter a string of characters: ") 3 4 # Process 5 extract = "" 6 for character in original: 7 # In this example, str.isalnum() is used for test() 8 if character.isalnum(): 9 extract += character 10 11 # Output 12 print(extract) </pre>	

▲ **Figure 5.38** Extracting required characters from a string

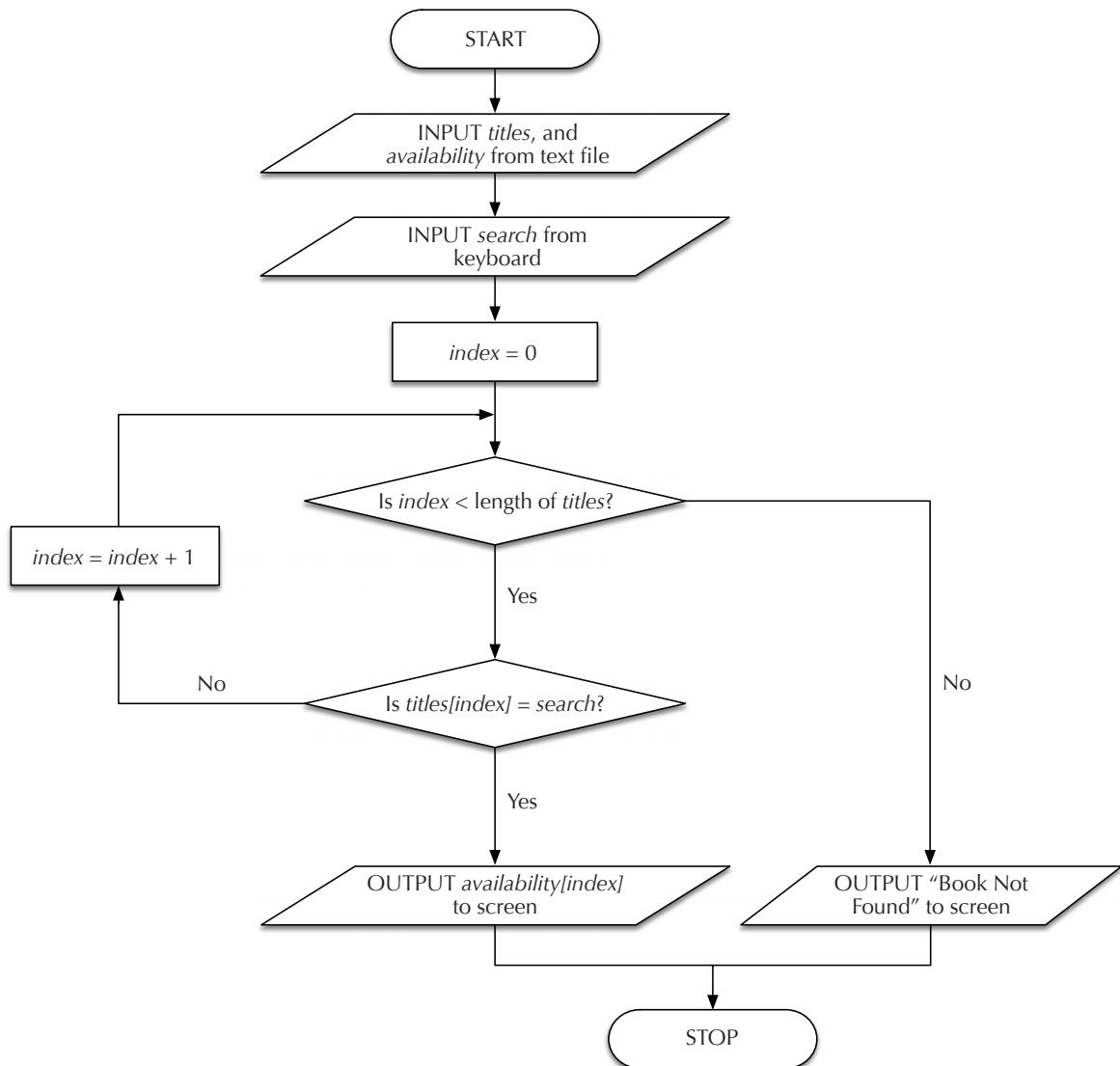
The solution starts by initialising the output variable `extract` to an empty string (line 5). Using an incremental approach, we then consider the characters of the original string, one by one and in order (line 6), to see if each character meets our requirements for extraction (line 8). Each character that meets our requirements is appended to our output (line 9).

When the loop ends, we would have considered all the characters in the original string and thus can conclude that `extract` has all the characters that meet our requirements.



! Review Questions

1. Siti has developed a program for checking the availability of library books. Based on the feedback she has received, Siti revisits the solution-planning stage and comes up with a new algorithm that she thinks would be faster, shown in Figure 5.39 below.



▲ **Figure 5.39** New solution for Siti’s problem

- Explain why this new solution may be faster than the solution in `availability_refined.py` shown in Figure 5.6.
- Write a program that corresponds to Siti’s new algorithm. You may wish to use `availability_refined.py` as a starting point.

2. Write a program that lets the user input a list of ints by asking for each item of the list in order. Receiving an empty input from the user would indicate the end of the list. The program would then output the smallest int in the list, followed by a space, followed by the second smallest int in the list. However, if the list has fewer than two items, it should output the word “None” instead.

Assume that the input data will always be valid and none of the ints are repeated.

Two test cases for how the program should behave with different inputs are as follows:

```
Enter item, blank to end: 19
Enter item, blank to end: 65
Enter item, blank to end: -20
Enter item, blank to end: 17
Enter item, blank to end:
-20 17
```

▲ Figure 5.40 Sample run – Test Case 1

```
Enter item, blank to end: -19
Enter item, blank to end:
None
```

▲ Figure 5.41 Sample run – Test Case 2

3. Write a program that lets the user input a list of floats by first asking for the number of items in the list, then asking for each item of the list in order. The program would then output the average of only the positive floats in the list, rounded to two decimal places. However, if the list has no positive floats, it should output the word “None” instead.

Assume that the input data will always be valid.

Two test cases for how the program should behave with different inputs are as follows:

```
Enter length: 4
Enter item: 20.17
Enter item: -19.65
Enter item: 201.8
Enter item: 2.020
74.66
```

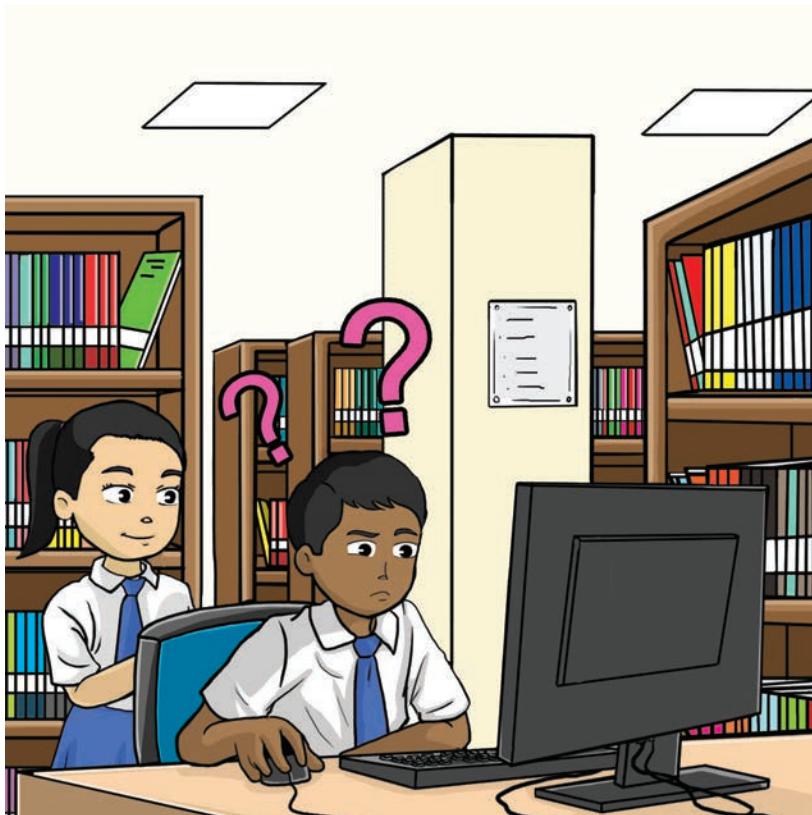
▲ Figure 5.42 Sample run – Test Case 1

```
Enter length: 0
None
```

▲ Figure 5.43 Sample run – Test Case 2

How Do I Ensure That a Program Works as Intended?

Bala has written a program to automate the process of shortening common phrases into acronyms. However, it does not seem to be working properly. He decides to ask Siti for help. How should Siti go about helping Bala to fix his program and ensure that it always functions as expected?



In the previous chapter, we learnt that testing and refining code is one of the five stages in developing a program. In this chapter, we will learn about the process of testing and refining code to identify and fix errors in greater detail, whether these errors are caused by invalid input from the user or mistakes made by the programmer.



By the end of this chapter, you should be able to:

- Identify and justify the use of data validation techniques.
- Validate input data for acceptance by performing:
 - Length check
 - Range check
 - Presence check
 - Format check
- Design appropriate test cases to cover normal, error and boundary conditions, and specify what is being tested for each test case.
- Understand and describe types of program errors and explain why they occur:
 - Syntax errors
 - Run-time errors
 - Logic errors
- Understand and apply debugging techniques to isolate/identify and debug program errors:
 - Using intermittent print statements
 - Walking through a program
 - Testing program in small chunks or by parts

6.1 Validating Input Data

6.1.1 Why Validation is Needed

For a problem to be solved, its inputs, outputs and processes need to be defined clearly. While the programmer often has control over the processes used and outputs produced by the code, the supplied inputs can come from many possible sources, and the programmer often has no control over what is supplied. This means that the supplied inputs may not actually meet the requirements for valid or acceptable input data as defined by the problem.

For example, consider the program we used in section 5.4.3 to find the average value in a list of numbers.

▼ **Table 6.1** Input and output requirements for the problem of calculating the average value in a list of numbers

Input	Output
<ul style="list-style-type: none"> • values: list to calculate average from (already provided) 	<ul style="list-style-type: none"> • average: average value in values; None if values is empty

#	Program 6.1 average_using_sum.py
1	<code># Input</code>
2	<code>values = [16, 64, 4, 128, 8, 2, 1, 32]</code>
3	
4	<code># Process</code>
5	<code>if len(values) > 0:</code>
6	<code> sum_values = sum(values)</code>
7	<code> average = sum_values / len(values)</code>
8	<code>else:</code>
9	<code> average = None</code>
10	
11	<code># Output</code>
12	<code>print(average)</code>

▲ **Figure 6.1** Finding the average value in a list of numbers

However, what would happen if `values` was initialised as a list of `strs` instead?

#	Program 6.2 average_with_invalid_input.py
1	<code># Input</code>
2	<code>values = ["16", "64", "4", "128", "8", "2", "1", "32"]</code>
3	
4	<code># Process</code>
5	<code>if len(values) > 0:</code>
6	<code> sum_values = sum(values)</code>
7	<code> average = sum_values / len(values)</code>
8	<code>else:</code>
9	<code> average = None</code>
10	
11	<code># Output</code>
12	<code>print(average)</code>

▲ **Figure 6.2** Attempting to find the average value in a list of `strs`

```
===== RESTART: C:/Examples/average_with_invalid_input.py =====
Traceback (most recent call last):
  File "average_with_invalid_input.py", line 6, in <module>
    sum_values = sum(values)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

▲ **Figure 6.3** Error caused by invalid input data (highlighted in the program)

Here, Python outputs an error message that the addition operator (+) used by the `sum()` function is unable to work with values that are of different data types (i.e., `int` and `str`). This is because `values` should contain only numeric values. While this error is due to invalid input provided by the user, we can prevent the program from behaving in an unexpected manner by performing **data validation** before the data is processed. This ensures that the input data is sensible, complete and within acceptable boundaries.

Key Term

Data validation

Process of ensuring that the input data supplied to a system satisfies a set of rules such that it is sensible, complete and within acceptable boundaries.

6.1.2 Common Data Validation Techniques

When performing data validation on data that is entered by the user, we should use a loop and keep asking for the data to be re-entered until valid data is entered or the maximum number of tries is exceeded.

The following sections will cover the four common checks that are performed in data validation:

1. Length check
2. Range check
3. Presence check
4. Format check

6.1.2.1 Length Check

Recall that the number of items in a `list` or number of characters in a `str` is called its length. The length of the inputs for a problem often needs to meet certain requirements in order to be valid. This is known as a **length check**.

For example, consider the input to a word-guessing game where all guesses must have a length of 3.

Key Term

Length check

Process of ensuring that input data is not too short or too long

▼ Table 6.2 Example of problem requiring a length check

Input	Output
<ul style="list-style-type: none"> • <code>guess: str</code> with a length of 3 (required and provided as many times as needed until program ends) 	<ul style="list-style-type: none"> • "Correct" or "Wrong", depending on whether the <code>guess</code> matches a secret <code>str</code> of length 3 (program ends after correct guess)

For the input to be valid, every guess must have a length of 3 exactly. The program below performs this length check using the `len()` function on line 16.

#	Program 6.3 game_with_length_check.py
1	<code>import random # For random.randint()</code>
2	
3	<code># List of possible secret words</code>
4	<code>WORDS = ["add", "bug", "cat", "dog", "eye", "for", "sky", "zip"]</code>
5	
6	<code># Choose a random secret word</code>
7	<code>random_index = random.randint(0, len(WORDS) - 1)</code>
8	<code>secret_word = WORDS[random_index]</code>
9	
10	<code># Outer loop to keep repeating until secret word is guessed</code>
11	<code>while True:</code>
12	
13	<code> # Input and input validation</code>
14	<code> while True:</code>
15	<code> guess = input("Enter guess of length 3: ")</code>
16	<code> if len(guess) == 3:</code>
17	<code> break</code>
18	<code> print("Data validation failed!")</code>
19	<code> print("Guess must have length of 3")</code>
20	
21	<code> # Process and output</code>
22	<code> if guess == secret_word:</code>
23	<code> print("Correct")</code>
24	<code> break # Exit outer main loop here</code>
25	<code> print("Wrong")</code>

▲ **Figure 6.4** Example of program that performs a length check

Note that this program uses two infinite `while` loops. To make the player keep guessing until a correct guess is made, the program enters an outer `while` loop on line 11 that keeps repeating until the correct word is guessed and a `break` statement is used to exit the outer loop on line 24. To make the player re-enter `guess` until it passes input validation, the program enters an inner `while` loop on line 14 that keeps repeating until the length check on line 16 is passed and a `break` statement is used to exit the loop on line 17.

In general, length checks make use of the `len()` function and are needed to ensure that `list` or `str` input data is not too short or too long.

6.1.2.2 Range Check

Another common requirement is to limit the input to a particular range of values. This is known as a **range check**. For example, consider the problem of converting rounded percentage scores to O-Level grades based on Table 6.3.

Key Term

Range check

Process of ensuring that input data is within the required range of values

▼ **Table 6.3** Converting rounded percentage scores to grade points

Rounded percentage score	Grade
75–100	A1
70–74	A2
65–69	B3
60–64	B4
55–59	C5
50–54	C6
45–49	D7
40–44	E8
Less than 40	F9

The inputs and outputs for this problem are specified below:

▼ **Table 6.4** Example of problem requiring a range check

Input	Output
<ul style="list-style-type: none"> score: rounded percentage score, should be a whole number between 0 and 100 inclusive 	<ul style="list-style-type: none"> Correct O-Level grade for given rounded percentage score

In this problem, the input is a rounded percentage and must be a whole number between 0 and 100. The following program shows how this range check might be performed:

#	Program 6.4 grade_with_range_check.py
	<pre> 1 # Input and input validation 2 score = None 3 while score == None: 4 score = int(input("Enter score: ")) 5 if score < 0 or score > 100: 6 print("Data validation failed!") 7 print("Your score should be between 0 to 100 inclusive") 8 score = None 9 10 # Process 11 if score >= 75: 12 grade = "A1" 13 elif score >= 70: 14 grade = "A2" 15 elif score >= 65: 16 grade = "B3" 17 elif score >= 60: 18 grade = "B4" 19 elif score >= 55: 20 grade = "C5" 21 elif score >= 50: 22 grade = "C6" 23 elif score >= 45: 24 grade = "D7" 25 elif score >= 40: 26 grade = "E8" 27 else: 28 grade = "F9" 29 30 # Output 31 print(grade) </pre>

▲ **Figure 6.5** Example of program that performs a range check

Like the previous example, this program keeps trying if data validation fails by asking the user to re-enter the score. In general, range checks make use of these operators:

- less than (<)
- less than or equal to (<=)
- greater than (>)
- greater than or equal to (>=)

They are needed to ensure that `int` or `float` input data is within the required range of values.

6.1.2.3 Presence Check

For some problems, portions of the input are required (or mandatory) while other parts may be optional. If it is possible to leave out any optional inputs, the program should perform a **presence check** to ensure that all the required inputs are provided.

For instance, let us look again at the word-guessing game where all guesses must have a length of 3.

▼ **Table 6.5** Example of problem requiring a presence check

Input	Output
<ul style="list-style-type: none"> • <code>guess: str</code> with a length of 3 (required and provided as many times as needed until program ends) 	<ul style="list-style-type: none"> • "Correct" or "Wrong", depending on whether the <code>guess</code> matches a secret <code>str</code> of length 3 (program ends after correct guess)

In this example, the input `guess` is required and must not be left blank.

The program in Figure 6.6 shows how a presence check for `guess` might be performed by checking whether the given input for `guess` is blank on line 16.

In general, presence checks use special functions or compare inputs to values such as `None` or an empty string ("") to ensure that all required inputs are provided.

Key Term

Presence check

Process of ensuring that all the required inputs are provided

#	Program 6.5 game_with_presence_check.py
1	import random # For random.randint()
2	
3	# List of possible secret words
4	WORDS = ['add', 'bug', 'cat', 'dog', 'eye', 'for', 'sky', 'zip']
5	
6	# Choose a random secret word
7	random_index = random.randint(0, len(WORDS) - 1)
8	secret_word = WORDS[random_index]
9	
10	# Main loop to keep repeating until secret word is guessed
11	while True:
12	
13	# Input and input validation
14	while True:
15	guess = input("Enter guess of length 3: ")
16	if guess != '':
17	break
18	print("Data validation failed!")
19	print("Guess is required and must not be left blank")
20	
21	# Process and output
22	if guess == secret_word:
23	print('Correct')
24	break # Exit main loop here
25	print('Wrong')

▲ **Figure 6.6** Example of program that performs a presence check

6.1.2.4 Format Check

Sometimes a problem requires the input to satisfy additional complex requirements such as following a particular pattern. This is known as a **format check**.

For instance, consider a program that prints an appropriate greeting for a given time of day.

▼ **Table 6.6** Example of problem requiring format check

Input	Output
<ul style="list-style-type: none"> time: time in the 24-hour clock in HH:MM format 	<ul style="list-style-type: none"> An appropriate greeting for the given time; "Good Morning" from 05:00 to 11:59, "Good Afternoon" from 12:00 to 17:59, "Good Evening" from 18:00 to 21:59, and "Good Night" from 22:00 to 04:59

In this problem, the input must follow the HH:MM format. This means that the user must key in exactly two digits followed by a colon and another two digits.

The program in Figure 6.7 shows how this format check might be performed.

Key Term

Format check

Process of ensuring that input data matches a required arrangement or pattern

#	Program 6.6 greeting_with_format_check.py
	<pre> 1 # Input and input validation 2 while True: 3 time = input("Enter time: ") 4 if len(time) == 5: 5 hours = time[:2] 6 minutes = time[-2:] 7 if time[2] == ":" and hours.isdigit() and minutes.isdigit(): 8 break 9 print("Data validation failed!") 10 print("Time should be in HH:MM format") 11 12 # Process 13 hours = int(hours) 14 if hours < 5: 15 greeting = "Good Night" 16 elif hours < 12: 17 greeting = "Good Morning" 18 elif hours < 18: 19 greeting = "Good Afternoon" 20 elif hours < 22: 21 greeting = "Good Evening" 22 else: 23 greeting = "Good Night" 24 25 # Output 26 print(greeting) </pre>

str.isdigit() method returns True if every character in the str is a digit.

▲ Figure 6.7 Example of program that performs a format check

Depending on the problem to be solved, more complex format checks, such as requiring that the input string uses a particular data format, may also be needed.

Did you know?

Performing format checks is a very common task in programming. However, the code for format checks can get very long. For instance, the following code is one possible way to check whether every letter in a `str` named `sleep` is 'z' using an output `bool` named `is_valid`:

```
is_valid = True
for letter in sleep:
    if letter != 'z':
        is_valid = False
        break
print(is_valid) # Outputs True only if every letter in sleep is 'z'
```

▲ **Figure 6.8** Format check for whether every letter in `sleep` is 'z'

To make the code for format checks shorter to write and read, programmers can use a specialised “regular expression language” that is designed for this purpose. For instance, the following code performs the same check as above but is much shorter:

```
import re
is_valid = bool(re.fullmatch("z*", sleep))
print(is_valid) # Outputs True only if every letter in sleep is 'z'
```

▲ **Figure 6.9** Equivalent format check using the regular expression "z*"

You can find out more about regular expressions and how to use Python’s `re` module (short for regular expressions) in this website:

<https://docs.python.org/3/library/re.html>



Quick Check 6.1

1. State whether each of the following extracts of input entry and validation code is performing a length check, range check, presence check or format check:

a) Extract 1:

```
while True:  
    s = input("Enter s: ")  
    if len(s) == 2 and s[0] in "ABCDEF" and s[1].isdigit():  
        break  
    print("Data validation failed!")  
...
```

▲ Figure 6.10 Extract 1

b) Extract 2:

```
while True:  
    first = input("Enter first name: ")  
    middle = input("Enter middle name (optional): ")  
    last = input("Enter last name: ")  
    if first == "" or last == "":  
        print("Data validation failed!")  
    else:  
        break  
...
```

▲ Figure 6.11 Extract 2

c) Extract 3:

```
p = float(input("Enter p:"))  
while p < 0.0 or p > 1.0:  
    print("Data validation failed!")  
    p = float(input("Enter p:"))  
...
```

▲ Figure 6.12 Extract 3

d) Extract 4:

```

n = int(input("Enter n:"))

while True:
    integers = []
    while True:
        input_text = input("Enter integer, blank to end: ")
        if input_text == "":
            break
        integers += [int(input_text)]
        if len(integers) == n:
            break
    print("Data validation failed!")

...

```

▲ Figure 6.13 Extract 4

2. Write the input entry and validation code for a program that needs to accept a postal code. If the input entered via the keyboard is invalid, your input validation code should keep trying by asking for the input to be entered again. The specification for the problem's input is provided below. (The specification for the output is not needed to solve this problem.)

▼ Table 6.7 Input and output requirements for a problem that uses a postal code for input

Input	Output
<ul style="list-style-type: none"> • <code>postal_code</code>: a <code>str</code> containing exactly six digits with no other characters 	(not needed)

6.2 Designing Test Cases

After data validation, we can be sure that all the inputs provided fulfil the requirements and do not have errors. However, there may still be errors and imperfections within our code. To detect these defects, it is necessary to perform testing using well-designed test cases that cover a wide variety of conditions.

Designing good test cases is important because it is usually impossible to test every possible input combination. We should focus on situations where errors are most likely to occur.

Before learning about designing test cases, we need to understand the types of program errors that are possible and which types can be detected by test cases.

6.2.1 Types of Program Errors

There are three main types of program errors:

1. Syntax errors
2. Run-time errors
3. Logic errors

Note that the three types of program errors are not mutually exclusive. In other words, some run-time errors may also be considered logic errors.

Key Terms

Crash

Sudden stop of a program due to an error

Hang

Unresponsiveness of a program due to an error

▼ Table 6.8 Description of program error types

Question	Syntax errors	Run-time errors	Logic errors
What are they?	Errors that are due to incorrect source code that does not follow the rules of the language	Errors that are detected while a program is running, usually causing the program to crash or hang	Errors that usually do not cause the program to crash or hang immediately; instead the program does not give the expected output
When are they detected?	When the compiler or interpreter tries to translate the source code to machine code	While the program is being run	While the program is being run
What causes them?	Spelling mistakes or the incorrect sequence of symbols in source code	Incorrect use of commands, input data that has not been properly validated or conditions occurring outside of the program's control (such as running out of memory)	Use of an incorrect or incomplete algorithm

6.2.1.1 Syntax Errors

Syntax errors are usually the easiest to detect as they are “caught” at the initial code translation stage. It is usually not necessary to design test cases to find syntax errors.

Let us look again at the program we used previously to find the average value in a list of numbers (see section 6.1.1). In the program in Figure 6.14, we introduced a syntax error on line 5 by using the assignment operator (=) instead of the equivalence operator (==).

#	Program 6.7 syntax_error.py
1	<code># Input</code>
2	<code>values = []</code>
3	<code>while True:</code>
4	<code> input_str = input("Enter integer, blank to end: ")</code>
5	<code> if input_str = "":</code>
6	<code> break</code>
7	<code> values += [int(input_str)]</code>
8	<code></code>
9	<code># Process</code>
10	<code>if len(values) > 0:</code>
11	<code> sum_values = sum(values)</code>
12	<code> average = sum_values / len(values)</code>
13	<code>else:</code>
14	<code> average = None</code>
15	<code></code>
16	<code># Output</code>
17	<code>print(average)</code>

▲ Figure 6.14 A syntax error is introduced on line 5 (highlighted)

```
=====
RESTART: C:/Examples/syntax_error.py =====
File "syntax_error.py", line 5
    if input_str = "":
               ^
SyntaxError: invalid syntax
```

▲ Figure 6.15 The program gives an error message that specifies the location of the error

By fixing this syntax error, the program is able to run successfully.

#	Program 6.8 syntax_error_fixed.py
1	<code># Input</code>
2	<code>values = []</code>
3	<code>while True:</code>
4	<code> input_str = input("Enter integer, blank to end: ")</code>
5	<code> if input_str == "":</code>
6	<code> break</code>
7	<code> values += [int(input_str)]</code>
8	<code></code>
9	<code># Process</code>
10	<code>if len(values) > 0:</code>
11	<code> sum_values = sum(values)</code>
12	<code> average = sum_values / len(values)</code>
13	<code>else:</code>
14	<code> average = None</code>
15	<code></code>
16	<code># Output</code>
17	<code>print(average)</code>

▲ **Figure 6.16** The syntax error on line 5 is fixed by using the correct operator (highlighted)

```
===== RESTART: C:/Examples/syntax_error_fixed.py =====
Enter integer, blank to end: 2
Enter integer, blank to end: 0
Enter integer, blank to end: 1
Enter integer, blank to end: 7
Enter integer, blank to end:
2.5
```

▲ **Figure 6.17** The program now runs successfully

6.2.1.2 Run-Time Errors

A **run-time error** refers to an error that occurs while a program is running.

For example, in Program 6.8, when the input is “seven” instead of the digit “7”, a run-time error occurs. This run-time error is caused by the input data that has not been properly validated by a format check to ensure that it is a valid integer (i.e., numeric digits only with a possible minus sign in front).

Key Term

Run-time error

Error that occurs while the program is running

The program gives an error message as shown below.

```
===== RESTART: C:/Examples/syntax_error_fixed.py =====
Enter integer, blank to end: 2
Enter integer, blank to end: 0
Enter integer, blank to end: 1
Enter integer, blank to end: seven
Traceback (most recent call last):
  File "syntax_error_fixed.py", line 7, in <module>
    values += [int(input_str)]
ValueError: invalid literal for int() with base 10: 'seven'
```

▲ **Figure 6.18** Example of a run-time error in Python

Did you know?

In compiled languages such as C or Java, since source code is fully translated to machine code before the program runs, it is not possible for a syntax error to cause a run-time error.

In interpreted languages such as Python, however, since source code is translated to machine code *while* the program is running, syntax errors can also be considered as run-time errors.

In Program 6.9, we have removed any checks for an empty list before calculating the average. When we run this program and immediately enter a blank, the program crashes with a division by zero error. This is another example of a run-time error.

#	Program 6.9 runtime_error.py
	<pre> 1 # Input 2 values = [] 3 while True: 4 input_str = input("Enter integer, blank to end: ") 5 if input_str == "": 6 break 7 values += [int(input_str)] 8 9 # Process 10 sum_values = sum(values) 11 average = sum_values / len(values) 12 13 # Output 14 print(average)</pre>

▲ **Figure 6.19** The checks for an empty list are removed

```
===== RESTART: C:/Examples/runtime_error.py =====
Enter integer, blank to end:
Traceback (most recent call last):
  File "C:/Examples/runtime_error.py", line 11, in <module>
    average = sum_values / len(values)
ZeroDivisionError: division by zero
```

▲ Figure 6.20 Example of another run-time error in Python

6.2.1.3 Logic Errors

While run-time errors are usually obvious when they occur, **logic errors** may be “hidden” and may not show any obvious sign that something is amiss. In such cases, the only way to detect the logic error is by running a test case and observing that the actual output of the program is different from the expected output.

For example, in Program 6.10, we have introduced a logic error on line 16 by using an incorrect formula that multiplies (instead of dividing) `sum_values` by the number of items in `values` to calculate the average.

Key Term

Logic error

Error that causes the output of a program to be different from what is expected

#	Program 6.10 logic_error.py
	<pre> 1 # Input 2 values = [] 3 while True: 4 input_str = input("Enter integer, blank to end: ") 5 if input_str == "": 6 break 7 values += [int(input_str)] 8 9 # Process 10 if len(values) > 0: 11 sum_values = sum(values) 12 average = sum_values * len(values) 13 else: 14 average = None 15 16 # Output 17 print(average)</pre>

▲ Figure 6.21 Example of a logic error

```
===== RESTART: C:/Examples/logic_error.py =====
Enter integer, blank to end: 2
Enter integer, blank to end: 0
Enter integer, blank to end: 1
Enter integer, blank to end: 7
Enter integer, blank to end:
40
```

▲ Figure 6.22 Output for example of a logic error

While the program appears to finish running with no errors, the output value of 40 is different from the expected output of 2.5 based on the input data of 2, 0, 1 and 7. Such unexpected or incorrect output is the most obvious sign that a program contains “hidden” logic errors.

Sometimes, logic errors may be obvious when the program crashes with a run-time error. However, the resulting run-time error may not occur immediately on the line of code containing the logic error. Such cases are tricky because the programmer would usually need to “work backwards” from the location of the crash to find out where the logic error is actually located. This is because the crash would usually be caused by a variable with an invalid value assigned to it by a previous line of code. This value could in turn be derived from another variable with an incorrect value assigned by a previous line of code, and so on.

For instance, in the program below, a run-time error occurs on line 11 because the contents of `values` are invalid for summing. However, this is actually caused by a logic error introduced on line 7, where `strs` are being inserted into `values` instead of `ints`.

#	Program 6.11 logic_error_2.py
	<pre> 1 # Input 2 values = [] 3 while True: 4 input_str = input("Enter integer, blank to end: ") 5 if input_str == "": 6 break 7 values += [input_str] Run-time error on line 11 is 8 9 # Process 10 if len(values) > 0: 11 sum_values = sum(values) 12 average = sum_values / len(values) 13 else: 14 average = None 15 16 # Output 17 print(average)</pre>

▲ Figure 6.23 Run-time error occurs on line 11 but is caused by logic error on line 7

```
===== RESTART: C:/Examples/logic_error_2.py =====
Enter integer, blank to end: 2
Enter integer, blank to end: 0
Enter integer, blank to end: 1
Enter integer, blank to end: 7
Enter integer, blank to end:
Traceback (most recent call last):
  File "C:/Examples/logic_error_2.py", line 11, in <module>
    sum_values = sum(values)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

▲ **Figure 6.24** The program alerts the run-time error but not the logic error

To detect run-time errors and logic errors like the ones described in these examples, it is important to run the programs that we write through multiple test cases. The test cases should cover three types of conditions:

1. Normal conditions
2. Boundary conditions
3. Error conditions

6.2.2 Conditions for Test Cases

To study the three types of conditions, let us consider the problem of converting a phrase into an acronym. An acronym is an abbreviation formed from the first letter of each word. Acronyms must usually be pronounceable, but for our problem, we simply want to combine the first letter of each word into a capitalised abbreviation regardless of whether the result is pronounceable.

The input and output requirements for this problem are specified in Table 6.9.

▼ **Table 6.9** Input and output requirements for the problem of converting a phrase into an acronym

Input	Output
<ul style="list-style-type: none"> • <code>input_text</code>: phrase to be used for generating the acronym, should contain only letters and spaces 	<ul style="list-style-type: none"> • Capitalised acronym of given phrase

6.2.2.1 Normal Conditions

First, design test cases to make sure that the program to be tested works as intended under **normal conditions** – situations where the input data is what we would expect to be provided with during normal use of the program. In general, under normal conditions, at least one test case should be designed for each type of expected input. In this case, only one type of input is expected (a `str`), so we will design a test case accordingly:

Key Term

Normal conditions (programming)

Situations where input data follows what is expected during normal use of the program

▼ **Table 6.10** Test Case 1 for acronym generation problem

→ Input	Expected Output →
• <code>input_text: "Computer Science"</code>	CS

However, we also want to be sure that the test cases for normal conditions accurately represent the variety of inputs that we are expecting. To test this, we shall design an additional test case so that words that start with both upper-case and lower-case letters are represented.

▼ **Table 6.11** Test Case 2 for acronym generation problem

→ Input	Expected Output →
• <code>input_text: "Man on earth"</code>	MOE

Test cases for normal conditions are the most straightforward and will usually be the first test cases to be written and passed successfully.

6.2.2.2 Boundary Conditions

Next, design test cases to make sure that the program works as intended under **boundary conditions** – situations where the input data is at the limit of what the program is designed for, or where special handling of the input data is required. The limits of a program are usually in terms of quantity (for `int` and `float` inputs) and/or length (for `list` and `str` inputs). On the other hand, special handling is usually needed when the problem definition specifies that certain values are to be treated as exceptions (such as using `-1` to represent missing data in a survey).

Key Term

Boundary conditions (programming)

Situations where input data is at the limit of what the program is designed to cope with or where special handling is required

For the acronym generation problem, there is no defined maximum length for the input phrase, so it is not possible to design a test case that would be at the upper limit of what the program can handle. However, we do want the program to work as intended even with the *minimum* length of input possible, that is, an empty `str`. In this case, we can choose the correct output to also be an empty `str`, as shown in Test Case 3.

▼ **Table 6.12** Test Case 3 for acronym generation problem

→ Input	Expected Output →
• <code>input_text: ""</code>	(blank output)

Did you know?

In the acronym generation problem, we could also have chosen the expected output to be an error. If you are unsure whether a set of inputs should generate an error or output something unusual, it is likely that you have encountered a boundary condition where the inputs are at the limit of what is considered normal. Usually, either option of generating an error or producing unusual output (such as an empty string) would be acceptable. However, depending on the situation, one option may be preferable to the other. For instance, if the output for the acronym generation problem will in turn be used as a required input for another problem, it might be better to generate an error instead so that a blank acronym is never produced.

Under normal conditions, we would expect the input phrase to be a sequence of words with a single space in between. However, the program must also be able to run in the special case where additional spaces are provided before and after words. Let us design an additional test case to cover such situations.

▼ **Table 6.13** Test Case 4 for acronym generation problem

→ Input	Expected Output →
• <code>input_text: " Computer Science "</code>	CS

For Test Case 4, the expected output for “Computer Science” is exactly the same as the expected output for “Computer Science” (with no additional spaces).

Determining the boundary conditions for a problem and the expected output for each case can be tedious as well as difficult. However, this process is necessary to ensure that the program behaves as intended regardless of the input supplied.

6.2.2.3 Error Conditions

Finally, it is important to design test cases to make sure that the program behaves as expected under **error conditions** – situations where the input data would normally be rejected by the program.

For the acronym generation problem, it is clearly stated that the input phrase should contain only letters and spaces. Based on this definition, any input that contains punctuation symbols or digits should generate an error. The following two test cases cover these situations:

Key Term

Error conditions (programming)

Situations where input data would normally be rejected by the program

▼ **Table 6.14** Test Case 5 for acronym generation problem

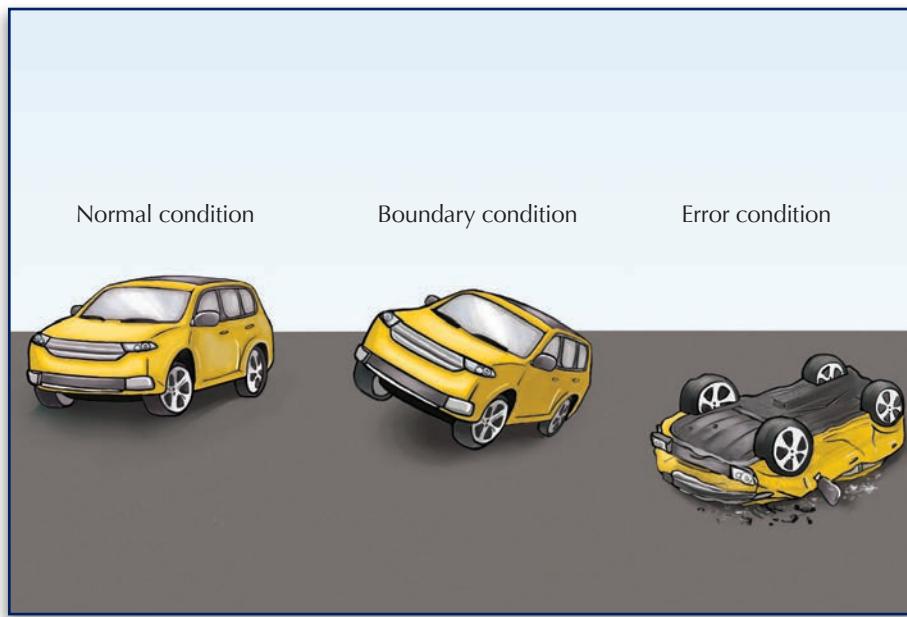
→ Input	Expected Output →
<ul style="list-style-type: none"> • input_text: "Lead, Care, Inspire" 	Data validation failed! Input text should only contain letters and spaces

▼ **Table 6.15** Test Case 6 for acronym generation problem

→ Input	Expected Output →
<ul style="list-style-type: none"> • input_text: "Computing 2017" 	Data validation failed! Input text should only contain letters and spaces

Test cases for error conditions are necessary because programs that fail to reject invalid input may end up performing unintended or even harmful actions. For instance, a large number of security flaws in programs today are caused by the improper handling of error conditions. A typical example in some programming languages would be forgetting to check whether the input data supplied by a user can fit into the memory space allocated to store the data. A program that tries to store this invalid data anyway will end up overwriting subsequent areas of memory and allow potential attackers to insert their own instructions into the program.

Figure 6.25 shows the analogy of a car to illustrate the three types of conditions that test cases should cover.



▲ **Figure 6.25** Analogy of normal, boundary and error conditions

Under normal conditions, the car has all four wheels on the ground. We expect the car to function perfectly under these conditions.

Under boundary conditions, the car may be balancing on two wheels. The car may or may not function correctly under these conditions, and depending on the problem context, the car may or may not be required to function under such conditions.

Under error conditions, the car has all four wheels up in the air. A well-designed car would fail to function in such a situation, and if possible, prevent any harmful actions from being taken by the driver.



Quick Check 6.2

1. A student wants to solve the following problem:

▼ **Table 6.16** Input and output requirements for the problem of drawing a square

Input	Output
<ul style="list-style-type: none"> • n: number of “*” characters needed for each side of the square, should be an integer between 1 and 20 inclusive 	<ul style="list-style-type: none"> • a text-based square made out of “*” characters with n “*” characters on each side, square should be filled in with “*” characters with no other characters inside

The following are two test cases under normal conditions for this problem. (Note that although the expected outputs may appear rectangular, they can be considered as squares as they have the same number of characters on each side.)

▼ **Table 6.17** Test Case 1 for square drawing problem

→ Input	Expected Output →
<ul style="list-style-type: none"> • n: 5 	<pre>* **** * * **** * * **** * * **** * * **** *</pre>

▼ **Table 6.18** Test Case 2 for square drawing problem

→ Input	Expected Output →
<ul style="list-style-type: none"> • n: 10 	<pre>* ***** * * * ***** * * * ***** * * * ***** * * * ***** * * * ***** * * * ***** * * * ***** * * * ***** * * * ***** * *</pre>

- Design another test case that covers normal conditions for this problem.
- Design two test cases that cover the boundary conditions for this problem. For each test case, explain why it covers a boundary condition.
- Design two test cases that cover the error conditions for this problem. For each test case, explain why it covers an error condition. (You may choose your own expected error message.)

Questions 2 and 3 involve programs meant to solve the following diamond drawing problem:

▼ **Table 6.19** Input and output requirements for the problem of drawing a diamond

Input	Output
<ul style="list-style-type: none"> • n: height of diamond in number of lines and also number of “*” characters needed for the widest part of the diamond, should be an odd integer between 1 and 19 inclusive 	<ul style="list-style-type: none"> • a text-based symmetrical diamond made out of “*” and “ ” (space) characters with a height of n lines and width of n “*” characters for the widest part of the diamond, diamond should be filled in with “*” characters with no other characters inside

The following is a test case under normal conditions for this problem:

▼ **Table 6.20** Test case for diamond drawing problem

→ Input	Expected Output →
<ul style="list-style-type: none"> • n: 5 	<ul style="list-style-type: none"> * *** ***** *** *

2. The program below for the diamond drawing problem has three syntax errors.

- a) Identify where the syntax errors are.
- b) Rewrite the program to fix these syntax errors.

#	Program 6.12 diamond_with_syntax_errors.py
1	# Input and input validation
2	while True:
3	input_text = input("Enter n: ")
4	if input_text.isdigit()
5	n = int(input_text)
6	if n >= 1 and n <= 19 and n % 2 == 1:
7	break
8	else:
9	print("Data validation failed!")
10	print("n should be odd and between 1 and 19 inclusive")
11	else:
12	print("Data validation failed!")
13	print("n should be a positive integer")
14	
15	# Process and output
16	for index in range(n // 2):
17	print(' ' * (n // 2 - index) + '*' * (2 * index + 1))
18	for index in range(n // 2 + 1):
19	print(' ' * index + '*' * (n - index * 2)))

▲ **Figure 6.26** Program for diamond drawing problem with syntax errors

3. The program below for the diamond drawing problem crashes with a run-time error on line 25 when it runs on the given test case shown in Table 6.21.

#	Program 6.13 diamond_crash.py
1	# Input and input validation 2 while True: 3 n_str = input("Enter n: ") 4 if not n_str.isdigit(): 5 print("Data validation failed!") 6 print("n should be a positive integer") 7 continue 8 n = int(n_str) 9 if n < 1 or n > 19 or n % 2 == 0: 10 print("Data validation failed!") 11 print("n should be odd and between 1 and 19 inclusive") 12 continue 13 break 14 15 # Process and output 16 stars = [] 17 index = 1 18 while index < n: 19 stars += [" " * ((n - index) // 2) + "*" * index] 20 index += 2 21 22 for index in range(n // 2): 23 print(stars[index]) 24 for index in range(n // 2 + 1): 25 print(stars[-index - 1])

▲ Figure 6.27 Program on diamond drawing problem as test case

▼ Table 6.21 Running diamond_crash.py on test case

→ Input	Expected Output →	
• n: 5	* *** ***** *** *	
Running diamond _ crash.py on test case →		Result
Enter n: 5 * *** *** * Traceback (most recent call last): File "diamond_crash.py", line 25, in <module> print(stars[-index - 1]) IndexError: list index out of range		Failed

Fix this error by changing only one line and explain whether line 25 should be the line of code to change.

6.3 Debugging Program Errors

6.3.1 Bugs and Debugging

The purpose of designing test cases is to detect when a program does not behave as intended by the programmer. When this happens, the program is said to have a **bug**. The process of fixing the program so that it behaves as intended is called **debugging**.

To demonstrate how debugging is performed, let us work on a draft program to solve the acronym generation problem. Since words usually have a space before them, we can use the general algorithm for extracting characters from a string (see section 5.4.6).

Key Terms

Bug

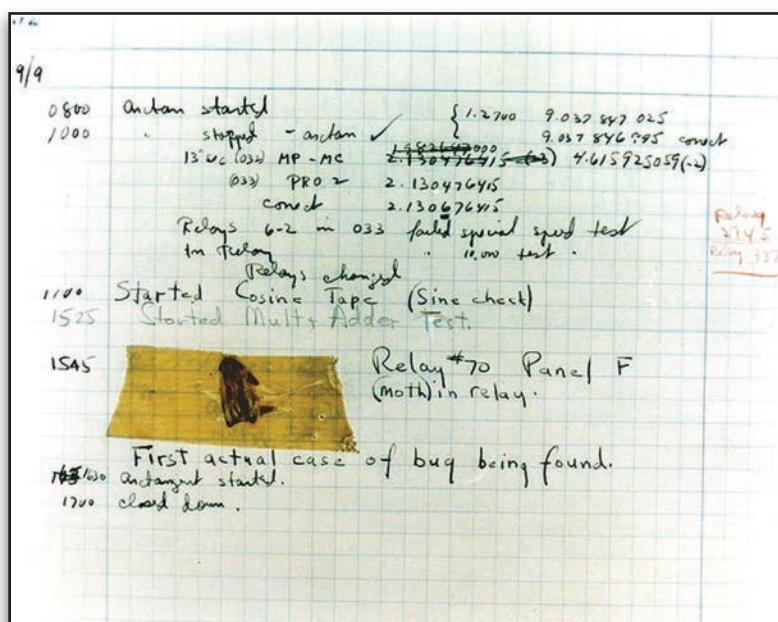
Defect in a program that causes it to behave in an unintended manner

Debugging

Process of identifying and removing defects from a program

Did you know?

Figure 6.28 shows a journal entry dated 9 September 1947 by users of an early electronic computer called the “Mark II”. They had discovered a moth trapped in part of the computer, causing it to malfunction. This dead moth is often considered the first actual computer “bug” to be found.



▲ Figure 6.28 A moth trapped in a computer was the first actual computer “bug”

In this case, we use the condition that extracts only characters that follow a space.

#	Program 6.14 acronym_draft.py
1	<code># Input</code>
2	<code>input_text = input("Enter the input text: ")</code>
3	
4	<code># Process</code>
5	<code>acronym = ""</code>
6	<code>for index in range(len(input_text)):</code>
7	<code> if input_text[index - 1].isspace():</code>
8	<code> acronym += input_text[index]</code>
9	
10	<code># Output</code>
11	<code>print(acronym)</code>

▲ **Figure 6.29** Draft program to solve the acronym generation problem

However, when we try to run the program on Test Case 1, we find that its output is different from what is expected.

▼ **Table 6.22** Running acronym_draft.py on Test Case 1

→ Input	Expected Output →	
• <code>input_text: "Computer Science"</code>	CS	
Running acronym_draft.py on Test Case 1 →		Result
<code>Enter the input text: Computer Science</code>		Failed
S		

This is a sign that there is a bug. Let us look at some common debugging techniques to fix it.

6.3.2 Common Debugging Techniques

6.3.2.1 Using Intermittent print Statements

One simple yet effective debugging technique is to keep track of how variables change as the program runs using the `print()` function. By adding a `print` statement that outputs the value of important variables inside a loop, we are able to get valuable information of how the variables change over time, just like what we were able to do with trace tables for flowcharts. For instance, the program below has been modified with the addition of line 9, which prints out the values of `index` and `acronym` each time the loop repeats.

#	Program 6.15 acronym_draft_with_debugging.py
1	<code># Input</code>
2	<code>input_text = input("Enter the input text: ")</code>
3	
4	<code># Process</code>
5	<code>acronym = ""</code>
6	<code>for index in range(len(input_text)):</code>
7	<code> if input_text[index - 1].isspace():</code>
8	<code> acronym += input_text[index]</code>
9	<code> print("Debug: when index=" + str(index) + ", acronym=" + acronym)</code>
10	
11	<code># Output</code>
12	<code>print(acronym)</code>
13	

▲ **Figure 6.30** Draft program with addition of `print` statement on line 10 (highlighted)

Running this new program on Test Case 1 reveals how these two variables change over time as characters are extracted from `input_text` to form the acronym.

▼ **Table 6.23** Running `acronym_draft_with_debugging.py` on Test Case 1

→ Input	Expected Output →	
• <code>input_text: "Computer Science"</code>	CS	
Running <code>acronym_draft_with_debugging.py</code> on Test Case 1 →		
Enter the input text: Computer Science Debug: when index=0, acronym= Debug: when index=1, acronym= Debug: when index=2, acronym= Debug: when index=3, acronym= Debug: when index=4, acronym= Debug: when index=5, acronym= Debug: when index=6, acronym= Debug: when index=7, acronym= Debug: when index=8, acronym= Debug: when index=9, acronym=S Debug: when index=10, acronym=S Debug: when index=11, acronym=S Debug: when index=12, acronym=S Debug: when index=13, acronym=S Debug: when index=14, acronym=S Debug: when index=15, acronym=S S		Failed

Note that in this example, all the output starting with “Debug” appears together. In more complex situations, however, different amounts of processing may occur between lines of debugging output, which is why this technique is called using “intermittent” print statements.

From the debugging output, we can see that the first letter of “Science” is correctly extracted into `acronym` when `index` reaches 9. However, the first letter of “Computer” is not.

Look again at line 7 in Program 6.15. When `index` is 0, the program is actually testing whether `input_text[-1]` (i.e., the last letter of `input_text`) is a space before deciding not to include `input_text[0]` in the acronym. This is not the intended behaviour and is caused by the fact that there is no “previous character” for the first character of a string.

Let us try to fix this bug by checking for this special case on line 7.

#	Program 6.16 acronym_draft_with_debugging_2.py
1	<code># Input</code>
2	<code>input_text = input("Enter the input text: ")</code>
3	
4	<code># Process</code>
5	<code>acronym = ""</code>
6	<code>for index in range(len(input_text)):</code>
7	<code> if input_text[index - 1].isspace() or index == 0:</code>
8	<code> acronym += input_text[index]</code>
9	<code> print("Debug: when index=" + str(index) + ", acronym=" + acronym)</code>
10	
11	<code># Output</code>
12	<code>print(acronym)</code>

▲ **Figure 6.31** Draft program with first character bug fixed by adding the highlighted code on line 7

Running this amended program on Test Case 1 now produces the expected output (if we ignore the lines of debugging output that start with “Debug”).

▼ **Table 6.24** Running acronym_draft_with_debugging_2.py on Test Case 1

→ Input	Expected Output →	
• input_text: "Computer Science"	CS	
Running acronym_draft_with_debugging_2.py on Test Case 1 →		
Enter the input text: Computer Science Debug: when index=0, acronym=C Debug: when index=1, acronym=C Debug: when index=2, acronym=C Debug: when index=3, acronym=C Debug: when index=4, acronym=C Debug: when index=5, acronym=C Debug: when index=6, acronym=C Debug: when index=7, acronym=C Debug: when index=8, acronym=C Debug: when index=9, acronym=CS Debug: when index=10, acronym=CS Debug: when index=11, acronym=CS Debug: when index=12, acronym=CS Debug: when index=13, acronym=CS Debug: when index=14, acronym=CS Debug: when index=15, acronym=CS CS		Passed

However, when we run the program on Test Case 2, once again we receive an unexpected output.

▼ **Table 6.25** Running `acronym_draft_with_debugging_2.py` on Test Case 2

→ Input	Expected Output →	
• <code>input_text: "Man on earth"</code>	MOE	
Running <code>acronym_draft_with_debugging_2.py</code> on Test Case 2 →		Result
<pre>Enter the input text: Man on earth Debug: when index=0, acronym=M Debug: when index=1, acronym=M Debug: when index=2, acronym=M Debug: when index=3, acronym=M Debug: when index=4, acronym=Mo Debug: when index=5, acronym=Mo Debug: when index=6, acronym=Mo Debug: when index=7, acronym=Moe Debug: when index=8, acronym=Moe Debug: when index=9, acronym=Moe Debug: when index=10, acronym=Moe Debug: when index=11, acronym=Moe Moe</pre>		Failed

Fortunately, the cause of this bug is quite clear: we did not convert the output to upper case. The program below fixes this by adding line 10 to convert the output to upper case when the program has exited the loop.

#	Program 6.17 <code>acronym_draft_with_debugging_3.py</code>
<pre> 1 # Input 2 input_text = input("Enter the input text: ") 3 4 # Process 5 acronym = "" 6 for index in range(len(input_text)): 7 if input_text[index - 1].isspace() or index == 0: 8 acronym += input_text[index] 9 print("Debug: when index=" + str(index) + ", acronym=" + acronym) 10 acronym = acronym.upper() 11 12 # Output 13 print(acronym)</pre>	

▲ **Figure 6.32** Draft program with letter case bug fixed by adding the highlighted code on line 10

Running the amended program on Test Case 2 now generates the expected output (if we ignore the lines of debugging output that start with “Debug”).

▼ **Table 6.26** Running acronym_draft_with_debugging_3.py on Test Case 2

→ Input	Expected Output →	
• input_text: "Man on earth"	MOE	
Running acronym_draft_with_debugging_3.py on Test Case 2 →		Result
Enter the input text: Man on earth Debug: when index=0, acronym=M Debug: when index=1, acronym=M Debug: when index=2, acronym=M Debug: when index=3, acronym=M Debug: when index=4, acronym=Mo Debug: when index=5, acronym=Mo Debug: when index=6, acronym=Mo Debug: when index=7, acronym=Moe Debug: when index=8, acronym=Moe Debug: when index=9, acronym=Moe Debug: when index=10, acronym=Moe Debug: when index=11, acronym=Moe MOE		Passed

6.3.2.2 Testing the Program in Chunks or Parts

Let us continue running our program on the test cases we designed. The program behaves as intended for Test Case 3 under the boundary conditions of having an empty str as input.

▼ **Table 6.27** Running acronym_draft_with_debugging_3.py on Test Case 3

→ Input	Expected Output →	
• input_text: ""	(blank output)	
Running acronym_draft_with_debugging_3.py on Test Case 3 →		Result
Enter the input text:		Passed

However, running the program on Test Case 4 produces something unexpected.

▼ **Table 6.28** Running `acronym_draft_with_debugging_3.py` on Test Case 4

→ Input	Expected Output →	
• <code>input_text: " Computer Science "</code>	CS	
Running <code>acronym_draft_with_debugging_3.py</code> on Test Case 4 →		
<pre>Enter the input text: Computer Science Debug: when index=0, acronym= Debug: when index=1, acronym= Debug: when index=2, acronym= C Debug: when index=3, acronym= C Debug: when index=4, acronym= C Debug: when index=5, acronym= C Debug: when index=6, acronym= C Debug: when index=7, acronym= C Debug: when index=8, acronym= C Debug: when index=9, acronym= C Debug: when index=10, acronym= C Debug: when index=11, acronym= C Debug: when index=12, acronym= C S Debug: when index=13, acronym= C S Debug: when index=14, acronym= C S Debug: when index=15, acronym= C S Debug: when index=16, acronym= C S Debug: when index=17, acronym= C S Debug: when index=18, acronym= C S Debug: when index=19, acronym= C S Debug: when index=20, acronym= C S C S</pre>	Failed	

From the output, it looks like the acronym has some extra spaces that should not be there. To investigate the cause of this bug, we shall decompose the program into smaller parts and test the parts one at a time. This allows us to narrow down the possible locations of any bugs and reduce the amount of code that we need to study at any one time.

To test only one part of a program at a time, we can temporarily disable lines of code by inserting a hash symbol (#) at the start of these lines. This process is called commenting out code since it works by temporarily turning the code into comments. (Recall in section 4.3.2 on adding comments to your program.)

Another method to break a program into smaller parts so that each part can be tested separately is by putting each part in its own function. Since functions allow the same instructions to be performed using different arguments, this makes it easy to test each part of the program repeatedly using different inputs. For instance, Figure 6.33 shows how the tasks of getting input and testing if a character should be included in the acronym can be broken out into two user-defined functions named `get_text()` and `should_include()` respectively.

#	Program 6.18 acronym_draft_with_functions.py
1	<code># Gets input text from the user</code>
2	<code>def get_text():</code>
3	<code> return input("Enter the input text: ")</code>
4	
5	<code># Returns whether text[i] should be included in the acronym</code>
6	<code>def should_include(text, i):</code>
7	<code> return text[i - 1].isspace() or i == 0</code>
8	
9	<code># Input</code>
10	<code>input_text = get_text()</code>
11	
12	<code># Process</code>
13	<code>acronym = ""</code>
14	<code>for index in range(len(input_text)):</code>
15	<code> if should_include(input_text, index):</code>
16	<code> acronym += input_text[index]</code>
17	<code>acronym = acronym.upper()</code>
18	
19	<code># Output</code>
20	<code>print(acronym)</code>

▲ **Figure 6.33** Breaking a program into smaller parts using functions

This modified program's overall behaviour is unchanged, so it still fails Test Case 4 by producing output with some extra spaces that should not be there. We can proceed to narrow down where the bug is located by decomposing the program line by line.

▼ **Table 6.29** Decomposition of `acronym_draft_with_debugging_3.py`

Line(s)	What the code does
1–8	Defines <code>get_text()</code> and <code>should_include()</code>
9–11	Uses <code>get_text()</code> to get input phrase from keyboard
12–13	Initialises <code>acronym</code> to empty <code>str</code>
14	Iterates through every character of input phrase
15	Uses <code>should_include()</code> to test if the current character of input phrase should be part of <code>acronym</code>
16	Adds the current character to <code>acronym</code> (if test succeeds)
17–18	Converts <code>acronym</code> to upper case
19–20	Displays <code>acronym</code> on screen

Since the letters “C” and “S” were extracted correctly, this implies that the following processes are all working:

- Using `get_text()` to get input from the keyboard (lines 9–11)
- Iterating through the input characters (line 14)
- Adding characters to the acronym (line 16)
- Displaying the acronym on the screen (lines 19–20)

Hence, we can deduce that the error is located in `should_include()` as part of the code that tests if the current character should be part of the acronym (line 15). We can also eliminate the possibility that the error is located at the code converting the acronym to upper case (lines 17–18) as it is not relevant in this case.

To confirm the location of the bug, we can run mini test cases. All the lines except for the function definitions are disabled by turning them into comments. At the end of the program, we then call `should_include()` repeatedly with different combinations of input arguments to determine if the function is working as expected.

#	Program 6.19 acronym_draft_with_mini_tests.py
1	<code># Gets input text from the user</code>
2	<code>def get_text():</code>
3	<code> return input("Enter the input text: ")</code>
4	
5	<code># Returns whether text[i] should be included in the acronym</code>
6	<code>def should_include(text, i):</code>
7	<code> return text[i - 1].isspace() or i == 0</code>
8	
9	<code># # Input</code>
10	<code># input_text = get_text()</code>
11	<code>#</code>
12	<code># # Process</code>
13	<code># acronym = ""</code>
14	<code># for index in range(len(input_text)):</code>
15	<code># if should_include(input_text, index):</code>
16	<code># acronym += input_text[index]</code>
17	<code># acronym = acronym.upper()</code>
18	<code>#</code>
19	<code># # Output</code>
20	<code># print(acronym)</code>
21	
22	<code>input_text = " Computer Science "</code>
23	<code>print(should_include(input_text, 0)) # Expected: False</code>
24	<code>print(should_include(input_text, 1)) # Expected: False</code>
25	<code>print(should_include(input_text, 2)) # Expected: True</code>

▲ Figure 6.34 Testing a function with different input arguments to confirm location of bug

In this case, the expected outputs for the three calls to `should_include()` are `False`, `False` and `True`. When `text` is `" Computer Science "` and `i` is either 0 or 1, we expect `should_include()` to return `False` since both `text[0]` and `text[1]` are spaces and spaces should not appear in acronyms. On the other hand, when `i` is 2 we expect `should_include()` to return `True` since `text[2]` is precisely the first letter of the word "Computer".

However, when we run the program, we do not get the expected outputs.

▼ **Table 6.30** Running `acronym_draft_with_mini_tests.py` on mini test cases

→ Input	Expected Output →	
Mini Test Case 1 • text: " Computer Science " • i: 0	False False True	
Running <code>acronym_draft_with_mini_tests.py</code> on mini test case →		Result
True True True		Failed

This result tells us that the bug is definitely in `should_include()`. To fix this bug, we will add an additional condition – the character being tested must not be a space.

#	Program 6.20 acronym_draft_fixed.py
1	# Gets input text from the user
2	def get_text():
3	return input("Enter the input text: ")
4	
5	# Returns whether text[i] should be included in the acronym
6	def should_include(text, i):
7	return [(text[i - 1].isspace() or i == 0) and not text[i].isspace()]
8	
9	# Input
10	input_text = get_text()
11	
12	# Process
13	acronym = ""
14	for index in range(len(input_text)):
15	if should_include(input_text, index):
16	acronym += input_text[index]
17	acronym = acronym.upper()
18	
19	# Output
20	print(acronym)

▲ **Figure 6.35** Draft program with unwanted spaces bug fixed by adding the highlighted code

This modified program now passes Test Case 4.

▼ **Table 6.31** Running acronym_draft_fixed.py on Test Case 4

➔ Input	Expected Output ➔	
• input_text: " Computer Science "	CS	
Running acronym_draft_fixed.py on Test Case 4 ➔		Result
Enter the input text: Computer Science CS		Passed

However, our program still does not pass Test Case 5 and Test Case 6 because it does not perform any data validation. To fix this, a format check is added to the program to ensure that the input phrase contains letters and spaces only, as shown in Figure 6.36.

#	Program 6.21 acronym_final.py
	<pre> 1 # Gets input text from the user 2 def get_text(): 3 while True: 4 input_text = input("Enter the input text: ") 5 input_is_valid = True 6 for c in input_text: 7 if not (c.isalpha() or c.isspace()): 8 print("Data validation failed!") 9 print("Input text should only contain " + 10 "letters and spaces") 11 input_is_valid = False 12 break 13 if input_is_valid: 14 return input_text 15 16 # Returns whether text[i] should be included in the acronym 17 def should_include(text, i): 18 return (text[i - 1].isspace() or i == 0) and not text[i].isspace() 19 20 # Input 21 input_text = get_text() 22 23 # Process 24 acronym = "" 25 for index in range(len(input_text)): 26 if should_include(input_text, index): 27 acronym += input_text[index] 28 acronym = acronym.upper() 29 30 # Output 31 print(acronym) </pre>

▲ Figure 6.36 Format check (highlighted) added to the program

This final version of the program, Program 6.21, passes Test Case 5 and Test Case 6.

▼ **Table 6.32** Running `acronym_final.py` on Test Case 5

→ Input	Expected Output →	
• <code>input_text: "Lead, Care, Inspire"</code>	Data validation failed! Input text should only contain letters and spaces	
Running <code>acronym_final.py</code> on Test Case 5 →		Result
Enter the input text: Lead, Care, Inspire Data validation failed! Input text should only contain letters and spaces Enter the input text:		Passed

▼ **Table 6.33** Running `acronym_final.py` on Test Case 6

→ Input	Expected Output →	
• <code>input_text: "Computing 2017"</code>	Data validation failed! Input text should only contain letters and spaces	
Running <code>acronym_final.py</code> on Test Case 6 →		Result
Enter the input text: Computing 2017 Data validation failed! Input text should only contain letters and spaces Enter the input text:		Passed

Since Program 6.21 passes all the test cases we have designed, we can confirm that it works as intended in solving the acronym generation problem.

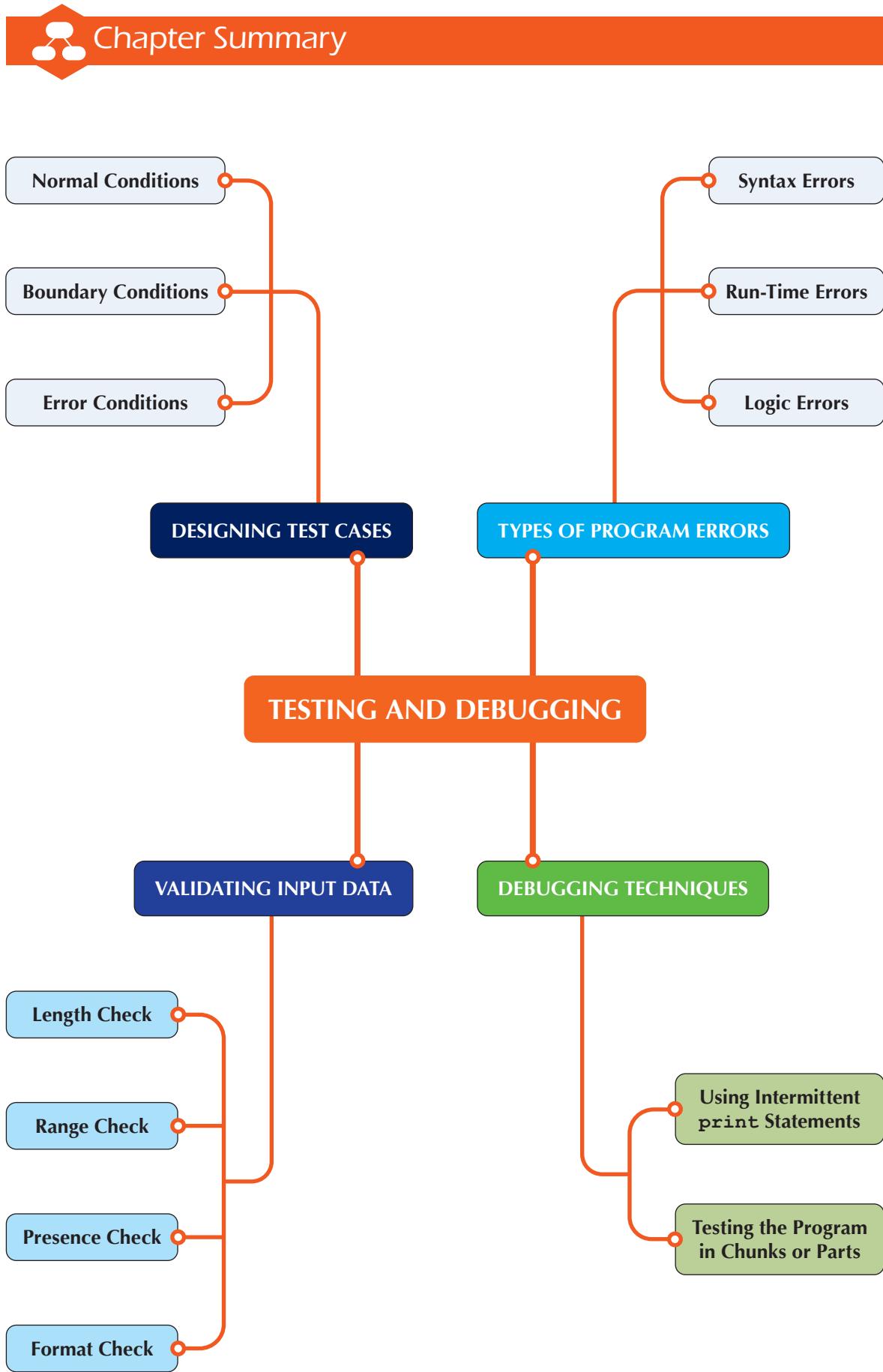


Quick Check 6.3

1. A new program has been written to solve the acronym generation problem. State whether it passes or fails each of the six test cases we have designed for the acronym generation problem.

#	Program 6.22 acronym_alternative.py
1	<code># Input</code>
2	<code>input_text = input("Enter the input text: ")</code>
3	
4	<code># Process</code>
5	<code>acronym = ""</code>
6	<code>previous_char = ' '</code>
7	<code>for char in input_text:</code>
8	<code> if previous_char.isspace() and not char.isspace():</code>
9	<code> acronym += char.upper()</code>
10	<code> previous_char = char</code>
11	
12	<code># Output</code>
13	<code>print(acronym)</code>

▲ Figure 6.37 New program for the acronym generation problem





Review Questions

1. A program is needed to solve the following word reversal problem:

▼ **Table 6.34** Input and output requirements for the word reversal problem

Input	Output
<ul style="list-style-type: none"> • <code>input_text</code>: phrase to reverse, should contain only letters and spaces 	<ul style="list-style-type: none"> • Given phrase with the order of the letters of each word reversed but the words themselves still in order, separated by one space between words

The draft program in Figure 6.38 is written for this purpose.

- a) The main algorithm for the program consists of three parts. Read through the code and state what each of the following parts of the program are intended to do:
- Line 5
 - Lines 7 to 11
 - Lines 13 to 16
- b) When this program is run on the following test case, the actual output does not match the expected output:

▼ **Table 6.35** Running `word_reversal_draft.py` on a test case

→ Input	Expected Output →	
<ul style="list-style-type: none"> • <code>input_text</code>: "Meetings on etiquette" 	sgniteem no etteuqite	
Running <code>word_reversal_draft.py</code> on test case →		Result
Enter input text: Meetings on etiquette Msgnitee on eetteuqit		Failed

- State the kind of condition (normal, boundary or error) covered by this test case.
- State whether the program has a syntax error or a logic error and explain why.
- Find out where the error is located by disabling lines of code and testing the program in chunks or parts. State the line on which the error is located and explain why the line is incorrect.
- Rewrite the program to fix the error so that it passes the test case.

#	Program 6.23 word_reversal_draft.py
1	<code># Input</code>
2	<code>input_text = input("Enter input text: ")</code>
3	
4	<code># Process</code>
5	<code>word_list = input_text.split()</code>
6	
7	<code>for i in range(len(word_list)):</code>
8	<code>current_word = ""</code>
9	<code>for letter_index in range(len(word_list[i])):</code>
10	<code>current_word += word_list[i][-letter_index]</code>
11	<code>word_list[i] = current_word</code>
12	
13	<code>result = ""</code>
14	<code>for word in word_list:</code>
15	<code>result += word + " "</code>
16	<code>result = result[:-1]</code>
17	
18	<code># Output</code>
19	<code>print(result)</code>

▲ **Figure 6.38** Draft program for the word reversal problem

2. The program in Figure 6.39 accepts a string of alphabetical letters only and outputs the string with its upper-case letters converted to lower-case and vice versa. For example, if the input were "comPUTing", then the output would be "COMpuTING".

The program uses three user-defined functions, `is_letters_only()`, `convert_letter()` and `convert_str()` to perform this task.

- a) The `is_letters_only()` function is supposed to return True if its parameter is made up of alphabetical letters only. Write down the expected return value for each of the following test cases:
 - i) `is_letters_only('hello')`
 - ii) `is_letters_only('Hello World')`
 - iii) `is_letters_only('HelloWorld')`
 - iv) `is_letters_only('Hello, World!')`
- b) Using an empty string as input is a boundary condition for the `is_letters_only()` function. Boundary conditions can be difficult to handle as there may be multiple options for expected behavior.
 - i) Suggest one possible argument for why the return value for `is_letters_only('')` should be True.
 - ii) Suggest one possible argument for why the return value for `is_letters_only('')` should be False.

- iii) Suppose an empty string should be treated as valid input for this program. Choose an appropriate return value for `is_letters_only('')` and fill in the function body for `is_letters_only()` accordingly. (To test your code, temporarily comment out the main program and run test cases.)

#	Program 6.24 opposite_case_unfinished.py
	<pre> 1 # Returns True if s has alphabetical letters only 2 def is_letters_only(s): 3 # TODO: See part (b) 4 5 # Converts single letter to opposite case 6 def convert_letter(l): 7 if l.islower(): 8 return l.upper() 9 return l.lower() 10 11 # Converts string of letters to opposite case 12 def convert_string(s): 13 result = '' 14 for c in s: 15 result = convert_letter(c) 16 return result 17 18 while True: 19 s = input('Enter string: ') 20 if is_letters_only(s): 21 break 22 print('String must have alphabetical letters only') 23 24 print(convert_string(s)) </pre>

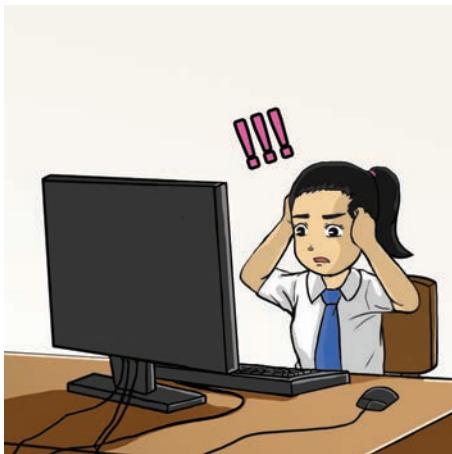
▲ **Figure 6.39** Draft program for the opposite case problem

- c) Even with `is_letters_only()` correctly filled in, there is a bug in the program that causes its output to be different from what is expected. For example, if the input were "comPUting", the program will output "G" instead of the expected "COMpuTING".
- i) Explain why a bug in `convert_letter()` can affect `convert_string()`.
 - ii) Find the bug in `convert_string()` and suggest how it can be fixed.

How Can I Be a Safe and Responsible Computer User?

Siti has just finished typing a book report on her computer in the library. She checks her email and leaves the computer for a moment to shelve some books. When she returns, however, she notices that someone has used her computer without permission. A program is sending mass emails from her email account and her report can no longer be opened. This worries Siti as the report is due very soon.

Alex learns about Siti's situation and wants to help. He manages to find one of his old book reports. Should he offer to let Siti submit his book report as her own?



In the previous chapters, we learnt how to give computers instructions by writing programs. With the growing speed, number and connectedness of computers today, it has become easier than ever to control one or more computers using programs and perform tasks which were previously thought to be too tedious or slow to accomplish.

Most programs are useful to us. However, not all programs perform tasks that are desirable or beneficial to society. For instance, in this situation, Siti is the victim of a program that used her email account without her permission and damaged her files. When we write programs, we should anticipate and guard against the possibility that these programs may be used in a harmful manner.

In this chapter, we will learn about data corruption and loss, unauthorised access to data, threats to the privacy and security of data, as well as the concept of intellectual property. We will also learn about the social and economic impact of computers in various fields and explore ethical issues related to the use of computers.



By the end of this chapter, you should be able to:

- Understand how data can be kept safe from accidental damage due to data corruption or human errors, malicious actions such as unauthorised viewing, deleting, copying and corrupting or malware.
- Understand the effects of threats to privacy and security of data from spam, spyware, cookies, phishing, pharming and unauthorised access, as well as defensive measures employed such as the use of appropriate hardware and software.
- Describe ethical issues that relate to the use of computers, public and private networks, freeware, shareware and open courseware; and the sharing of information.
- Compare the positive and negative social and economic impacts of technology in education, communication (e.g., via mobile apps and social media), finance, medicine/ healthcare, transportation and entertainment; and explain plagiarism and software piracy.

7.1 Data Corruption and Loss

After Siti returned from shelving some books, she found that her book report could no longer be opened. Like Siti, many of us store important information in our computers. Unfortunately, just as information written on a piece of paper can be torn up or misplaced, information stored in computers can also be damaged or lost.

Data corruption occurs when computer data is made unusable by errors or alterations. This kind of damage can happen during the reading, writing or transmission of data. If the corrupted data cannot be recovered or replaced, this also results in **data loss**.

Key Terms

Data corruption

When data is made unusable by errors or alterations

Data loss

When data is destroyed and cannot be recovered

7.1.1 Effects of Data Corruption and Loss

The effects of data corruption can vary depending on the amount of corrupted data and the type of data that is represented.

If the corrupted data is not needed to read other data, then only the corrupted data itself is lost. This situation is more likely if the amount of corrupted data is small. This is similar to having smudged cells in a printed table of data – only the smudged data is lost.

ID	Height	Time	Rank
221	1.54	19	4
222	1.48	20	6
223	1.79	23	5
224	Smudged header		

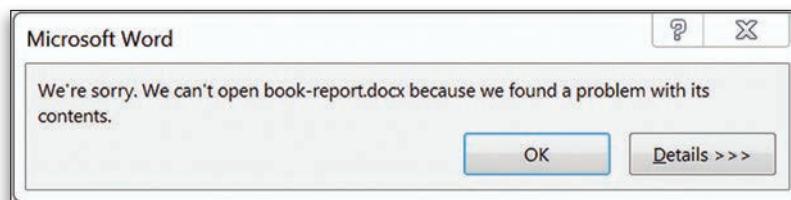
▲ **Figure 7.1** Only the data in the smudged cells is lost

On the other hand, if the corrupted data is related to other data in the computer, then both the corrupted data and its related data may be lost. This is because the corrupted data may contain information that is required to read or interpret the related data. This situation is more likely if the amount of corrupted data is large. This is similar to having smudged headers in a printed table of data. While the cells of the table are still readable, it is not possible to interpret what the contents mean, so the data stored in the table is meaningless and effectively lost.

23	9982	145	102.3
12	9983	166	115.7
16	9985	151	133.5
45	9986	176	129.9

▲ **Figure 7.2** Data in the entire table is effectively lost due to the smudged header

In Siti's case, she is probably unable to open her book report because one or more bytes in the file that are required to read the rest of the file have been corrupted. Figure 7.3 shows a typical error message that a user may get when trying to open a corrupted file in a word processor.



▲ **Figure 7.3** Example of data corruption error message

7.1.2 Causes of and Ways to Prevent Data Corruption and Loss

Data corruption and loss may be caused by human error, power or hardware failures, or malicious software, as shown in Table 7.1. Note that in all cases, making regular **backups** of data will help to prevent the loss of data in the event of data corruption.

Key Terms

Backup

Copy of data that is made in case the original is damaged or lost

Uninterruptible power supply (UPS)

Device that provides enough emergency power for a computer to properly shut down in case of a power failure

▼ **Table 7.1** Causes of and ways to prevent data corruption and loss

Cause	Explanation	Preventive measures
Human error	<ul style="list-style-type: none"> Storage devices may be accidentally damaged during transport. Multiple users working on the same file may accidentally overwrite each other's data. 	<ul style="list-style-type: none"> Make regular backups of data. Use adequate protection when transporting storage devices. Set up rules when collaborating with multiple users to prevent them from writing to the same file at the same time.
Power failure	<ul style="list-style-type: none"> If the power supply to a computer fails, data that is in the process of being written to a storage device may become corrupted and data that is stored in volatile memory but not yet written to a storage device will be lost. 	<ul style="list-style-type: none"> Make regular backups of data. Set up a backup power supply or uninterruptible power supply (UPS) so storage devices can complete any write operations in case of a power failure.

▼ **Table 7.1** Causes of and ways to prevent data corruption and loss (continued)

Cause	Explanation	Preventive measures
Hardware failure or damage	<ul style="list-style-type: none"> All magnetic, optical and solid-state storage devices can fail, either due to overuse, manufacturing defects or age. 	<ul style="list-style-type: none"> Make regular backups of data. Check storage devices regularly and replace them immediately when signs of failure are detected.
Malicious software or viruses	<ul style="list-style-type: none"> Some malicious software (see section 7.3) may purposely damage and corrupt data as a way of attacking the computer. 	<ul style="list-style-type: none"> Make regular backups of data. Avoid opening email/chat attachments or files from unknown sources. Install and configure a firewall (see section 7.2.2.2) to prevent them from spreading through the network. Install anti-virus and anti-spyware software (see section 7.3.2.1), as well as perform regular scans and updates.



Quick Check 7.1

- State three ways by which data can become corrupted.

7.2 Unauthorised Access

Besides losing her book report, Siti was also upset that someone had used her computer and email account without her permission. Like Siti, most people want to have control over how others access their data or property, such as online accounts and personal information. For instance, most people would not want to share their passwords with others but would not mind sharing less private details such as their birthdates.

Unauthorised access occurs whenever data owned by someone is used by someone else, such as an intruder or even a member of the public, without permission. Besides data loss, this can lead to many other undesirable consequences. For instance, passwords or bank account information can be used by an intruder to impersonate the owner's identity, steal his or her money, and commit fraud. An intruder can also publicise private information that can lead to unwanted attention or bullying.

Unauthorised access can occur due to one or more of the following reasons:

1. Poor authentication
2. Poor access control or authorisation
3. Poor understanding of privacy policies

These reasons are described further in the following sections.

7.2.1 Authentication

Authentication is the process of verifying the identity of a user. Authentication usually requires the user to prove his or her identity by providing evidence from one or more of the following categories:

1. Something the user knows, such as a password
2. Something the user owns, such as a mobile phone
3. Something unique that is measured from a physical part of the user, such as a thumbprint

Each category of evidence that is used for authentication is called an **authentication factor**.

Key Terms

Authentication

Process of verifying the identity of a user

Authentication factor

Category of evidence that is used for authentication: something the user knows or owns, or something that is measured from a physical part of the user

Unauthorised access

Situation where data owned by someone is used by someone else without permission

7.2.1.1 Passwords

The most common authentication method is to ask for a secret password or phrase that is known only to the user. Most computer users are probably familiar with the process of entering a password to use a computer or online account. Such passwords are usually entered together with a user name that identifies who the user is claiming to be.

Unfortunately, passwords can be a poor form of authentication if they are chosen poorly or not well-kept as a secret. Avoid using birthdates and surnames as passwords as they can be easily guessed by an intruder. Instead, use hard-to-guess passwords that are a mixture of lower-case letters, upper-case letters, numbers and symbols.

Avoid re-using passwords or leaving them unchanged for a long time as doing so makes it easier for an intruder to guess the password. Instead, use unique passwords for each computer or online account and update them at least once every 90 days.

7.2.1.2 Security Tokens (Two-Factor Authentication)

More stringent authentication systems often require evidence from more than one authentication factor. For instance, banks typically issue a device called a **security token** to users who wish to access their bank accounts online.



▲ Figure 7.4 A security token

To access his or her bank account online, the user has to confirm his or her identity by providing a secret password or personal identification number (PIN), followed by a one-time password (OTP) generated from the security token or a mobile phone that the user owns. This kind of authentication that uses evidence from both something the user knows and something the user owns is called **two-factor authentication**.

Key Terms

Security token

Device that is used specifically for authentication purposes

Two-factor authentication

Type of authentication that uses evidence from both something the user knows and something the user owns

Did you know?

An OTP is usually a string of seemingly random numeric or alphanumeric characters. For security reasons, an OTP is usually valid only for a short period of time, after which a new one has to be requested. This reduces the possibility that the OTP is obtained by an intruder between the time it is generated and when it is accepted by the bank.

OTPs may either be generated by the bank and sent securely to the user's device, or generated directly on the device using a secret algorithm that only the bank knows.

Two-factor authentication has become increasingly important in banking as large amounts of money may be at risk as a result of authenticating a user wrongly. In Singapore, most banks require two-factor authentication for their online banking services.

Two-factor authentication is stronger than using only a password as it is much more difficult for an intruder to both guess a password and steal the user's security token. This is why it is important to keep the security token in a secure location at all times and to report a missing security token as soon as possible.

However, if an OTP is sent wirelessly to the user's mobile phone, it may be intercepted and used by an intruder during the transmission process. Alternatively, if the secret algorithm used to generate OTPs is poorly chosen or accidentally revealed, an intruder may find out how to generate OTPs without needing the security token at all. Unfortunately, there is not much that a user can do to prevent such intrusion attempts.

7.2.1.3 Biometrics

Biometrics is a type of authentication that is based on the measurement of human physical characteristics. For example, biometrics is used to identify a user by fingerprint or voice. Other common characteristics used in biometrics include the face, iris, retina, and deoxyribonucleic acid (DNA).

Compared to passwords, the use of biometric identification is more secure as the physical characteristics measured are typically unique to the individual and cannot be easily replicated. Thus, biometrics can help guard against attempts to establish fraudulent identities and prevent **identity theft**.

Key Terms

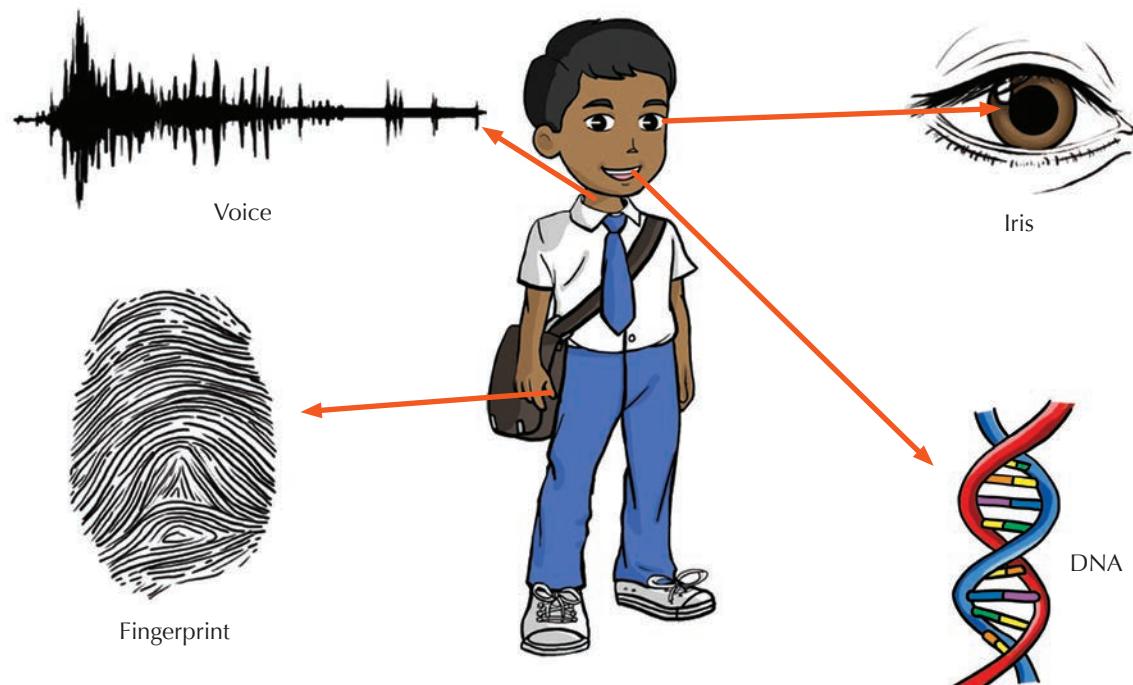
Biometrics

Type of authentication based on the measurement of human physical characteristics

Identity theft

Impersonation of another person to steal personal details such as name and identity number for fraudulent purposes

Figure 7.5 shows some of the common human physical characteristics used in biometrics.



▲ **Figure 7.5** Human physical characteristics used in biometrics

7.2.2 Access Control or Authorisation

Once a user is authenticated, the ability of a computer to control the access of data and resources by that user is called **access control** or **authorisation**. Computers provide access control through a variety of means, which are described in the following sections.

7.2.2.1 File Permissions

Most operating systems have settings to control the ability of users to view or make changes to specific files or folders. These settings are called **permissions**.

Key Terms

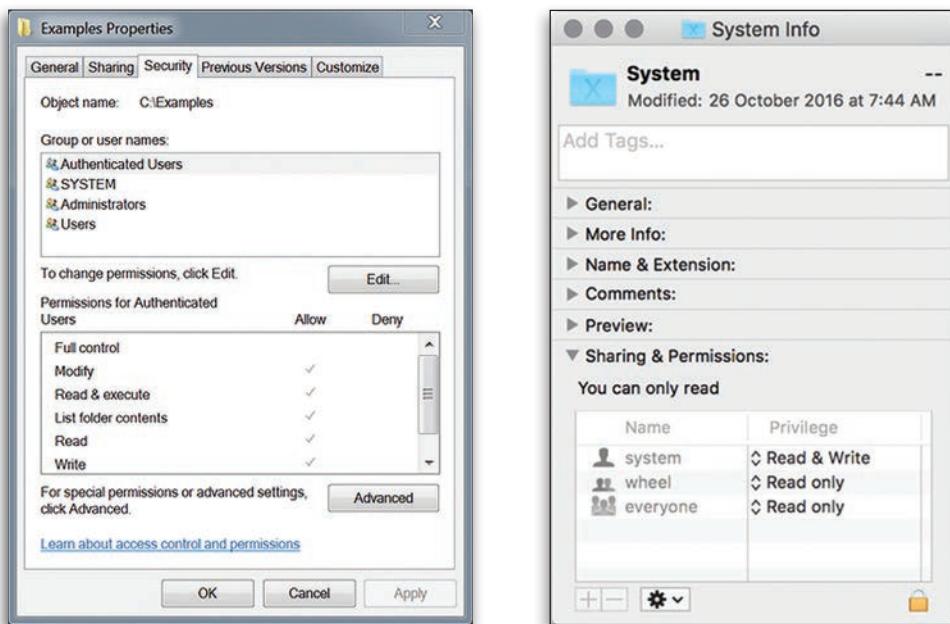
**Access control
(or authorisation)**

Ability of a computer to control a user's access to data and resources

Permissions

Settings to control the ability of users to view or make changes to the contents of a file or folder

Figure 7.6 shows an example of a file permissions dialogue box in the Windows and Mac operating systems.



▲ Figure 7.6 Viewing file permissions in the Windows (left) and Mac operating systems (right)

For example, in a computer lab, students in a group may wish to set the permissions for a group assignment file such that only members of the group can access the file. Students from other groups would thus be unable to access the file. Similarly, a teacher may set the permissions for a presentation file so that students may only read but not modify the file. This lets students view the presentation without accidentally changing its contents.

Typically, users can only change the permissions for any files or folders that they own. However, most operating systems allow for a special user called an **administrator** to override the permissions for almost any file or folder. A normal user may also be given special **administrator rights** that allow him or her to override the permissions for certain files or folders, just like an administrator.

Key Terms

Administrator

Special user who can override the permissions for almost any file or folder

Administrator rights

Ability of a user to perform tasks related to authentication and authorisation, such as creating and removing user accounts, resetting passwords and overriding file permissions

Did you know?

Besides being able to override file permissions, an administrator is usually able to perform other tasks related to authentication and authorisation, such as creating and removing user accounts or resetting passwords.

Unfortunately, managing permissions and administrative rights can be a complex task and it is possible to unintentionally grant access to a file or administrative rights to an unauthorised user. Such a user can then make use of such mistakes to gain unauthorised access to data and resources. In addition, authentication for the administrator must be especially strong, as an intruder that successfully claims to be the administrator can bypass file permissions entirely.

File permissions also do not prevent an intruder with physical access to a storage device from accessing files or folders directly without going through the operating system. To prevent such unauthorised access, it is necessary to use encryption (see section 7.2.2.3).

File permissions can be used as access control for both computers that are connected to a network as well as computers that are not connected to a network but are shared by multiple users.

7.2.2.2 Firewalls

Computers that are connected to a network are naturally more susceptible to intrusion as unauthorised access can occur even without the physical presence of an intruder. Hence, computers connected to a network usually require another layer of access control called a **firewall**. Just as how a fireproof barrier prevents fire from spreading and destroying valuable property, a firewall prevents harmful content from passing through it to reach other computers connected to the network.

A firewall can be either a device or a computer program. It works by monitoring each piece of data that is transmitted through a network. Then the data would be either blocked or allowed to pass through based on a set of rules configured by an administrator.

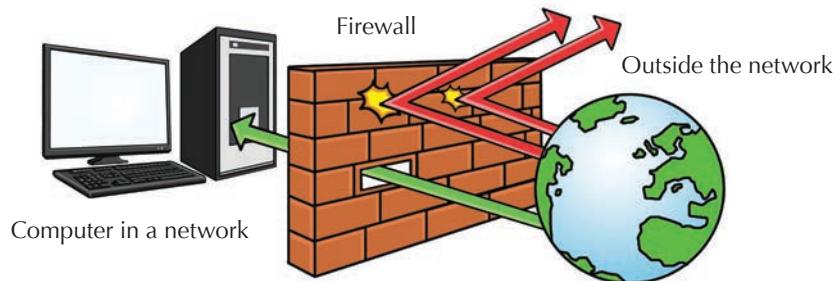
Key Terms

Firewall

Device or computer program that prevents unauthorised access to or from a private network

Traffic

Transmission of data over a network



▲ **Figure 7.7** A firewall protects a network from external threats

When properly configured, a firewall can protect the computers within a network from unauthorised access. For instance, a firewall can be configured to block the transmission of data (known as **traffic**) between any unauthorised senders and/or receivers, especially requests for data coming from anonymous users on the Internet. This prevents intruders from gaining access to the computers within a network.

As a firewall can also block traffic based on the type of **application** that is transmitting the data, it can also stop certain harmful programs from sending copies of themselves to other computers through the network.

Conversely, however, configuring a firewall correctly can be a complex task and a misconfigured firewall may unintentionally allow an intruder to gain access to computers on the network.

In general, a properly configured firewall allows for a private network, or **intranet**, to be set up such that any external traffic is blocked and only authenticated and authorised users are able to access it. Since the users on a private network are generally trusted and expected to keep information found on the network secret, there are usually fewer concerns about unauthorised access when sharing data on a private network.

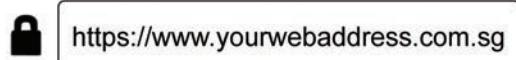
Conversely, a public network such as the Internet allows anyone to connect to it and share data. Since a public network has few or no restrictions, users need to be wary of possible security and privacy risks when accessing it (see section 7.2.3).

7.2.2.3 Encryption

Encryption is the process of encoding data so that a secret key is required to read the data. Like passwords, the secret key is usually provided as a sequence of bytes. Before the encrypted data is decoded using the secret key, it appears as random, meaningless data.

Encryption is often used to protect data from unauthorised access by allowing only authorised users to have the secret key. It can also be used in combination with file permissions so that an unauthorised user who is able to bypass file permissions would still be unable to use the accessed data without knowing the secret key.

A website uses encryption if its address starts with “`https://`” and a padlock icon appears next to its address on a web browser.



▲ **Figure 7.8** A secure website is indicated by an address that starts with “`https://`” and a padlock icon

Key Terms

Application

Software designed for users to perform specific tasks

Encryption

Process of encoding a message so that a secret key is needed to decode the data

Intranet

Private network that only authorised users within an organisation can access

7.2.3 Understanding of Privacy Policies

In the previous sections, we have discussed how unauthorised access can occur when an intruder interacts directly with the user who owns the data or the user's computer. However, unauthorised access can also occur indirectly due to the actions of third-party users or services. For instance, a user alters file permissions to let a classmate access some private files. The classmate in turn shares those files with others without the original user's knowledge. In another example, an online journaling service lets an advertising company access a user's private entries without the user's knowledge. In these cases, the problem is not caused by poor authentication or authorisation. Instead, it is a result of incorrect assumptions made by the owner of the data or improper behaviour by another authorised party.

The ability to keep specific data or resources from being known by others is called **privacy**. In many countries, organisations are required by law to publish or make available a **privacy policy** about the rules and practices they follow regarding the collection, protection and use of personal or private data provided by users. For instance, in Singapore, organisations are required by the Personal Data Protection Act (PDPA) to make their privacy policies available upon request.

Unfortunately, even as an increasing number of users share personal information such as photos and location data using online services, many of them are still unfamiliar with the relevant privacy policies or how such sharing habits may indirectly result in unauthorised access. A poor understanding of the privacy policies of these services can often result in unauthorised access.

7.2.3.1 Social Networking Sites

Social networking sites such as Facebook, Twitter, Instagram and TikTok are popular because of their wide reach and convenience. They allow users to share photographs and information quickly with their families and friends. They can also be used to help promote businesses or raise awareness of particular campaigns or causes.

However, social networking sites can also pose many privacy concerns because most users do not read or consider the repercussions of the privacy policies used by these sites regarding personal information such as status updates, notes, photographs and location data. For instance, the privacy policies for many social networking sites do not guarantee that the personal data they collect will never be exposed to unauthorised users and may even require that some personal data be shared with advertisers in order to use their sites. In this way, personal data can potentially be harvested for spam and other threats to privacy (see section 7.3) that users did not authorise directly.

Key Terms

Privacy

Ability to keep specific data or resources from being known by others

Privacy policy

Rules and practices followed by an organisation regarding the collection, protection and use of personal or private data provided by users

Users of social networking sites should be aware that once any data is digitised and uploaded to a public network such as the Internet, it can potentially remain there forever. This is because the digital data is easily copied and may be republished in ways that are no longer under the control of the original uploader.

Many users are also unaware that the privacy policies for some social networking sites do not guarantee that their personal data will be deleted from the site completely or immediately even after they close their account or remove the personal data from their account.

Personal data is sensitive and should not be shared publicly. Some companies may decide not to hire candidates after reviewing the information and photographs posted on their social networking accounts, even if this information was posted while they were still students in school.

Some good measures to prevent unauthorised access when using social networking sites:

- Read and fully understand the privacy policy of the social networking site.
- Set your sharing settings to “private” so that only people you know in real life can read your posts.
- Think twice before posting any personal photographs or information that you may feel uncomfortable sharing.
- Accept friend requests wisely. Make sure you know everyone in your friends list.

Table 7.2 summarises some ways to prevent unauthorised access.

▼ **Table 7.2** Ways to prevent unauthorised access

Category	Type	Preventive measures
Authentication	Passwords	<ul style="list-style-type: none"> • Keep passwords secret and safe. • Avoid obvious password choices such as birthdates and surnames. • Use passwords that are a mixture of lower-case letters, upper-case letters, numbers and symbols. • Avoid re-using passwords and use unique passwords for each computer or online account. • Update passwords regularly – at least once every 90 days.
	Security tokens	<ul style="list-style-type: none"> • Keep the security token stored in a secure location at all times. • Report a missing security token as soon as possible.

▼ **Table 7.2** Ways to prevent unauthorised access (continued)

Category	Type	Preventive measures
Authentication	Biometrics	<ul style="list-style-type: none"> Choose an appropriate biometric measurement that is difficult to replicate (e.g., fingerprint).
Access Control or Authorisation	File permissions	<ul style="list-style-type: none"> Take care not to accidentally grant file access or administrative rights to unauthorised users. Make authentication for the administrator especially strong (such as by using two-factor authentication) to avoid having an intruder successfully claim to be the administrator and bypass file permissions entirely. Use file permissions in combination with encryption.
	Firewalls	<ul style="list-style-type: none"> Configure the firewall properly to block traffic between any unauthorised senders and/or receivers. Configure the firewall to block traffic from certain well-known harmful programs.
	Encryption	<ul style="list-style-type: none"> Keep secret keys private and safe. Use encryption in combination with file permissions. Verify that a website's address starts with "https://" and a padlock icon appears next to its address on a web browser before sending confidential customer data such as credit card information.
Understanding of Privacy Policies	Social networking sites	<ul style="list-style-type: none"> Read and fully understand the privacy policy of the social networking site. Set your sharing settings to "private" so that only people you know in real life can read your posts. Think twice before posting any personal photographs or information that you may feel uncomfortable sharing. Accept friend requests wisely. Make sure you know everyone in your friends list.



Quick Check 7.2

1. Search online to find out more about how biometric systems are used in schools to track student attendance. State three advantages of using biometric systems over paper attendance sheets.
2. Describe three measures that an e-commerce website can take to protect customer data from unauthorised access.
3. Some websites use a feature called a CAPTCHA to perform authentication. Search online and explain briefly what a CAPTCHA tries to authenticate and how it works.

Security check

Enter both pieces of text below, separated by a space.

RtYqpp98 6a745H

Type the text in the box:

▲ Figure 7.9 Example of a CAPTCHA

7.3 Threats to Privacy and Security

Recall the situation at the beginning of this chapter. After Siti returned to her computer, she noticed that a program was sending mass emails using her account. The act of sending unwelcome mass emails to others is called **spamming**.

Spamming is one of many threats to privacy and security. Such threats usually involve **malicious software** (also known as **malware**) that is downloaded or installed on the affected computer.

Key Terms

Malicious software (or malware)

Software that is intentionally used to damage, disrupt or gain unauthorised access to a computer system

Spamming

Mass distribution of unwanted messages or advertising to email addresses which are collected from sources such as public mailing lists, social networking sites, company websites and personal blogs

7.3.1 Types of Privacy and Security Threats

Table 7.3 shows some common types of privacy and security threats.

▼ **Table 7.3** Common types of privacy and security threats

Term	Description
Cookie	<ul style="list-style-type: none"> A small file used by websites to store personal information on a user's web browser Although usually not malicious in nature, cookies are sometimes misused to collect personal information about users
Pharming	<ul style="list-style-type: none"> The interception of requests sent from a computer to a legitimate website and redirection to a fake website to steal personal data or credit card details The stolen data can then be used for unauthorised access to even more of the victim's data More difficult to detect than phishing as the fake website uses the same address as the real website
Phishing	<ul style="list-style-type: none"> The use of emails and fake websites that appear to be from reputable companies in order to steal personal information such as passwords and credit card numbers from users The stolen data can then be used for unauthorised access to even more of the victim's data
Spamming	<ul style="list-style-type: none"> The mass distribution of unwanted messages or advertising to email addresses which are collected from sources such as public mailing lists, social networking sites, company websites and personal blogs
Spyware	<ul style="list-style-type: none"> A hidden program that secretly collects personal information about its users and transmits this information to attackers without the users' knowledge The collected data can then be used for unauthorised access to even more of the victim's data
Trojan horse	<ul style="list-style-type: none"> A computer program that pretends to be a harmless file or useful application Once a Trojan horse is run, it does something harmful such as giving intruders unauthorised access to the computer instead

▼ **Table 7.3** Common types of privacy and security threats (continued)

Term	Description
Unauthorised access	<ul style="list-style-type: none"> • The use of data owned by someone by someone else, such as an intruder, without permission (see section 7.2) • May arise as a result of other privacy and security threats that bypass authentication and authorisation by exploiting software bugs or tricking the user into performing harmful actions • Unauthorised access to passwords or private information can lead to identity theft, stolen money or public embarrassment
Virus	<ul style="list-style-type: none"> • A computer program that attaches itself to a normally harmless program and modifies it • When the modified program is run by a user, the virus attaches copies of itself to any other programs it can find, thus “infecting” them
Worm	<ul style="list-style-type: none"> • A computer program that runs automatically and attempts to spread by sending copies of itself to other computers • Unlike a virus, a worm does not need to attach itself to an existing program

7.3.2 Defensive Measures Against Privacy and Security Threats

The loss of privacy and/or security can be highly damaging, thus it is best to take preventive measures to avoid becoming a victim.

7.3.2.1 Install Anti-Virus and Anti-Spyware Programs

Viruses, worms, spyware and Trojan horses are all examples of malware that need to run on a user's computer in order to perform their respective attacks. **Anti-virus** and **anti-spyware** programs can be used to:

- detect when malware is about to be run and stop it from running;
- detect malware that is already running and try to stop it; or
- scan the user's storage and email to detect and remove malware; if a program has been infected by a virus, it may also try to restore the original program.

While powerful, most anti-virus and anti-spyware programs rely on a list of **signatures**, or unique evidence, for each known version of malware. This list needs to be updated regularly to ensure that the protection provided by these programs continues to be effective against new malware. For convenience, most anti-virus and anti-spyware programs can update this list automatically through the Internet.

Some especially devious Trojan horses may appear to be anti-virus and anti-spyware programs. To be safe, only anti-virus and anti-spyware programs provided by reputable companies or websites, or as part of the computer's **operating system** should be trusted.

7.3.2.2 Update Software Regularly

Viruses, spyware and Trojan horses typically require human interaction to start running, but worms can run automatically. They typically do this by exploiting bugs or unintended behaviour in otherwise legitimate programs that are already running on the computers. For instance, a flawed web browser may have a bug that allows websites to run malicious programs without the user's knowledge.

To avoid such situations, it is important to update software regularly so that bugs that were discovered since the last update can be fixed. This is especially important for software that is used to interact with public networks such as the Internet, as data from public networks is more likely to be malicious and designed to take advantage of known bugs.

Key Terms

Anti-spyware

Software to detect, remove and stop spyware and other malware from running

Anti-virus

Software to detect, remove and stop viruses and other malware from running

Operating system

Software designed to support a computer's basic functions

Signature (malware)

Unique evidence that is used to detect a known version of some malicious software

7.3.2.3 Identify Phishing

Phishing is the use of emails and fake websites that appear to be from reputable companies in order to steal personal information such as passwords and credit card numbers from users. Such emails should be ignored and deleted. Figure 7.10 shows an example of a phishing email that appears to be from an online game website.



▲ Figure 7.10 Example of a phishing email

Some tell-tale signs to identify phishing emails:

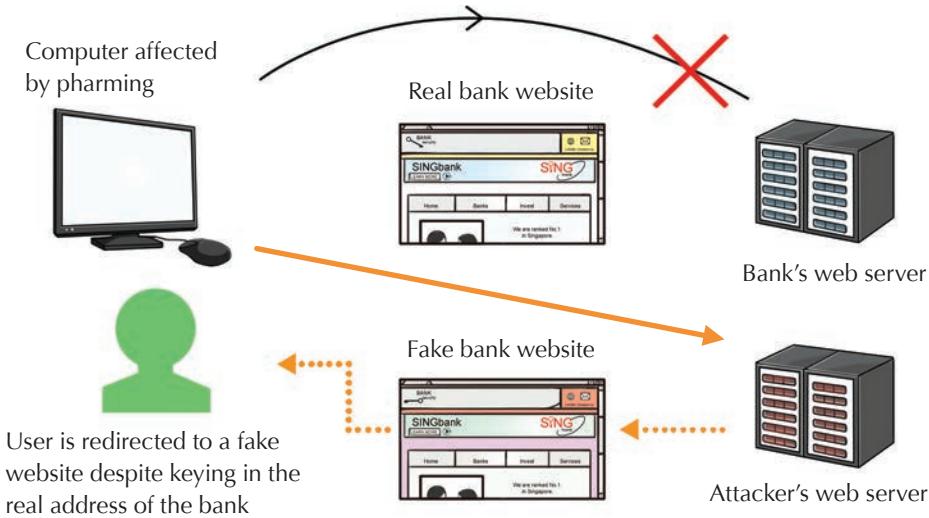
- The email claims to be from a company or bank and asks for personal data or confidential information. Most companies or banks will never ask for such information via email. When in doubt, call the company or bank to verify.
- The email uses a generic greeting such as “Dear Customer” or “Dear User”. This is a sign that the email was sent automatically and not by a person.
- The email has inaccurate logos or grammatical and spelling errors that suggest it is not from a legitimate source.
- The email seems to come from a fake sender or from an address or contact that does not match the supposed source of the email. Fake senders usually use email accounts from free providers such as Gmail and Yahoo or from sites made up of random letters or common misspellings of legitimate sites.
- The email contains hyperlinks with destinations that do not match what the hyperlink text says or are otherwise unexpected. To check the actual destination of a hyperlink, place your mouse cursor over the hyperlink and its destination will usually appear either as a pop-up or on the status bar.
- The tone of the email or chat is excessively urgent or threatening. Phishers often use such scare tactics to make victims act before they can think through their actions properly. Alternatively, the email may promise an offer that seems too good to be true. This is to tempt victims into revealing their personal information.

Did you know?

Phishing is a play on the word “fishing”. The idea is that bait is thrown out with the hope that while most fish might ignore the bait, some will be tempted into biting. Similarly, phishing emails are sent to many recipients in the hope that some recipients will eventually fall for the scam and give away their personal data.

7.3.2.4 Identify Pharming

Pharming is a more serious form of phishing. In pharming, the attacker will attempt to intercept requests sent from a computer to a legitimate website and redirect the user to a fake website to steal personal data or credit card details.



▲ **Figure 7.11** Pharming is an attempt to redirect website traffic to a fake website

For example, when victims of pharming enter the web address of their bank into a web browser, they would be presented with a website that appears to be genuine but is actually provided by the attacker's web server. When they try to log in to the fake website, their user names, passwords and account details would be recorded by the attacker, who can then use these details to access the victims' bank account on the bank's actual website.

For pharming to be successful, the attacker must either have malware running on the victim's computer or have taken control of a network device such as a router or server. This can occur as the software that runs on such devices is also susceptible to bugs.

Unlike phishing, it can be difficult to spot pharming attacks as everything seems to be normal while the attack is taking place. Nevertheless, there are some measures that can be taken to protect against pharming:

- Ensure that encryption (see section 7.2.2.3) is used when submitting credit card or other sensitive information via the Internet. Clicking on the padlock icon that appears beside the address bar usually displays more information to verify that the recipient is legitimate and that encryption is working correctly.
- Regularly check bank, debit/credit card and other statements to ensure all transactions are legitimate.
- Regularly update web browsers and the software running on network hardware so that all known bugs are fixed.
- Make sure that two-factor authentication (see section 7.2.1.2) is enabled for all bank transactions. This ensures that even if an attacker is able to access the bank account, no unauthorised transactions can take place as the attacker would not be able to provide the required OTP.

7.3.2.5 Manage Spam

Most Internet users with email accounts should be familiar with the large number of unwanted emails that may be received daily, also known as spam. Due to the ease with which emails can be sent, it is almost impossible to avoid spam entirely. Nonetheless, there are some measures that can be taken to reduce the impact of spam on email accounts:

- Avoid giving your email address to unfamiliar contacts or untrusted websites. If it really is necessary to provide one, some sites can help generate a temporary email address that will automatically expire after a short period of time. Alternatively, set up and use a secondary email address that is dedicated to unimportant emails.
- Read and understand the privacy policy (see section 7.2.3) of any website, trusted or untrusted, that requests an email address before providing it. Some websites, even those run by reputable companies, share email addresses with advertisers who may be guilty of spamming. The way such websites claim to use email addresses is usually communicated in full via the privacy policy.
- Look out for options to turn off email updates or participation in mailing lists when signing up for or changing the settings of an online account. Many sites leave such options turned on by default and additional effort is needed to turn them off.
- Most email services have a filter feature that allow users to block specific senders or to only allow emails from specific senders through. Some filtering systems also have advanced spam detection algorithms that can be trained by having the user identify examples of spam that the filter is not yet able to detect. This lets the filter become more effective in detecting spam over time.

7.3.2.6 Manage Cookies

Cookies are small pieces of data stored by the web browser when a user visits a website. Each time a user visits a website that uses cookies, the web browser checks whether it has a relevant cookie and if so, it sends the information contained in that cookie back to the website. The website is thus aware that the user has visited before and, in some cases, will customise what appears on the page for the user. On the other hand, if no relevant cookie is found, the website may request for a new cookie to be created.

In general, cookies are not malicious and are needed to keep track of authentication information for identifying which user is currently logged in. However, they can also be used to keep track of user movements and preferences within the website, such as which pages were most recently visited by the user or how the user wants the site to be presented. Even worse, advertising companies with advertisements on multiple websites can also use cookies to keep track of users as they move from one website to another.

For users who want to keep their movement on the Internet private, the use of cookies in this manner is undesirable. Thankfully, most web browsers have settings that allow users to manually delete or prevent cookies from being created by untrusted websites. These settings can also be configured to disable cookies or allow only selected websites to use cookies.

To fully determine what information is being tracked by a website's cookies and how that information will be used, it is necessary to read and understand the website's privacy policy (see section 7.2.3).



Quick Check 7.3

1. You receive an unexpected email that advertises a free program. The advertisement has links to download the program or to learn more about the product. The program seems useful but you need more information. What should you do?
 - A Click on the link to download and run the program to try it out.
 - B Click on the link to learn more about the product.
 - C Close the email and search online for more information.
 - D Reply to the email and ask for more information.
2. Attackers may use spyware to monitor what you do on your computer. Which one of the following is *not* likely to be a sign that your computer is affected by spyware?
 - A The light next to your computer's camera cannot be turned off.
 - B Your computer's network is disabled and there is no Internet access.
 - C Your computer seems to function more slowly than usual when it is online.
 - D Your web browser automatically visits an unfamiliar website at regular intervals.
3. State whether each of the following is a measure that can be taken to avoid spam:
 - a) Set up an email filter to block out unwanted emails.
 - b) Change web browsers regularly.
 - c) Use the same email address for schoolwork and online games.
 - d) Configure social networking accounts to keep email address information private.
 - e) Provide email addresses to a website only after checking its privacy policy to ensure that personal information will not be shared.
 - f) Open email attachments as soon as possible.
4. A movie company uses the mobile phone numbers and email addresses of its customers to send electronic tickets and to promote upcoming movies. As a result of unauthorised access, an attacker manages to access the company's collection of mobile phone numbers and email addresses and releases this data to the public.

Suggest how such an incident may negatively affect the company and its customers.
5. Siti receives an email message from a stranger claiming that she has won a cash prize from a lucky draw. The stranger wants to know her bank account details in order to transfer the prize money to her. Suggest briefly what Siti should do.

7.4 Intellectual Property

In the situation at the beginning of the chapter, Alex was considering whether he should let Siti submit one of his old book reports as her own. Unfortunately, since the book report is meant to represent each student's individual work, it would be dishonest for Siti to claim Alex's work as her own.

Creative works such as Alex's book report, songs or computer programs take effort to create and are valuable as they benefit their owners. The owner of a song, for instance, can benefit from feeling entertained by listening to the song. This is similar to how physical property such as a chair or a house requires materials to produce and also benefit their owners. However, unlike physical property, some works are creations of the mind and can exist purely as data with no physical form. Such creative works are called **intellectual property**.

The owner of physical property such as a chair or a house can easily use locks or chains to prevent the property from being stolen or used by others without permission. However, it is not as easy to do the same for intellectual property such as music or computer programs. For instance, while it is easy to copy music files or programs from one computer to another, doing so may be illegal if the owner of this intellectual property disallows it. The legal right of owners to control the use and distribution of their intellectual property is called **copyright**. In particular, copyright protects programmers against theft, misuse and unauthorised copying of their software.

However, some owners may intentionally want to give away their intellectual property or to openly declare that they have granted permission for use of the intellectual property under certain conditions. An official description of activities that are authorised or forbidden by the owner of intellectual property is called a **licence**. While it is possible to write a licence from scratch, most intellectual property owners typically re-use existing licences. A wide range of licences with different levels of restrictiveness are available. A licence can also refer to the authorisation that is granted by the owner of the intellectual property after a fee is paid or certain conditions satisfied.

Key Terms

Copyright

The legal right of owners to control the use and distribution of their intellectual property

Intellectual property

Creations of the mind that have value but can exist purely as data with no physical form

Licence

Official description of activities that are authorised or forbidden by the owner of intellectual property

7.4.1 Types of Software Licences

It is important to understand the different types of software licences in order to avoid infringing copyright laws.

7.4.1.1 Public Domain

While it is not technically a licence, **public domain software** refers to software where the legal protections that are typically granted to intellectual property have either expired, been surrendered or are simply inapplicable. For instance, the first-ever program for running a web server was released in 1990 under the public domain. Some authors purposely release software under the public domain to benefit the public, while others are simply so old that the legal protections of copyright have expired.

Public domain software is not protected by copyright and anyone can legally copy, modify and distribute public domain software. Public domain software also need not always come with source code, although most do.

7.4.1.2 Free and Open-Source

Free and open-source software (FOSS) refers to software

where users are given freedom to change, copy, study and share the software. The “free” in “free software” refers to the freedom to use and not the price. Like public domain software, anyone can legally copy, modify and distribute FOSS. However, unlike public domain software, the software is still protected under copyright and the copyright owners may use this protection to require that the software must always be distributed with source code, that attribution to the original authors must always be provided or that any modifications must use a similar licence if they are also distributed.

Key Terms

Free and open-source software (FOSS)

Software where users are given freedom to change, copy, study and share the software and its source code

Public domain software

Software where the legal protections that are typically granted to intellectual property have either expired, been surrendered or are simply inapplicable

Did you know?

There are many FOSS licences created by various institutions with varying levels of restrictiveness, such as the Apache License, the Berkeley Software Distribution (BSD) License, the Massachusetts Institute of Technology (MIT) License, the GNU Lesser General Public License (LGPL), and the GNU General Public License (GPL). Knowledge of these licences and the differences between them is not covered in this textbook.

Besides software, other kinds of copyrighted works such as books, photographs and music can be licensed in a similar manner using Creative Commons (CC) licences. Note, however, that while CC licences are similar to FOSS licences in granting users the freedom to copy, modify and distribute copyrighted works, CC licences are not designed for software and should not be used for this purpose. On the other hand, CC licences are suitable for licensing *content* that is delivered using software. For instance, higher-education course materials such as videos and notes created by universities and distributed for free on the Internet, also known as **open courseware**, often use CC licences.

The Linux operating system is an example of FOSS licensed under the GPL, in which the source code is freely available for modification. The Python interpreter used in Chapter 4 of this textbook is also distributed under an FOSS license.

7.4.1.3 Proprietary Software

Proprietary software usually refers to commercial software for which most of the legal protections under copyright are retained. In other words, unlike FOSS, it is usually *not* legal to copy, modify or distribute proprietary software. The terms and conditions for which the proprietary software may or may not be used under copyright protection law are usually communicated to users through an End User License Agreement (EULA) contract that the user must accept to use the software. In addition to these restrictions, the source code for proprietary software is usually kept secret.

While these restrictions may seem severe compared to FOSS licences, it is important to remember that software is a form of intellectual property and it is the right of the owner to be compensated (usually by selling licences) for use of the property.

The Windows operating system is an example of proprietary software where unauthorised copying is illegal and the majority of source code is kept secret.

7.4.1.4 Freeware and Shareware

Freeware is similar to proprietary software except that it is available for use at no cost. Some freeware are so-called “lite” versions of proprietary software, which allow users to try a limited version of the software while promoting the full version. Similarly, **shareware** is demonstration software that is distributed for free but for a specific evaluation period only. After the evaluation period, the program expires and the user can no longer access the program unless the user pays a registration fee.

Key Terms

Freeware

Proprietary software that is available for use at no cost

Open courseware

Higher-education course materials such as videos and notes created by universities and distributed for free on the Internet

Proprietary software

Commercial software for which most of the legal protections under copyright are retained

Shareware

Demonstration software that is distributed for free but for a specific evaluation period only

Like proprietary software, the source code for freeware and shareware is usually kept secret and modifying the software is usually illegal. However, unlike proprietary software, it may be legal to copy and distribute freeware and shareware. Adobe Reader and Skype are examples of freeware, while Camtasia Studio and WinRAR are examples of shareware.

7.4.2 Software Piracy

The crime of copying, distributing and/or using proprietary software illegally is called **software piracy**. Despite being illegal, software piracy is still prevalent and can take place in many forms. For instance, in 2015, a private school in Singapore and its director were fined \$38,000 for running proprietary training programs without a licence. Installing multiple copies of proprietary software without purchasing additional licences or sharing proprietary software with unlicensed users is considered software piracy and can result in similar legal repercussions. Software piracy causes significant loss of revenue for the owners of intellectual property, which in turn leads to higher prices for legitimate buyers.

Some users may unintentionally install pirated software without realising it. However, the distribution and installation of pirated software usually involves some unusual steps that should warn the user that something illegal is taking place. For instance, some software pirates may distribute the installation files through illegitimate websites or use programs called **cracks** to modify the proprietary software so that the software cannot detect that it is being used illegally. Pirated software for mobile phones may require users to perform complex steps to modify their phone before installation or to bypass the built-in software store for installation.

If you suspect that you have unintentionally installed pirated software on your computer or device, have a teacher or parent help you check whether the software you have installed is legal. In most cases, pirated software can be replaced with an alternative open-source program that uses a less restrictive licence and can be downloaded legally for free.

7.4.3 Copyright Infringement

Software piracy is an example of **copyright infringement**, which is the use or distribution of copyrighted work without the permission of the copyright owner. While software piracy is specific to software, copyright infringement can occur for any copyrighted materials, such as text or pictures that are on the Internet, and it is equivalent to theft.

Key Terms

Copyright infringement

Use or distribution of copyrighted work without the permission of the copyright owner

Crack

Program that modifies proprietary software so that the software cannot detect that it is being used illegally

Software piracy

Crime of copying, distributing and/or using proprietary software in a manner that is not permitted by its licence

In particular, users may even copy, modify or distribute copyrighted materials from the Internet without realising that they have committed copyright infringement. Such infringement is illegal even if the act of copying is unintentional. Students using materials from the Internet for schoolwork should adhere to the following guidelines to avoid committing copyright infringement:

1. Check and follow the website's terms and conditions.

Many websites state the terms and conditions for using the site's content, such as whether copying, downloading or hyperlinking to any materials is permitted. In some cases, the content may be under a licence, such as a CC licence, where such permissions are explicitly granted or denied. Otherwise, if no notice is present, it may be necessary to write to the copyright owners and seek written permission for use of the materials.

2. Limit reproduction of a copyrighted work to 10% (for educational purposes only).

For the special case of using copyrighted materials for research, study or other educational purposes, students may reproduce no more than 10% of the electronic edition of a work. The reproduced work should not be modified and must be properly cited to avoid plagiarism (see section 7.4.4).

3. Consider using public domain material instead.

Like public domain software, some works are released to the public domain on purpose or are so old that their copyrights have expired. Note that although these materials can be copied without the need to seek permission, it is still necessary to acknowledge or cite the authors appropriately to avoid plagiarism (see section 7.4.4).

7.4.4 Plagiarism

Related to copyright infringement is **plagiarism**. Plagiarism is the act of passing off someone else's original work as your own. In other words, it is the act of claiming to be the author of a piece of work without providing proper credit or attribution to the actual author. Unlike copyright infringement, plagiarism may not always be illegal, but it is nevertheless an act of dishonesty and is usually a punishable offence when detected in schoolwork.

Key Term

Plagiarism

Passing off someone else's original work as one's own

For example, in the situation at the beginning of this chapter, suppose Siti agrees to use one of Alex's old book reports as her own. Even if Alex gives Siti permission to do this, it is still plagiarism as the book report is meant to represent a student's individual work and Siti is falsely claiming to be the author of Alex's book report. This is *not* an example of copyright infringement, however, as the author Alex explicitly gave Siti permission to copy the work.

On the other hand, some cases of copyright infringement are also cases of plagiarism. For example, a student who copies more than 10% of copyrighted text and pictures from a website without the owner's permission and submits the copied materials in a school assignment without acknowledging the actual author is committing both copyright infringement and plagiarism. Such acts are illegal and dishonest.

Did you know?

Published books and articles that use materials from other authors acknowledge and provide credit to these authors using statements known as **citations**. Citations can also be used for schoolwork to avoid plagiarism and give proper credit to the original authors of any reproduced work. There are multiple standards for writing citations that differ by the relevant fields of study and preferred conventions. Knowledge of these citation standards will not be covered in this textbook.

Key Term

Citation

Statement to acknowledge and provide credit to the author(s) of reproduced materials in published books and articles



Quick Check 7.4

1. For each of the following scenarios, state whether the person described is guilty of copyright infringement, plagiarism, both or neither.
 - a) Alice takes her friend's artwork and submits it for a contest as her own artwork without her friend's permission.
 - b) Bob installs a computer game on two different computers even though the licence he purchased allows for one installation only.
 - c) Cindy purchases an educational licence for a spreadsheet program and installs the program on her home laptop for school projects.
 - d) Dan downloads some public domain source code and submits it as his own for a homework assignment.

7.5 Impact of Technology on Everyday Life

So far in this chapter, we have learnt about the effects of data corruption and loss, the various threats to privacy and security of data, as well as the legal and ethical issues surrounding intellectual property. These topics have risen to prominence in recent years due to the rapid rate at which technology has influenced our daily lives.

At this point, it is useful to look at the bigger picture and evaluate the overall impact of technology. Since the World Wide Web's invention in 1989 and the widespread availability of smart phones, technology has completely changed the way we work, live and play.

Recently, with advancements in computing power and means of collecting data, computer scientists have also developed algorithms that are successful in automating complex tasks such as classifying images, making shopping recommendations and even generating works of art.

Such algorithms and methods that allow a computer to perform complex tasks without constant human guidance and improve its performance as more data is collected form the basis of **artificial intelligence (AI)**. The emergence of AI will change our lives even further as companies and governments find new ways to collect data and automate processes that were previously done manually. These changes are also likely to bring about their own set of legal and ethical concerns.

Key Term

Artificial intelligence (AI)

Ability of a computer to perform complex tasks without constant human guidance and improve its performance as more data is collected

Table 7.4 provides a brief overview of the social and economic impact of technology on various areas of life, as well as related ethical issues.

▼ **Table 7.4** Impact of technology on various areas of life

Area	Impact/Issues
Communication	<p>Social impact</p> <ul style="list-style-type: none"> On the positive side, the Internet has enabled diverse cultures to interact and share ideas with each other. Social networking sites have also allowed users to remain connected with friends, family and colleagues even over long distances. AI has also made it possible for anyone to automatically transcribe and translate speech into different languages with remarkable speed and accuracy. On the negative side, some people use the Internet to reinforce their existing opinions or to spread rumours and misinformation. This is worsened by the use of AI by some news and social networking sites to promote content that the reader is likely to be interested in, since most sites promote content based on how much they are able to elicit responses and not by their level of accuracy. <p>Economic impact</p> <ul style="list-style-type: none"> The rise of smart phones has led to an increased focus on mobile devices and mobile applications in the computing industry. The rise of social media has led to the increased use of social media for marketing purposes and has helped businesses to better understand buying habits and consumer needs by analysing social media posts. Improvements in communications technology have also reduced business costs through the use of cheap and effective video conferencing calls in place of face-to-face meetings. <p>Ethical issues</p> <ul style="list-style-type: none"> Should some kinds of false information on the Internet be blocked or would this be taking censorship too far? Is it right to collect and analyse social media posts in ways that the original authors may not have intended?

▼ Table 7.4 Impact of technology on various areas of life (continued)

Area	Impact/Issues
Education	<p>Social impact</p> <ul style="list-style-type: none"> On the positive side, technology has provided better videos, simulations and collaboration tools for lessons to be more engaging and relevant to students, both inside and outside the classroom. The open sharing of information on the Internet has made education accessible to all, even learners from disadvantaged backgrounds. AI has also made it possible to provide personalised learning and testing for thousands of students without the need for excessive manual effort. On the negative side, technology has made it more challenging for educators as students' attention spans are getting shorter due to the technology and social media that they are used to today. <p>Economic impact</p> <ul style="list-style-type: none"> The rise of open courseware has led to a large number of for-profit and non-profit organisations that offer customised lessons on a variety of topics for self-study online. Many educational institutions have also invested heavily in the materials and infrastructure needed to conduct lessons over the Internet. Some traditional educators find that they need to retrain and pick up new skills to better utilise these new technologies for education. <p>Ethical issues</p> <ul style="list-style-type: none"> Should students be allowed to use mobile phones in school? Can education technology ever be more effective than a good teacher?

▼ **Table 7.4** Impact of technology on various areas of life (continued)

Area	Impact/Issues
Finance	<p>Social impact</p> <ul style="list-style-type: none"> On the positive side, the rise of financial technology has enabled consumers to spend, borrow, invest and save money through low-cost and easy-to-use mobile and web applications. There is no longer a need to perform such transactions in person. Individuals have also become better-educated on how to make smart financial decisions using information that is freely available on the Internet. In retail, many consumers also enjoy the convenience of making payments via biometric authentication (e.g., face or fingerprint recognition) that is only possible with the use of AI. On the negative side, threats to the privacy and security of data as well as the ease of obtaining false information on the Internet has made some people more vulnerable to financial scams and other get-rich-quick schemes. <p>Economic impact</p> <ul style="list-style-type: none"> Financial technology is currently an area of growth in the financial industry. Numerous companies have been started to make financial services more efficient for both individuals and businesses. These companies typically use technology and software to reduce the time, cost and effort needed for payments, investments, fundraising, trading, and/or data analytics for both businesses and individuals. With the evolution of technology, the time needed to perform a financial trade has decreased from seconds down to mere microseconds. This has led to the rise of algorithmic trading, which is the study and refinement of algorithms to make trading decisions at speeds not possible by a normal human being. Many banks also use AI to analyse transaction data and automatically identify unusual spending patterns or money transfers. This helps them to automatically detect and prevent instances of financial fraud without the need for manual intervention. <p>Ethical issues</p> <ul style="list-style-type: none"> Should financial technology be limited to protect more vulnerable users? Is it safe or acceptable to have financial markets controlled by secret algorithms instead of humans?

▼ **Table 7.4** Impact of technology on various areas of life (continued)

Area	Impact/Issues
Healthcare	<p>Social impact</p> <ul style="list-style-type: none"> On the positive side, technology has enabled telemedicine, which is the use of video conferencing and other technology, for doctors to provide medical consultations and diagnoses over the Internet or applications in mobile devices. This gives patients who are located in remote places or have limited mobility better access to healthcare. By analysing medical data, AI can automatically identify warning signs of possible health problems and provide doctors with more accurate diagnoses. On the negative side, some patients find the use of robots and other technology in healthcare impersonal and mistrust the ability of machines to provide proper healthcare. Other patients may misuse information from the Internet and make potentially dangerous decisions based on incorrect diagnoses. Patients may also be uncomfortable with the collection of medical data necessary to improve the performance of healthcare-related AI. <p>Economic impact</p> <ul style="list-style-type: none"> Technology has created new areas of growth in the healthcare industry, such as the provision of telemedicine solutions to existing healthcare businesses. In particular, many of these solutions provide a way for patients to securely transfer potentially sensitive medical information over the Internet. There is also an increased focus in automating healthcare processes through the use of robots to dispense medicine and other more menial tasks. This may in turn cause such jobs to disappear from the job market. The rise of 3D-printing technology has also opened up new opportunities in the building and customisation of prosthetic limbs, hearing aids and dental fixtures. <p>Ethical issues</p> <ul style="list-style-type: none"> Is it acceptable for robots to replace humans in providing certain kinds of healthcare? Is it acceptable to transfer private medical information over the Internet?

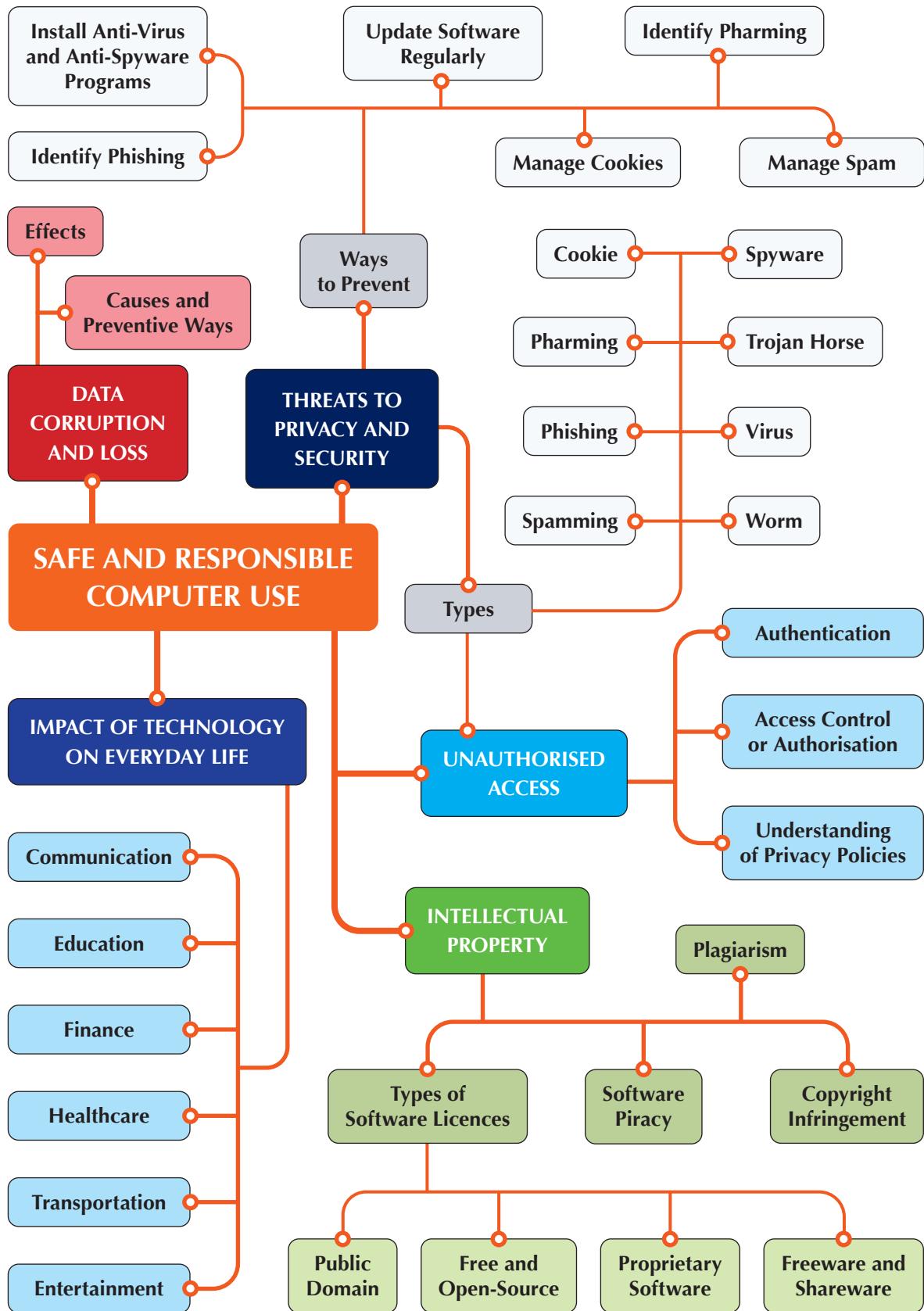
▼ **Table 7.4** Impact of technology on various areas of life (continued)

Area	Impact/Issues
Transportation	<p>Social impact</p> <ul style="list-style-type: none"> On the positive side, transport has become less stressful and more predictable for travellers due to the availability of detailed maps as well as real-time information on bus frequencies, traffic congestion levels and street-level photographs of neighbourhoods around the world. All this information is also available at low cost through popular mobile devices. Location data from such mobile devices may also be collected by AI to improve the accuracy and relevance of any map or traffic data that is displayed. On the negative side, some people are uncomfortable with how pictures and information about themselves or their home may be used by mapping companies without their permission in order to build more accurate maps for travellers. This is especially true with regards to the collection of location data from mobile devices as this data can reveal personal details such as home and work addresses that users may wish to keep private. <p>Economic impact</p> <ul style="list-style-type: none"> The rise of self-driving vehicles with the use of AI to recognise and adapt to road conditions is likely to open new areas of growth in the travel industry. Singapore is one of the first countries where self-driving cars are being tested, and if successful, the technology will likely revolutionise the motor industry. There are also multiple new companies that offer on-demand rides via mobile applications. These developments have led to sweeping changes in the taxi service industry as well as employment opportunities for taxi drivers. Mapping technology is also another area of growth with an increased focus on making 3D maps and geospatial data more accessible and useful to travellers. <p>Ethical issues</p> <ul style="list-style-type: none"> Is it acceptable for mapping companies to drive through neighbourhoods and take pictures to put them online? Is the sharing of personal transport preferences worth the conveniences offered by AI?

▼ **Table 7.4** Impact of technology on various areas of life (continued)

Area	Impact/Issues
Entertainment	<p>Social impact</p> <ul style="list-style-type: none"> On the positive side, technology has enabled more exciting and engaging forms of entertainment. Many computer games have active online communities and mobile games have even managed to bring participants together in the real world through in-game incentives to meet or team up. AI has also made it possible to provide more accurate recommendations for consumers based on collected data of their previous choices of entertainment. On the negative side, some people may be addicted to computer games or social networking sites. There is an increasing concern that such technology is causing people to become deficient in real-life social skills or abandon their responsibilities. AI has also made it possible for anyone to create doctored images and videos that appear remarkably convincing to the average viewer. While such images and videos have been used mostly for entertainment, they can also be used to cause public alarm or spread damaging falsehoods. <p>Economic impact</p> <ul style="list-style-type: none"> Games, animation and media are areas of strong growth in the entertainment industry with new opportunities being opened up by the rise of high-quality virtual reality, augmented reality and motion-tracking technology. Many businesses are also using monitoring technology and strategies from game design to provide rewards and incentives for work-related achievements. <p>Ethical issues</p> <ul style="list-style-type: none"> Is it right for game companies to make money off addicted players? Is it right for businesses to use technology to monitor employees more extensively?

Chapter Summary





Review Questions

1. Recently, the customers of a bank received an email with “Transaction Advice” as the subject. The email contained a hyperlink to a fake website which looked like the bank’s real website. Some customers did not pay attention to the address of the hyperlink and tried to log in as usual. As a result, their personal information was stolen.
 - a) Explain the difference between phishing and pharming.
 - b) State whether phishing or pharming occurred in this case.
 - c) Describe two ways how the bank customers could have avoided getting their personal data stolen.

2. For each of the following threats, identify and explain whether unauthorised access is likely to occur. For each threat where unauthorised access is likely to occur, suggest one preventive measure that can be taken.
 - a) Cookies
 - b) Spam
 - c) Trojan horses
 - d) Viruses

3. Across the world, the music, television and film industries have been hit hard by acts of piracy.
 - a) Describe two economic effects on the music and film industries if piracy is not curbed.
 - b) Siti enjoys watching a television series that is only available on an online video site. After some research, she realises that she can download the television series onto her computer hard drive so that she can watch the videos without going online. Explain whether doing so is considered to be copyright infringement.
 - c) Alex finds a photograph of a wall mural on a website. He notices that the photograph was labelled with the icon shown in Figure 7.12.

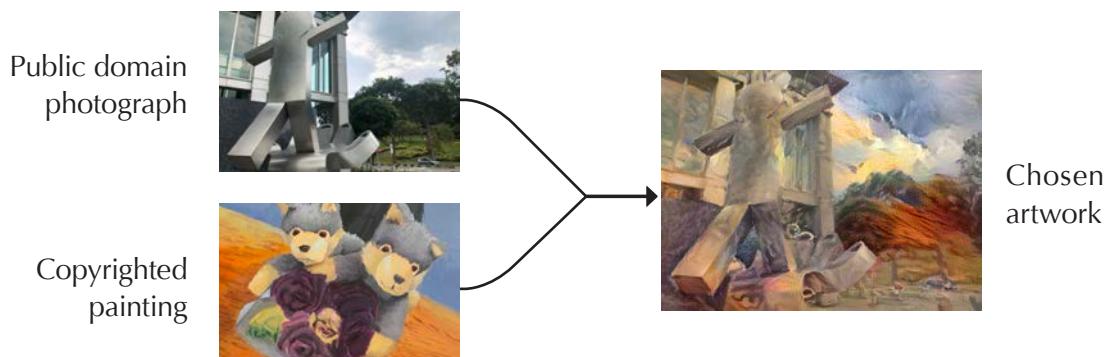


▲ **Figure 7.12** Icon used in a photograph’s label

Search online and explain what this icon represents.

- d) Describe a situation where music can be downloaded legally without copyright infringement.

4. A company conducts a digital art competition where individual members of the public are invited to upload their original artworks via a website. The uploaded artworks are automatically stored on the website's server.
- a) Suggest two things that the company should do to prevent the uploaded artworks from being lost due to sudden hardware or power failure.
 - b) After submissions are closed, the competition's judges are surprised to discover that the artworks submitted by two different members of the public are identical. Identify the act of dishonesty that is likely to have taken place.
 - c) After choosing an artwork to win the competition, the judges are surprised again when it is revealed that the chosen artwork was generated using a public domain artificial intelligence (AI) program. Its creator reveals that the chosen artwork is actually the automated result of combining content extracted from a public domain photograph with textures extracted from a copyrighted painting. More precisely, the resulting artwork depicts the same objects as in the photograph but is formed using colours and textures copied from the painting. Figure 7.13 shows the chosen artwork and the two source images that were used to generate it.



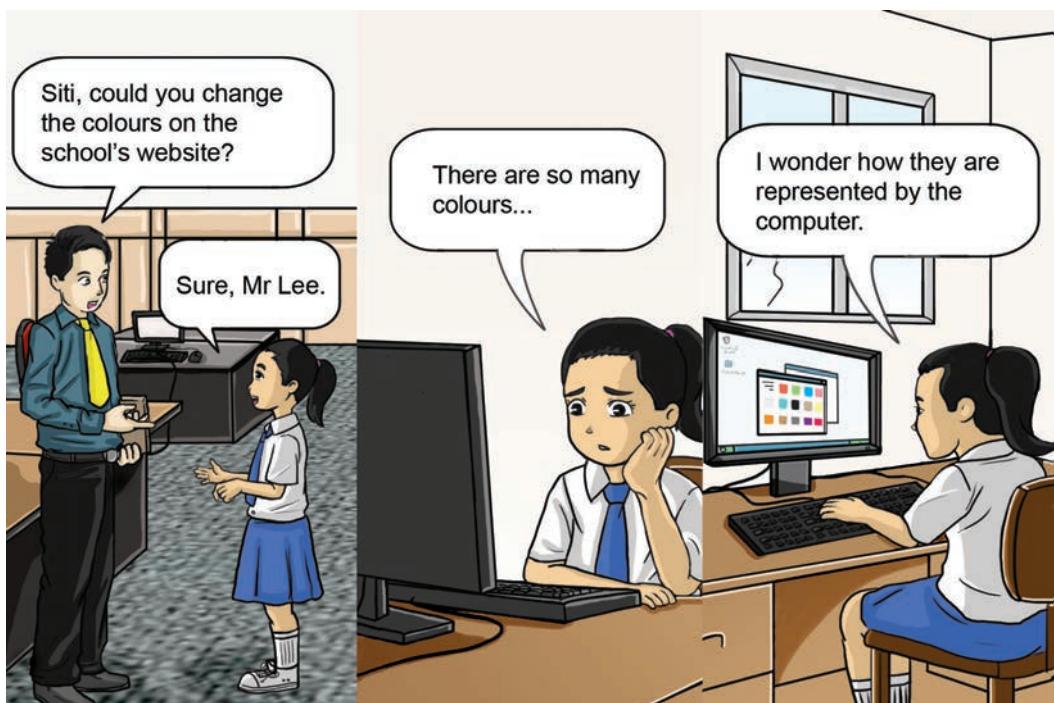
▲ **Figure 7.13** Chosen artwork and its two source images

The artwork's creator also submitted his artwork without asking permission from the painting's copyright owner. However, he claims that he did nothing wrong because his artwork only uses a small portion of the original painting.

State whether you think the artwork's creator should win the competition and explain your reasoning.

How are Number Systems Applied in Real Life?

Siti is a member of her school's Infocomm Club. She was asked by her teacher to modify her school's website to give it a new look.



Siti notices that many colours are used in the website. She wonders how each colour is represented by the computer and what may be some common ways of expressing this information.

When we work with computers, we often need to express numbers in different ways. This chapter will introduce the binary, denary and hexadecimal number systems. You will learn about the purpose of each number system, the situations in which they are used and how to convert numbers from one system to another.



By the end of this chapter, you should be able to:

- Represent positive whole numbers in binary form.
- Convert positive whole numbers from one number system to another – binary, denary and hexadecimal; and describe the technique used.
- Describe situations in which the number systems are used such as ASCII codes, IP (Internet Protocol) and Media Access Control (MAC) addresses, and RGB codes.

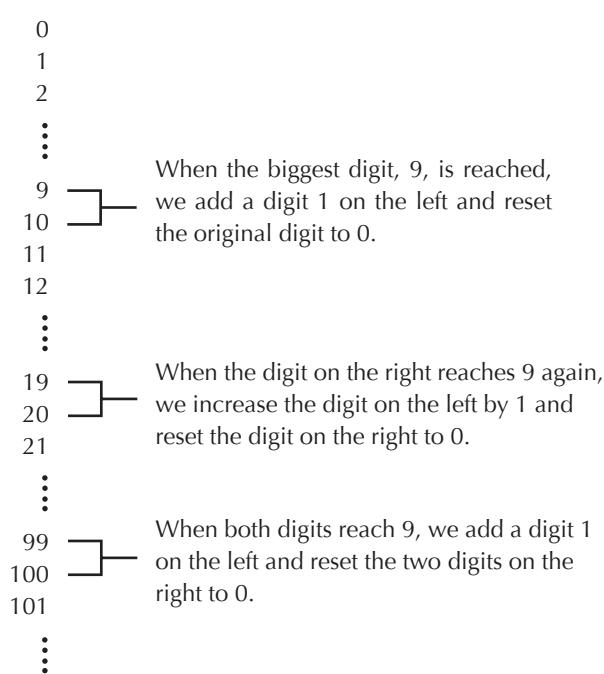
8.1 The Denary Number System

The number system that we are most familiar with and typically use is called the **denary number system**. It is also known as the decimal number system and is made up of 10 distinct digits.

▼ **Table 8.1** Digits of the denary number system

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

As the denary number system has 10 digits, it can also be called a base-10 system. This is how we count using the denary number system:



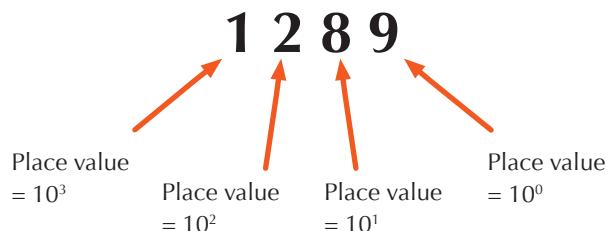
▲ **Figure 8.1** Counting in the denary number system

The position of a digit in a denary number determines its place value, which is represented by 10^N . The place value starts from 10^0 for the rightmost digit, with the power (N) increasing by 1 for each subsequent digit on the left, as shown in Figure 8.2. This means that the place value of each digit in the denary number system is 10 times the place value of the digit on its immediate right.

Key Term

Denary number system (or decimal number system)

Number system that is made up of 10 unique digits



▲ Figure 8.2 Place values of a denary number

For example, 1289 represents 1 thousand, 2 hundreds, 8 tens and 9 ones.

▼ Table 8.2 1289 in the denary number system

Place value	hundred thousands	ten thousands	thousands	hundreds	tens	ones
	10^5	10^4	10^3	10^2	10^1	10^0
Denary digit			1	2	8	9

Figure 8.3 shows how the value of 1289 can be calculated as the sum of each digit multiplied by its place value:

$$(1 \times 10^3) + (2 \times 10^2) + (8 \times 10^1) + (9 \times 10^0) = 1289$$

▲ Figure 8.3 Calculating the value of a denary number

Note that 1289 does not have any digits to the left of the digit in the thousands place. The missing digits in the ten thousands place and higher are thus implied to be 0. These implied digits are usually left out for denary numbers, but may sometimes be included, as **leading zeros**, when an exact number of digits is needed. For instance, 1289 may be written as 001289 to use exactly six digits in the denary number system. This may be useful to make numbers printed in a column appear neater or easier to read.

Key Term

Leading zero

Zero digit (0) that is to the left of the first non-zero digit in a number

Figure 8.4 shows that 001289 still has a value of 1289 even with the leading zeros included:

$$(0 \times 10^5) + (0 \times 10^4) + (1 \times 10^3) + (2 \times 10^2) + (8 \times 10^1) + (9 \times 10^0) = 1289$$

▲ **Figure 8.4** Calculating the value of a denary number with leading zeros

Denary numbers with many digits can also be difficult to read. To make interpreting such numbers easier, denary numbers larger than 999 are often shown with a separator (either a space or comma) after every group of three denary digits, starting from the right. For instance, the number 1234567890 can also be written as 1,234,567,890 or 1 234 567 890.

8.2 The Binary Number System

As computers store data using bits or binary digits, they cannot use the denary number system with its 10 distinct digits directly. Instead, they use the **binary number system** that is made up of two distinct digits, 0 and 1, in order to store numbers.

Key Term

Binary number system
Number system that is made up of two unique digits

▼ **Table 8.3** Digits of the binary number system

0	1
---	---

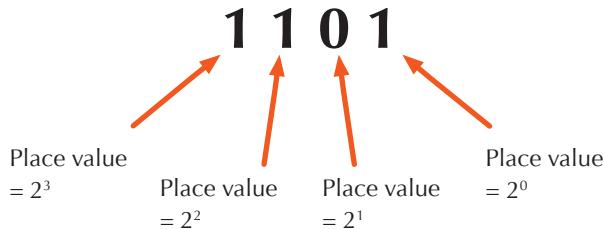
As the binary number system has two digits, it can also be called a base-2 system.

This is how we count using the binary number system:

0	When the biggest digit, 1, is reached, we add a digit 1 on the left and reset the original digit to 0.
1	
10	
11	When both digits reach 1, we add a digit 1 on the left and reset the two digits on the right to 0.
100	
101	
110	
111	When all the digits reach 1, we add a digit 1 on the left and reset the digits on right to 0.
1000	
1001	
1010	
1011	
⋮	

▲ **Figure 8.5** Counting in the binary number system

The position of a digit in a binary number determines its place value, which is represented by 2^N . The place value starts from 2^0 for the rightmost digit, with the power (N) increasing by 1 for each subsequent digit on the left, as shown in Figure 8.6. This means that the place value of each digit in the binary number system is two times the place value of the digit on its immediate right.



▲ Figure 8.6 Place values of a binary number

To distinguish binary numbers from denary numbers, the binary numbers are appended the number 2 as a subscript, like this: 1101_2 . Alternatively, the binary number may be prefixed with the characters “0b”, like this: $0b1101$.

Just like in denary numbers, leading zeros do not affect the value of binary numbers but may still be shown when an exact number of digits is needed. This is common when using binary numbers in computer systems as each byte is defined to have exactly eight binary digits. Leading zeros are often needed so that all eight binary digits of each byte are shown.

For readability, large binary numbers may also be shown with a separator (usually a space) between groups of four digits, starting from the right. For instance, the binary number 10010110_2 can also be written as $1001\ 0110_2$.

Did you know?

Another acceptable way to write binary numbers is to enclose the binary digits in parentheses, like this: $(1101)_2$. This format is less ambiguous in situations where normal digits and subscript digits may appear similar in handwriting.

8.2.1 Converting from Binary to Denary

To convert a binary number to its denary equivalent, we add up the product of each digit and its place value. Let us use the binary number 111001_2 as an example.

▼ Table 8.4 111001_2 in the binary number system

Place value	256	128	64	32	16	8	4	2	1
Binary digit	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
				1	1	1	0	0	1

$$(1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\ = 32 + 16 + 8 + 0 + 0 + 1 = 57$$

▲ **Figure 8.7** Converting $11\ 1001_2$ to denary

Hence, the denary equivalent of $11\ 1001_2$ is 57.

8.2.2 Converting from Denary to Binary

There are two methods of converting denary numbers to binary numbers. Let us use the denary number 135 as an example.

8.2.2.1 Method 1: Division by 2

In this method, we divide the denary number repeatedly by 2. The remainders of each division are then used to derive the binary number. The general algorithm is as follows:

- Step 1:** Draw a table with three columns – one column for denary numbers, one column for the quotients and one column for the remainders.
- Step 2:** Fill in the denary number in the first row.
- Step 3:** Divide the denary number by 2 and fill in its quotient and remainder in the same row.
- Step 4:** If the quotient is 0, proceed to Step 5. Otherwise, copy the quotient to the denary number column of the next row and repeat Step 3.
- Step 5:** The equivalent binary number is the remainder column read from the bottom up.

Using 135 as an example:

1. Draw a table with three columns – one column for denary numbers, one column for quotients and one column for remainders. Then fill in the denary number that we wish to convert, 135, in the first row.

▼ **Table 8.5** Steps 1 and 2 in the division by 2 method

Denary	Quotient	Remainder
135		

2. Divide the denary number 135 by 2 and fill in its quotient and remainder in the same row. In this case, 135 divided by 2 gives a quotient of 67 and a remainder of 1. As the quotient 67 is non-zero, copy 67 to the denary number column of the next row and repeat Step 3 of the algorithm.

▼ **Table 8.6** First iteration of Steps 3 and 4 in the division by 2 method

Denary	Quotient	Remainder
135	67	1
67		

- Divide the denary number 67 by 2 and fill in its quotient and remainder in the same row. In this case, 67 divided by 2 gives a quotient of 33 and a remainder of 1. As the quotient 33 is non-zero, copy 33 to the denary number column of the next row and repeat Step 3 of the algorithm.

▼ **Table 8.7** Second iteration of Steps 3 and 4 in the division by 2 method

Denary	Quotient	Remainder
135	67	1
67	33	1
33		

- This sequence is repeated until the last iteration, when the denary number reaches 1. In this case, 1 divided by 2 gives a quotient of 0 and a remainder of 1. As the quotient is 0, we proceed to Step 5 of the algorithm.

▼ **Table 8.8** Last iteration of Steps 3 and 4 in the division by 2 method

Denary	Quotient	Remainder
135	67	1
67	33	1
33	16	1
16	8	0
8	4	0
4	2	0
2	1	0
1	0	1

Direction to
read the digits

- The converted binary number is the remainder column read from the bottom up. Hence, the binary equivalent of 135 is $1000\ 0111_2$.

8.2.2.2 Method 2: Sum of Place Values

In this method, we put 1s under the place values, starting from the left, so that the total adds up to the value that we want to convert. The general algorithm is as follows:

- Step 1:** Write out the place value of each position in a binary number.
- Step 2:** Find the highest place value that is equal to or less than the denary number.
- Step 3:** Fill in the binary digit 1 below this place value.
- Step 4:** Subtract this place value from the denary number. If the result is 0, proceed to Step 5. Otherwise, repeat Step 2.
- Step 5:** Fill in the binary digit 0 below any remaining place values.
- Step 6:** The equivalent binary number is formed by the row of binary digits below the place values.

Using 135 as an example:

- Write out the place value of each position in a binary number, as well as the denary number we wish to convert, 135.

▼ Table 8.9 Step 1 in the sum of place values method

Place value	256	128	64	32	16	8	4	2	1	Denary
	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	135
Binary digit										

- Find the highest place value that is equal to or less than the denary number 135. In this case, 2^7 (128) is the highest place value that is less than 135, so we place a 1 under 2^7 , as shown in Table 8.10.
- Subtract the place value 128 from 135, which gives us 7, as shown in Table 8.10. As the result is non-zero, we repeat Step 2 of the algorithm with the number 7.

▼ Table 8.10 First iteration of Steps 2 to 4 in the sum of place values method

Place value	256	128	64	32	16	8	4	2	1	Denary
	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	135
Binary digit		1								7

4. Find the highest place value that is equal to or less than the denary number 7. In this case, 2^2 (4) is the highest place value that is less than 7, so we place a 1 under 2^2 , as shown in Table 8.11.
5. Subtract the place value 4 from 7, which gives us 3, as in Table 8.11. As the result is non-zero, we repeat Step 2 of the algorithm with the number 3.

▼ **Table 8.11** Second iteration of Steps 2 to 4 in the sum of place values method

Place value	256	128	64	32	16	8	4	2	1	Denary
	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	135 7 3
Binary digit		1					1			

6. Following the same method, we place 1s under 2^1 and 2^0 . As the result is now 0, we continue with Step 5 of the algorithm.

▼ **Table 8.12** Last iteration of Steps 2 to 4 in the sum of place values method

Place value	256	128	64	32	16	8	4	2	1	Denary
	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	135 7 3 0
Binary digit		1					1	1	1	

7. Fill in 0s under the remaining place values, as shown in Table 8.13.

▼ **Table 8.13** Steps 5 and 6 in the sum of place values method

Place value	256	128	64	32	16	8	4	2	1
	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Binary digit		1	0	0	0	0	1	1	1

Hence, the binary equivalent of 135 is $1000\ 0111_2$.



Quick Check 8.2

1. Convert the following denary numbers to binary using both the division by 2 and sum of place values methods:
 - a) 54
 - b) 89
 - c) 100
2. Convert the denary numbers 7, 8, 9 and 10 to their binary equivalents.
3. Convert the following binary numbers to denary:
 - a) 10011_2
 - b) $110\ 0000_2$
 - c) $1000\ 0110_2$

8.3 The Hexadecimal Number System

Another number system that is often used when working with computers is the **hexadecimal number system** that is made up of 16 different distinct digits – the digits 0 to 9 and the letters A to F.

In this system, the letters A to F are also called “digits” and behave just like the digits 0 to 9. Both upper-case “A” to “F” and lower-case “a” to “f” can be used as digits in this manner such that the hexadecimal digit “A” is the equivalent to the hexadecimal digit “a” and so on.

Key Term

Hexadecimal number system

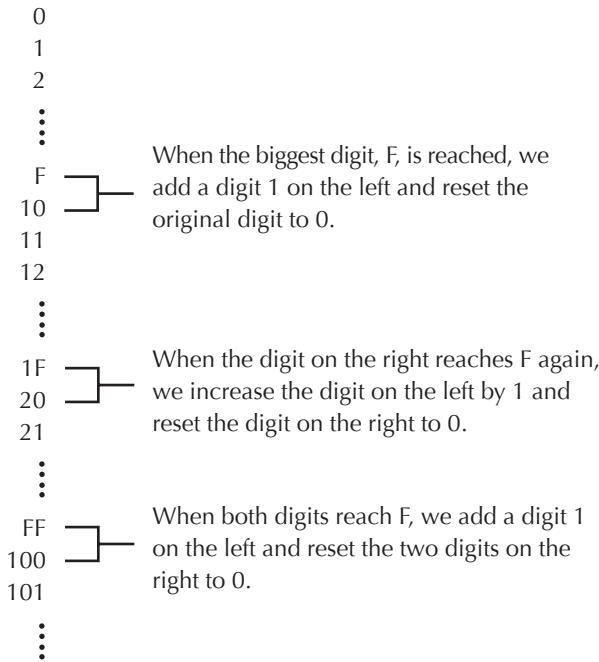
Number system that is made up of 16 unique digits

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

▼ Table 8.14 Digits of the hexadecimal number system

As the hexadecimal number system has 16 digits, it can also be called a base-16 system.

This is how we count using the hexadecimal number system:



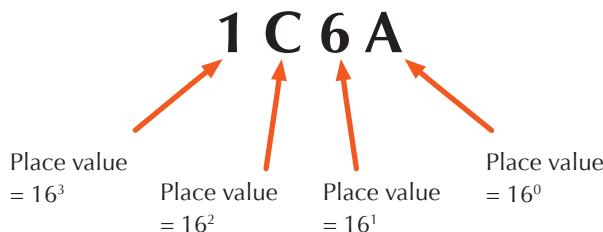
▲ **Figure 8.8** Counting in the hexadecimal number system

Note that this method of counting also tells us that the digits A to F are equivalent to the denary values of 10 to 15 respectively, as shown in Table 8.15.

▼ **Table 8.15** Denary equivalents of the hexadecimal digits

Hexadecimal digit	0	1	2	3	4	5	6	7
Denary equivalent	0	1	2	3	4	5	6	7
Hexadecimal digit	8	9	A	B	C	D	E	F
Denary equivalent	8	9	10	11	12	13	14	15

The position of a digit in a hexadecimal number determines its place value, which is represented by 16^N . The place value starts from 16^0 for the rightmost digit, with the power (N) increasing by 1 for each subsequent digit on the left, as shown in Figure 8.9. This means that the place value of each digit in the hexadecimal number system is 16 times the place value of the digit on its immediate right.



▲ **Figure 8.9** Place values of a hexadecimal number

To distinguish hexadecimal numbers from binary and denary numbers, hexadecimal numbers are appended the number 16 as a subscript, like this: 1C6A₁₆. Alternatively, the hexadecimal number may be prefixed with the characters “0x”, like this: 0x1C6A.

Did you know?

As with binary numbers, another acceptable way to write hexadecimal numbers is to enclose the hexadecimal digits in parentheses, like this: (1C6A)₁₆.

Just like denary and binary numbers, leading zeros do not affect the value of hexadecimal numbers but may still be shown when an exact number of digits is needed. This is common when using hexadecimal numbers in computer systems as each byte needs at most two hexadecimal digits to be shown in full. Leading zeros are often needed so that each byte can be expressed using exactly two hexadecimal digits.

For readability, hexadecimal numbers with more than three digits may also be shown with a separator (usually a space) between groups of two or four digits, starting from the right. For instance, the hexadecimal number 1C6A₁₆ can also be written as 1C 6A₁₆.

Did you know?

An easy way to convert values between different number systems is to use the Calculator program in your computer. In Windows, you can find this program under the Accessories folder of the Start Menu. In Mac, you can find this program in the Applications folder. For either version, set the calculator to Programmer mode by clicking on the “View” menu and selecting “Programmer” to perform conversions between different number systems.

8.3.1 Converting from Hexadecimal to Binary

To convert a hexadecimal number to binary, we break up the number into its individual hexadecimal digits and, using Table 8.16 as a reference, replace each hexadecimal digit with its four-digit binary equivalent.

▼ **Table 8.16** Hexadecimal digits and their four-digit binary equivalents

Hexadecimal digit	0	1	2	3	4	5	6	7
Binary equivalent	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal digit	8	9	A	B	C	D	E	F
Binary equivalent	1000	1001	1010	1011	1100	1101	1110	1111

For example, let us convert $3C_{16}$ to binary.

1. Break up $3C_{16}$ into the individual hexadecimal digits 3 and C.
2. From Table 8.17, we can see that the four-digit binary equivalents of 3 and C are 0011 and 1100 respectively.

▼ **Table 8.17** Converting $3C_{16}$ to binary

Hexadecimal digit	3	C
Binary equivalent	0011	1100

3. Hence, as shown in Table 8.17, the binary equivalent of $3C_{16}$ is $0011\ 1100_2$, which is the same as $11\ 1100_2$ if the leading zeros are omitted.

Next, let us convert CF 07₁₆ to binary.

▼ **Table 8.18** Converting CF 07₁₆ to binary

Hexadecimal digit	C	F	0	7
Binary equivalent	1100	1111	0000	0111

As shown in Table 8.18, the binary equivalent of CF 07₁₆ is 1100 1111 0000 0111₂.

8.3.2 Converting from Binary to Hexadecimal

To convert a binary number to its hexadecimal equivalent, we split the number into groups of four binary digits, starting from the right. If we are left with a group that does not have the full set of four binary digits, we add leading zeros to that group until it is a full set of four binary digits. Using Table 8.16 as a reference, we then convert each four-digit group into its corresponding hexadecimal digit.

For example, let us convert 10 1000 0110 1011₂ to hexadecimal.

- Starting from the right, we split the number into four-digit groups, as shown in Table 8.19.

▼ **Table 8.19** Split the binary number into groups of four digits, starting from the right

10	1000	0110	1011
----	------	------	------

- Since the leftmost group only contains two digits, we add two leading zeros to its left to form a full set of four digits, as shown in Table 8.20.

▼ **Table 8.20** Add leading zeros to the first group from the left if it does not have four digits

0010	1000	0110	1011
------	------	------	------

3. Using Table 8.16, we then convert each group of four binary digits into a hexadecimal digit, as shown in Table 8.21.

▼ **Table 8.21** Converting each group of four binary digits into a hexadecimal digit

Binary digit	0010	1000	0110	1011
Hexadecimal digit	2	8	6	B

4. Hence, the hexadecimal equivalent of $10\ 1000\ 0110\ 1011_2$ is $286B_{16}$.

8.3.3 Converting from Hexadecimal to Denary

To convert a hexadecimal number to its denary equivalent:

1. Convert each hexadecimal digit to its equivalent denary number.
2. Multiply each denary number by its place value.
3. Sum the product of each denary number and its place value.

Let us convert $C7F_{16}$ to denary.

1. From Table 8.15, the denary equivalents of C, 7 and F are 12, 7 and 15 respectively.

▼ **Table 8.22** $C7F_{16}$ in the hexadecimal number system

Place value	65536	4096	256	16	1
	16^4	16^3	16^2	16^1	16^0
Hexadecimal digit			C	7	F
Denary equivalent			12	7	15

2. Multiply the denary equivalent of each digit with its place value as shown in Figure 8.10 and sum up the products.

$$(12 \times 16^2) + (7 \times 16^1) + (15 \times 16^0) = 3072 + 112 + 15 = 3199$$

▲ **Figure 8.10** Converting $C7F_{16}$ to denary

3. Hence, the denary equivalent of $C7F_{16}$ is 3199.

8.3.4 Converting from Denary to Hexadecimal

There are two methods of converting denary numbers to hexadecimal numbers. Let us use the denary number 1899 as an example.

8.3.4.1 Method 1: Division by 16

In this method, we divide the denary number repeatedly by 16. The remainders of each division are then used to derive the hexadecimal number. The general algorithm is as follows:

- Step 1:** Draw a table with three columns – one column for denary numbers, one column for quotients and one column for remainders.
- Step 2:** Fill in the denary number in the first row.
- Step 3:** Divide the denary number by 16 and fill in its quotient and remainder in the same row.
- Step 4:** If the quotient is 0, proceed to Step 5. Otherwise, copy the quotient to the denary number column of the next row and repeat Step 3.
- Step 5:** Convert the numbers in the remainder column to hexadecimal digits.
- Step 6:** The converted hexadecimal number is the remainder column read from the bottom up.

Using 1899 as an example:

1. Draw a table with three columns – one column for denary numbers, one column for quotients and one column for remainders. Then fill in the denary number that we wish to convert, 1899, in the first row.

▼ **Table 8.23** Steps 1 and 2 in the division by 16 method

Denary	Quotient	Remainder
1899		

2. Divide the denary number 1899 by 16 and fill in its quotient and remainder in the same row. In this case, 1899 divided by 16 gives a quotient of 118 and a remainder of 11. As the quotient 118 is non-zero, copy 118 to the denary number column of the next row and repeat Step 3 of the algorithm.

▼ **Table 8.24** First iteration of Steps 3 and 4 in the division by 16 method

Denary	Quotient	Remainder
1899	118	11
118		

3. Divide the denary number 118 by 16 and fill in its quotient and remainder in the same row. In this case, 118 divided by 16 gives a quotient of 7 and a remainder of 6. As the quotient 7 is non-zero, copy 7 to the denary number column of the next row and repeat Step 3 of the algorithm.

▼ **Table 8.25** Second iteration of Steps 3 and 4 in the division by 16 method

Denary	Quotient	Remainder
1899	118	11
118	7	6
7		

4. Divide the denary number 7 by 16 and fill in the quotient and remainder in the same row. In this case, 7 divided by 16 gives a quotient of 0 and a remainder of 7. As the quotient is 0, we will proceed to Step 5 of the algorithm.

▼ **Table 8.26** Last iteration of Steps 3 and 4 in the division by 16 method

Denary	Quotient	Remainder
1899	118	11
118	7	6
7	0	7

5. Convert the numbers in the remainder column to hexadecimal digits.

▼ **Table 8.27** Steps 5 and 6 in the division by 16 method

Denary	Quotient	Remainder
1899	118	11 = B ₁₆
118	7	6 = 6 ₁₆
7	0	7 = 7 ₁₆



Direction to read the digits

6. The converted hexadecimal number is the remainder column read from the bottom up. Hence, the hexadecimal equivalent of 1899 is 76B₁₆.

8.3.4.2 Method 2: Sum of Place Values

In this method, we put hexadecimal digits under the place values, starting from the left, so that the total adds up to the value that we want to convert. This method is similar to the method used in section 8.2.2.2 to convert from denary to binary. However, it is significantly more complex as there are more digits to work with in the hexadecimal number system compared to the binary number system.

The general algorithm is as follows:

- Step 1:** Write out the place value of each position in a hexadecimal number.
- Step 2:** Find the highest place value that is equal to or less than the denary number.
- Step 3:** Fill in the largest hexadecimal digit below this place value such that its product with this place value is equal to or less than the denary number.
- Step 4:** Subtract this product from the denary number. If the result is 0, proceed to Step 5. Otherwise, repeat Step 2.
- Step 5:** Fill in the hexadecimal digit 0 below any remaining place values.
- Step 6:** The converted hexadecimal number is formed by the row of hexadecimal digits below the place values.

Using 1899 as an example:

1. Write out the place value of each position in a hexadecimal number, as well as the denary number we wish to convert, 1899.

▼ **Table 8.28** Step 1 in the sum of place values method

Place value	65536	4096	256	16	1	Denary
	16^4	16^3	16^2	16^1	16^0	1899
Hexadecimal digit						

2. Find the highest place value that is equal to or less than the denary number 1899. In this case, 16^2 (256) is the highest place value that is less than 1899. Hence, under the 16^2 column, we fill in the largest hexadecimal digit such that its product with 16^2 is equal to or less than 1899. Since $7_{16} \times 16^2 = 7 \times 256 = 1792$ (which is less than 1899) and $8_{16} \times 16^2 = 8 \times 256 = 2048$ (which exceeds 1899), the largest hexadecimal digit needed here is 7_{16} , as shown in Table 8.29.

3. Subtract the product 1792 from 1899, which gives us 107, as shown in Table 8.29. As the result is non-zero, we repeat Step 2 of the algorithm with the number 107.

▼ **Table 8.29** First iteration of Steps 2 to 4 in the sum of place values method

Place value	65536	4096	256	16	1	Denary
	16^4	16^3	16^2	16^1	16^0	
Hexadecimal digit				7		1899 107

4. Find the highest place value that is equal to or less than the denary number 107. In this case, 16^1 (16) is the highest place value that is less than 107. Hence, under the 16^1 column, we fill in the largest hexadecimal digit such that its product with 16^1 is equal to or smaller than 107. Since $6_{16} \times 16^1 = 6 \times 16 = 96$ (which is less than 107) but $7_{16} \times 16^1 = 7 \times 16 = 112$ (which exceeds 107), the largest hexadecimal digit needed here is 6_{16} , as shown in Table 8.30.
5. Subtract the product 96 from 107, which gives us 11, also shown in Table 8.30. As the result is non-zero, we repeat Step 2 of the algorithm with the number 11.

▼ **Table 8.30** Second iteration of Steps 2 to 4 in the sum of place values method

Place value	65536	4096	256	16	1	Denary
	16^4	16^3	16^2	16^1	16^0	
Hexadecimal digit			7	6		1899 107 11

6. Find the highest place value that is equal to or less than the denary number 11. In this case, 16^0 (or 1) is the highest place value that is less than 11. Hence, under the 16^0 column, we fill in the largest hexadecimal digit such that its product with 16^0 is equal to or smaller than 11. Since $B_{16} \times 16^0 = 11 \times 1 = 11$ (which is equal to 11) but $C_{16} \times 16^0 = 12 \times 1 = 12$ (which exceeds 11), the largest hexadecimal digit needed here is B_{16} , as shown in Table 8.31.

- Subtract the product 11 from 11, which gives us 0, also shown in Table 8.31. As the result is 0, we proceed to Step 5 of the algorithm.

▼ **Table 8.31** Last iteration of Steps 2 to 4 in the sum of place values method

Place value	65536	4096	256	16	1	Denary
	16^4	16^3	16^2	16^1	16^0	
Hexadecimal digit			7	6	B	0

- Fill in 0s under the remaining place values. In this case, all the place values after the first non-zero hexadecimal digit are already filled.

Hence, the hexadecimal equivalent of 1899 is $76B_{16}$.



Quick Check 8.3

- Write down the hexadecimal and binary equivalents of the denary values from 32 to 47.
- Convert the following hexadecimal numbers to binary:
 - 49_{16}
 - $AB6_{16}$
 - $DA47_{16}$
- Convert the following hexadecimal numbers to denary:
 - $9D_{16}$
 - $BB6_{16}$
 - $AD90_{16}$
- Convert the following denary numbers to hexadecimal:
 - 1000
 - 3789
 - 4220

8.4 Applications of Binary and Hexadecimal Number Systems

There are three applications which use the binary and hexadecimal number systems:

1. RGB colour codes
2. Network addresses
3. ASCII and Unicode

8.4.1 RGB Colour Codes

RGB is an abbreviation of the colours red, green and blue. They are primary colours because a wide range of colours can be formed by adding different intensities of red, green and blue light together. For instance, red and green lights of equal intensity can be combined (such as by shining both lights on a screen) to produce yellow. This allows us to represent a wide range of colours by varying the intensities of the red, green and blue components.

This representation of colour is perfect for computers as we can use numbers to control the intensity of each component. Each intensity value is stored in a single byte, which contains eight bits. Hence, each value can only vary from $0000\ 0000_2$ to $1111\ 1111_2$. This corresponds to 00_{16} to FF_{16} and 0 to 255 in hexadecimal and denary respectively.

If all three colour intensities are 0, the resulting colour is black. If all three colour intensities are at their maximum (FF_{16}), the resulting colour is white. Table 8.32 shows some common colours and the corresponding intensities of their red, green and blue components.

▼ **Table 8.32** Intensities of the red, green and blue components of some common colours

Colour		Red component	Green component	Blue component
Black		00_{16}	00_{16}	00_{16}
Red		FF_{16}	00_{16}	00_{16}
Orange		FF_{16}	80_{16}	00_{16}
Yellow		FF_{16}	FF_{16}	00_{16}
Green		00_{16}	FF_{16}	00_{16}
Cyan		00_{16}	FF_{16}	FF_{16}
Blue		00_{16}	00_{16}	FF_{16}
Purple		80_{16}	00_{16}	80_{16}
White		FF_{16}	FF_{16}	FF_{16}

An RGB colour code is displayed in the form of #RRGGBB, where RR, GG and BB are two-digit hexadecimal numbers that represent the red (R), green (G) and blue (B) components of the colour. Note that the six hexadecimal digits do not represent a single number but instead represent *three* separate numbers with two hexadecimal digits each.

RGB colour codes are used to represent colours in websites, which are typically written using the Hypertext Markup Language (HTML) and the Cascading Style Sheets (CSS) language.

Did you know?

While the Python language that you have learnt is specialised for describing instructions that a computer performs, the HTML and CSS languages are specialised for describing documents that are transmitted online. This textbook will not cover the HTML and CSS languages other than how they use RGB colour codes to represent colours.

For instance, at the beginning of this chapter, Siti finds that the background of her school's website has an RGB colour code of #FF8000. This RGB colour code represents a colour with a red component intensity of $FF_{16} = 255$, a green component intensity of $80_{16} = 128$ and a blue component intensity of $00_{16} = 0$, which results in a shade of orange.

If Siti wishes to change this background colour to cyan, she first has to find out what the red, green and blue component intensities of cyan are, and convert these values to two-digit hexadecimal numbers. From Table 8.32, we see that cyan has a red component intensity of $0 = 00_{16}$, a green component intensity of $255 = FF_{16}$ and a blue component intensity of $255 = FF_{16}$. Hence, the RGB colour code for cyan is #00FFFF. Siti can use this colour code in the website's HTML and CSS codes to change the background colour to cyan. Table 8.33 shows some common colours and their corresponding RGB colour codes.

▼ **Table 8.33** RGB colour codes of some common colours

Colour	RGB colour code
Black	#000000
Red	#FF0000
Orange	#FF8000
Yellow	#FFFF00
Green	#00FF00
Cyan	#00FFFF
Blue	#0000FF
Purple	#800080
White	#FFFFFF

8.4.2 Network Addresses

For computers to successfully communicate or exchange data over a computer network, they must be able to locate each other so that transmitted data can be directed to the correct destination. This can be done by giving each computer a unique name in the form of a sequence of bytes called a **network address**.

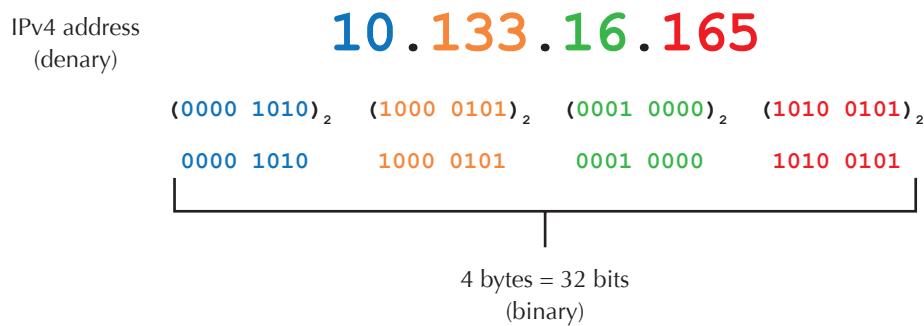
This is similar to how mailing addresses are used to locate buildings or apartments for postal mail. For instance, mail can be successfully delivered to any location with a mailing address as the mailing address can be used to locate the correct destination. On the other hand, mail cannot be delivered to a location that does not have a mailing address as there would be no way to locate the destination.

The Internet is an example of a computer network, and just like how buildings and apartments have mailing addresses, each computer on the Internet has an **Internet Protocol (IP) address** that serves as its network address.

“IP” refers to a standard system of rules used by computers on the Internet to communicate with one another. There are two versions of IP used today, namely IP version 4 (IPv4) and IP version 6 (IPv6).

8.4.2.1 IPv4 Addresses

In IPv4, an IP address is made up of 4 bytes, or 32 bits. It is usually shown as a sequence of four denary numbers, one for each byte of the address, separated by dots. Since the value of a byte can only vary from 0 to 255, none of the four denary numbers in an IPv4 address can be less than 0 or more than 255. An example of a valid IPv4 address is 10.133.16.165, as shown in Figure 8.11.



▲ **Figure 8.11** Example of an IPv4 address

An IPv4 address contains 32 bits, so the maximum number of possible IPv4 addresses is 2^{32} , or approximately 4.3 billion. However, with the rapid growth of the Internet in the 1980s and 1990s, it became obvious that this number of possible IPv4 addresses would not be enough.

Key Terms

Internet Protocol (IP) address

Sequence of bytes that is used to identify a computer or device on the Internet

Network address

Unique name or sequence of bytes that is used to identify a computer or device in a network

8.4.2.2 IPv6 Addresses

To provide a long-term solution to the shortage of IPv4 addresses, IP version 6 (IPv6) was introduced in 1998, and expanded the size of IP addresses from 4 bytes (32 bits) to 16 bytes (128 bits). This meant that the maximum number of IP addresses was increased to 2^{128} (approximately 3.4×10^{38}), which is large enough to overcome the problem of insufficient IP addresses for the foreseeable future. Almost all computers connected to the Internet have an IPv4 address, and many use the IPv6 address system as well.

Due to the increased length of IPv6 addresses, however, the old format of expressing IPv4 addresses was no longer suitable, as having 16 denary numbers separated by dots would take up a maximum of $16 \times 3 = 48$ digits, which is too long.

As a result, a new format of showing IPv6 addresses using hexadecimal digits was developed. In this new format, the 128 bits of an IPv6 address are divided into eight groups of 16 bits. Each group of 16 bits can vary between $0000\ 0000\ 0000\ 0000_2$ to $1111\ 1111\ 1111\ 1111_2$, which is 0000_{16} to $FFFF_{16}$ in hexadecimal. As we can see, each group of 16 bits requires only four hexadecimal digits. Therefore, an IPv6 address expressed in this format would only take up at most $8 \times 4 = 32$ digits (including leading zeros).

An example of a full IPv6 address is 2001:0db8:0000:0000:0020:0017:0bad:c0de, shown in Figure 8.12. (Note that, by convention, IPv6 addresses use lower-case instead of upper-case letters.)

IPv6 address (hexadecimal) **2001:0db8:0000:0000:0020:0017:0bad:c0de**

$$\begin{array}{l}
 (\textcolor{blue}{0010} \textcolor{blue}{0000} \textcolor{blue}{0000} \textcolor{blue}{0001})_2 \quad (\textcolor{orange}{0000} \textcolor{orange}{1101} \textcolor{orange}{1011} \textcolor{orange}{1000})_2 \quad (\textcolor{green}{0000} \textcolor{green}{0000} \textcolor{green}{0000} \textcolor{green}{0000})_2 \\
 (\textcolor{red}{0000} \textcolor{red}{0000} \textcolor{red}{0000} \textcolor{red}{0000})_2 \quad (\textcolor{blue}{0000} \textcolor{blue}{0000} \textcolor{blue}{0010} \textcolor{blue}{0000})_2 \quad (\textcolor{black}{0000} \textcolor{black}{0000} \textcolor{black}{0001} \textcolor{black}{0111})_2 \\
 \qquad (\textcolor{blue}{0000} \textcolor{blue}{1011} \textcolor{blue}{1010} \textcolor{blue}{1101})_2 \quad (\textcolor{black}{1100} \textcolor{black}{0000} \textcolor{black}{1101} \textcolor{black}{1110})_2
 \end{array}$$

0010	0000	0000	0001	0000	1101	1011	1000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0010	0000	0000	0000	0000	0001	0111
					0000	1011	1010	1101	1100	0000	1101	1110

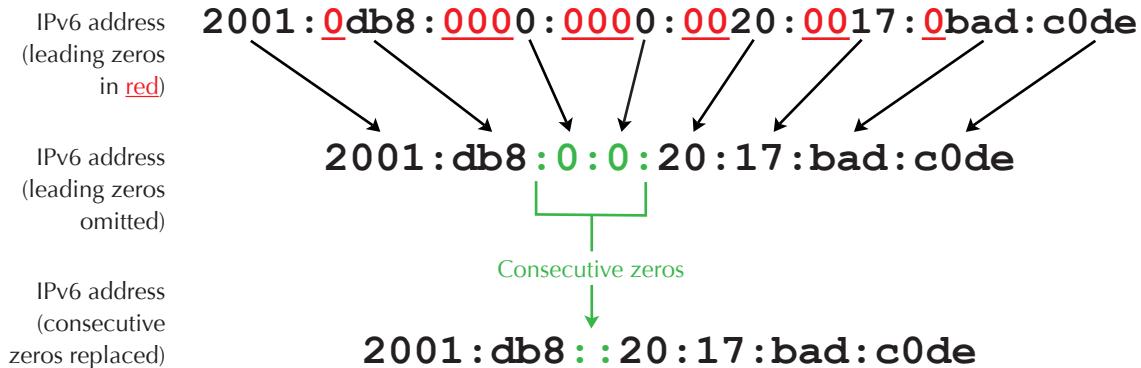
16 bytes = 128 bits
(binary)

▲ **Figure 8.12** Example of an IPv6 address

To make this format more compact, the leading zeros can be omitted. For instance, the IPv6 address in Figure 8.12 can also be written as 2001:db8:0:0:20:17:bad:c0de.

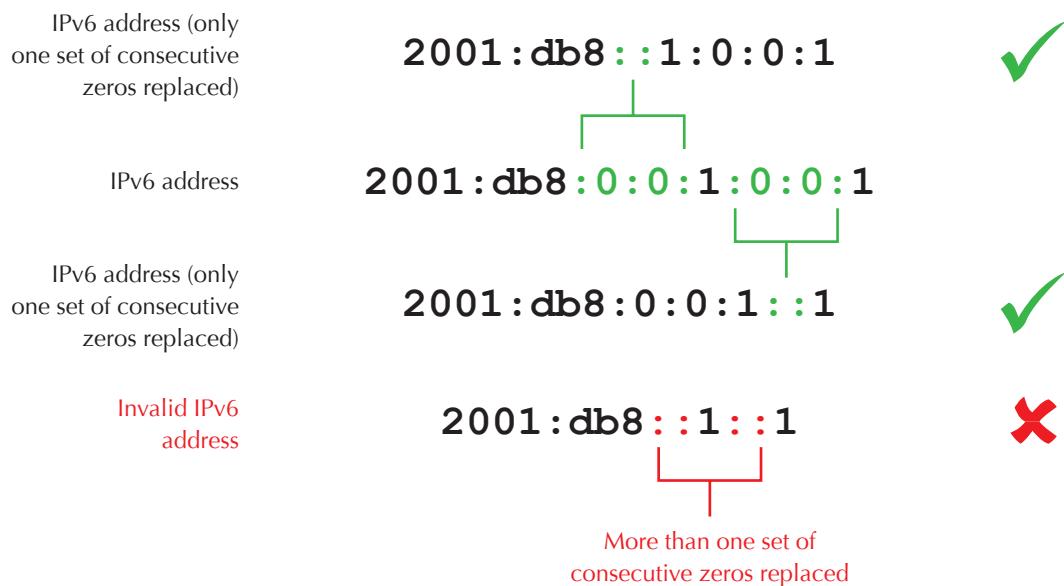
To be even more compact, a section of consecutive zeros can also be replaced with two colon characters. For instance, the IPv6 address in Figure 8.12 can be written with the third and fourth sections omitted as 2001:db8::20:17:bad:c0de.

These methods of shortening IPv6 addresses are illustrated in Figure 8.13.



▲ Figure 8.13 Shortening an IPv6 address

However, no more than one such section of consecutive zeros can be replaced with double colons. For instance, `2001:db8:0:0:1:0:0:1` can be written as `2001:db8::1:0:0:1` or `2001:db8:0:0:1::1`, but *not* `2001:db8::1::1`. Figure 8.14 illustrates this.



▲ Figure 8.14 Only one section of consecutive zeros can be replaced in an IPv6 address

Did you know?

IP versions 0, 1, 2, 3 and 5 were experimental and not meant to be widely used. This is why only IP versions 4 and 6 are in use today.

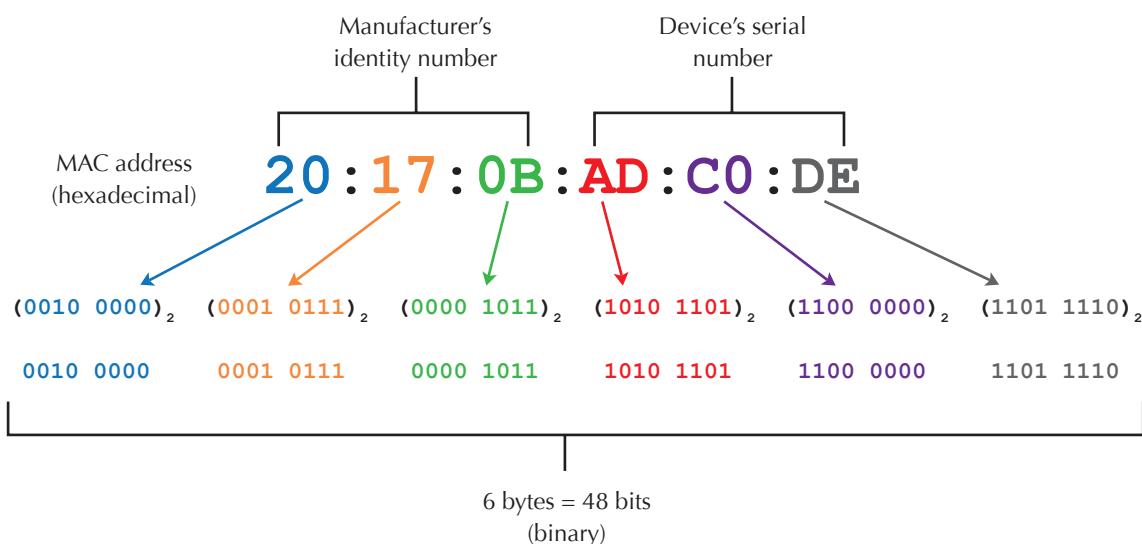
8.4.2.3 MAC Addresses

As the IP address (in both IPv4 and IPv6) of a computer or device may change each time it connects to the Internet, it is not considered permanent. If a more permanent way to locate or identify a particular computer or device is needed, then its **Media Access Control (MAC) address** (also called hardware address or physical address) should be used instead. In some ways, an IP address is like a mailing address while a MAC address is like a thumbprint.

For instance, a person who travels frequently may have a constantly changing mailing address, but his or her thumbprint does not change. Similarly, a computer may have a different IP address each time it connects to the Internet, but its MAC address is unlikely to change each time it connects.

Unlike IP addresses, which are used to direct transmitted data over the Internet, MAC addresses are only used to direct data between devices in a local area network (LAN). MAC addresses are not usually transmitted in full over the Internet. You will learn more about the difference between the Internet and a LAN in Chapter 11.

A MAC address is made up of 6 bytes, or 48 bits. It is usually shown as a sequence of six hexadecimal numbers, one for each byte of the address, separated by hyphens or colons. Hence, a typical MAC address uses exactly two hexadecimal digits for each byte (with a leading zero if needed) so that its format is NN-NN-NN-DD-DD-DD (or NN:NN:NN:DD:DD:DD), where NN-NN-NN (or NN:NN:NN) is the manufacturer's identity number and DD-DD-DD (or DD:DD:DD) is the device's serial number. An example of a valid MAC address is 20:17:0B:AD:C0:DE, as shown in Figure 8.15.



▲ **Figure 8.15** Example of a MAC address

Key Term

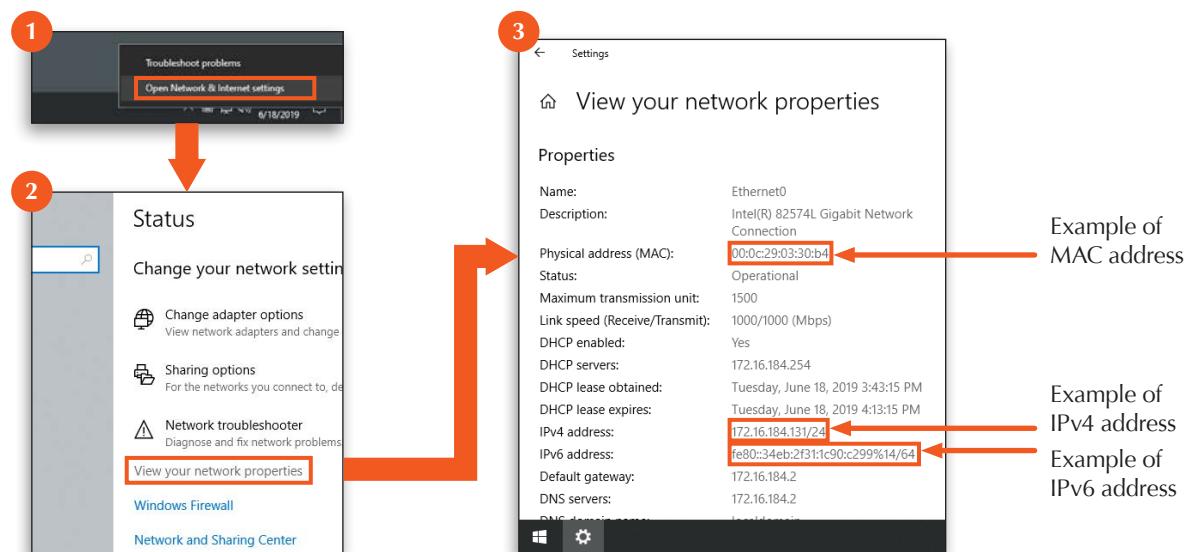
Media Access Control (MAC) address

Sequence of bytes (usually permanent in nature) that is used to identify a particular network interface controller

Did you know?

To be precise, the MAC address only identifies the computer's network interface controller (NIC) and not the whole computer. Furthermore, while they are usually considered permanent, it is actually possible to change or fake the MAC address of some NICs. You will learn more about NICs in Chapter 11.

On Windows, you can view your computer's IPv4 address, IPv6 address (if any) and MAC address by opening Network and Internet Settings and clicking on the "View your network properties" link.



▲ Figure 8.16 Viewing your computer's IPv4, IPv6 and MAC addresses on Windows

On Mac, you can open the Network panel under System Preferences and click on the "Advanced..." button. Your computer's IPv4 address and IPv6 address (if any) can be found on the "TCP/IP" tab, while your computer's MAC address can be found on the "Hardware" tab.

8.4.3 ASCII and Unicode

As explained in section 5.3.1, the ASCII standard defines how numbers are used to represent common characters that can be typed using a keyboard, such as upper-case and lower-case letters. Using ASCII, only 128 distinct characters can be represented because ASCII codes are defined to be exactly seven bits long. This means that the value of ASCII codes can only vary from $000\ 0000_2$ to $111\ 1111_2$, which is the same as 00_{16} to $7F_{16}$ or 0 to 127.

The first 32 ASCII codes (0 to 31) represent non-printing characters that are not written symbols or the space character. These codes are instead used to control devices or to represent special input such as typing the Backspace and Esc keys on the keyboard. The last ASCII code 127 is also used to represent the special input of typing the Delete key on the keyboard.

Hexadecimal digits are often used to express ASCII codes as only two hexadecimal digits are needed to express any ASCII code. This is more compact than using three denary digits or seven binary digits.

Table 8.34 shows how the ASCII codes from 32 to 127 may be expressed using the denary (base-10), binary (base-2) and hexadecimal (base-16) number systems along with the corresponding characters being represented.

▼ **Table 8.34** ASCII codes converted to binary and hexadecimal numbers and their corresponding characters

Base-10	Base-2	Base-16	Character	Base-10	Base-2	Base-16	Character
32	010 0000	20	(space)	55	011 0111	37	7
33	010 0001	21	!	56	011 1000	38	8
34	010 0010	22	"	57	011 1001	39	9
35	010 0011	23	#	58	011 1010	3A	:
36	010 0100	24	\$	59	011 1011	3B	;
37	010 0101	25	%	60	011 1100	3C	<
38	010 0110	26	&	61	011 1101	3D	=
39	010 0111	27	'	62	011 1110	3E	>
40	010 1000	28	(63	011 1111	3F	?
41	010 1001	29)	64	100 0000	40	@
42	010 1010	2A	*	65	100 0001	41	A
43	010 1011	2B	+	66	100 0010	42	B
44	010 1100	2C	,	67	100 0011	43	C
45	010 1101	2D	-	68	100 0100	44	D
46	010 1110	2E	.	69	100 0101	45	E
47	010 1111	2F	/	70	100 0110	46	F
48	011 0000	30	0	71	100 0111	47	G
49	011 0001	31	1	72	100 1000	48	H
50	011 0010	32	2	73	100 1001	49	I
51	011 0011	33	3	74	100 1010	4A	J
52	011 0100	34	4	75	100 1011	4B	K
53	011 0101	35	5	76	100 1100	4C	L
54	011 0110	36	6	77	100 1101	4D	M

▼ **Table 8.34** ASCII codes converted to binary and hexadecimal numbers and their corresponding characters (continued)

Base-10	Base-2	Base-16	Character	Base-10	Base-2	Base-16	Character
78	100 1110	4E	N	103	110 0111	67	g
79	100 1111	4F	O	104	110 1000	68	h
80	101 0000	50	P	105	110 1001	69	i
81	101 0001	51	Q	106	110 1010	6A	j
82	101 0010	52	R	107	110 1011	6B	k
83	101 0011	53	S	108	110 1100	6C	l
84	101 0100	54	T	109	110 1101	6D	m
85	101 0101	55	U	110	110 1110	6E	n
86	101 0110	56	V	111	110 1111	6F	o
87	101 0111	57	W	112	111 0000	70	p
88	101 1000	58	X	113	111 0001	71	q
89	101 1001	59	Y	114	111 0010	72	r
90	101 1010	5A	Z	115	111 0011	73	s
91	101 1011	5B	[116	111 0100	74	t
92	101 1100	5C	\	117	111 0101	75	u
93	101 1101	5D]	118	111 0110	76	v
94	101 1110	5E	^	119	111 0111	77	w
95	101 1111	5F	_	120	111 1000	78	x
96	110 0000	60	`	121	111 1001	79	y
97	110 0001	61	a	122	111 1010	7A	z
98	110 0010	62	b	123	111 1011	7B	{
99	110 0011	63	c	124	111 1100	7C	
100	110 0100	64	d	125	111 1101	7D	}
101	110 0101	65	e	126	111 1110	7E	~
102	110 0110	66	f	127	111 1111	7F	(delete)

Recall that, besides ASCII, an expanded standard called Unicode is used to represent characters in languages other than English such as Tamil and Mandarin. Like ASCII, the Unicode standard uses numbers to represent characters. However, while ASCII codes are defined to be exactly seven bits long, Unicode is not as restrictive. In Unicode, the number of bits used to represent each character can vary from 8 to 32 bits depending on the encoding scheme used. This flexibility means that Unicode can be used to represent over a million unique characters from many different written languages all over the world, which makes it a preferred choice for use over the ASCII standard.

Did you know?

The largest number that can be used to represent a character in Unicode is 1,114,111, which is 10000 1111 1111 1111 1111₂ and requires 21 binary digits. This means that if Unicode values are stored directly as binary numbers, each character would take up 21 bits. Doing so would be wasteful, however, as not all Unicode characters are used equally often. The designers of Unicode realised that it would be more efficient to use fewer bits for frequently used characters at the expense of using more bits for rarely used characters. This explains why the number of bits used to represent a Unicode character can vary between 8 and 32.

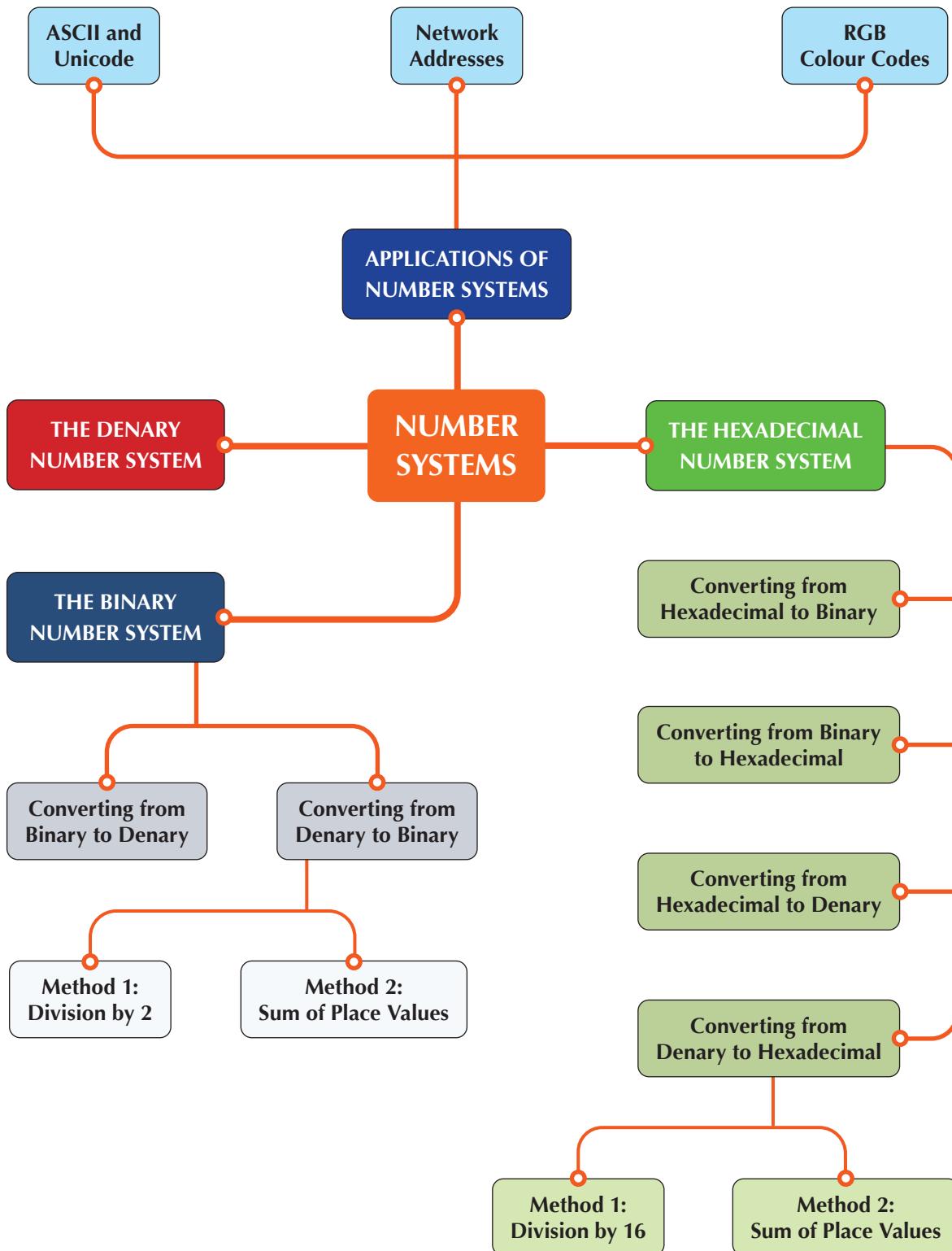


Quick Check 8.4

1. Write down a possible RGB colour code for the colour magenta.
(Hint: Magenta can be formed by adding red light to blue light at maximum intensity.)
2. Write a program that accepts a string as input and outputs “Valid” if the input string is a valid IPv4 address (i.e., four denary numbers between 0 and 255 separated by dots), or “Invalid” if the input string is not a valid IPv4 address.



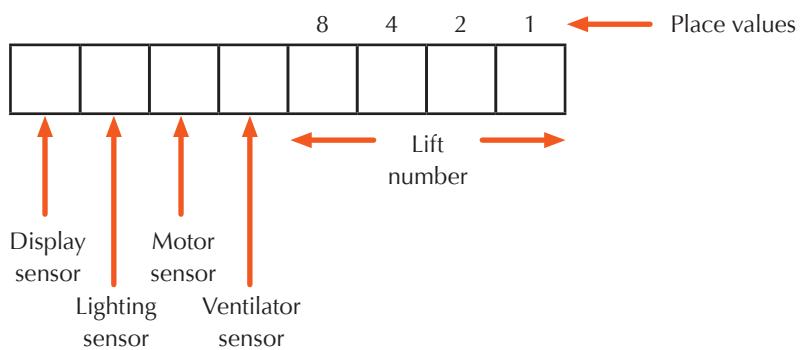
Chapter Summary





Review Questions

1. Why is the hexadecimal number system preferred over the binary number system in representing large numbers?
2. Convert the following hexadecimal numbers to binary:
 - a) A_{16}
 - b) FF_{16}
 - c) $10B_{16}$
3. Convert the following binary numbers to hexadecimal:
 - a) 1101_2
 - b) 1001_2
 - c) 101_2
 - d) $1110\ 1101_2$
 - e) $101\ 0101_2$
 - f) $10110\ 1100\ 1101_2$
4. a) Convert the denary number 108 to an eight-bit binary number.
b) Convert the denary number 108 to hexadecimal.
c) State two common uses of the hexadecimal number system.
5. A lift system in a building uses an eight-bit register shown in Figure 8.17. The first four bits indicate whether any of the four sensors has picked up a fault (denoted by the value 1) in any of the subsystems. The last four bits indicate the lift number. There are 15 lifts in the building.



▲ **Figure 8.17** The eight-bit register used in a lift system

For example, the register shown in Figure 8.18 indicates that there are faults in the lighting and motor subsystems for lift number 3.

0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

▲ **Figure 8.18** Register indicating that lift number 3 has faults in the lighting and motor subsystems

- a) What is indicated by the register shown in Figure 8.19?

1	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---

▲ **Figure 8.19** Reading on eight-bit register

- b) What is indicated by the register shown in Figure 8.20?

0	0	1	1	1	0	1	1
---	---	---	---	---	---	---	---

▲ **Figure 8.20** Reading on eight-bit register

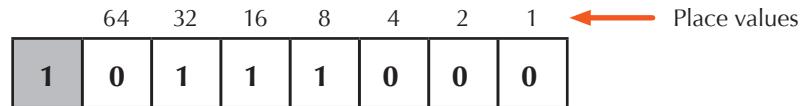
- c) What would be the reading indicated by the register if there are faults in the display and lighting subsystems for lift number 14? Give your answer as a binary number.
6. In a factory, there are 127 electrical points that provide power to various machinery. An eight-bit register is used to represent the state of each electrical point. The first bit indicates the operational state of the electrical point – a value of 1 means that it is operational and a value of 0 means that it is faulty. The next seven bits indicate the electrical point number.

For example, the eight-bit register shown in Figure 8.21 indicates that electrical point number 41 is operational.

64	32	16	8	4	2	1	Place values
1	0	1	0	1	0	0	1

▲ **Figure 8.21** Register indicating that electrical point number 41 is operational

- a) What is indicated by the register shown in Figure 8.22?



▲ Figure 8.22 Reading on eight-bit register

- b) What would the register show if electrical point number 36 is faulty? Give your answer as an eight-bit binary number.
- c) Convert the number indicated by the register shown in Figure 8.23 to hexadecimal.



▲ Figure 8.23 Reading on eight-bit register

How Do Logic Circuits Make Decisions?

Every day, we make decisions based on the information that is available to us. Suppose Siti needs to decide between taking the bus or walking to school. Her decision would depend on the weather and timing. If it is raining or she is late, Siti would choose to take the bus to avoid getting drenched or to reach the school faster. Otherwise, she would choose to walk to school instead.



We have already learnt how to write algorithms that output the result of decisions that depend on one or more inputs. In this case, the output of whether Siti decides to take the bus or to walk depends on two inputs: whether it is raining and whether she is late.

Like Siti, computers and other electrical systems use inputs to make decisions. However, while Siti makes these decisions using her brain, computers and electrical systems make these decisions using electrical circuits called logic gates. For instance, an automatic lighting system may use logic gates to decide if the lights in a room should be switched on based on whether the sun is shining and whether anyone is in the room. In computers, the logic gates in their processors make the decisions based on the given inputs.

In this chapter, you will learn how logic gates make use of Boolean logic, the five different types of logic gates and how logic circuits can be constructed from logic gates. You will also learn about some applications of logic circuits.



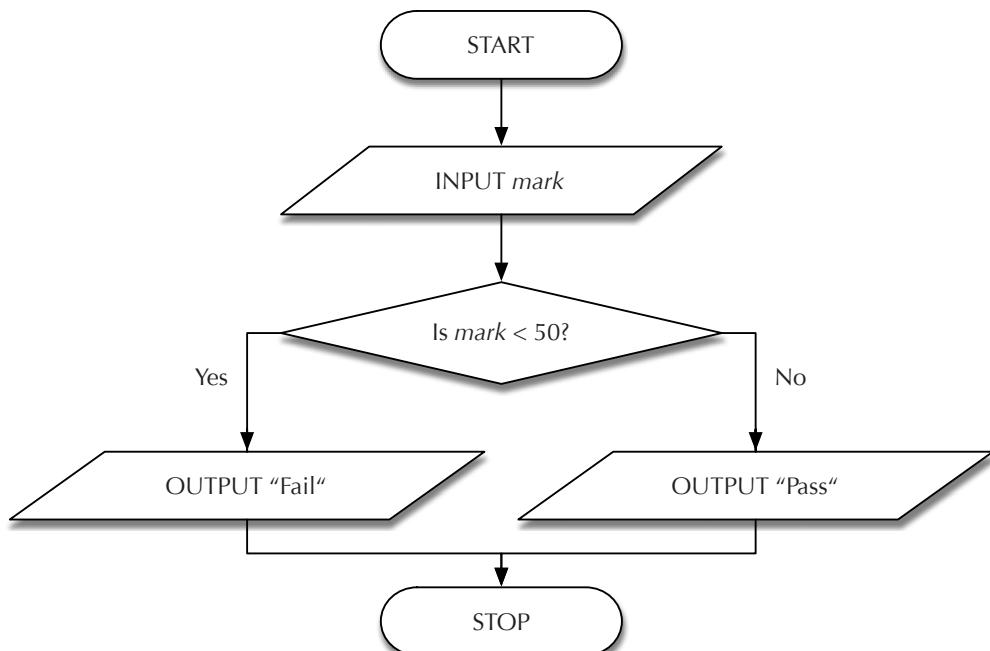
By the end of this chapter, you should be able to:

- Recognise a logic gate from its truth table and evaluate Boolean statements by means of truth tables.
- Construct the truth tables for given logic circuits (maximum three inputs).
- Design and construct simple logic circuits using AND, OR, NOT, NAND and NOR gates.

9.1 Boolean Logic

In section 4.5.4, we learnt that Boolean values have only two possible values: True and False. The study of Boolean values, called Boolean logic, was developed by the English mathematician George Boole in 1847.

Boolean values are often used to determine the status of a system or a logical condition in an algorithm. For example, the following algorithm produces an output of either "Pass" or "Fail" based on whether the input mark is less than 50. The logical condition of whether "mark < 50" in the decision symbol can be expressed using a Boolean value and must be either True or False at any one time.



▲ **Figure 9.1** An algorithm consisting of a logical condition

Besides True and False, Boolean values can be represented in other ways as shown in Table 9.1.

▼ **Table 9.1** Ways of representing the two Boolean values

0	1
False	True
Off	On
Low	High
No	Yes

Boolean logic is very important as the data in a computer is stored as bits (binary digits) which have a value of either 0 or 1. From Table 9.1, we see that 0 and 1 also correspond to the Boolean values of False and True respectively. Hence, the data in a computer can be represented using Boolean values, and common operations such as addition, subtraction, multiplication and division are actually performed by the computer's processor using Boolean logic.

The inputs and outputs of simple problems and decisions can each be represented using a single bit or Boolean value as they are usually binary in nature. This means that there are only two possible options for each input and output. For instance, in the example at the beginning of the chapter, the input of whether it is raining can only be either True or False, with no other possible options. Similarly, the output of Siti's choice can only be either taking the bus or walking, with no other possible options. For this chapter, we will consider all inputs and outputs to be binary in nature.

9.2 Truth Tables

We use a **truth table** to show the resulting output for every possible combination of inputs. Tables 9.2 to 9.4 show the truth tables for logic gates with one, two and three inputs respectively. Notice that there is only one output column in the truth tables. This is because we will be using these tables for logic gates that have only one output.

Key Term

Truth table

Table that shows the resulting output for every possible combination of inputs

▼ **Table 9.2** Truth table for logic gate with one input

Input A	Output
0	
1	

▼ **Table 9.3** Truth table for logic gate with two inputs

Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

▼ **Table 9.4** Truth table for logic gate with three inputs

Input A	Input B	Input C	Output
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

To construct a truth table, we need to:

1. Determine the number of rows in the truth table, which is equal to 2^n , where n refers to the number of inputs.

For example, if there are two inputs, the truth table will have $2^2 = 4$ rows.

If there are three inputs, the truth table will have $2^3 = 8$ rows.

2. Fill in the input columns in ascending order using the binary number system.

For example, in a three-input system, the inputs start from 000 in the first row, followed by 001 in the second, 010 in the third and so on, up to 111 in the last row.

The output is determined by the type of logic gate, as described in the next section.

9.3 Logic Gates

A **logic gate** is an electrical circuit that performs **logical operations** based on one or more inputs and produces a single output. (Recall the use of logical operators in section 4.6.2.4.)

In this section, you will learn about the five basic types of logic gates (OR, AND, NOT, NOR and NAND) as well as their functions and applications.

9.3.1 OR Gate

The OR gate has two inputs and one output. The logic symbol and Boolean statement representing the OR gate are shown in Table 9.5. Boolean statements are also known as logic statements.

Key Terms

Logic gate

An electrical circuit that performs logical operations based on one or more inputs and produces a single output

Logical operation

An operation that acts on binary inputs to produce an output according to the laws of Boolean logic

▼ Table 9.5 The OR gate

Type	Symbol	Boolean statement
OR		$Q = A \text{ OR } B$

The OR gate performs the OR operation, which gives an output of 1 when at least one of its inputs is 1. The truth table in Table 9.6 illustrates this.

▼ Table 9.6 Truth table of the OR gate

Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	1

9.3.2 AND Gate

The AND gate has two inputs and one output. The logic symbol and Boolean statement representing the AND gate are shown in Table 9.7.

▼ Table 9.7 The AND gate

Type	Symbol	Boolean statement
AND		$Q = A \text{ AND } B$

The AND gate performs the AND operation, which gives an output of 1 only when both inputs are 1. The truth table in Table 9.8 illustrates this.

▼ Table 9.8 Truth table of the AND gate

Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1

9.3.3 NOT Gate

The NOT gate has only one input and one output. The logic symbol and Boolean statement representing the NOT gate are shown in Table 9.9.

▼ Table 9.9 The NOT gate

Type	Symbol	Boolean statement
NOT		$Q = \text{NOT } A$

The small circle at the output of the NOT gate symbol represents an inversion.

The NOT gate performs the NOT operation, which inverts the input to its opposite state. Thus, it is also known as the inverter gate.

The truth table in Table 9.10 illustrates this.

▼ **Table 9.10** Truth table of the NOT gate

Input A	Output Q
0	1
1	0

9.3.4 NOR Gate

The NOR gate has two inputs and one output. The logic symbol and Boolean statement representing the NOR gate are shown in Table 9.11.

▼ **Table 9.11** The NOR gate

Type	Symbol	Boolean statement
NOR		$Q = \text{NOT}(A \text{ OR } B)$

The NOR gate performs the NOR operation, which gives an output of 0 when at least one of its inputs is 1. The truth table in Table 9.12 illustrates this.

▼ **Table 9.12** Truth table of the NOR gate

Input A	Input B	Output Q
0	0	1
0	1	0
1	0	0
1	1	0

An OR gate followed by a NOT gate is equivalent to a NOR gate.



▲ **Figure 9.2** An OR gate followed by a NOT gate is equivalent to a NOR gate

9.3.5 NAND Gate

The NAND gate has two inputs and one output. The logic symbol and Boolean statement representing the NAND gate are shown in Table 9.13.

▼ Table 9.13 The NAND gate

Type	Symbol	Boolean statement
NAND		$Q = \text{NOT}(A \text{ AND } B)$

The NAND gate performs the NAND operation, which gives an output of 0 only when both inputs are 1. The truth table in Table 9.14 illustrates this.

▼ Table 9.14 Truth table of the NAND gate

Input A	Input B	Output Q
0	0	1
0	1	1
1	0	1
1	1	0

An AND gate followed by a NOT gate is equivalent to a NAND gate.



▲ Figure 9.3 An AND gate followed by a NOT gate is equivalent to a NAND gate



Quick Check 9.3

1. State whether each of the statements below is true or false.
 - a) An OR gate returns an output of 1 when either input is 1.
 - b) An AND gate returns an output of 1 only when both its inputs are 1.
 - c) A NOT gate returns an output of 1 when its input is 1.

2. When using a NOR gate, which of the following conditions will return an output of 1?
 - A Both inputs are 0
 - B Both inputs are 1
 - C Either input is 1
 - D None of the above

3. When using a NAND gate, which of the following conditions will return an output of 0?
 - A Both inputs are 0
 - B Both inputs are 1
 - C Either input is 0
 - D None of the above

9.4 Logic Circuit Diagrams

Logic gates are basic building blocks of logic circuits. A **logic circuit** can consist of only one logic gate but we can build more complex and useful logic circuits by connecting two or more logic gates together. A **logic circuit diagram** shows how these logic gates are connected together. It is formed by using symbols to represent logic gates and lines to represent the wires connecting the inputs and outputs of different logic gates together.

9.4.1 Intermediate Inputs

Within a logic circuit, it is common for the output of a logic gate to be also the input of another logic gate. In Figure 9.4, the output of the first OR gate is also one of the inputs of the second OR gate. In this textbook, we will call these “intermediate inputs”.

Key Terms

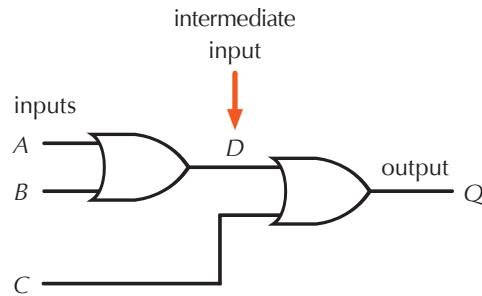
Logic circuit

A circuit that consists of only one logic gate or multiple logic gates connected together

Logic circuit diagram

Visual representation using symbols to show how the component logic gates are connected together in a logic circuit

The logic circuit in Figure 9.4 has three inputs (A , B and C), one intermediate input (D) and one output (Q).



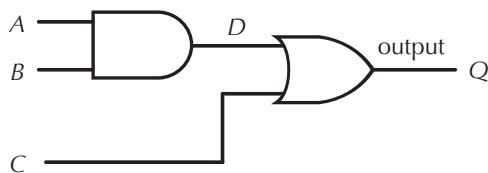
▲ Figure 9.4 Logic circuit with an intermediate input

9.4.2 Constructing Truth Tables from Logic Circuit Diagrams

To generate a truth table from a logic circuit diagram, follow these steps:

1. Determine the total number of columns and rows in the truth table.
Number of columns = total number of inputs, intermediate inputs and output
Number of rows = 2^n , where n is the number of inputs.
2. Draw the truth table and fill in all the input combinations.
3. Work out the intermediate inputs.
4. Work out the final output.

Consider the logic circuit in Figure 9.5. Let us use the steps above to generate its truth table.



▲ Figure 9.5 A logic circuit diagram

Step 1: Determine the total number of columns and rows in the truth table.

Since there are three inputs (A , B and C), one output (Q) and one intermediate input (D):

Number of columns = total number of inputs, intermediate inputs and output = 5
Number of rows = $2^3 = 8$

Step 2: Draw the truth table and fill in all the input combinations.

▼ **Table 9.15** Filling in all the input combinations in the truth table

Input A	Input B	Input C	Intermediate Input D	Output Q
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Step 3: Work out the intermediate input.

Work out the intermediate input D using A and B .

▼ **Table 9.16** Filling in the column for intermediate input D

Input A	Input B	Input C	Intermediate Input D	Output Q
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	
1	1	1	1	

Step 4: Work out the final output.

Work out the final output Q using C and D .

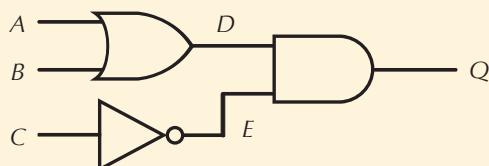
▼ **Table 9.17** Filling in the column for the final output Q

Input A	Input B	Input C	Intermediate Input D	Output Q
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1



Quick Check 9.4

1. Consider the logic circuit below.



▲ **Figure 9.6** A logic circuit

- a) State the number of inputs.
- b) State the number of outputs.

9.5 Boolean Statements

In the previous section, you learnt that a logic gate can be represented by a Boolean statement. Like a logic gate, a logic circuit can also be represented by a Boolean statement. Boolean statements are useful in two ways:

1. They can be evaluated using truth tables to determine the output of a logic circuit.
2. They can determine the arrangement of the logic gates in a logic circuit.

9.5.1 Order of Operations

To evaluate a Boolean statement, the order of operations must always be followed.

In mathematics, you would have learnt about order of operations in evaluating algebraic equations. For instance, in $1 + 2 \times 3$, the multiplication operation is evaluated before the addition operation to give the result of 7. A similar order applies when evaluating Boolean statements. The order of operations for Boolean statements is summarised in Table 9.18.

▼ **Table 9.18** Order of operations for Boolean statements

Order	Logical operation
1	Parentheses
2	NOT
3	AND
4	OR

9.5.2 Constructing Truth Tables from Boolean Statements

Consider the Boolean statement below:

$$Q = (A \text{ AND } B) \text{ OR } (A \text{ AND } C)$$

We can evaluate it by expressing all combinations of its inputs and output using a truth table.

Step 1: Determine the total number of columns and rows in the truth table.

Since there are three inputs (A , B and C), one output (Q) and two intermediate inputs ($(A \text{ AND } B)$ and $(A \text{ AND } C)$):

$$\begin{aligned} \text{Number of columns} &= \text{total number of inputs, intermediate inputs and output} = 6 \\ \text{Number of rows} &= 2^3 = 8 \end{aligned}$$

Using the order of operations, the parts of the Boolean statement in parentheses are to be evaluated first, followed by the part with the OR operation. This means that “ $A \text{ AND } B$ ” as well as “ $A \text{ AND } C$ ” are evaluated first, from left to right, since they are contained within parentheses. They form the two intermediate inputs, D and E .

$$Q = \overbrace{(A \text{ AND } B)}^{\text{Order 1}} \text{ OR } \overbrace{(A \text{ AND } C)}^{\text{Order 2}} \underbrace{\text{OR}}_{\text{Order 3}}$$

▲ **Figure 9.7** Evaluating a Boolean statement following the order of operations

Step 2: Draw the truth table and fill in all the input combinations.

▼ **Table 9.19** Filling in the input combinations

Input A	Input B	Input C	Intermediate Input D (A AND B)	Intermediate Input E (A AND C)	Output Q
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Step 3: Work out the intermediate inputs.

To work out the intermediate input D , use A and B .

To work out the intermediate input E , use A and C .

▼ **Table 9.20** Filling in the intermediate inputs D and E

Input A	Input B	Input C	Intermediate Input D (A AND B)	Intermediate Input E (A AND C)	Output Q
0	0	0	0	0	
0	0	1	0	0	
0	1	0	0	0	
0	1	1	0	0	
1	0	0	0	0	
1	0	1	0	1	
1	1	0	1	0	
1	1	1	1	1	

Step 4: Work out the final output.

Work out the final output Q using D and E .

▼ **Table 9.21** Filling in the final output Q

Input A	Input B	Input C	Intermediate Input D (A AND B)	Intermediate Input E (A AND C)	Output Q
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

9.5.3 Constructing Boolean Statements from Truth Tables

In the previous section, we learnt how to construct a truth table from a Boolean statement. In many situations, it is also useful to do the reverse, which is to construct a Boolean statement from a truth table.

The Boolean statement can be formed by creating an AND term for each input combination with a final output of 1, and then ORing all the AND terms together.

We will illustrate this using the truth table in Table 9.22.

▼ **Table 9.22** Truth table used to derive the Boolean statement

Input A	Input B	Input C	Output Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Step 1: Determine the combinations which result in an output of 1.

From the truth table in Table 9.22, we can see that there are two combinations which result in output $Q = 1$:

- $A = 0, B = 1$ and $C = 1$
- $A = 1, B = 1$ and $C = 0$

Step 2: Create an AND term for each combination that results in an output of 1.

Since these conditions need to be all present for output Q to be 1, we should AND them together. Note that NOT A is used to represent $A = 0$. This is because when $A = 0$, NOT A will give 1.

- ((NOT A) AND B) AND C
- (A AND B) AND (NOT C)

Step 3: OR the AND terms together.

Since either of these two input combinations will result in an output of 1, we should join the partial Boolean statements together using OR. Thus we obtain the final Boolean statement below:

$$Q = ((\text{NOT } A) \text{ AND } B) \text{ AND } C \text{ OR } (A \text{ AND } B) \text{ AND } (\text{NOT } C)$$

This form of representation where the AND terms are ORed together is known as **Sum-of-Product (SOP)**.

Key Term

Sum-of-Product (SOP)
Boolean statement
where the AND terms
are ORed together

9.5.4 Constructing Logic Circuit Diagrams from Boolean Statements

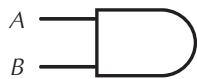
To determine how logic circuits look like, we should use the order of operations to evaluate the Boolean statements.

Let us draw the logic circuit diagram represented by this Boolean statement:

$$Q = (A \text{ AND } B) \text{ OR } C$$

Recall that in Table 9.18, the AND operation is evaluated before the OR operation. This means that the logic gate representing “A AND B” should be drawn first. The output of “A AND B” is then connected as an input to an OR gate with input C.

Step 1: Draw an AND gate with inputs A and B.



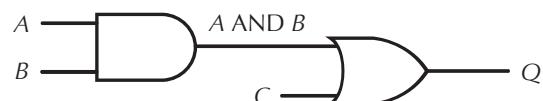
▲ **Figure 9.8** Drawing the AND gate first

Step 2: Connect the output of “A AND B” as an input to an OR gate.



▲ **Figure 9.9** Connecting the output of the AND gate to the OR gate

Step 3: Connect C to the other input of the OR gate.



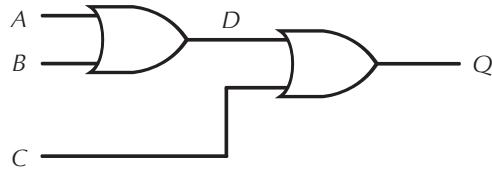
▲ **Figure 9.10** Complete logic circuit diagram of $Q = (A \text{ AND } B) \text{ OR } C$

9.5.5 Constructing Boolean Statements from Logic Circuit Diagrams

In the previous section, we learnt to draw logic circuit diagrams from Boolean statements. In this section, we will learn how to do the reverse, which is to derive the Boolean statement from a logic circuit diagram.

Example 1:

Let us work out the Boolean statement of the logic circuit diagram shown in Figure 9.11.

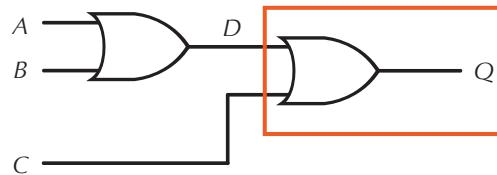


▲ **Figure 9.11** Deriving the Boolean statement from a logic circuit diagram

To work out the Boolean statement of a logic circuit diagram, we can work backwards from its final output.

Step 1: Work out the Boolean statement involving the final output.

An OR gate with inputs D and C is connected to the final output Q .



▲ **Figure 9.12** Working backwards from the final output

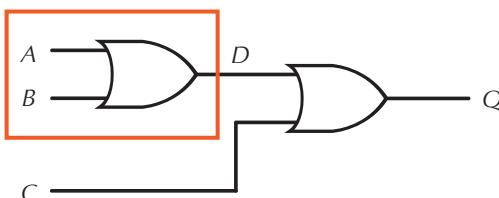
Hence, the Boolean statement for this OR gate is:

$$Q = D \text{ OR } C$$

Note that D is an intermediate input.

Step 2: Work out the Boolean statement involving the intermediate input.

Notice that the intermediate input D is also the output D of an OR gate with inputs A and B .



▲ **Figure 9.13** Working backwards from the intermediate input D

Hence, the Boolean statement for this OR gate is:

$$D = A \text{ OR } B$$

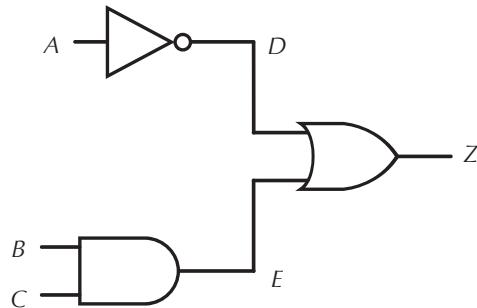
Step 3: Combine the Boolean statements.

Substituting the Boolean statement from Step 2 into the statement from Step 1 will give us the Boolean statement which represents the logic circuit diagram:

$$Q = (A \text{ OR } B) \text{ OR } C$$

Example 2:

Let us now work out the Boolean statement of the logic circuit diagram in Figure 9.14.

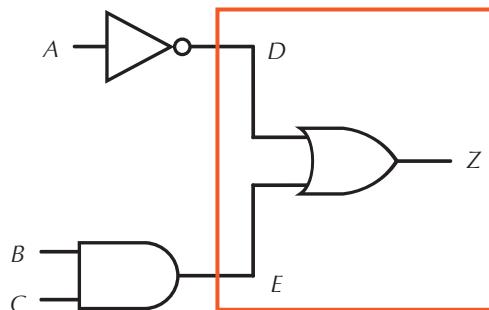


▲ **Figure 9.14** Deriving the Boolean statement from a logic circuit diagram

Again, we start by working backwards from the final output.

Step 1: Work out the Boolean statement involving the final output.

An OR gate with inputs D and E is connected to the final output Z .



▲ **Figure 9.15** Working backwards from the final output Z

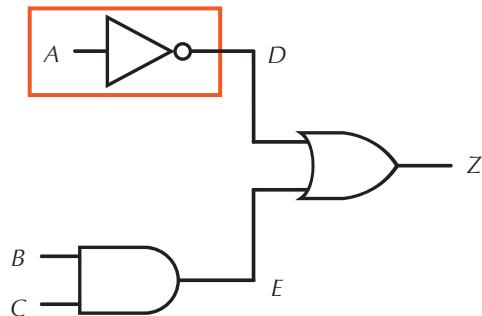
Hence, we can work out the Boolean statement for this OR gate as:

$$Z = D \text{ OR } E$$

Note that D and E are intermediate inputs.

Step 2: Work out the Boolean statements involving the intermediate inputs.

Notice that the intermediate input D is also the output D of a NOT gate with input A .

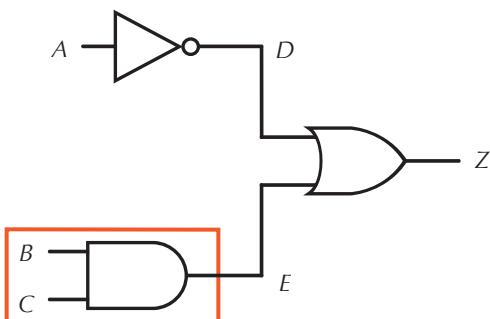


▲ **Figure 9.16** Working backwards from the intermediate input D

Hence, the Boolean statement for this NOT gate is:

$$D = \text{NOT } A$$

Next, notice that the intermediate input E is also the output E of an AND gate with inputs B and C .



▲ **Figure 9.17** Working backwards from the intermediate input E

Hence, the Boolean statement for this AND gate is:

$$E = B \text{ AND } C$$

Step 3: Combine the Boolean statements.

Substituting the Boolean statements from Step 2 into the statement from Step 1 will give us the Boolean statement which represents the logic circuit diagram:

$$Z = (\text{NOT } A) \text{ OR } (B \text{ AND } C)$$



Quick Check 9.5

1. Refer to the Boolean statement below.

$$Q = ((\text{NOT } A) \text{ OR } B) \text{ AND } C$$

- a) Draw the truth table by evaluating the statement.
- b) Draw the logic circuit diagram that represents the statement.

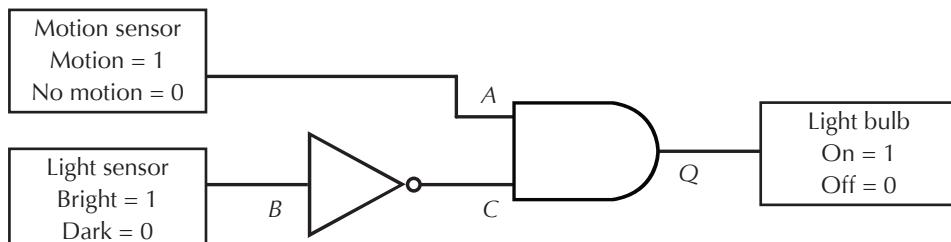
9.6 Applications of Logic Circuits

Computing devices and electrical appliances perform logic decisions using logic circuits. This section introduces three applications of logic circuits that are used to make logic decisions.

9.6.1 Motion-Sensing Lights

In toilets or offices installed with motion-sensing lights, the logic circuit is designed to make a logic decision of when to switch on the lights depending on the input conditions.

One example of a logic circuit used to operate motion-sensing lights is found in Figure 9.18.



▲ Figure 9.18 Logic circuit for motion-sensing light

The corresponding Boolean statement for this logic circuit is $Q = A \text{ AND } (\text{NOT } B)$.

▼ Table 9.23 Truth table for $Q = A \text{ AND } (\text{NOT } B)$

Input A	Input B	Intermediate Input C	Output Q
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

The motion-sensing light is switched on when both of its input conditions are fulfilled – its surroundings are dark, and motion is detected within the toilet or office area. At all other times, the light is switched off. This ensures that the light is not switched on when no one is around, helping to reduce electricity consumption.

When the logic circuit is used to control a motion-sensing light, inputs *A* and *B* are provided by a motion sensor and a light sensor respectively, and output *Q* is connected to a light bulb. Table 9.24 shows how the truth table should be interpreted for this circuit.

▼ **Table 9.24** Truth table for the logic circuit of a motion-sensing light

Input A	Input B	Intermediate Input C	Output Q
0 (no motion)	0 (dark)	1	0 (light off)
0 (no motion)	1 (bright)	0	0 (light off)
1 (motion)	0 (dark)	1	1 (light on)
1 (motion)	1 (bright)	0	0 (light off)

9.6.2 Car Door Lock

A car door can be unlocked either manually with a key or by receiving an unlock signal via a remote control sensor. The key switch provides an input of 1 when the key is turned. The remote control switch provides an input of 1 when the unlock signal is detected.

We can then work out that there are two types of inputs (the key switch and remote control sensor), resulting in $2^2 = 4$ possible outputs for the lock.

Let us work out the truth table for the car door lock. If we connect inputs *A* and *B* to a key switch and remote control sensor respectively, and the output *Q* to the lock, we can create the truth table shown in Table 9.25.

▼ **Table 9.25** Truth table for the car door lock

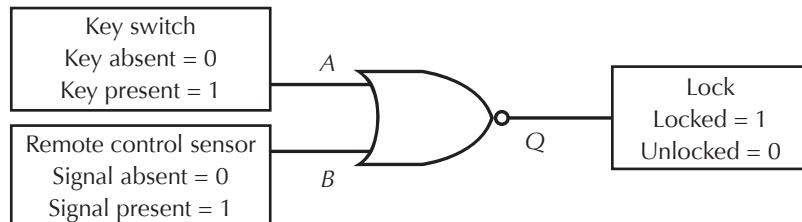
Input A	Input B	Output Q
0 (key absent)	0 (signal absent)	1 (locked)
0 (key absent)	1 (signal present)	0 (unlocked)
1 (key turned)	0 (signal absent)	0 (unlocked)
1 (key turned)	1 (signal present)	0 (unlocked)

The Boolean statement for this car door lock is:

$$Q = \text{NOT } (A \text{ OR } B)$$

This is equivalent to using a NOR gate.

Hence, the logic circuit for this car door lock can be designed as follows:

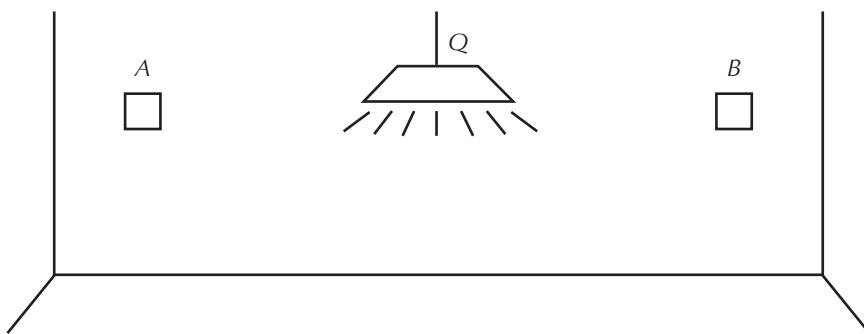


▲ **Figure 9.19** Logic circuit for car door lock

9.6.3 Two-Way Switches

Two-way switches allow a single light to be switched on or off from two different locations easily. Two-way switches are often used at the opposite ends of corridors or staircases to control a light that is located between the switches.

In Figure 9.20, the light Q at the centre of a long corridor is controlled by switches A and B at both ends of the corridor. Each switch has two states: 0 for “off” and 1 for “on”. Light Q also has two states: 0 for “off” and 1 for “on”. Light Q will be on only when both switches A and B are off or on concurrently. However, if only one switch is on and the other is off, light Q will be off.



▲ **Figure 9.20** Light Q is controlled by switches A and B , which are placed at both ends of the corridor

If we connect the inputs of the logic circuit to switches A and B and the output to light Q , we can create the truth table shown in Table 9.26.

▼ **Table 9.26** Truth table for the two-way switches

Input A	Input B	Output Q
0	0	1
0	1	0
1	0	0
1	1	1

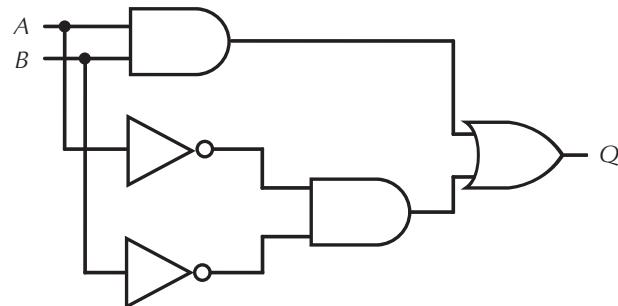
To derive the Boolean statement from a truth table, recall the following two steps:

1. Create an AND term for each input combination that results in an output of 1.
2. OR the AND terms together.

The Boolean statement for this two-way switch will be:

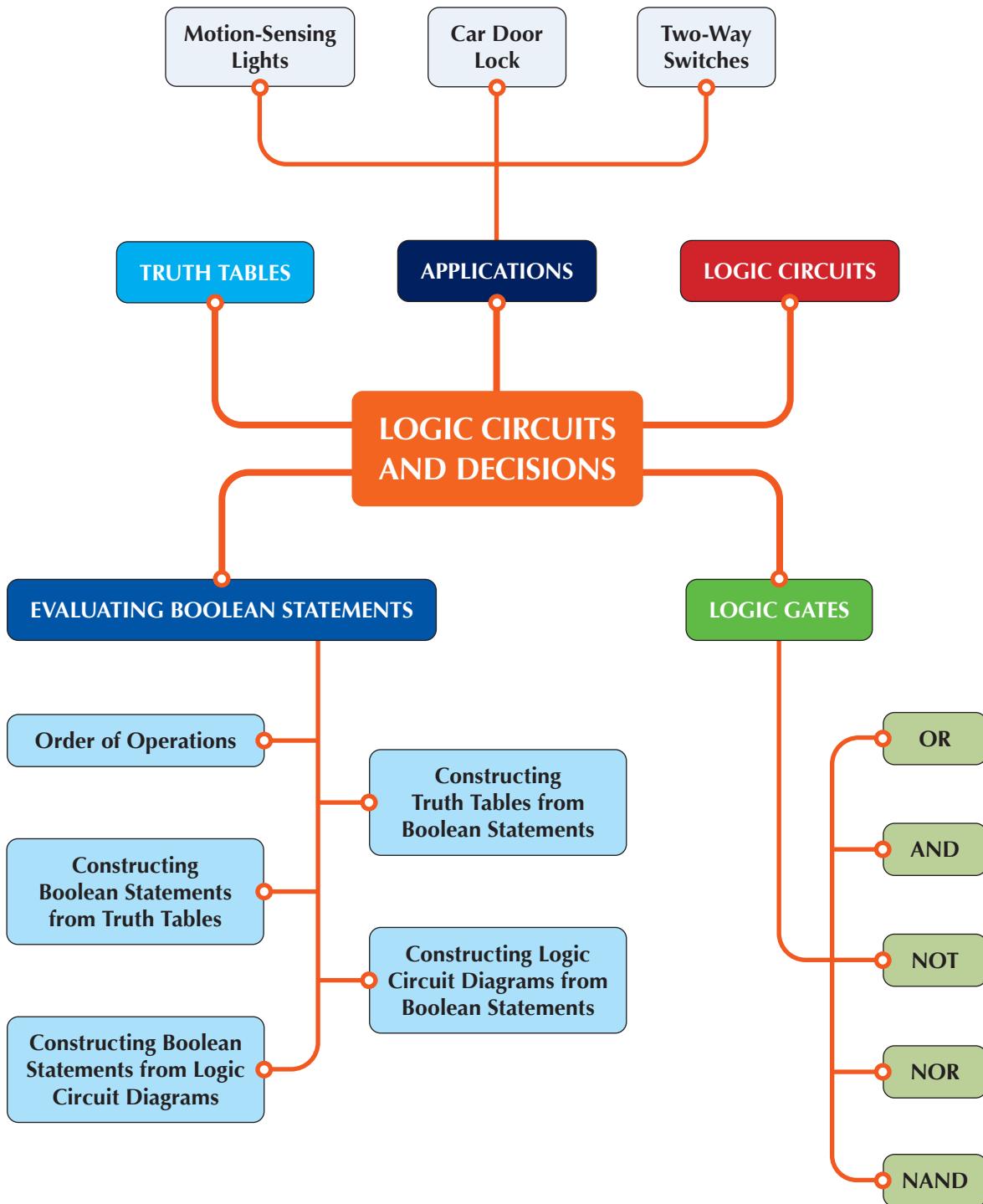
$$Q = (A \text{ AND } B) \text{ OR } ((\text{NOT } A) \text{ AND } (\text{NOT } B))$$

Hence, the logic circuit for the two-way switches can be designed as follows:



▲ **Figure 9.21** Logic circuit for the two-way switches

Chapter Summary



Review Questions

1. Draw the truth tables for the following Boolean statements.

- a) $Q = A \text{ OR } (B \text{ AND } C)$
- b) $X = (\text{NOT } A) \text{ AND } (\text{NOT } B)$
- c) $Y = A \text{ AND } (\text{NOT } B)$
- d) $Z = (A \text{ OR } B) \text{ AND } (C \text{ OR } D)$

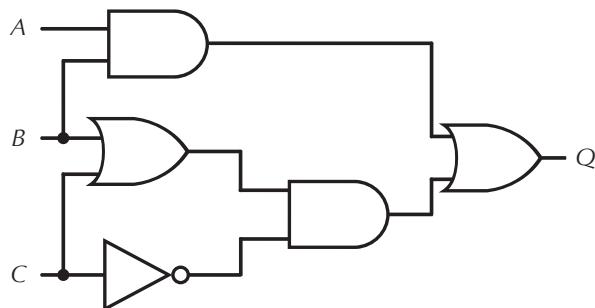
2. For the statement:

If A is True then Q is True
else if B is True then Q is True

determine its:

- a) Boolean statement; and
- b) truth table.

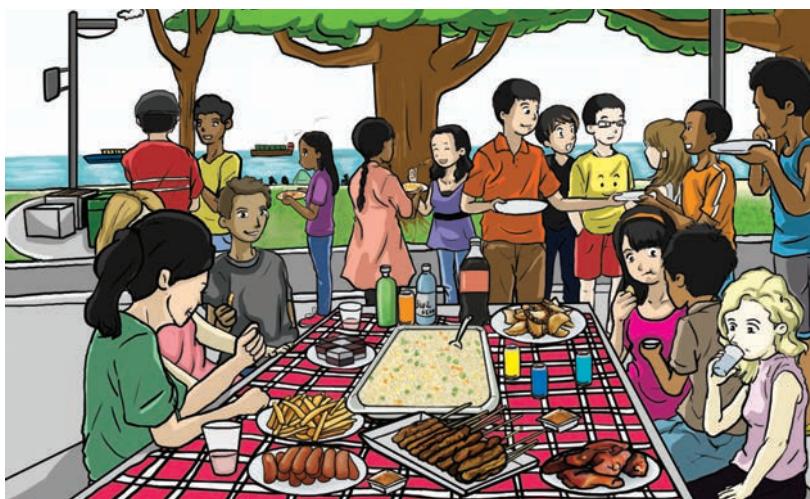
3. Draw the truth table for the logic circuit shown below.



▲ Figure 9.22 Logic circuit

How are Spreadsheets Used to Process and Analyse Data?

To celebrate the end of their examinations, Alex, Bala, Siti and their classmates decide to hold a party. After considering a number of options and their respective costs, they agree on a potluck party. Each person in the class would contribute one dish and also a small amount of money to pay for other party items. The students also agree to invite their family members to the party.



Alex, Bala, Siti and their classmates have to decide on the type of party to organise, manage the costs of various party items and track the contributions made by attendees to the party. These tasks may be difficult as there is a large amount of data that needs to be collected, processed and analysed.

In the previous chapters, you learnt how computer programs can be written to help solve such problems. A spreadsheet is a type of program that is useful for solving problems that involve a large amount of data.

In this chapter, you will learn about the basics of using spreadsheets as well as how they can be used to analyse data in various statistical contexts. You will also learn how spreadsheets can be used to search for the best answer to meet a particular goal and highlight details in a large set of data.



By the end of this chapter, you should be able to:

- Tabulate data under appropriate column headings (i.e., data field names) and data types (e.g., numeric, text and date).
- Use mathematical operators, functions and what-if data analysis (goal seeking) to prepare spreadsheets and solve real-world problems such as:
 - Find the total of a list of numbers, average of a list of numbers, minimum/maximum of a list of numbers, square root of a number, simple interest and remainder after division of numbers
 - Round values
 - Randomise values
 - Convert data from one type to another (e.g., getting integer values from decimal values)
 - Count the number of data items
- Understand and use conditional statements (simple and nested): COUNTIF and IF with relational and Boolean operators such as AND, NOT and OR.
- Use functions effectively to look up data in rows or columns (horizontal and vertical table lookups) in a list or table for data processing.
- Use absolute and relative cell addressing.

10.1 Understanding Spreadsheets

A **spreadsheet** is an electronic worksheet used to manage and manipulate data arranged in columns and rows.

Columns are labelled with letters and rows with numbers. The rectangular space located in a particular column and row is called a **cell**.

Key Terms

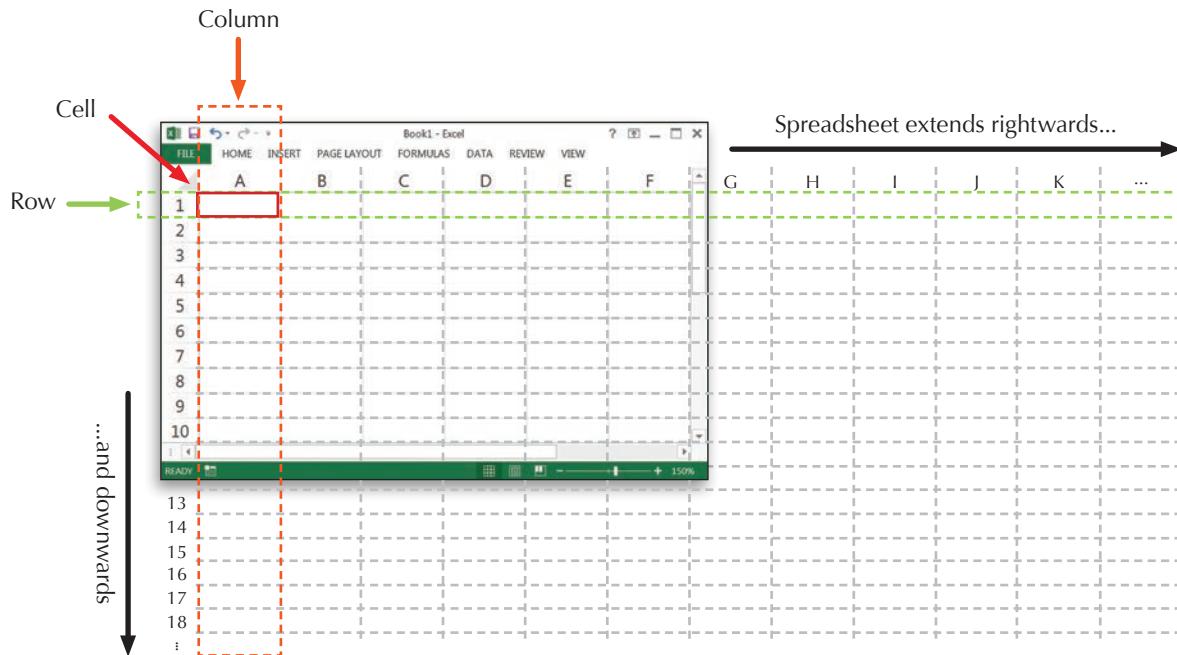
Cell

Space for data in a particular column and row of a spreadsheet

Spreadsheet

Electronic worksheet used to manage and manipulate data arranged in columns and rows

Figure 10.1 shows a typical spreadsheet with a grid of cells that starts from column A and row 1 at the top-left corner and extends rightwards and downwards.



▲ **Figure 10.1** A spreadsheet starts from cell A1 at the top-left corner and extends rightwards and downwards

10.1.1 Cell and Range Addresses

We often need to refer to the location of a cell or range of cells in a spreadsheet. For a single cell, we use its **cell address**, which is the cell's column name (one or more letters) followed immediately by its row number.

For instance, each cell in Figure 10.2 displays its cell address.

Key Term

Cell address

Location name for a cell formed by its column name and row number

	A	B	C
1	A1	B1	C1
2	A2	B2	C2
3	A3	B3	C3
4	A4	B4	C4
5	A5	B5	C5
6	A6	B6	C6
7	A7	B7	C7
8	A8	B8	C8
9	A9	B9	C9
10	A10	B10	C10

▲ **Figure 10.2** Cells and their cell addresses

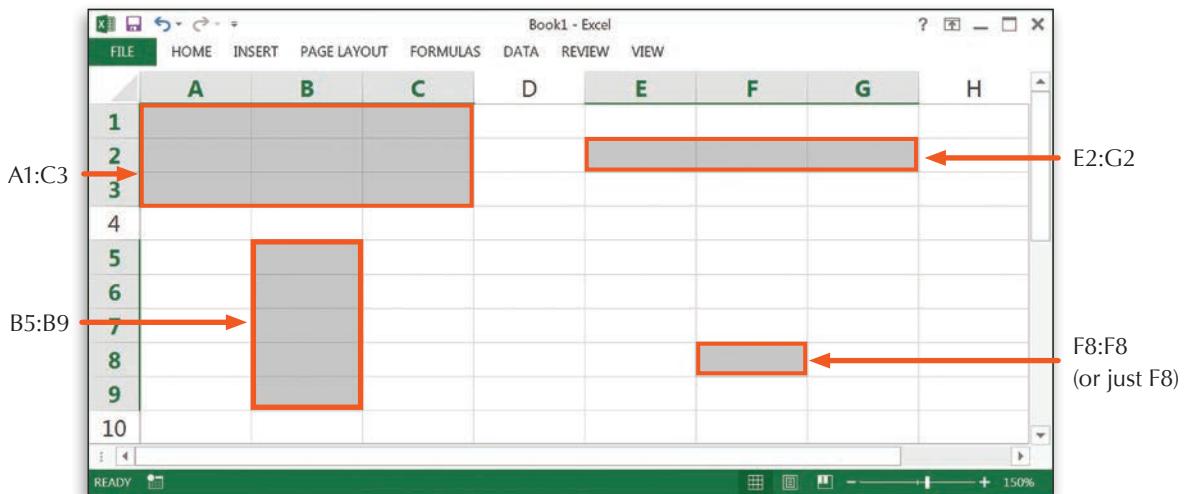
When we want to refer to a rectangular range of cells, we use a **range address** instead. A range address is the top-left cell's address followed by a colon (:) and the bottom-right cell's address.

Figure 10.3 shows the different range addresses for various selections of cells in a spreadsheet. For instance, "A1:C3" is the range address for a 3-by-3 range of cells in the top-left corner of a spreadsheet.

Key Term

Range address

Location name for a rectangular range of cells formed by the top-left cell's address, a colon (:) and the bottom-right cell's address

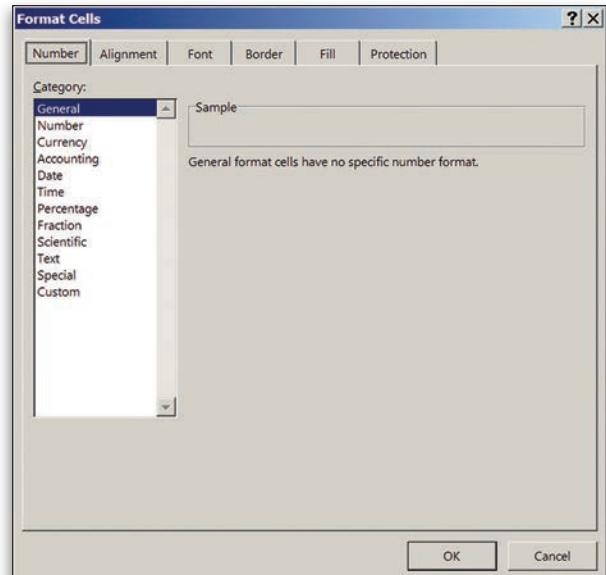


▲ Figure 10.3 Range addresses

10.1.2 Data Types

In Python, each value has a data type such as `bool`, `float`, `int` or `str`. In spreadsheets, cells also have data types.

Figure 10.4 shows the built-in data types available in a typical spreadsheet application. To set the data type of a particular cell, right-click on it and select "Format Cells..." from the context menu.



▲ Figure 10.4 Built-in data types in a typical spreadsheet application

Some common data types are described in Table 10.1.

▼ **Table 10.1** Common data types

Data type	Description	Examples
General	Used for data with no specific type Typically used if the type of data cannot be known ahead of time	2017 19.65 Hello, World!
Number	Used for numbers The number of decimal places to display and the type of digit separator (if any) can be specified Most spreadsheet applications do not distinguish between integers and floating-point numbers	2017 19.65 -0.05
Currency	Used for money Typically displayed with a currency symbol and two decimal places; the type of currency symbol can be specified (the default is the dollar sign (\$))	\$20.17 \$2017.00 \$0.17
Date	Used for dates Can be displayed and entered in a variety of styles depending on the spreadsheet application	1-Jan-17 1/1/2017 2017-01-01
Time	Used for times and durations Can be displayed and entered in a variety of styles depending on the spreadsheet application	0:20 20:17:00 10:00 AM 10:00:00 AM
Percentage	Used for rates, ratios and proportions Typically displayed and entered as a number with a "%" appended; treated as the number divided by 100 when used in calculations, e.g., 5% * 20 will calculate 0.05 * 20, which is equal to 1	5% 20.17% 0.05%
Text	Used for textual information Typically displays exactly what is entered To force a cell to treat an input as text, start it with an apostrophe ('), e.g., '20-12	Hello, World! 20-12
Logical value	Used for Boolean data Must be either TRUE or FALSE	TRUE FALSE

Most spreadsheet programs can automatically detect and change the data type of a cell based on what is entered. For instance, entering “\$2017” into a cell with a General data type will automatically change it to use a Currency data type instead. However, the detected data type may not always be what is intended by the user.

For instance, a user may want to enter “1–1” into a cell as Text, but if the cell initially has a General data type, it will automatically be converted to use a Date data type instead. This can be avoided by setting the cell to use a Text data type beforehand, or by entering an apostrophe in front of the data like this: '1–1. Thus, it is a good practice to always set the data type of cells to the most appropriate option before entering any values. You will learn how to do this as part of data collection in section 10.2.1.

Did you know?

Some spreadsheet applications such as Microsoft Excel do not consider Currency and Date as data types. Instead, they are treated as number formats that determine how a cell is displayed, but not how it is actually stored. The difference between data types and number formats in Microsoft Excel is not covered in this textbook.

10.1.3 Formulas and Automatic Recalculation

In Python, calculations are performed by providing the computer with step-by-step instructions. In spreadsheets, however, calculations are performed by telling the computer how cell values are related to each other. This is done by entering a **formula** into a cell instead of a **constant**. In spreadsheets, a formula is the result of entering an equals sign (=) followed by an expression that can contain functions, operators as well as values from other cells.

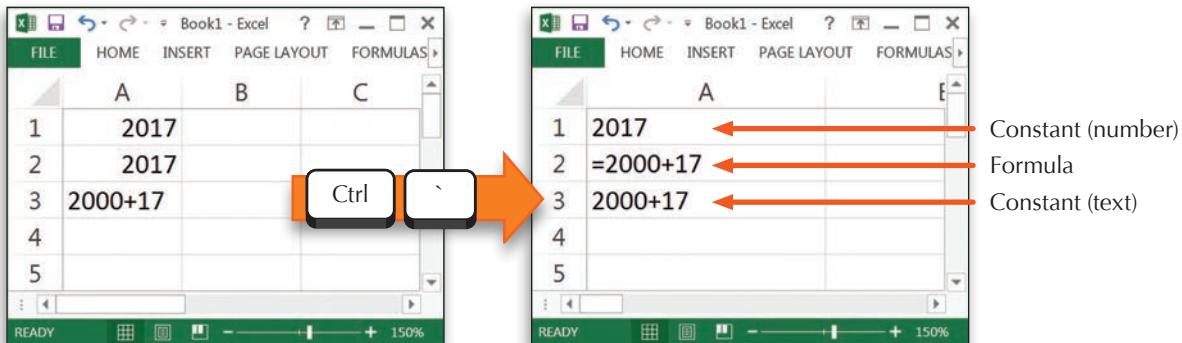
To reveal the cells which have formulas, press the Ctrl key followed by the grave accent (`) key, i.e., $\text{Ctrl}+\text{`}$. The grave accent key is usually located to the left of the digit 1 on the keyboard. Instead of showing the calculated result, cells with formulas will now show an equals sign followed by the cell’s formula. To show the calculated values once more, press $\text{Ctrl}+\text{`}$ again.

Key Terms

Constant (spreadsheet)
Non-calculated value that is entered directly into a cell and does not change

Formula
Performs calculations on the data in a spreadsheet; may contain functions, operators as well as cell and range references

For example, in Figure 10.5, cell A1 has a constant with a value of 2017 while cell A2 has a formula that calculates the sum of 2000 and 17. Cell A3, on the other hand, has a constant with a text value of “2000+17” and does *not* have a formula. This is because the expression “2000+17” was entered in cell A3 without an equals sign in front.



▲ Figure 10.5 Example of formulas and constants

To describe how cell values are related to each other, formulas use **cell references**, which specify the column and row of another cell. There are two types of cell references: relative cell references and absolute cell references.

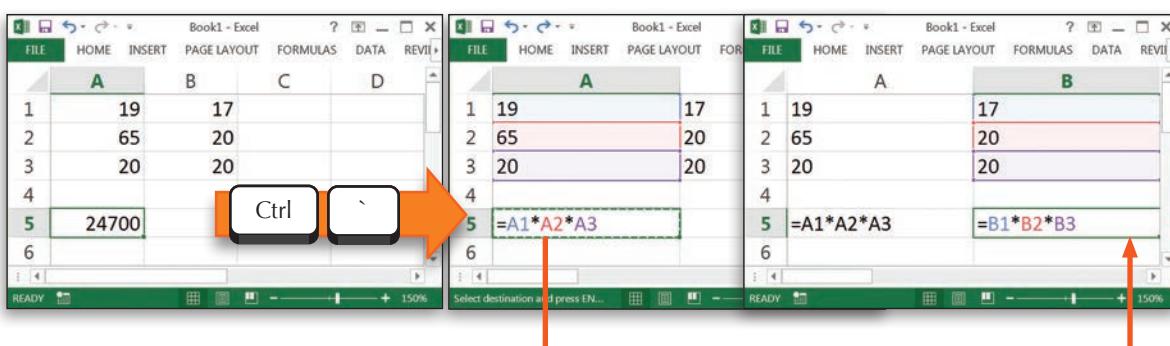
Key Term

Cell reference

Description of a cell that can be used in a spreadsheet formula

10.1.3.1 Relative Cell References

To make a relative cell reference, use a cell address or range address directly in a formula. For instance, in Figure 10.6, cell A5 uses relative cell references to calculate the product of the values in cells A1, A2 and A3.



Cell A5 uses relative cell references to calculate the product of the values in cells A1, A2 and A3.

When cell A5 is copied to cell B5, the cell references in the formula automatically change to B1, B2 and B3.

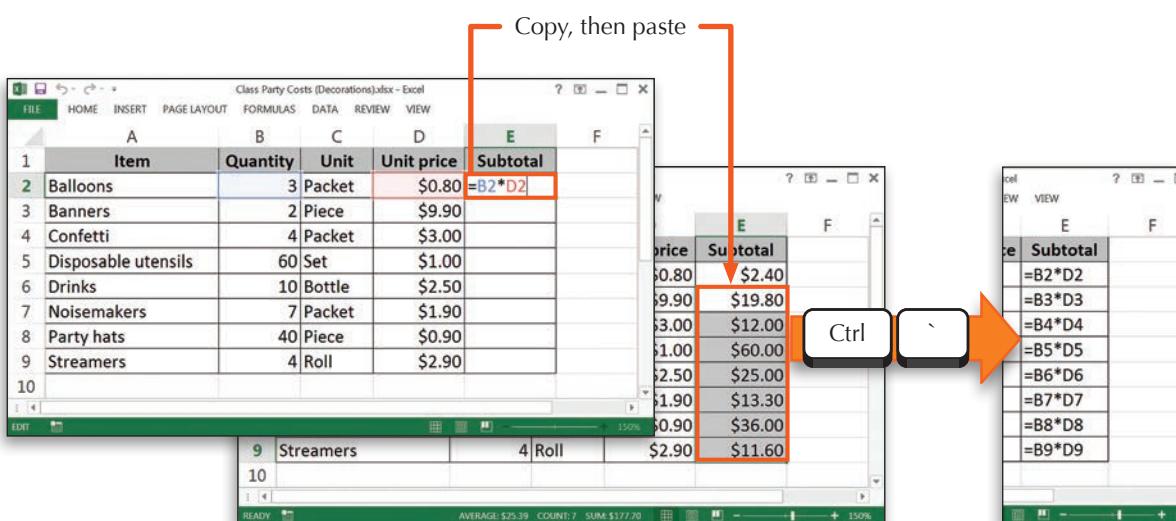
▲ Figure 10.6 Using relative cell references

An important feature of relative cell references is that when a formula from one cell is copied to another cell, all the relative cell references in the copied formula are changed so that they maintain the same relative positions to the new cell.

In Figure 10.6, when cell A5 is copied to cell B5, the resulting formula in cell B5 automatically has its relative cell references change from A1, A2 and A3 to B1, B2 and B3 respectively. This allows repetitive calculations or relationships between cell values to be set up quickly by copying the formula from one cell to multiple cells.

Suppose Bala is helping to manage the costs for the party described at the beginning of this chapter using the spreadsheet shown in Figure 10.7. In row 2, the unit cost of balloons is \$0.80 (cell D2), so the total cost for balloons is \$0.80 (cell D2) \times 3 (cell B2) = \$2.40.

Bala notices that the formula to calculate the total cost for subsequent rows follows the same pattern of multiplying the quantity in column B by the unit cost in column D. Hence, instead of entering the formula for rows 3 to 10 manually, he enters the formula “=B2*D2” into cell E2 and copies it into cells E3 to E9.

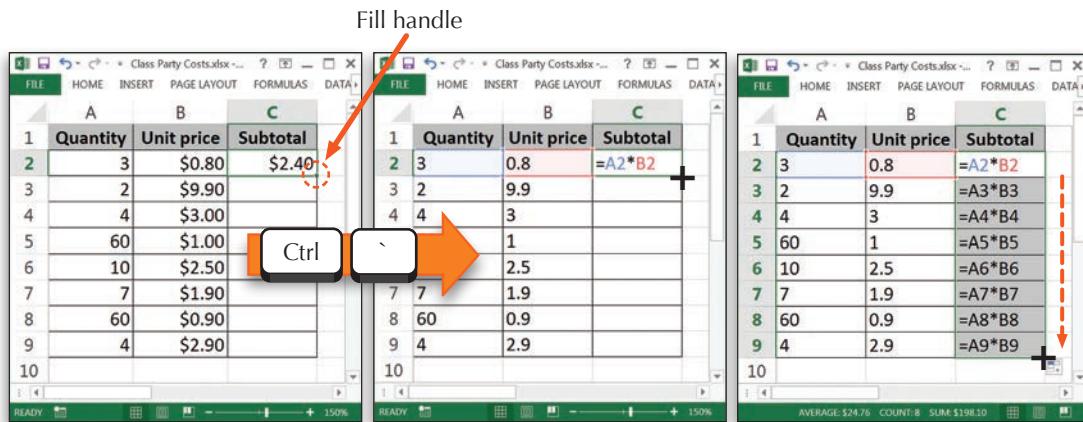


▲ Figure 10.7 Using relative cell references to calculate the total cost of party items

In Figure 10.7, the two relative cell references of B2 and D2 are automatically changed to B3 and D3 for row 3, B4 and D4 for row 4, B5 and D5 for row 5, and so on for the subsequent rows. Note that this occurs automatically to all relative cell references in the copied formula.

Did you know?

An alternative to copying and pasting formulas manually is to use the “fill” feature of a spreadsheet. Select the cell you wish to copy from. Notice the square on the bottom-right corner of the cell. This is called the fill handle. Drag this square to cover the adjacent cells that you wish to paste the formula into.



▲ Figure 10.8 Dragging the fill handle (indicated by the small square) to copy C2 into C3:C9

10.1.3.2 Absolute Cell References

Absolute cell references are used when we do not want the cell references in formulas to change when copied to other cells. To make an absolute cell reference, type a dollar sign (\$) before the column letter or row number that should not change. This indicates that the particular column letter or row number will *not* be automatically changed when the formula is copied into other cells. For instance, in Figure 10.9, “\$A\$1” is an absolute cell reference that means the cell reference to cell A1 does not change even when the formula is copied to another cell.

Cell A5 uses absolute cell references to calculate the product of the values in cells A1, A2 and A3.

When cell A5 is copied to cell B5, the cell references in the formula remain unchanged.

▲ Figure 10.9 Using absolute cell references

Suppose Alex is helping to track the contributions made by the attendees for the class party using the spreadsheet shown in Figure 10.10. The amount each attendee needs to contribute is \$12.00 (cell C1).

Class Party Contributions.xlsx - Excel							
FILE	HOME	INSERT	PAGE LAYOUT	FORMULAS	DATA	REVIEW	VIEW
A	B	C	D				
1	Contribution per attendee:	\$12.00					
2							
3	Name	Additional family members	Amount				
4	Aisha	1					
5	Alex	2					
6	Bala	3					
7	Denise	1					
8	Farhan	4					
9	Gopi	5					
10	Irfan	1					
11	Jun Ming	4					
12	Lily	3					
13	Mei Ling	5					
14	Muthu	4					
15	Nurul	5					
16	Priya	5					
17	Siti	3					
18							

▲ **Figure 10.10** Spreadsheet to track contributions from class party attendees

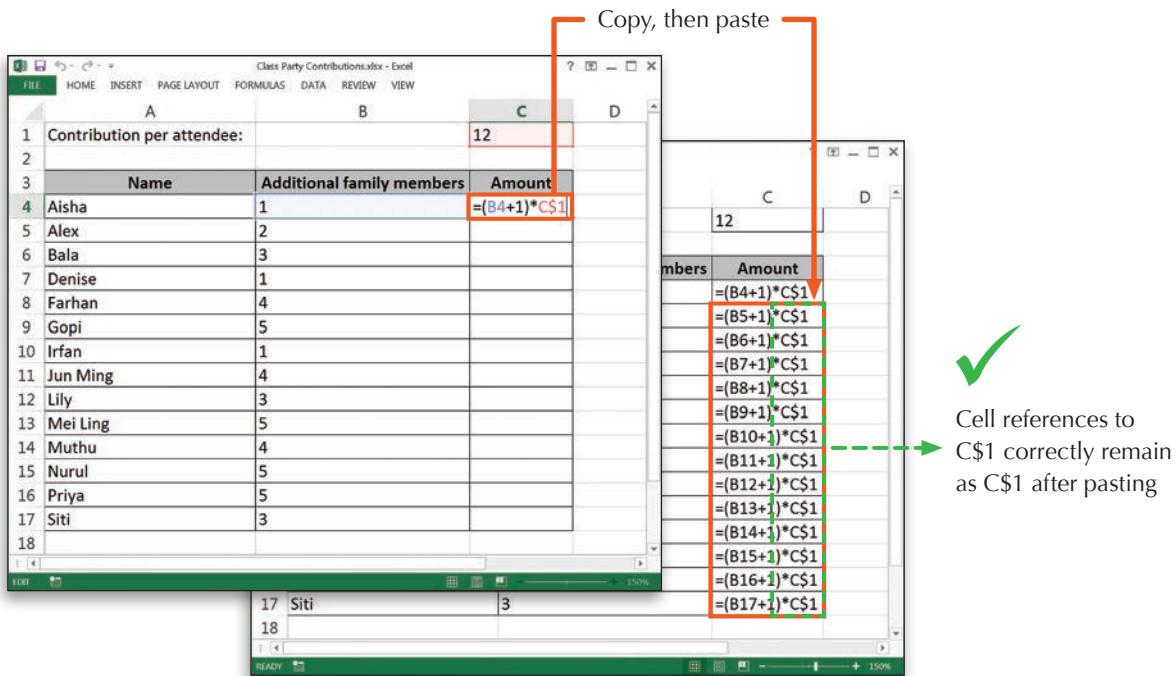
The number of additional family members attending for Aisha is 1 (cell B4), so the total contribution by Aisha and her family is $2 \times \$12.00$ (cell C1) = \$24.00.

Without absolute cell references, Alex may enter the formula “=(B4+1)*C1” into cell C4. However, Figure 10.11 shows that copying this formula into cells C5 to C17 will result in “=(B5+1)*C2” for row 5, “=(B6+1)*C3” for row 6, “=(B7+1)*C4” for row 7, and so on. This is incorrect as the formulas in these cells should only refer to the cell C1.

	A	B	C	D
1	Contribution per attendee:		12	
2				
3	Name	Additional family members	Amount	
4	Aisha	1	=B4+1)*C1	
5	Alex	2		
6	Bala	3		
7	Denise	1		
8	Farhan	4		
9	Gopi	5		
10	Irfan	1		
11	Jun Ming	4		
12	Lily	3		
13	Mei Ling	5		
14	Muthu	4		
15	Nurul	5		
16	Priya	5		
17	Siti	3		
18				

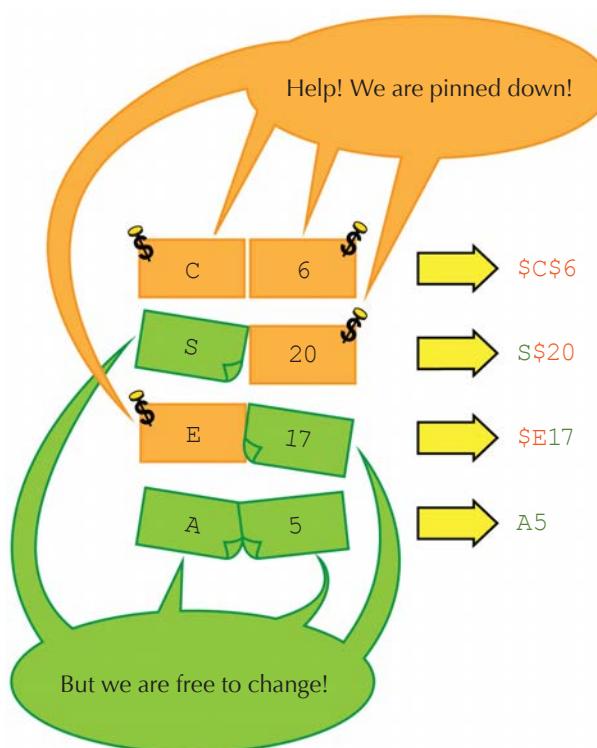
▲ **Figure 10.11** The cell reference to C1 should not change when copied to C5:C17

To prevent the row number 1 from changing in the cell reference “C1”, Alex should use the absolute cell reference “C\$1” instead. The dollar sign in front of “1” indicates that it will not be automatically changed when the formula is copied into other cells. Hence, Alex can enter the formula “=(B4+1)*C\$1” into cell C4 before copying the formula into cells C5 to C17, as shown in Figure 10.12. This results in the correct formulas “=(B5+1)*C\$1” for row 5, “=(B6+1)*C\$1” for row 6, “=(B7+1)*C\$1” for row 7, and so on. Note that since the column letter C should not change as well, he can also use “\$C\$1”.



▲ **Figure 10.12** Using C\$1 prevents the cell reference's row number from being changed when copied to other cells

You can think of the dollar sign as a pin or lock that prevents the column letter or row number in the cell reference from being changed when copied to other cells.



▲ **Figure 10.13** The dollar sign acts like a pin to prevent the column letter or row number from being changed

Did you know?

While editing a cell reference, you can press F4 to cycle through all the possible relative and absolute versions of the cell reference. For instance, pressing F4 while editing the formula “=C2” will cycle through “=\$C\$2”, “=C\$2”, “=\$C2” and back to “=C2” again.

10.1.3.3 Range References

Cell references only refer to one cell at a time. However, some functions may refer to a rectangular range of cells. To do this, we use a **range reference** instead, which is the cell reference for the top-left cell in the range followed by a colon (:) and the cell reference for the bottom-right cell in the range.

Like cell references, there are also relative and absolute range references. For instance “A1:C3” is a relative range reference while “\$A\$1:\$C\$3” is an absolute range reference.

Key Term

Range reference

Description of multiple cells that can be used in a spreadsheet formula

10.1.3.4 Automatic Recalculation

The advantage of using cell and range references is that the spreadsheet can automatically recalculate the formulas in all affected cells if any referenced cells are updated.

For instance, in Bala’s situation in Figure 10.14, updating the unit cost for balloons (cell D2) from \$0.80 to \$0.50 will automatically recalculate the total cost for balloons (cell E2), which would change from \$2.40 to \$1.50. This occurs because the spreadsheet already understands the relationship between cells D2 and E2.

After updating the unit price in D2 from \$0.80 to \$0.50...

...the subtotal in E2 is automatically recalculated

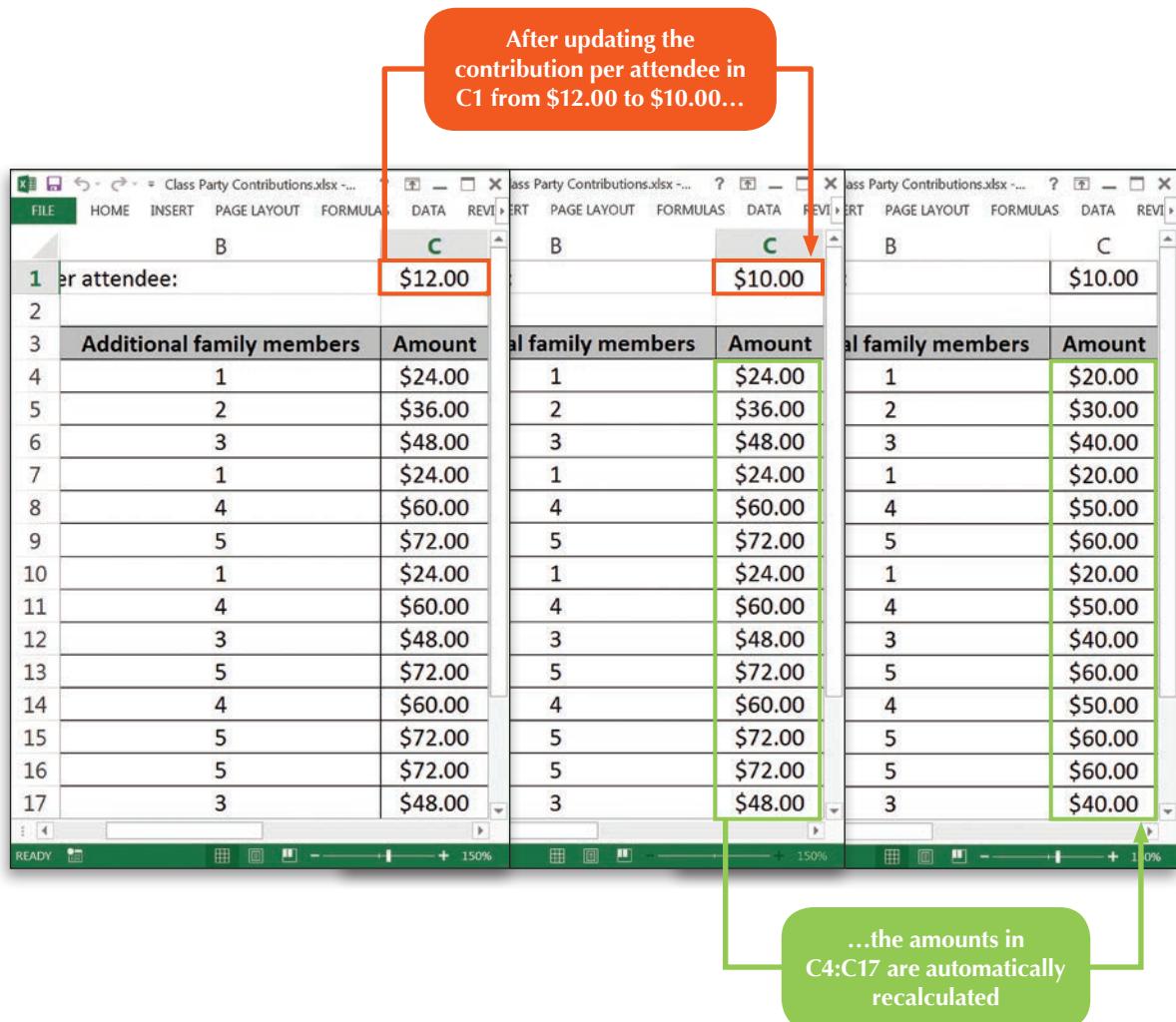
	C	D	E
1	Unit	Unit price	Subtotal
2	Packet	\$0.80	\$2.40
3	Piece	\$9.90	\$19.80
4	Packet	\$3.00	\$12.00
5	Set	\$1.00	\$60.00
6	Bottle	\$2.50	\$25.00

	C	D	E
1	Unit	Unit price	Subtotal
2	Packet	\$0.50	\$2.40
3	Piece	\$9.90	\$19.80
4	Packet	\$3.00	\$12.00
5	Set	\$1.00	\$60.00
6	Bottle	\$2.50	\$25.00

	C	D	E
1	Unit	Unit price	Subtotal
2	Packet	\$0.50	\$1.50
3	Piece	\$9.90	\$19.80
4	Packet	\$3.00	\$12.00
5	Set	\$1.00	\$60.00
6	Bottle	\$2.50	\$25.00

▲ Figure 10.14 Changing D2 causes E2 to be automatically recalculated

Similarly, in Alex's situation in Figure 10.15, updating the amount each attendee needs to contribute (cell C1) from \$12.00 to \$10.00 will automatically recalculate *all* the cells from C4 to C17 using the new amount. Once again, this is because the spreadsheet already understands how cell C1 is related to the calculations in range C4:C17.



▲ **Figure 10.15** Changing C1 causes C4:C17 to be automatically recalculated



Quick Check 10.1

- The spreadsheet in Figure 10.16 produces different multiplication tables.

Multiplication table for:		3			
Number	Times	Multiplier	Equals	Product	
1	×	3	=	3	
2	×	3	=	6	
3	×	3	=	9	
4	×	3	=	12	
5	×	3	=	15	
6	×	3	=	18	
7	×	3	=	21	
8	×	3	=	24	
9	×	3	=	27	
10	×	3	=	30	
11	×	3	=	33	
12	×	3	=	36	

▲ Figure 10.16 Multiplication table spreadsheet

When $\text{Ctrl}+\text{`}$ is pressed, the spreadsheet looks like this:

Multiplication table for:		3			
Number	Times	Multiplier	Equals	Product	
1	×	=C\$1	=	=A4*C4	
2	×	=C\$1	=	=A5*C5	
3	×	=C\$1	=	=A6*C6	
4	×	=C\$1	=	=A7*C7	
5	×	=C\$1	=	=A8*C8	
6	×	=C\$1	=	=A9*C9	
7	×	=C\$1	=	=A10*C10	
8	×	=C\$1	=	=A11*C11	
9	×	=C\$1	=	=A12*C12	
10	×	=C\$1	=	=A13*C13	
11	×	=C\$1	=	=A14*C14	
12	×	=C\$1	=	=A15*C15	

▲ Figure 10.17 Multiplication table with formulas shown

- a) State the data type of cell C1.
- b) Suggest a common data type that is appropriate for cell A3.
- c) Suggest how the equals sign may be entered as text in the range D4:D15.
- d) Identify the cell(s) which use(s) absolute cell references.
- e) Identify the cell(s) which would be recalculated if cell A4 is changed.
- f) Identify the cell(s) which would be recalculated if cell C1 is changed.

2. The spreadsheet in Figure 10.18 calculates speeds in units of m/s.

	A	B	C	D
1	Distance travelled (m)	Time taken (s)	Speed (m/s)	
2	121.2	24.25		
3	134.5	29.54		
4	20.5	3.44		
5	56.8	20.17		
6	44.7	19.65		
7				

▲ Figure 10.18 Spreadsheet to calculate speeds

Suggest a formula for cell C2 that can be copied to range C3:C6 in order to complete the spreadsheet correctly.

10.2 Data Management

While the class party described at the beginning of this chapter involves only a small amount of data, more complex tasks such as managing the accounts of a company require the handling of vast amounts of data. Hence, it is important to manage this data effectively to make good decisions. To do this, we need clear and efficient processes to collect, analyse and store data for repeated use. This series of processes is called the data life cycle.

There are four stages in the data life cycle:

1. Data collection
2. Data processing
3. Data analysis
4. Data distribution

10.2.1 Data Collection

Data can come from various sources and may be collected automatically by a computer or manually by performing interviews, making observations, conducting surveys, and so on. A spreadsheet can be used to collate and organise the collected data into columns and rows.

10.2.1.1 Organising Data by Columns

Typically, the different columns of a spreadsheet represent different types of data. Each column stores only *one* type of data. For instance, in Figure 10.19, one column is used for classmates' names while another column is used for the number of family members attending.

Columns are also called **fields**. To identify and describe the data stored in each column or field, there should always be a row of **column headings** or **field names** at the top of the spreadsheet.

Figure 10.19 shows how an empty spreadsheet is prepared for data collection by providing appropriate column headings in the first row to describe the data that will be entered in that column.

Key Terms	
Column heading (or field name)	Description of data stored in a particular column or field
Field (spreadsheet)	Column of data in a spreadsheet

	A	B
1	Name	Additional family members
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		

▲ Figure 10.19 Empty spreadsheet with proper column headings and data type settings

Since each column or field should only store one type of data, it is also a good practice to set the data type of each column to the most appropriate option before any data is entered. For instance, in Figure 10.19, the data type for the "Name" column should be set to "Text" while the data type for the "Additional family members" column should be set to "Number" with zero decimal places.

10.2.1.2 Organising Data by Rows

In a spreadsheet, each row typically represents a set of related data. For instance, in Figure 10.20, each row describes a classmate's name and the associated number of family members that classmate is bringing to the party.

Each row is also called a **record** and represents a set of related data that can be divided into fields.

Key Term

Record

Set of related data that describes a person or thing; usually refers to a row of data in a spreadsheet



	A	B
1	Name	Additional family members
2	Aisha	1
3	Alex	2
4	Bala	3
5	Denise	1
6	Farhan	4
7	Gopi	5
8	Irfan	1
9	Jun Ming	4
10	Lily	3
11	Mei Ling	5
12	Muthu	4
13	Nurul	5
14	Priya	5
15	Siti	3

▲ **Figure 10.20** A row of related data from multiple fields is called a record

10.2.2 Data Processing

In this stage, the collected data is checked for accuracy. This is because raw data is usually imperfect and must be processed to ensure that the analysis done later will produce results that make sense.

Raw data may be imperfect due to input errors, missing data and/or contradictory data. For instance, suppose the data for the class party is collected using paper forms before being entered into a spreadsheet. Input errors may occur if a student's name is misspelled, missing data may occur if a required question is left blank and contradictory data may occur if two mutually exclusive responses are both indicated (e.g., a student cannot claim to be both a vegetarian and a non-vegetarian at the same time).

The data processing stage is similar to the process of input validation in programming and a spreadsheet application typically has built-in functions that can help in data processing tasks. You will learn about these functions in section 10.3.

10.2.3 Data Analysis

In this stage, the processed data is studied to answer questions and to make decisions.

For example, the price of party items may be summed up to determine if there are enough contributions to cover the total cost, or the mean score for a test may be calculated to decide whether class performance has improved. Spreadsheets typically have built-in functions to perform such data analysis. You will learn about these functions in section 10.3.

10.2.4 Data Distribution

After data is analysed, the results need to be distributed and shared with the relevant parties. It is important to document the data's distribution policy as well as its copyright and intellectual property status. This information may be recorded in the rows above the column header of a spreadsheet.



Quick Check 10.2

- The spreadsheet template in Figure 10.21 is prepared for the data collection of purchases made for the class party.

	A	B	C	D
1	Item name	Unit price	Quantity	
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				

▲ **Figure 10.21** Spreadsheet template for data collection of purchases for the class party

- State the number of fields present in this spreadsheet.
- Suggest a common data type that is appropriate for range A2:A10.
- Suggest a common data type that is appropriate for range B2:B10.

10.3 Functions

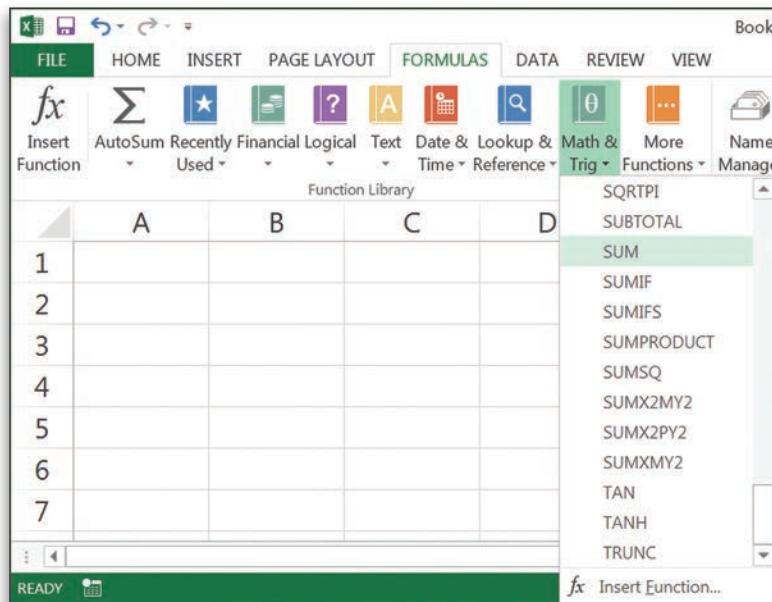
Just like functions in Python, spreadsheet functions are predefined formulas used to perform particular operations. Each function takes in zero or more inputs (arguments) and returns one output (return value). Most spreadsheet applications have various categories of built-in functions. In Microsoft Excel, these categories are found under the Formulas tab, as shown in Figure 10.22.



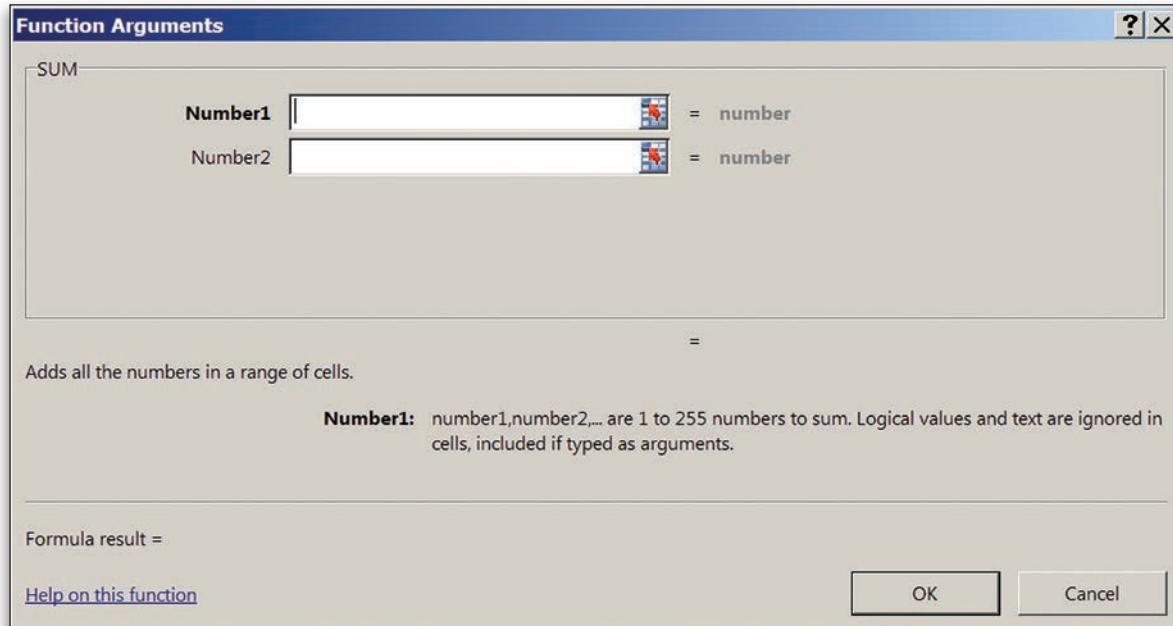
▲ Figure 10.22 Categories of built-in functions in a spreadsheet application

Functions can only be used in formulas. When using a function, type its name, followed by zero or more comma-separated arguments enclosed within parentheses.

For more details on each function, open its function arguments window by selecting the appropriate category under the Formulas tab and clicking on the function's name as shown in Figures 10.23 and 10.24. You will learn about some of the most common functions in the following sections.



▲ Figure 10.23 Selecting the SUM() function from the “Math & Trig” category



▲ Figure 10.24 Function arguments window for the SUM () function

10.3.1 Logical Functions

In section 10.1.2, we learnt that cells can contain logical values which are similar to Boolean values in Python. Besides using the explicit values of `TRUE` and `FALSE`, logical values can also be formed using the relational operators in Table 10.2 to compare two values. (Note that while most of these operators are the same as those in Python, the equivalence and non-equivalence operators are not.)

▼ Table 10.2 Relational operators

Operator	Meaning
<	Less than
\leq	Less than or equal to
>	Greater than
\geq	Greater than or equal to
=	Equal to (equivalence)
\neq	Not equal to (non-equivalence)

In Python, we typically use the conjunction (`and`), disjunction (`or`) and negation (`not`) operators to combine Boolean values and make decisions using `if-else` statements. In spreadsheets, however, we use logical functions instead, as described in Table 10.3.

▼ **Table 10.3** Common logical functions

Function	Syntax	Description
<code>IF()</code>	<code>=IF(logical_test, value_if_true, value_if_false)</code>	Returns <code>value_if_true</code> (i.e., the second argument) when <code>logical_test</code> is <code>TRUE</code> and <code>value_if_false</code> (i.e., the third argument) when <code>logical_test</code> is <code>FALSE</code>
<code>AND()</code>	<code>=AND(logical1, logical2, ...)</code> <ul style="list-style-type: none"> • Only the argument <code>logical1</code> is compulsory. The arguments from <code>logical2</code> onwards are optional. • <code>logical1</code>, <code>logical2</code>, etc. can be either logical values or range/cell references. 	Returns <code>TRUE</code> when all of the given logical values (<code>logical1</code> , <code>logical2</code> , etc.) or values in the given range/cell references are <code>TRUE</code> ; otherwise, returns <code>FALSE</code>
<code>OR()</code>	<code>=OR(logical1, logical2, ...)</code> <ul style="list-style-type: none"> • Only the argument <code>logical1</code> is compulsory. The arguments from <code>logical2</code> onwards are optional. • <code>logical1</code>, <code>logical2</code>, etc. can be either logical values or range/cell references. 	Returns <code>TRUE</code> when any of the given logical values (<code>logical1</code> , <code>logical2</code> , etc.) or values in the given range/cell references is <code>TRUE</code> ; otherwise, returns <code>FALSE</code>
<code>NOT()</code>	<code>=NOT(logical)</code>	Returns <code>TRUE</code> when <code>logical</code> is <code>FALSE</code> and <code>FALSE</code> when <code>logical</code> is <code>TRUE</code>

Figure 10.25 shows a spreadsheet that tracks the attendees for the class party. Suppose Bala would like to fill column C with information that indicates whether each classmate is bringing more than three additional family members. To do this, he could enter “=B2>3” into C2 and copy this formula into C3:C15.

A	B	C	
1	Name	Additional family members	More than 3?
2	Aisha	1	=B2>3
3	Alex	2	
4	Bala	3	
5	Denise	1	
6	Farhan	4	
7	Gopi	5	
8	Irfan	1	
9	Jun Ming	4	
10	Lily	3	
11	Mei Ling	5	
12	Muthu	4	
13	Nurul	5	
14	Priya	5	
15	Siti	3	
16			

▲ Figure 10.25 Using relational operators to produce logical values

In this case, using the greater than (>) relational operator produces a logical value that is either TRUE or FALSE. Now, suppose Bala wishes for the column to use “Yes” and “No” instead of “TRUE” and “FALSE”. He could use the IF() function to accomplish this, as shown in Figure 10.26.

A	B	C	
1	Name	Additional family members	More than 3?
2	Aisha	1	=IF(B2>3, "Yes", "No")
3	Alex	2	=IF(B3>3, "Yes", "No")
4	Bala	3	=IF(B4>3, "Yes", "No")
5	Denise	1	=IF(B5>3, "Yes", "No")
6	Farhan	4	=IF(B6>3, "Yes", "No")
7	Gopi	5	=IF(B7>3, "Yes", "No")
8	Irfan	1	=IF(B8>3, "Yes", "No")
9	Jun Ming	4	=IF(B9>3, "Yes", "No")
10	Lily	3	=IF(B10>3, "Yes", "No")
11	Mei Ling	5	=IF(B11>3, "Yes", "No")
12	Muthu	4	=IF(B12>3, "Yes", "No")
13	Nurul	5	=IF(B13>3, "Yes", "No")
14	Priya	5	=IF(B14>3, "Yes", "No")
15	Siti	3	=IF(B15>3, "Yes", "No")
16			

▲ Figure 10.26 Using IF() to return different values based on a condition

As explained in Table 10.3, the `IF()` function returns the second argument if the first argument is `TRUE`. Otherwise, it returns the third argument instead.

Table 10.4 shows some examples of how `IF()`, `AND()`, `OR()` and `NOT()` may be used to determine and display different information about each classmate. In each of these examples, note that the formula quoted is for the student in row 2 only (i.e., Aisha) and should be copied into rows 3 to 15 to show the same information for all classmates.

▼ **Table 10.4** Examples of using `IF()`, `AND()`, `OR()` and `NOT()`

Example	Information to display	Formula for row 2
1	<p>Whether each classmate has between 2 to 4 (both inclusive) additional family members attending</p> <p>Result must be either "Yes" or "No" only</p>	$=IF(AND(B2>=2, B2<=4), "Yes", "No")$ This formula returns "Yes" when the number of additional family members attending is <i>both</i> greater than or equal to 2 <i>and</i> less than or equal to 4. Otherwise, it returns "No".
2	<p>Whether each classmate has less than 2 or more than 4 additional family members attending</p> <p>Result must be either "Yes" or "No" only</p>	<p><u>Option A</u></p> $=IF(OR(B2<2, B2>4), "Yes", "No")$ This formula returns "Yes" when the number of additional family members attending is <i>either</i> less than 2 <i>or</i> greater than 4. Otherwise, it returns "No". <p><u>Option B</u></p> $=IF(AND(B2>=2, B2<=4), "No", "Yes")$ Alternatively, since the information to display is exactly the opposite of Example 1, we could also swap the responses of "Yes" and "No" from the formula given in Example 1. <p><u>Option C</u></p> $=IF(NOT(AND(B2>=2, B2<=4)), "Yes", "No")$ We could also use the <code>NOT()</code> function to negate the condition in the formula for Example 1.

▼ Table 10.4 Examples of using IF(), AND(), OR() and NOT() (continued)

Example	Information to display	Formula for row 2
3	<p>Whether each classmate has less than, equal to or more than 3 additional family members attending</p> <p>Result must be either "Less than 3", "Equal to 3" or "More than 3"</p>	<pre>=IF(B2<3, "Less than 3", IF(B2=3, "Equal to 3", "More than 3"))</pre> <p>This formula uses a nested IF() function. If the number of additional family members attending is less than 3, the formula returns "Less than 3". Otherwise, it performs another test for whether the number is equal to 3. If so, it returns "Equal to 3". Otherwise, it returns "More than 3".</p> <p>(Besides this formula, there are multiple alternative options to achieve the same result.)</p>

10.3.2 Text Functions

Spreadsheets have many text functions to manipulate and customise text. Table 10.5 shows some common functions for finding the length of some text or extracting a subset of characters from some text.

▼ Table 10.5 Common text functions

Function	Syntax	Description
LEN()	<pre>=LEN(text)</pre> <p>Example: <code>=LEN("Computing")</code> This would return 9.</p>	Returns the number of characters in the text value <code>text</code>
MID()	<pre>=MID(text, start_num, num_chars)</pre> <p>Example: <code>=MID("Computing", 3, 4)</code> This would return "mput".</p>	<p>Returns <code>num_chars</code> characters starting from position <code>start_num</code> of the text value <code>text</code></p> <p>(Note that, unlike Python, the position of the first character is 1, not 0.)</p>

▼ Table 10.5 Common text functions (continued)

Function	Syntax	Description
LEFT()	=LEFT(text, num_chars) • num_chars is an optional argument Example: =LEFT("Computing", 5) This would return "Compu".	Returns the first num_chars characters of the text value text If num_chars is left out, the function returns only the first character of text
RIGHT()	=RIGHT(text, num_chars) • num_chars is an optional argument Example: =RIGHT("Computing", 5) This would return "ting".	Returns the last num_chars characters of the text value text If num_chars is left out, the function returns only the last character of text

10.3.3 Date Function

Table 10.6 shows the TODAY() function which returns the current date.

▼ Table 10.6 Date function

Function	Syntax	Description
TODAY()	=TODAY()	Returns the current date

10.3.4 Lookup Functions

In spreadsheets, we often need to find the row in which a particular value is located. Suppose we have a class register list with registration numbers in the first column, birthdays in the second column and student names in the third column, as shown in Figure 10.27. If we want to know the name of the student with registration number 10, we would perform the following steps:

Step 1: Search vertically down the first column of class register list A1:C15 (called the **lookup table**) for the registration number 10. The value to look for is called the **lookup value**. (In this case, the lookup value is 10.)

Step 2: If the lookup value is found, output the matching row's third-column value. The column number is the column to output. (In this case, the column number is 3.)

Otherwise, if no matching row is found, output an error.

Key Terms

Lookup table

Array or data structure that is searched through in order to find a particular value

Lookup value

Value to search for in order to find a particular row or column

	Column	Column	Column
1	A	B	C
1	Registration Number	Birthday	Name
2	1	4/7/2002	Aisha
3	2	14/7/2002	Alex
4	3	5/5/2002	Bala
5	4	25/10/2002	Denise
6	5	20/4/2002	Farhan
7	6	22/7/2002	Gopi
8	7	19/3/2002	Irfan
9	8	13/12/2002	Jun Ming
10	9	4/8/2002	Lily
11	10	24/3/2002	Mei Ling
12	11	24/7/2002	Muthu
13	12	3/8/2002	Nurul
14	13	29/7/2002	Priya
15	14	27/1/2002	Siti
16			

▲ **Figure 10.27** The lookup value of 10 and column number of 3 are used to find the value “Mei Ling”

Spreadsheets typically provide a VLOOKUP() function to perform these steps if you provide a lookup value, a lookup table and a column number. For spreadsheets that use rows for fields and columns for records, the HLOOKUP() function can be used instead. Note that VLOOKUP() performs a *vertical* search while HLOOKUP() performs a *horizontal* search. These functions are described in Table 10.7.

▼ **Table 10.7** Lookup functions

Function	Syntax	Description
HLOOKUP()	=HLOOKUP(lookup_value, table_array, row_index_num, range_lookup)	<p>Looks for <code>lookup_value</code> in the first row of <code>table_array</code> and returns the value in row <code>row_index_num</code> of the matching column.</p> <ul style="list-style-type: none"> If <code>range_lookup</code> is TRUE, the function performs an approximate match. If <code>range_lookup</code> is FALSE, the function performs an exact match. If <code>range_lookup</code> is left out, the function performs an approximate match.
VLOOKUP()	=VLOOKUP(lookup_value, table_array, col_index_num, range_lookup)	<p>Looks for <code>lookup_value</code> in the first column of <code>table_array</code> and returns the value in column <code>col_index_num</code> of the matching row.</p> <ul style="list-style-type: none"> If <code>range_lookup</code> is TRUE, the function performs an approximate match. If <code>range_lookup</code> is FALSE, the function performs an exact match. If <code>range_lookup</code> is left out, the function performs an approximate match.

Note that the last argument to both HLOOKUP() and VLOOKUP() is a logical value named `range_lookup` that determines whether an exact match or approximate match is used.

An exact match looks for the exact lookup value and returns the error value #N/A if no match is found. An approximate match, on the other hand, looks for the largest value that is less than or equal to the lookup value. It only returns an error value if all the values in the first column or row of the lookup table are larger than the lookup value.

When using an exact match, the only way to make sure that all lookups are successful is to include all possible lookup values in the lookup table. This can make the lookup table very long. Using an approximate match lets us shorten the lookup table as the table does not need to contain the exact lookup values. This is useful if a single row or column of the lookup table is used for multiple lookup values, as explained in the following examples.

Figure 10.28 shows how `VLOOKUP()` is used to look up the names of three students based on their registration numbers (5, 6 and 9).

The screenshot shows an Excel spreadsheet titled "Class Registration (Lookup).xlsx". The main table (A1:C15) contains columns for Registration Number, Birthday, and Name. A smaller table (E2:F4) is highlighted with a red dashed border and labeled "Lookup table". Three formulas are shown in the adjacent cells (F2, F3, F4): =VLOOKUP(E2, \$A\$1:\$C\$15, 3), =VLOOKUP(E3, \$A\$1:\$C\$15, 3), and =VLOOKUP(E4, \$A\$1:\$C\$15, 3). A callout diagram on the right side of the screenshot illustrates the formula structure: "These grades are obtained from A2:C15 using VLOOKUP ()". It points to the first formula with arrows labeled "Lookup value" (pointing to E2), "Column number" (pointing to 3), and "Lookup table" (pointing to \$A\$1:\$C\$15). Below the formulas, there is a large orange arrow pointing right, with a small box containing "Ctrl" above it, indicating the use of the Ctrl+C keyboard shortcut to copy the formula.

▲ **Figure 10.28** Using `VLOOKUP()` to look up names based on registration number

In this case, the first argument for each use of `VLOOKUP()` is a registration number in column E (that is, E2, E3 or E4). This is the lookup value that we are searching for.

The second argument for all three uses of `VLOOKUP()` refers to the class list in A1:C15. This is the lookup table that we wish to search through and retrieve a value from. (An absolute range reference is used here to make copying the formula to new rows easier.)

Finally, the third argument of 3 tells `VLOOKUP()` to return the value in the third column of the lookup table if it finds a match. This is because the names we wish to retrieve are in the third column of the lookup table.

Figure 10.29 shows how VLOOKUP() can also be used to convert rounded percentage scores into grades (such as A1, A2 and B3). However, the lookup table in this case is very long as it includes every possible rounded percentage score from 0 to 100.

The figure displays two Microsoft Excel windows side-by-side. The left window shows a table with 'Percentage' in column A and 'Grade' in column B. The formula in cell B2 is =VLOOKUP(A2, \$D\$2:\$E\$102, 2). The right window shows a detailed 'lookup table' with columns 'Percentage' and 'Grade'. The table has 102 rows, starting from 0 and ending at 100. The 'Grade' column contains values like F9, A1, B3, B4, E8, etc. An orange arrow points from the right window to the formula in the left window. A callout box on the right states 'Lookup table is 102 rows long!'. Labels above the tables identify the 'Lookup value' (A2), 'Column number' (2), and 'Lookup table' (\$D\$2:\$E\$102).

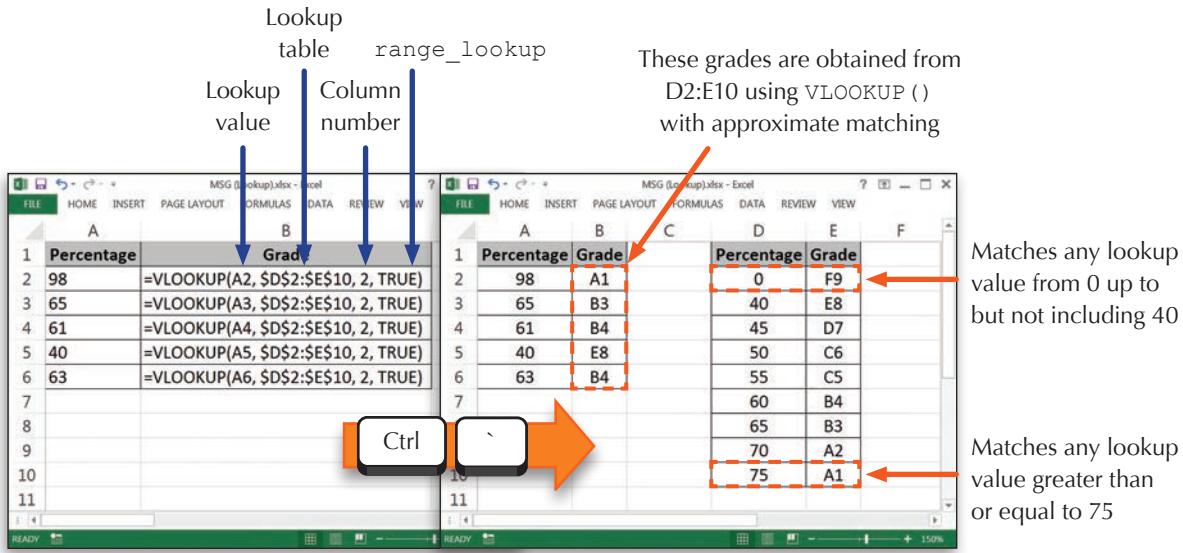
Percentage	Grade
98	=VLOOKUP(A2, \$D\$2:\$E\$102, 2)
65	=VLOOKUP(A3, \$D\$2:\$E\$102, 2)
61	=VLOOKUP(A4, \$D\$2:\$E\$102, 2)
40	=VLOOKUP(A5, \$D\$2:\$E\$102, 2)
63	=VLOOKUP(A6, \$D\$2:\$E\$102, 2)

Percentage	Grade
98	A1
65	B3
61	B4
40	E8
63	B4
93	91
94	92
95	93
96	94
97	95
98	96
99	97
100	98
101	99
102	100
103	A1

▲ **Figure 10.29** Using VLOOKUP() to convert rounded percentage scores into grades

However, notice that the entries in the second column of the lookup table are the same for every value in the first column from 0 up to but not including 40, from 40 up to but not including 45, from 45 up to but not including 50, and so on. This indicates that we can achieve the same result using approximate matching and a shorter lookup table that only includes the boundaries for the different grades in the first column. This works because approximate matching looks for the largest value that is less than or equal to the lookup value.

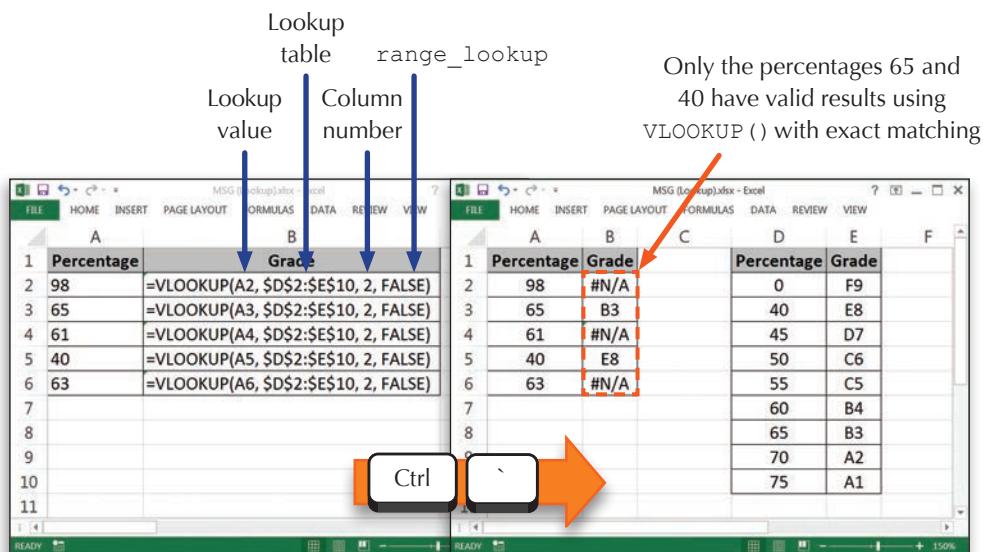
Approximate matching is used by default when the fourth argument is left out. Nonetheless, to make it clear that we wish to use approximate matching, we shall include the fourth argument and set it to TRUE, as shown in Figure 10.30.



▲ **Figure 10.30** Using approximate matching to convert rounded percentage scores into grades

Figure 10.30 shows how VLOOKUP() with approximate matching can be used to convert rounded percentage scores into grades (such as A1, A2 and B3). For example, the formula in cell B2 looks for the largest value in column D (the first column) that is less than or equal to the lookup value of 98, i.e., the value of 75, which is found in row 10. It then returns the value found in column E (the second column) of the same row, i.e., "A1". Hence, the formulas in column B are able to return valid results even though most of the values in column A do not have exact matches in column D.

On the other hand, if the fourth argument is set to FALSE, VLOOKUP() will perform an exact match instead, as shown in Figure 10.31. In this case, only the scores of 65 and 40, which exactly match rows 3 and 5 of range D2:E10 respectively, will have a valid result. The other cells will return the error value #N/A.



▲ **Figure 10.31** Using exact matching returns an error value if an exact match cannot be found

Figure 10.32 shows how `VLOOKUP()` with approximate matching can be used to group BMI values into the Underweight, Acceptable and Overweight categories.

The screenshot shows two Excel windows. The left window displays a formula-based approach where the lookup value is compared against a range of BMI values (A2:A7) to find the corresponding result in E4. The right window shows the final results categorized into Underweight, Acceptable, and Overweight based on the BMI values. A legend table is also shown.

BMI	Result
0	Underweight
16	Acceptable
27	Overweight

These results are obtained from D2:E4 using `VLOOKUP()` with approximate matching

▲ Figure 10.32 Using `VLOOKUP()` with approximate matching to categorise BMI values

Suppose that the lookup table D2:E4 is now “flipped” such that rows are used for fields and columns are used for records instead. This means that the BMI values of 0, 16 and 27 are now on row 1 and the categories “Underweight”, “Acceptable” and “Overweight” are now on row 2. In this case, we will have to use `HLOOKUP()` instead of `VLOOKUP()` to perform the same task.

The screenshot shows two Excel windows. The left window displays a formula-based approach using HLOOKUP to find the category for a given BMI value. The right window shows the final results categorized into Underweight, Acceptable, and Overweight based on the BMI values. A legend table is also shown.

BMI	Result
0	Underweight
16	Acceptable
27	Overweight

These results are obtained from E1:G2 using `HLOOKUP()` with approximate matching

Lookup table E1:G2 has rows and columns swapped

▲ Figure 10.33 Using `HLOOKUP()` with approximate matching to categorise BMI values

When using `HLOOKUP()`, the first two arguments still refer to the lookup value and lookup table range respectively. However, the third argument is now a *row number* instead of a *column number*. In this case, the row number is still 2 as the groupings we want to retrieve are on the second row of the lookup table.

Note that for approximate matching to work, the first column (for `VLOOKUP()`) or row (for `HLOOKUP()`) of the lookup table must have its values arranged in increasing order. For instance, the values in the ranges D2:D10 in Figure 10.30 and D2:D4 in Figure 10.32 are arranged in increasing order from top to bottom.

Similarly, the values in range E1:G1 in Figure 10.33 are arranged in increasing order from left to right. Exact matching, on the other hand, does not require the values in the first column or row of the lookup table to be arranged in increasing order.

10.3.5 Math Functions

Table 10.8 shows some common spreadsheet functions for performing mathematical calculations and generating random numbers.

▼ **Table 10.8** Common math functions

Function	Syntax	Description
<code>CEILING.MATH()</code>	=CEILING.MATH(number) Example: =CEILING.MATH(12.34) This would return 13.	Returns number rounded <i>up</i> to the nearest whole number (Other uses of this function are not covered in this textbook.)
<code>FLOOR.MATH()</code>	=FLOOR.MATH(number) Example: =FLOOR.MATH(98.76) This would return 98.	Returns number rounded <i>down</i> to the nearest whole number (Other uses of this function are not covered in this textbook.)
<code>POWER()</code>	=POWER(number, power) Example: =POWER(2, 8) This would return 256.	Returns number raised to the exponent power
<code>SQRT()</code>	=SQRT(number) Example: =SQRT(2017) This would return 44.9 (3 s.f.).	Returns the square root of number

▼ Table 10.8 Common math functions (continued)

Function	Syntax	Description
ROUND ()	=ROUND (number, num_digits) • Note that num_digits is a compulsory argument. Example: =ROUND (12.34, 0) This would return 12.	Returns number rounded to num_digits decimal place(s)
RANDBETWEEN ()	=RANDBETWEEN (lowest, highest) Example: =RANDBETWEEN (1965, 2017) This would return a random integer between 1965 and 2017 (both inclusive).	Returns a random number between the whole numbers lowest and highest (both inclusive)
RAND ()	=RAND () Example: =RAND () *2017 This would return a random floating-point number between 0 (inclusive) and 2017 (exclusive).	Returns a random number greater than or equal to 0 and less than 1
SUM ()	=SUM (number1, number2, ...) • Only the argument number1 is compulsory. The arguments from number2 onwards are optional. • number1, number2, etc. can be either numbers or range/ cell references. Example 1: =SUM (2, 0, 1, 7) This would return 10. Example: 2 =SUM (A1:C3) This would return the total of all the numbers in the range A1:C3.	Returns the total of the given numbers (number1, number2, etc.) or numbers in the given range/cell references

▼ Table 10.8 Common math functions (continued)

Function	Syntax	Description
SUMIF()	<p>=SUMIF(range, criteria, sum_range)</p> <ul style="list-style-type: none"> • <code>sum_range</code> is an optional argument. <p>Suppose table A2:C40 is used to record the test scores of students identified by registration number and group name. The table has registration numbers in column A, group names in column B and scores in column C.</p> <p>Example 1: <code>=SUMIF(A2:A40, ">6", C2:C40)</code></p> <p>This would return the total score of students with registration numbers greater than 6.</p> <p>Example 2: <code>=SUMIF(B2:B40, "Computing", C2:C40)</code></p> <p>This would return the total score for the group “Computing”.</p>	<p>Returns the total of the numbers in <code>sum_range</code> where the corresponding value in <code>range</code> is equal to or satisfies the condition in <code>criteria</code></p> <p>Any <code>criteria</code> that includes relational operators must be enclosed in double-quotes.</p> <p>If <code>sum_range</code> is left out, then <code>range</code> is used as <code>sum_range</code>.</p>
QUOTIENT()	<p>=QUOTIENT(number, divisor)</p> <p>Example: <code>=QUOTIENT(2017, 1000)</code></p> <p>This would return 2.</p>	Returns the quotient when <code>number</code> is divided by <code>divisor</code>
MOD()	<p>=MOD(number, divisor)</p> <p>Example: <code>=MOD(2017, 1000)</code></p> <p>This would return 17.</p>	<p>Returns the remainder when <code>number</code> is divided by <code>divisor</code></p> <p>Similar to Python’s % operator</p>

10.3.6 Statistical Functions

Table 10.9 shows some common spreadsheet functions for performing statistical calculations and gathering statistical information.

▼ **Table 10.9** Common statistical functions

Function	Syntax	Description
MAX()	<p>=MAX (number1, number2, ...)</p> <ul style="list-style-type: none"> • Only the argument number1 is compulsory. The arguments from number2 onwards are optional. • number1, number2, etc. can be either numbers or range/cell references. <p>Example 1: $=\text{MAX}(2, 0, 1, 7)$ This would return 7.</p> <p>Example 2: $=\text{MAX}(\text{A1:C3})$ This would return the largest of all the numbers in the range A1:C3.</p>	Returns the largest number out of the given numbers (number1, number2, etc.) or the numbers in the given range/cell references
MIN()	<p>=MIN (number1, number2, ...)</p> <ul style="list-style-type: none"> • Only the argument number1 is compulsory. The arguments from number2 onwards are optional. • number1, number2, etc. can be either numbers or range/cell references. <p>Example 1: $=\text{MIN}(2, 0, 1, 7)$ This would return 0.</p> <p>Example 2: $=\text{MIN}(\text{A1:C3})$ This would return the smallest of all the numbers in the range A1:C3.</p>	Returns the smallest number out of the given numbers (number1, number2, etc.) or numbers in the given range/cell references

▼ Table 10.9 Common statistical functions (continued)

Function	Syntax	Description
COUNT()	=COUNT(range1, range2, ...) • Only the argument range1 is compulsory. The arguments from range2 onwards are optional. Example: =COUNT(A1:C3) This would return the number of cells that contain numbers, currencies, dates, times and percentages in the range A1:C3.	Returns the number of cells that contain numbers, currencies, dates, times and percentages in the given range references (range1, range2, etc.) Empty cells and cells with text or logical values are not counted.
COUNTA()	=COUNTA(range1, range2, ...) • Only the argument range1 is compulsory. The arguments from range2 onwards are optional. Example: =COUNTA(A1:C3) This would return the number of non-empty cells in the range A1:C3.	Returns the number of <i>non-empty</i> cells in the given range references (range1, range2, etc.) Empty cells are not counted while cells with any other data type (e.g., numbers and text) are counted.
COUNTIF()	=COUNTIF(range, criteria) Example 1: =COUNTIF(A1:A20, ">6") This would return the number of cells in range A1:A20 that have a value greater than 6. Example 2: =COUNTIF(B1:B20, "Computing") This would return the number of cells in range B1:B20 that match the text "Computing".	Returns the number of cells in range that are equal to or satisfy the condition in criteria Any criteria that includes relational operators must be enclosed in double-quotes.
COUNTBLANK()	=COUNTBLANK(range) Example: =COUNTBLANK(A1:C3) This would return the number of empty cells in the range A1:C3.	Returns the number of empty cells in range

▼ Table 10.9 Common statistical functions (continued)

Function	Syntax	Description
MODE.SNGL()	=MODE.SNGL(number1, number2, ...) <ul style="list-style-type: none">Only the argument number1 is compulsory. The arguments from number2 onwards are optional.number1, number2, etc. can be either numbers or range/cell references. <p>Example 1: =MODE.SNGL(1, 9, 6, 5, 2, 0, 1, 7) This would return 1 as it is most repeated value out of all the arguments.</p> <p>Example 2: =MODE.SNGL(A1:C3) This would return the most repeated number in the range A1:C3.</p>	Returns the most frequently occurring or repeated value in the given numbers (number1, number2, etc.) or the given range/cell references
MEDIAN()	=MEDIAN(number1, number2, ...) <ul style="list-style-type: none">Only the argument number1 is compulsory. The arguments from number2 onwards are optional.number1, number2, etc. can be either numbers or range/cell references. <p>Example: =MEDIAN(2, 0, 1, 7) This would return 1.5 (which is the average of the middle numbers 1 and 2 when the numbers are sorted in ascending order).</p>	Returns the median of the given numbers (number1, number2, etc.) or numbers in the given range/cell references
AVERAGE()	=AVERAGE(number1, number2, ...) <ul style="list-style-type: none">Only the argument number1 is compulsory. The arguments from number2 onwards are optional.number1, number2, etc. can be either numbers or range/cell references. <p>Example: =AVERAGE(2, 0, 1, 7) This would return 2.5 (which is the average of the four numbers).</p>	Returns the mean or average of the given numbers (number1, number2, etc.) or numbers in the given range/cell references

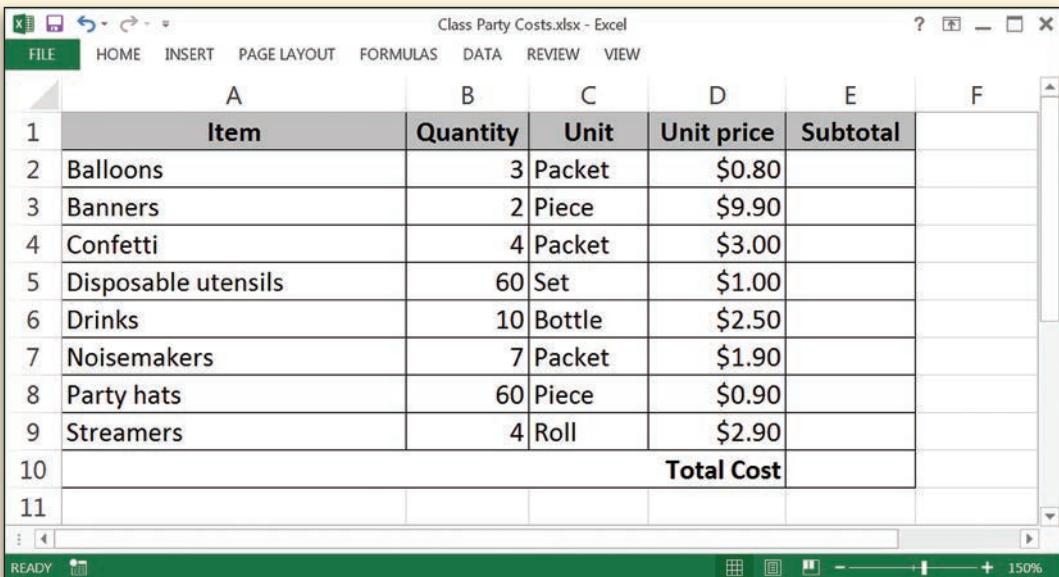
▼ Table 10.9 Common statistical functions (continued)

Function	Syntax	Description
SMALL ()	=SMALL(range, k) Example: =SMALL(A1:C3, 2) This would return the second-smallest number in the range A1:C3.	Returns the kth smallest number in range
LARGE ()	=LARGE(range, k) Example: =LARGE(A1:C3, 2) This would return the second-largest number in the range A1:C3.	Returns the kth largest number in range



Quick Check 10.3

1. The spreadsheet in Figure 10.34 is used to tabulate costs for a class party.



	A	B	C	D	E	F
1	Item	Quantity	Unit	Unit price	Subtotal	
2	Balloons	3	Packet	\$0.80		
3	Banners	2	Piece	\$9.90		
4	Confetti	4	Packet	\$3.00		
5	Disposable utensils	60	Set	\$1.00		
6	Drinks	10	Bottle	\$2.50		
7	Noisemakers	7	Packet	\$1.90		
8	Party hats	60	Piece	\$0.90		
9	Streamers	4	Roll	\$2.90		
10	Total Cost					
11						

▲ Figure 10.34 Costs for a class party

- a) Suggest a formula for cell E2 that can be copied into range E3:E9 in order to complete the spreadsheet.

- b) Suggest a formula for cell E10 to calculate the total cost from the subtotals.
 - c) Predict the value that will be returned for the following formulas:
 - i) =COUNTIF(B2:B9, ">3")
 - ii) =IF(A6="Drinks", IF(B6<10, "A", "B"), "C")
2. The formulas below are two ways of comparing the value in cell A1 with the number 3. They both return a text value of "Less than 3", "Equal to 3" or "Greater than 3".
- =IF(A1<3, "Less than 3", IF(A1=3, "Equal to 3", "Greater than 3"))
 - =IF(A1<3, "Less than 3", IF(A1>3, "Greater than 3", "Equal to 3"))
- Suggest two other formulas that can achieve the same result by rearranging the IF() functions, arguments and/or changing the conditions.
3. The spreadsheet in Figure 10.35 is used to track scores in a race.

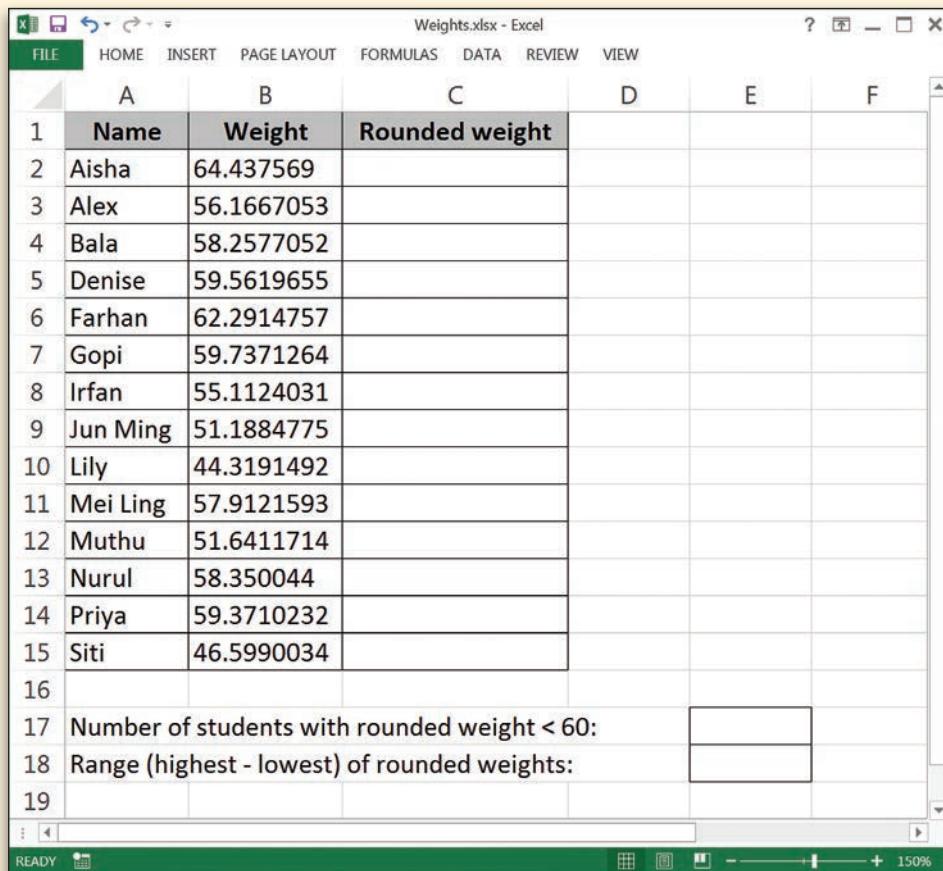
	A	B	C	D	E	F	G
1	Name	Timing	Score		Timing	Score	
2	Alex	0:01:44			0:00:00	5	
3	Farhan	0:02:01			0:01:00	2	
4	Jun Ming	0:07:19			0:03:00	1	
5	Nurul	0:00:56					
6	Siti	0:01:36					
7							
8	Average score:						
9	Median score:						
10							
11							

▲ Figure 10.35 Scores in a race

Timings of one minute or less get a score of 5. Timings between one minute and three minutes get a score of 2. Timings exceeding three minutes get a score of 1.

- a) Suggest a formula for cell C2 that can be copied to range C3:C6 in order to look up the correct score from E2:F4. The formula must not require any further changes after it is copied into C3:C6.
- b) Suggest a formula for cell C8 to calculate the average of the five scores.
- c) Suggest a formula for cell C9 to calculate the median of the five scores.

4. The spreadsheet in Figure 10.36 is used to track the weight readings of students in a class.



The screenshot shows an Excel spreadsheet with the following data and formulas:

	A	B	C	D	E	F
1	Name	Weight	Rounded weight			
2	Aisha	64.437569				
3	Alex	56.1667053				
4	Bala	58.2577052				
5	Denise	59.5619655				
6	Farhan	62.2914757				
7	Gopi	59.7371264				
8	Irfan	55.1124031				
9	Jun Ming	51.1884775				
10	Lily	44.3191492				
11	Mei Ling	57.9121593				
12	Muthu	51.6411714				
13	Nurul	58.350044				
14	Priya	59.3710232				
15	Siti	46.5990034				
16						
17	Number of students with rounded weight < 60:					
18	Range (highest - lowest) of rounded weights:					
19						

Formulas in cells E17 and E18:

- E17: =COUNTIF(C2:C15, "<60")
- E18: =MAX(C2:C15) - MIN(C2:C15)

▲ Figure 10.36 Weight readings for a class

- Suggest a formula for cell C2 that can be copied to range C3:C15 in order to round the weight readings to one decimal place.
- Suggest a formula for cell E17 to count the number of students with a rounded weight reading that is less than 60 kg.
- Suggest a formula for cell E18 to calculate the range of rounded weight readings.

10.4 What-If Analysis

As you have learnt, a spreadsheet is able to perform automatic recalculation of all affected cells if any referenced cells are changed. We can use this ability to answer “what if” questions and predict the outcomes that may occur when particular values are changed. In particular, a spreadsheet lets us search for the best answer to meet a particular goal (using the Goal Seek feature).

10.4.1 Goal Seek

After defining the relationships between cells in a spreadsheet, we may wish to adjust the value in one cell so that another cell can reach a particular value (i.e., the goal). Goal Seek automatically solves the problem using a trial-and-error approach by plugging in guesses until the goal is reached.

For instance, suppose we have a spreadsheet that calculates the BMI (cell C2) of a student, given his height (cell A2) and weight (cell B2) using the formula:

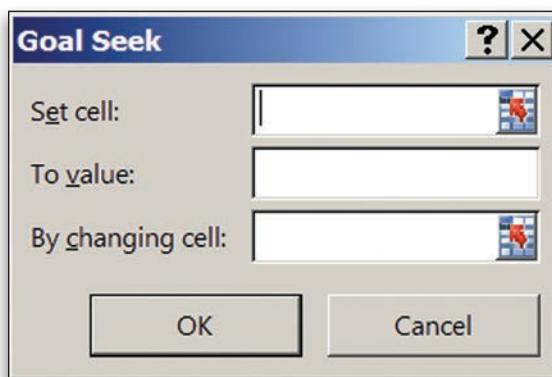
$$\text{BMI} = (\text{Weight in kg}) / (\text{Height in m})^2$$

	A	B	C	D
1	Height in m	Weight in kg	BMI	
2	1.45	60	29	
3				

▲ Figure 10.37 Spreadsheet to calculate a student’s BMI

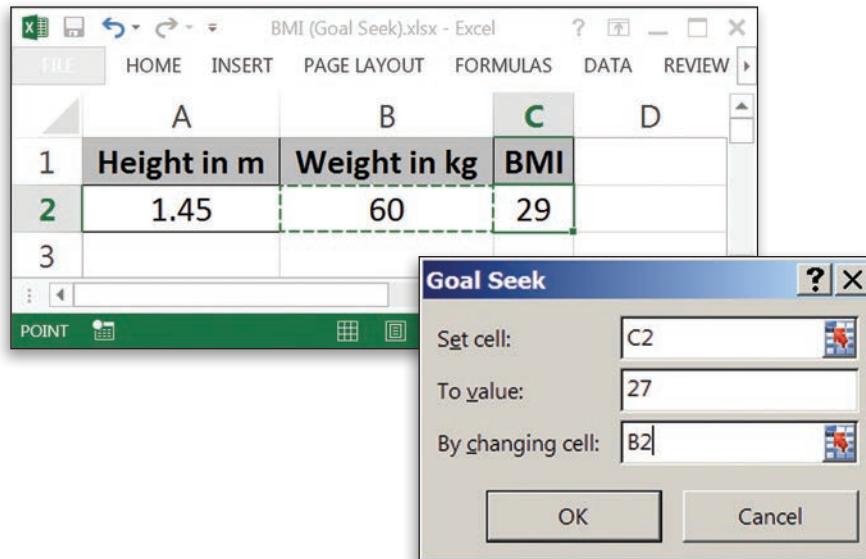
As shown in Figure 10.37, the student is currently overweight as his BMI is above 27 (based on the BMI guideline for 15- to 16-year-olds). To find out what his weight must be to reach an acceptable BMI of 27, we can use the Goal Seek feature that can be found under the “What-If Analysis” menu of the “Data” tab.

Figure 10.38 shows what the Goal Seek window looks like when it is first opened.



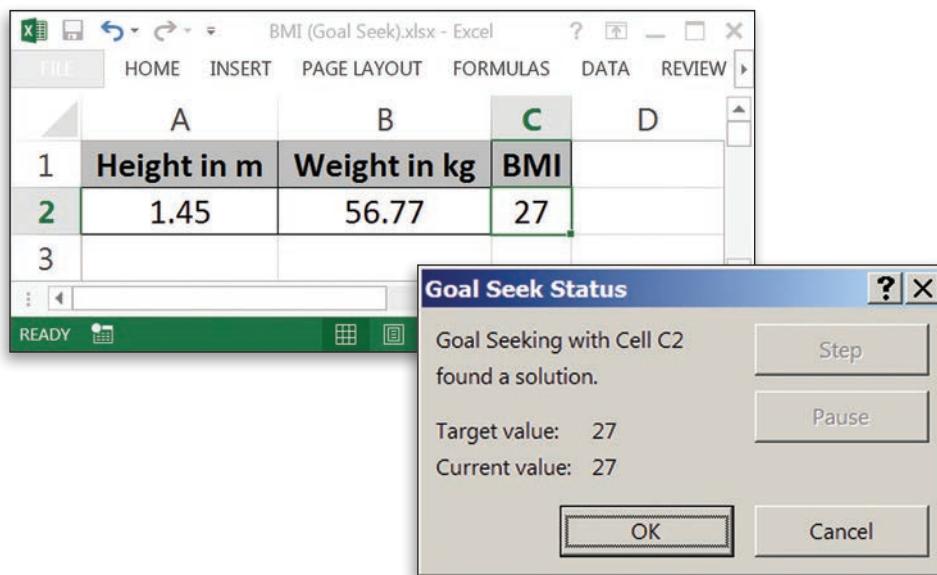
▲ Figure 10.38 Goal Seek window when initially opened

In this case, we want to set the BMI (cell C2) to 27 by changing the weight (cell B2). Figure 10.39 shows how these requirements can be entered into the Goal Seek window.



▲ Figure 10.39 Using Goal Seek by keying in the required values

After clicking on “OK”, the spreadsheet automatically calculates the value of 56.77 for the student’s weight in order to achieve a BMI of exactly 27, as shown in Figure 10.40. This means that the student with a height of 1.45 m can only reach a BMI of 27 by reducing his weight to 56.77 kg.



▲ Figure 10.40 The solution (weight in kg) is found by Goal Seek

The spreadsheet in Figure 10.41 collates the expected attendance, funding and costs for the class party described at the beginning of this chapter.

Formulas are used to calculate the total funding (cell C18) based on the amount each attendee is expected to contribute (cell C1).

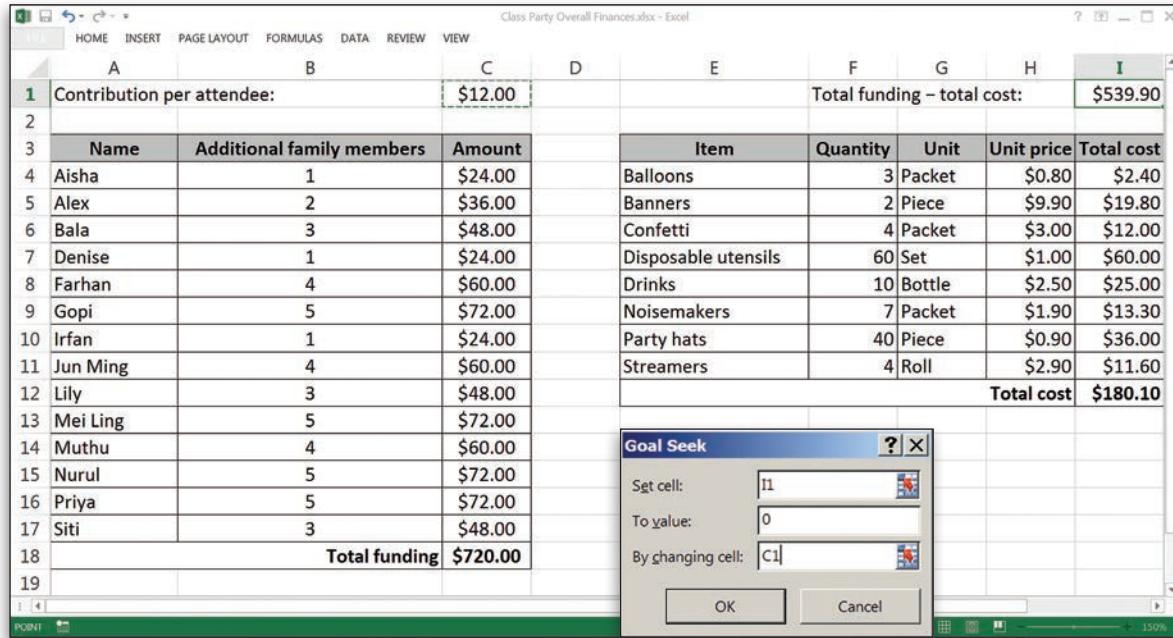
Formulas are also used to calculate the total cost (cell I12) based on the quantities of items needed to organise a potluck. In particular, the formula in cell I1 shows whether the class party makes a net profit or loss by calculating the difference between total funding and total cost.

The screenshot shows an Excel spreadsheet with two main sections: 'Contributions' and 'Party Supplies'. The 'Contributions' section (rows 1-18) lists 17 attendees with their names in column A, additional family members in column B, and contribution amounts in column C. The total funding is calculated in cell C18 (\$720.00). The 'Party Supplies' section (rows 1-12) lists items, quantities, unit prices, and total costs. The total cost is calculated in cell I12 (\$180.10). The difference between total funding and total cost is shown in cell I1 (\$539.90).

Contributions per attendee:				Total funding – total cost:			
1	Name	Additional family members	Amount				
2	Aisha	1	\$24.00	Balloons	3	Packet	\$0.80
3	Alex	2	\$36.00	Banners	2	Piece	\$9.90
4	Bala	3	\$48.00	Confetti	4	Packet	\$3.00
5	Denise	1	\$24.00	Disposable utensils	60	Set	\$1.00
6	Farhan	4	\$60.00	Drinks	10	Bottle	\$2.50
7	Gopi	5	\$72.00	Noisemakers	7	Packet	\$1.90
8	Irfan	1	\$24.00	Party hats	40	Piece	\$0.90
9	Jun Ming	4	\$60.00	Streamers	4	Roll	\$2.90
10	Lily	3	\$48.00				
11	Mei Ling	5	\$72.00				
12	Muthu	4	\$60.00				
13	Nurul	5	\$72.00				
14	Priya	5	\$72.00				
15	Siti	3	\$48.00				
16			Total funding \$720.00				
17							
18							
19							

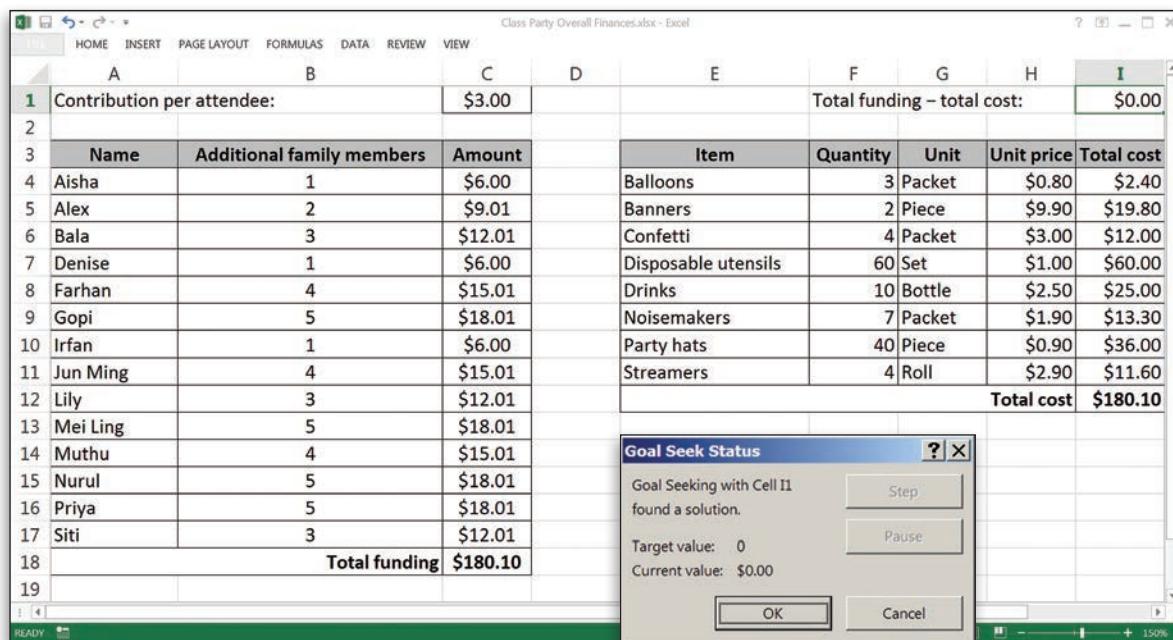
▲ Figure 10.41 Spreadsheet for class party finances

Suppose the organisers of the class party wish to change the contribution per attendee so that the class party breaks even exactly and neither makes nor loses money. This means that we should set the difference between total funding and total cost (cell I1) to 0 by changing the contribution per attendee (cell C1), as shown in Figure 10.42.



▲ **Figure 10.42** Using Goal Seek by setting the value for cell I1 to 0

After clicking on “OK”, the spreadsheet automatically calculates that each attendee should contribute \$3.00 in order to break even, as shown in Figure 10.43.



▲ **Figure 10.43** The solution (contribution per attendee) found by Goal Seek

10.5 Conditional Formatting

When trying to understand a large set of data, it is often useful to highlight important details to make them more noticeable. This can be done using a spreadsheet feature called **conditional formatting**.

Conditional formatting allows a cell to automatically vary its appearance based on its contents, ranking, relation to the average, uniqueness or some other formula.

To use conditional formatting, we need to create **rules**. Each rule specifies three pieces of information:

1. Which range of cells the rule is applied to
2. What condition is being tested
3. What formatting should be applied if the condition is true

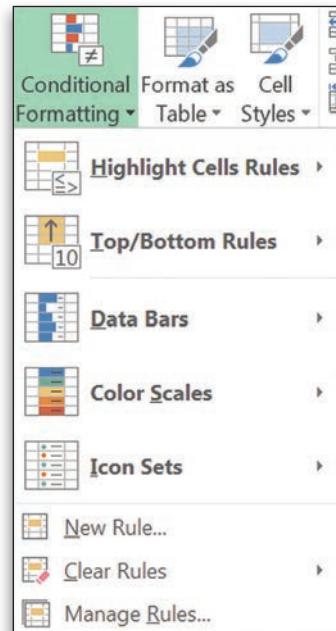
To create a rule, select the range of cells that the rule should be applied to, then choose “Conditional Formatting” under the “Home” tab, as shown in Figure 10.44.

The most common cases and options for using conditional formatting are listed under this menu. More complex rules can be created using the “New Rule...” option while existing rules can be edited or removed using the “Manage Rules...” and “Clear Rules” options respectively.

Key Terms

Conditional formatting
Spreadsheet feature which automatically formats a cell based on given criteria

Rule (conditional formatting)
Criterion for conditional formatting; specifies a range of cells, a condition to be tested for and the formatting to apply if the condition is true



▲ **Figure 10.44** Conditional formatting menu under the “Home” tab

For instance, suppose we wish to highlight the cells in the second column of the spreadsheet in Figure 10.45 that contain a number that is less than 3.

The screenshot shows a Microsoft Excel spreadsheet titled "Class Party Contributions.xlsx". The spreadsheet has a header row with columns A, B, C, and D. Row 1 contains the text "Contribution per attendee:" in cell A1 and "\$12.00" in cell C1. Rows 2 through 18 contain data for 17 individuals. The first 16 rows have three columns: Name, Additional family members, and Amount. The last row (row 17) has two columns: Name and Amount. The data is as follows:

	A	B	C
1	Contribution per attendee:		\$12.00
2			
3	Name	Additional family members	Amount
4	Aisha	1	\$24.00
5	Alex	2	\$36.00
6	Bala	3	\$48.00
7	Denise	1	\$24.00
8	Farhan	4	\$60.00
9	Gopi	5	\$72.00
10	Irfan	1	\$24.00
11	Jun Ming	4	\$60.00
12	Lily	3	\$48.00
13	Mei Ling	5	\$72.00
14	Muthu	4	\$60.00
15	Nurul	5	\$72.00
16	Priya	5	\$72.00
17	Siti	3	\$48.00
18			

▲ **Figure 10.45** Spreadsheet for tracking class party contributions by each classmate's family

To do this, we would follow the instructions in Table 10.10:

▼ **Table 10.10** Using conditional formatting to highlight cells that contain a number less than 3

Instructions

Step 1:

Select the range of cells that the new rule should be applied to.

	A	B	C	D
1	Contribution per attendee:		\$12.00	
2				
3	Name	Additional family members	Amount	
4	Aisha	1	\$24.00	
5	Alex	2	\$36.00	
6	Bala	3	\$48.00	
7	Denise	1	\$24.00	
8	Farhan	4	\$60.00	
9	Gopi	5	\$72.00	
10	Irfan	1	\$24.00	
11	Jun Ming	4	\$60.00	
12	Lily	3	\$48.00	
13	Mei Ling	5	\$72.00	
14	Muthu	4	\$60.00	
15	Nurul	5	\$72.00	
16	Priya	5	\$72.00	
17	Siti	3	\$48.00	
18				

In this case, we select the cells containing numbers under the second column (range B4:B17).

Step 2:

Select “Conditional Formatting” under the “Home” tab.

Since we wish to highlight the cells that contain a number that is less than 3, we can select “Highlight Cells Rules” and choose the “Less Than...” option.

▼ **Table 10.10** Using conditional formatting to highlight cells that contain a number less than 3 (continued)

	Name	Additional family members	Contribution
3			
4	Aisha	1	\$24.00
5	Alex	2	\$60.00
6	Bala	3	\$48.00
7	Denise	1	\$72.00
8	Farhan	4	\$60.00
9	Gopi	5	\$72.00
10	Irfan	1	\$24.00
11	Jun Ming	4	\$60.00
12	Lily	3	\$48.00
13	Mei Ling	5	\$72.00
14	Muthu	4	\$60.00
15	Nurul	5	\$72.00
16	Priya	5	\$72.00
17	Siti	3	\$48.00
18			

Besides varying the font, fill and border of cells, we can also use the “Data Bars”, “Color Scales” and “Icon Sets” (outlined in green) options to display tiny bar graphs called data bars, or icons that update automatically based on each cell’s contents.

Step 3:

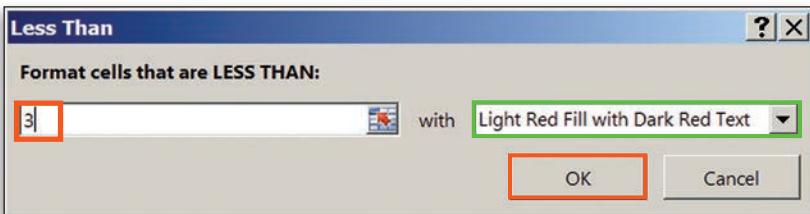
Complete the “LESS THAN” condition by entering a value of 3.

(Note that cell references can also be used here instead of constants. Relative cell references will be changed relative to the top-leftmost cell of the range when applied to other cells in the range.)

The box outlined in green describes how cells that match the condition will be formatted. Besides the default options, we can also choose the “Custom Format...” option to customise the formatting.

▼ **Table 10.10** Using conditional formatting to highlight cells that contain a number less than 3 (continued)

Instructions



In this case, we will use the default option of “Light Red Fill with Dark Red Text” and click “OK”.

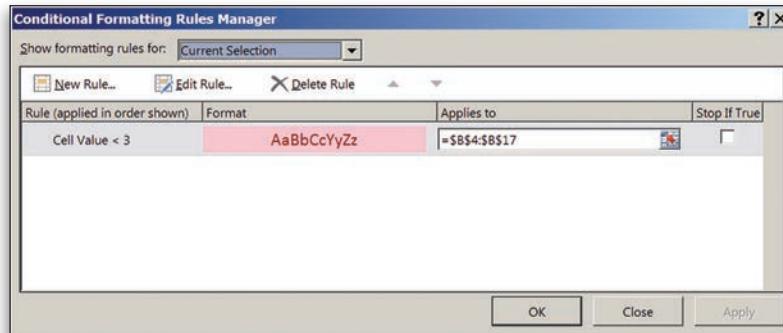
Result:

The cells under the second column which contain a number less than 3 are now highlighted using a light red fill and dark red text, as expected.

Class Party Contributions.xlsx - Excel		
	A	B
1	Contribution per attendee:	\$12.00
2		
3	Name	Additional family members
4	Aisha	1
5	Alex	2
6	Bala	3
7	Denise	1
8	Farhan	4
9	Gopi	5
10	Irfan	1
11	Jun Ming	4
12	Lily	3
13	Mei Ling	5
14	Muthu	4
15	Nurul	5
16	Priya	5
17	Siti	3
18		

Note that changing the contents of highlighted cell to a number that is 3 or more will automatically remove its highlighting. Similarly, changing the contents of a non-highlighted cell in B4:B17 to any number that is less than 3 will automatically highlight it.

If we select the “Manage Rules...” option under the conditional formatting menu while the highlighted cells are selected, we see that there is now a rule applied to range \$B\$4:\$B\$17 that formats any cells in this range containing a value less than 3 with a light red fill and dark red text, as shown in Figure 10.46.



▲ **Figure 10.46** Conditional formatting rules can be edited and/or deleted in the Conditional Formatting Rules Manager window

To remove a conditional formatting rule, select it in the Conditional Formatting Rules Manager window and click “Delete Rule”. To change the rule’s range, condition and/or highlighting style, select it and click “Edit Rule...”.

Sometimes, none of the common cases and options under the “Conditional Formatting” menu are suitable for what is required. In that case, we will need to create a custom rule by selecting “New Rule...” under the conditional formatting menu.

Suppose we wish to highlight the cells in the first column of the spreadsheet in Figure 10.47 that contain a name starting with the letter M.

Class Party Contributions.xlsx - Excel			
FILE	HOME	INSERT	PAGE LAYOUT FORMULAS DATA REVIEW VIEW
A	B	C	D
1	Contribution per attendee:	\$12.00	
2			
3	Name	Additional family members	Amount
4	Aisha	1	\$24.00
5	Alex	2	\$36.00
6	Bala	3	\$48.00
7	Denise	1	\$24.00
8	Farhan	4	\$60.00
9	Gopi	5	\$72.00
10	Irfan	1	\$24.00
11	Jun Ming	4	\$60.00
12	Lily	3	\$48.00
13	Mei Ling	5	\$72.00
14	Muthu	4	\$60.00
15	Nurul	5	\$72.00
16	Priya	5	\$72.00
17	Siti	3	\$48.00
18			

▲ **Figure 10.47** Spreadsheet where conditional formatting is needed

To do this, we would follow the instructions in Table 10.11:

▼ **Table 10.11** Using conditional formatting with a custom rule for names starting with the letter M

Instructions

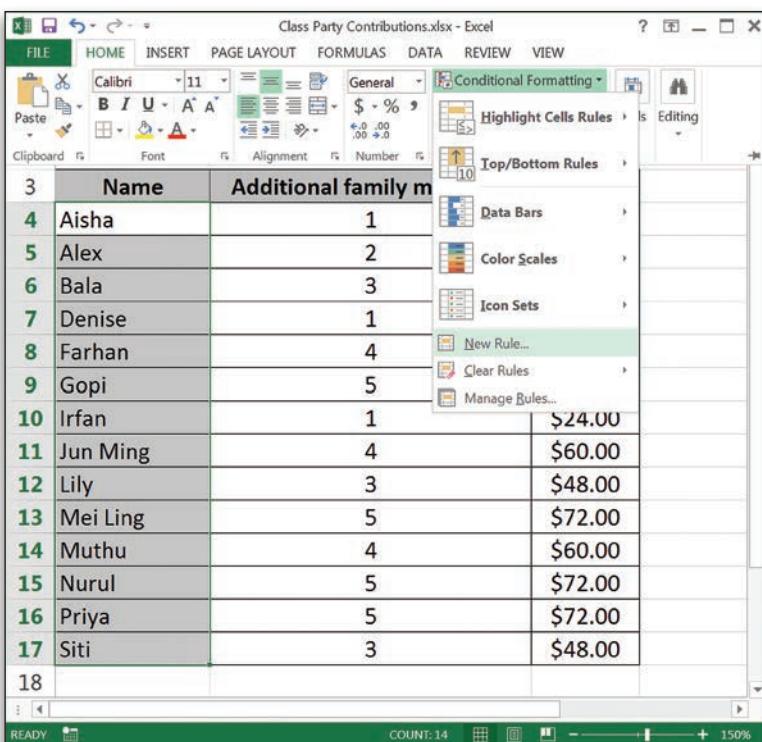
Step 1:
Select the range of cells that the new rule should be applied to.



	A	B	C	D
1	Contribution per attendee:		\$12.00	
2				
3	Name	Additional family members	Amount	
4	Aisha	1	\$24.00	
5	Alex	2	\$36.00	
6	Bala	3	\$48.00	
7	Denise	1	\$24.00	
8	Farhan	4	\$60.00	
9	Gopi	5	\$72.00	
10	Irfan	1	\$24.00	
11	Jun Ming	4	\$60.00	
12	Lily	3	\$48.00	
13	Mei Ling	5	\$72.00	
14	Muthu	4	\$60.00	
15	Nurul	5	\$72.00	
16	Priya	5	\$72.00	
17	Siti	3	\$48.00	
18				

In this case, we select the cells containing the names under the first column (range A4:A17).

▼ **Table 10.11** Using conditional formatting with a custom rule for names starting with the letter M (continued)

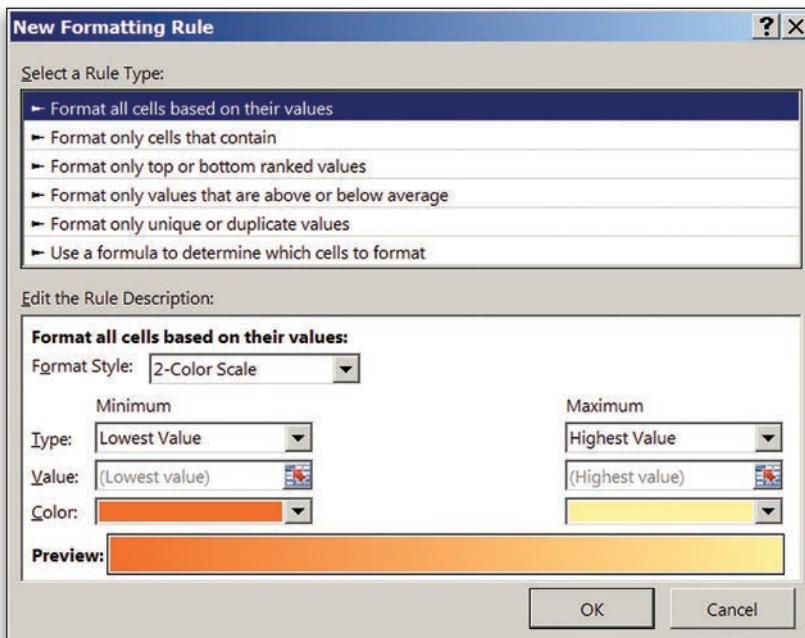
Instructions			
Step 2:	<p>Select “Conditional Formatting” under the “Home” tab.</p> <p>We wish to highlight the cells that contain names starting with the letter M. Unfortunately, none of the common options under the “Conditional Formatting” menu appear suitable for this requirement.</p> <p>This means that we will need to create a custom rule by selecting “New Rule...”.</p>  <p>The screenshot shows a Microsoft Excel spreadsheet titled "Class Party Contributions.xlsx". The data is organized into columns: "Name" and "Additional family members" (colored green), and "Amount" (colored blue). The "Name" column contains names like Aisha, Alex, Bala, etc. The "Amount" column contains values like 1, 2, 3, etc. The "Additional family members" column contains values like 1, 2, 3, etc. The "Amount" column contains values like \$24.00, \$60.00, \$48.00, etc. The "Conditional Formatting" dropdown menu is open, showing options like "Highlight Cells Rules", "Top/Bottom Rules", "Data Bars", "Color Scales", "Icon Sets", "New Rule...", "Clear Rules", and "Manage Rules...". The "New Rule..." option is highlighted with a green background.</p>		

▼ **Table 10.11** Using conditional formatting with a custom rule for names starting with the letter M (continued)

Instructions

Step 3:

The New Formatting Rule window will appear.



None of the first five options appear suitable for checking whether any cells start with the letter M, so we will select the last option: “Use a formula to determine which cells to format”

Step 4:

Now we need a formula that will produce a logical value based on whether the first letter of the name in each cell is the letter M.

We always write custom formulas for conditional formatting rules relative to the top-left cell of the range. In this case, it is cell A4.

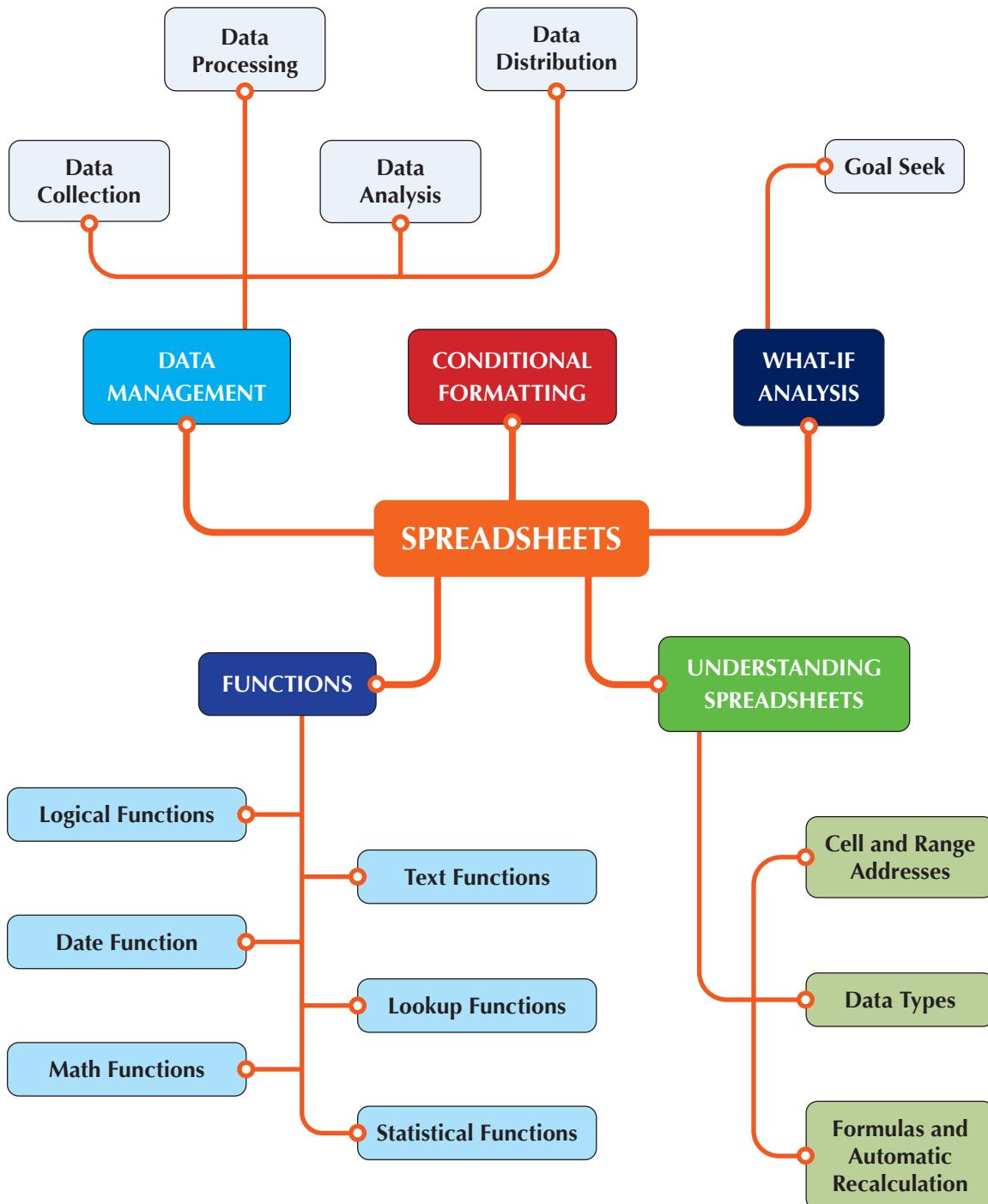
To check whether the first character of A4 is the letter M, we use the formula
“=LEFT(A4, 1) = "M"”.

We then customise the formatting to have a yellow fill and click “OK”.

▼ **Table 10.11** Using conditional formatting with a custom rule for names starting with the letter M (continued)

Instructions																																																		
<pre> New Formatting Rule Select a Rule Type: - Format all cells based on their values - Format only cells that contain - Format only top or bottom ranked values - Format only values that are above or below average - Format only unique or duplicate values - Use a formula to determine which cells to format Edit the Rule Description: Format values where this formula is true: =LEFT(A4, 1) = "M" Preview: AaBbCcYyZz Format... OK Cancel </pre>																																																		
<p>Result:</p> <p>Cells under the first column containing a name which starts with the letter M are now highlighted using a yellow fill.</p> <table border="1"> <thead> <tr> <th colspan="3">Contribution per attendee:</th> </tr> <tr> <th>Name</th> <th>Additional family members</th> <th>Amount</th> </tr> </thead> <tbody> <tr> <td>Aisha</td> <td>1</td> <td>\$24.00</td> </tr> <tr> <td>Alex</td> <td>2</td> <td>\$36.00</td> </tr> <tr> <td>Bala</td> <td>3</td> <td>\$48.00</td> </tr> <tr> <td>Denise</td> <td>1</td> <td>\$24.00</td> </tr> <tr> <td>Farhan</td> <td>4</td> <td>\$60.00</td> </tr> <tr> <td>Gopi</td> <td>5</td> <td>\$72.00</td> </tr> <tr> <td>Irfan</td> <td>1</td> <td>\$24.00</td> </tr> <tr> <td>Jun Ming</td> <td>4</td> <td>\$60.00</td> </tr> <tr> <td>Lily</td> <td>3</td> <td>\$48.00</td> </tr> <tr> <td>Mei Ling</td> <td>5</td> <td>\$72.00</td> </tr> <tr> <td>Muthu</td> <td>4</td> <td>\$60.00</td> </tr> <tr> <td>Nurul</td> <td>5</td> <td>\$72.00</td> </tr> <tr> <td>Priya</td> <td>5</td> <td>\$72.00</td> </tr> <tr> <td>Siti</td> <td>3</td> <td>\$48.00</td> </tr> </tbody> </table>			Contribution per attendee:			Name	Additional family members	Amount	Aisha	1	\$24.00	Alex	2	\$36.00	Bala	3	\$48.00	Denise	1	\$24.00	Farhan	4	\$60.00	Gopi	5	\$72.00	Irfan	1	\$24.00	Jun Ming	4	\$60.00	Lily	3	\$48.00	Mei Ling	5	\$72.00	Muthu	4	\$60.00	Nurul	5	\$72.00	Priya	5	\$72.00	Siti	3	\$48.00
Contribution per attendee:																																																		
Name	Additional family members	Amount																																																
Aisha	1	\$24.00																																																
Alex	2	\$36.00																																																
Bala	3	\$48.00																																																
Denise	1	\$24.00																																																
Farhan	4	\$60.00																																																
Gopi	5	\$72.00																																																
Irfan	1	\$24.00																																																
Jun Ming	4	\$60.00																																																
Lily	3	\$48.00																																																
Mei Ling	5	\$72.00																																																
Muthu	4	\$60.00																																																
Nurul	5	\$72.00																																																
Priya	5	\$72.00																																																
Siti	3	\$48.00																																																

Chapter Summary



Review Questions

- The spreadsheet in Figure 10.48 is used to tabulate the monthly interest earned and money withdrawn from a bank account.

Figure 10.48 Interest earned and money withdrawn from a bank account

	A	B	C	D	E	F	G	H
1	Month	Balance at Start	Interest Earned	Money Withdrawn		Interest rate:	0.40%	
2	January	\$5,000.00		\$34.84				
3	February	\$4,965.16		\$140.73				
4	March	\$4,824.43		\$96.34				
5	April	\$4,728.09		\$56.97				
6	May	\$4,671.12		\$102.42				
7	June	\$4,568.70		\$33.31				
8	July	\$4,535.39		\$157.24				
9	August	\$4,378.15		\$151.30				
10	September	\$4,226.85		\$68.14				
11	October	\$4,158.71		\$35.26				
12	November	\$4,123.45		\$168.53				
13	December	\$3,954.92		\$133.33				
14								
15	Amount left at end of December:			\$3,821.59				
16								

When Ctrl-` is pressed, the spreadsheet looks like this:

Figure 10.49 Spreadsheet for bank account with formulas shown

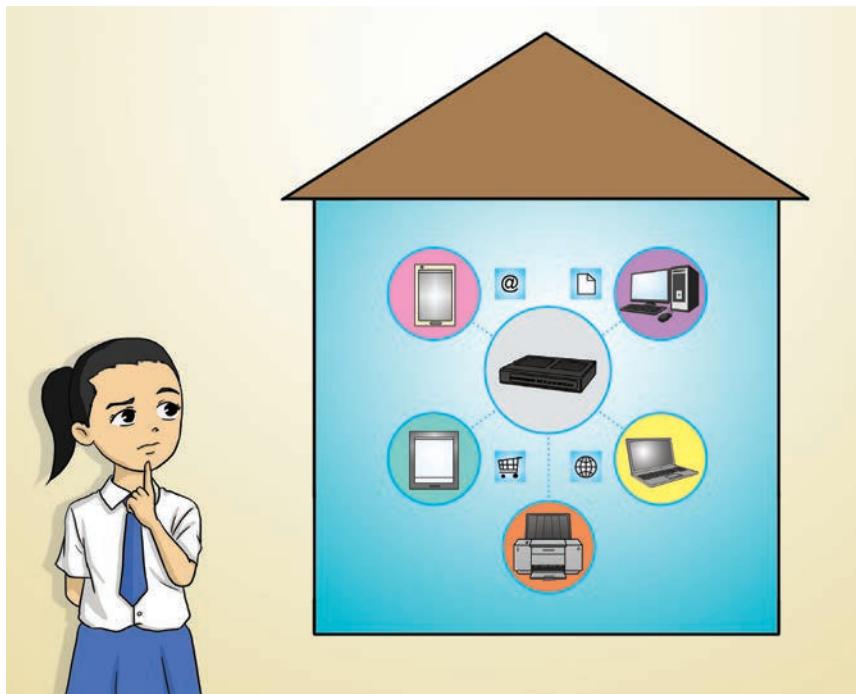
	A	B	C	D	E	F	G
1	Month	Balance at Start	Interest Earned	Money Withdrawn		Interest rate:	0.004
2	January	5000		34.84			
3	February	=B2+C2-D2		140.73			
4	March	=B3+C3-D3		96.34			
5	April	=B4+C4-D4		56.97			
6	May	=B5+C5-D5		102.42			
7	June	=B6+C6-D6		33.31			
8	July	=B7+C7-D7		157.24			
9	August	=B8+C8-D8		151.3			
10	September	=B9+C9-D9		68.14			
11	October	=B10+C10-D10		35.26			
12	November	=B11+C11-D11		168.53			
13	December	=B12+C12-D12		133.33			
14							
15	Amount left at end of December:			=B13+C13-D13			
16							

Create this spreadsheet with the formulas shown in Figure 10.58 using a spreadsheet program, then answer the following questions:

- a) Suggest a formula for cell C2 that can be copied into range C3:C13 in order to calculate the interest earned for each month before money is withdrawn, using the interest rate that is specified in cell G1.
- b) In the program, use Goal Seek to determine what the monthly interest rate must be to have \$4,200 left at the end of December. Give your answer to two decimal places.
- c) In the program, use conditional highlighting to automatically colour the text of cells in column D red if they show a withdrawal amount greater than \$100.

How Do I Create a Simple Network?

Siti's family has several devices, such as laptops, mobile phones, tablets and a wireless printer. She wants to create a simple network to connect all these devices together in her new home so that data can be transferred between them. Previously, she could only print her assignments from her brother's desktop computer but not from her laptop. She would like to print from both devices.



In previous chapters, we learnt about ethical issues that relate to the use of computers in public and private networks. The world is becoming increasingly connected with the growing number of network hardware being linked to the Internet. Computer networking has enabled the fast and effective transfer of data, which is essential for the daily operations of businesses and our way of life.

In this chapter, we will learn about what makes up a network and the different types of networks. We will also learn about the practical concerns of implementing a wired and a wireless network, the differences between client-server and peer-to-peer network architectures, and the components of a simple home network.



By the end of this chapter, you should be able to:

- Identify and explain the function of different network hardware: modem, network interface controller, hub, switch and router.
- Describe the difference between wired and wireless networks and explain the factors that will determine the use of each type of network.
- Describe the components for a simple home network and design a simple home network.
- Compare and contrast client-server and peer-to-peer network strategies with emphasis on:
 - Purpose
 - Function
 - Organisation
 - Bandwidth
- Explain the use of parity and checksums in data transmission.

11.1 Computer Network

A computer network is a system of two or more computers that are connected together by a **transmission medium** for the exchange of data.

Table 11.1 on the next page shows some advantages and disadvantages of a computer network.

Key Term

Transmission medium
Means of connecting two or more computers together, such as using copper cables, radio waves or light pulses, so that they may exchange data and interact with each other

▼ **Table 11.1** Advantages and disadvantages of a computer network

Advantages	Disadvantages
<ul style="list-style-type: none"> • Shared resources: A network allows a group of computers to make use of shared resources such as printers or files. • Shared Internet access: Depending on the network's configuration, every user who logs on to the network may have access to the Internet. • Shared software: Software can be stored on the central server of a network and deployed to other computers over a network. • Shared storage: Data files can be stored on a central server for ease of access and backup purposes. • Communication: Computers in the same network are often able to share instant messages and emails for communication. 	<ul style="list-style-type: none"> • Initial costs: Installing a network could be costly due to the high setup and equipment costs. • Maintenance costs: There are also subsequent costs associated with administering and maintaining the network. • Security risks: As files are shared through a network, there is the risk of virus or worm attacks spreading throughout the network even with just one infected computer. • Risk of data loss: Data may become lost due to hardware failures or errors. Using a network means regular data backups are needed. • Server outage: If the server fails, the network will not be able to function, thus affecting work processes.

11.2 Types of Computer Networks

Computer networks can be classified according to their geographical size, transmission medium and network organisation type.

11.2.1 Geographical Size

Depending on the geographical coverage of the network, a network can be classified into one of the following types:

1. Local area network
2. Metropolitan area network
3. Wide area network

The number of users that it serves will also vary according to its geographical size.

11.2.1.1 Local Area Network

A **local area network (LAN)** is a network of computing devices connected within a small geographical area, typically within the same building, such as a home, school or office. Due to the small number of connections supported and the close proximity of the devices, LANs typically provide faster data transfer than the other networks.

The size of the network in Siti's home is relatively small. Thus, her home network is a LAN.

11.2.1.2 Metropolitan Area Network

A **metropolitan area network (MAN)** is a network of computing devices covering a larger geographical area – two or more buildings within the same town or city – than a LAN. A MAN is typically owned and operated by a large organisation such as a business or government body.

11.2.1.3 Wide Area Network

A **wide area network (WAN)** is a network of computing devices covering a large-scale geographical area, typically across multiple geographical locations.

A WAN generally consists of multiple smaller networks such as LANs or MANs.

A WAN differs from a LAN in several ways. Unlike a LAN, a WAN is not limited to a single geographical location and can span long distances via long-range transmission media such as telephone lines, fibre optic cables or satellite links.

WANs can be either private or public. Private WANs are built and maintained by large multinational companies and government organisations. The Internet is an example of a public WAN. A WAN also uses more expensive and high-speed technology than a LAN.

Key Terms

Local area network (LAN)

Network of computing devices connected within a small geographical area, typically within the same building, such as a home, school or office

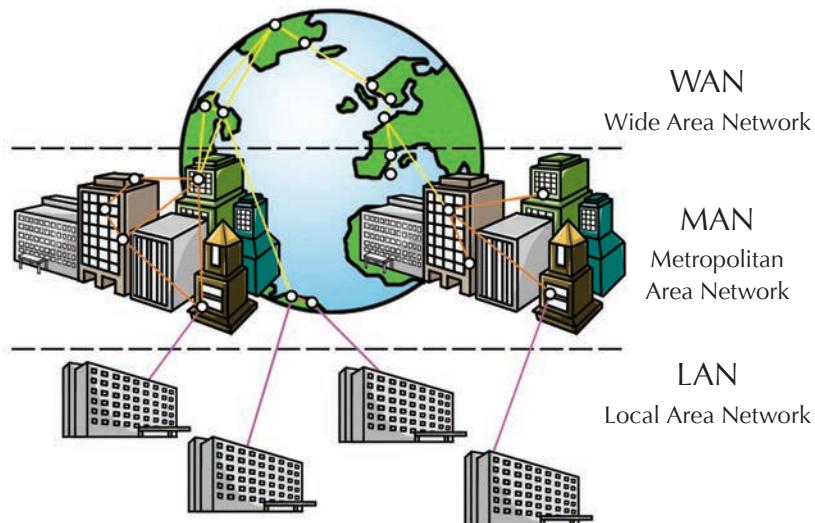
Metropolitan area network (MAN)

Network of computing devices typically spanning across two or more buildings within the same town or city

Wide area network (WAN)

Network of computing devices covering a large-scale geographical area, typically across multiple geographical locations

Figure 11.1 compares the geographical distribution of these three types of network.



▲ **Figure 11.1** Geographical distribution of LAN, MAN and WAN

11.2.2 Transmission Medium

Networks can also be classified according to their transmission medium, such as cables or electromagnetic waves. The following sections will describe wired and wireless networks and their transmission media.

11.2.2.1 Wired Networks

A **wired network** is a network of devices connected by a physical medium, such as cables. Data transfer is typically faster and more secure in a wired network. However, as the number of devices and the distance between devices increase, the cost of setting up the network increases as well.

Computers in a network communicate using a set of **network protocols**, just like how people communicate using languages. The **Ethernet** is the most widely used wired network protocol in LANs and MANs. Figure 11.2 shows the connector of an Ethernet cable that is used to connect devices in a wired network.

Key Terms

Ethernet

Most commonly used wired network protocol for local and metropolitan area networks

Network protocol

Set of standards and rules that govern how two or more devices communicate over a network

Wired network

Network of devices connected by a physical medium, such as cables



▲ **Figure 11.2** An Ethernet cable connector

11.2.2.2 Wireless Networks

A **wireless network** is a network of devices in which signals are transmitted without the use of a physical medium. The transmission is in the form of electromagnetic waves, such as radio waves and microwaves. Users can be connected to a wireless network as long as they are within range of the network coverage.

The most common wireless network protocol is Wi-Fi, which uses radio waves to transmit data.

A **wireless access point (WAP)** is network hardware that provides a connection between wireless devices up to 100 metres away and can connect to wired networks. Obstacles such as walls or metal frames can reduce the strength of Wi-Fi signals. Radio signals such as those from microwave ovens can also interfere with Wi-Fi signals.

Wireless networks are becoming increasingly popular in homes and businesses as they are becoming lower in cost and are easy to configure and manage.

11.2.3 Organisation

Computers in a network are generally classified as either clients or servers.

A **client** is a computer that initiates a connection to a server to request for resources and services to perform operations. Employees in offices or students in schools would normally use client computers to do their work.

A **server** is a computer that shares resources and responds to requests from devices and other servers on the network. It usually has a higher capacity and is more powerful than a client as it needs to manage resources and services. These might include:

- Providing central storage of files
- Sharing hardware such as printers
- Controlling logins and network access

Although any computer can function as a server, computers that are designed specifically for use as servers are built to be reliable and less prone to failure. This means that they may be much more expensive than normal computers.

Computers can be organised in either a client-server or peer-to-peer network.

Key Terms

Client

Computer that initiates a connection to a server to request for resources and services to perform operations

Server

Computer that shares resources with and responds to requests from devices and other servers on the network

Wireless access point (WAP)

Network hardware that provides a connection between wireless devices up to 100 metres away and can connect to wired networks

Wireless network

Network of devices in which signals are transmitted without the use of a physical medium

11.2.3.1 Client-Server Network

In a client-server network, one or more higher-capacity computers will act as servers while the remaining computers are clients. Each server contains data and other resources to be shared with clients. The server also fulfils requests from clients.

Table 11.2 shows the advantages and disadvantages of a client-server network.

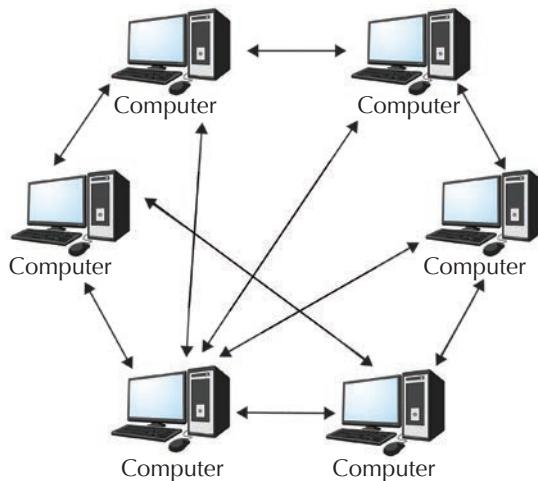
▼ **Table 11.2** Advantages and disadvantages of a client-server network

Advantages	Disadvantages
<ul style="list-style-type: none"> Centralised control of data and resources Easy to schedule backups of all shared files at regular intervals Security may be enhanced with the use of specialised software or operating system features that are designed for servers 	<ul style="list-style-type: none"> Higher initial cost due to the need for a server Administrative costs needed for the maintenance of server and clients

11.2.3.2 Peer-To-Peer Network

Peer-to-peer (P2P) is a type of network where all computers are considered as equals and the load is distributed among all computers. Each computer in the network is able to act as both a client and a server, communicating directly with other computers. Users are able to share files and resources located on their computers as well as access shared resources found on other computers in the network. These networks are low in cost.

Figure 11.3 shows an example of a P2P network.



▲ **Figure 11.3** A P2P network

Table 11.3 shows the advantages and disadvantages of a P2P network.

▼ **Table 11.3** Advantages and disadvantages of a P2P network

Advantages	Disadvantages
<ul style="list-style-type: none"> • Cheaper to set up as there is no cost related to dedicated servers • Easy to set up as no specialised software or operating system features are needed 	<ul style="list-style-type: none"> • More effort is required to access and back up resources as they are stored locally within each computer instead of centrally in a server • Security is an issue as access rights are not administered by a central server

Table 11.4 compares client-server and P2P networks.

Key Term

Bandwidth

The average number of bits of data that can be transmitted from a source to a destination over the network in one second; usually measured in megabits per second (Mbps) or gigabits per second (Gbps)

▼ **Table 11.4** Comparison between client-server and P2P networks

Feature	Client-server	P2P
Function	Data and resources are shared using one or more dedicated servers; each computer has a distinct role – client or server	Data and resources are shared directly between computers; each computer acts as both a client and server
Organisation of hardware	Each client is connected to one or more dedicated servers	Each computer in the network can serve as a client and server at the same time
Bandwidth	Typically high but limited by the capability of the server	Varies depending on how data needs to be transmitted; bandwidth may be reduced if a single computer must handle a large request, but may be increased if a large request can be divided into smaller requests that are handled by multiple computers simultaneously

▼ Table 11.4 Comparison between client-server and P2P networks (continued)

Feature	Client-server	P2P
Security	High as access rights can be controlled centrally at a server	Low as security is handled by each computer and not by a central server
Setup cost	High as the use of specialised high-performance servers would be needed	Low as basic computers can act as servers to share resources
Storage	Centralised and carried out only at the server; usually managed by a network administrator	Decentralised and can be carried out by individual users at each computer
Application	Found in businesses or organisations with a large number of users	Found in homes or small businesses where there are fewer users



Quick Check 11.2

1. Company X has a remote office in another country that needs to access information from its main headquarters. What is the type of network used in this case?
 - A LAN
 - B MAN
 - C WAN
 - D None of the above

2. The main office of Y Corporation occupies five floors in their building. What is the type of network used in this case?
 - A LAN
 - B MAN
 - C WAN
 - D None of the above

3. Write down three advantages of a client-server network.

4. State whether each of the features below describes a client-server network or a peer-to-peer network:
 - a) Every computer is able to directly share resources with other computers in the network.
 - b) Each computer has a distinct role in the network.
 - c) It is easier to schedule backups of shared files on this type of network.

11.3 Factors Affecting Choice of Transmission Medium

Both wired and wireless networks have their advantages and disadvantages.

Table 11.5 shows the factors that should be considered when deciding between a wired or wireless network.

▼ **Table 11.5** Comparison between wired and wireless networks

Factor	Wired	Wireless
Cost	Initially cheaper but becomes more expensive as network grows in size due to the cost of cables	Initially expensive due to the cost of wireless networking equipment but becomes more cost-effective as network grows in size
Speed of transmission and bandwidth	Faster and higher bandwidth as cables provide dedicated connection	Generally slower and lower bandwidth due to possible interference from radio waves or microwaves; varies according to user location in relation to network
Reliability	More reliable as data transmission is unaffected by interference	Less reliable due to potential interference from radio waves and microwaves or blockage from physical obstructions
Security	More secure as the network is less susceptible to interception and hacking	Less secure due to possible intrusion by hackers
Mobility of users	Lower as network connections are fixed at specific spots and users cannot move to other locations	Higher as users can move about freely within the range of the wireless network
Scalability	More cumbersome to add new devices to the network as physical constraints and the running of cables need to be considered	Easier to add new devices to the network as the router can be easily configured
Physical organisation	Tends to look more disorganised due to cables running across floors	More organised without cables



Quick Check 11.3

1. Siti wants to send print jobs from both her laptop as well as tablet whenever she is at home. Describe two advantages of using a wireless network to enable printing from multiple devices in her home.
2. Give two reasons why wireless networks are preferred over wired networks at cafes and canteens.

11.4 Identifiers

Networks typically have many computers and programs using them at any one time. In order to identify the different networks, computers and programs, we need to give them names and describe their respective functions.

In Chapter 8, you learnt how IP addresses (sections 8.4.2.1 and 8.4.2.2) and MAC addresses (section 8.4.2.3) are used to identify individual computers so that data transmitted over a network can be directed to the correct destination. The following sections will discuss other ways that networks, computers and programs can be identified – using port numbers and SSIDs.

11.4.1 Port Numbers

A **port number** is used in combination with an IP address to identify a program that is running on a network. Each program that is active on the network must be identifiable by one or more unique combinations of an IP address and a port number. Both clients and servers can run programs that use port numbers to identify themselves.

A computing device can run multiple programs at the same time. For example, a single computer can run both a web server program with port number 80 and a mail server program with port number 143. All port numbers are assigned in a range from 0 to 65,535.

Key Term

Port number

Number that is used together with an IP address to uniquely identify a program that is running on a network

Did you know?

You can list all the port numbers that are in use on your computer by entering “netstat -na” at the command prompt.

11.4.2 Service Set Identifiers

A **service set identifier (SSID)** is a string of up to 32 bytes that identifies a wireless access point (WAP) and all the devices connected to it. All wireless devices connected to the same WAP must use the same SSID.

If Siti decides to set up a wireless network at home using a WAP, she will need to configure an SSID for her WAP so that her devices can select this WAP and connect to it.

Figure 11.4 shows an example of the SSID of a wireless network.

Key Term

Service set identifier (SSID)

A 32-byte string that identifies a wireless access point (WAP) and all devices connected to it



▲ Figure 11.4 An example of the SSID of a wireless network



Quick Check 11.4

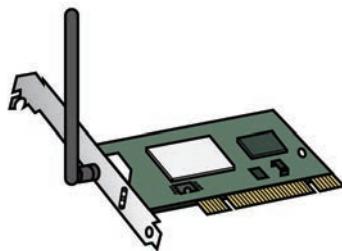
1. On a computer or laptop connected to the network, use the command line interface or network tools available to find its MAC and IPv4 addresses.
2. Choose the correct words in the statements below from the given choices:
 - a) A program running on a network can be uniquely identified by its (*MAC address / port number / SSID*) and IP address.
 - b) All (*wired/wireless*) devices connected to the same WAP must use the same SSID.

11.5 Network Hardware and Their Functions

Data is seldom sent in a single stream over networks. Instead, it is broken up into smaller **packets** before being sent independently over the network through devices such as hubs, switches and routers. Besides containing a piece of the original data, each packet also contains a header with information about the source and destination addresses that are needed for transmission. These data packets are then reassembled at the destination device.

11.5.1 Network Interface Controller

A network interface controller (NIC) provides the hardware interface to enable the transfer of data between a device and a network. An NIC may connect to a network physically or wirelessly. Most devices are equipped with a built-in NIC. Figure 11.5 shows an example of a wireless NIC.



▲ Figure 11.5 A wireless NIC

As discussed in section 8.4.2.3, each NIC also has a unique 48-bit MAC address that can be used to identify the particular computer or device with that NIC.

11.5.2 Network Hub

A **network hub** is the simplest way to connect multiple devices to the same network. When a hub receives a packet, the packet is transmitted to all the devices that are connected to the hub. Likewise, when a device responds, the response is sent to every device that is connected to the hub.

In this way, a hub acts like a loudspeaker as it broadcasts the data to all its connected devices without limiting the data to only the specific device it was intended for.

Key Terms

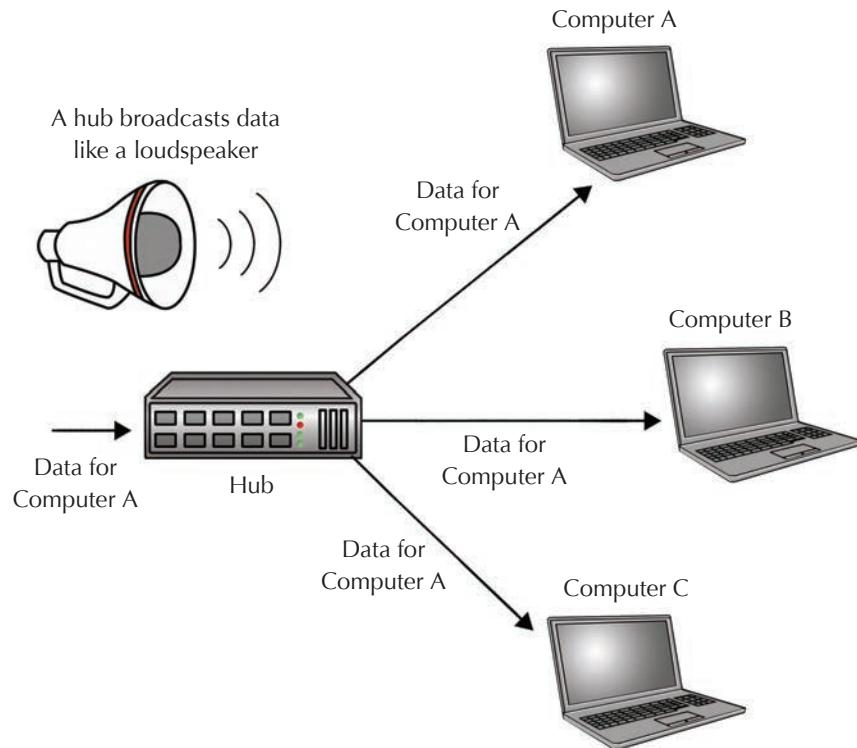
Network hub (or hub)

Device that transmits received packets to all connected devices

Packet (computing)

A unit of broken-up data containing a header with information about the source and destination addresses that are needed for transmission

Figure 11.6 shows how a network hub works.



▲ Figure 11.6 How a network hub works

A hub is the cheapest option among all the network connection devices as it does not store any information about the devices that are connected to it. However, because the hub transmits data to all connected devices regardless of the intended recipient, it can cause bottlenecks that reduce the overall efficiency of the network.

11.5.3 Network Switch

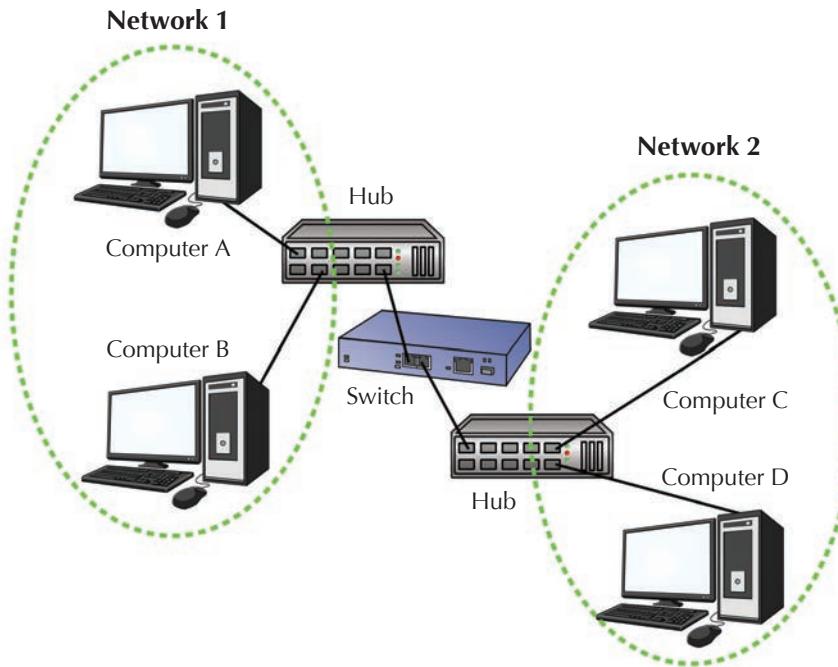
A **network switch** constructs a single network by connecting multiple similar networks together. Switches are typically used to connect multiple LANs that use the same protocol so that the combined network can cover a larger physical area.

Unlike a hub that simply repeats data to all connected devices, a switch uses MAC addresses to keep track of the devices that are connected to it. This lets the switch intelligently decide whether it should drop or forward the packets that it receives.

Key Term

Network switch (or switch)
Device that constructs a single network by connecting multiple similar networks together

In Figure 11.7, suppose the switch receives a packet from computer A. The switch first examines the destination MAC address stored in the packet's header and decides whether to forward or drop the packet.



▲ **Figure 11.7** How a switch connects two networks

If the destination MAC address is that of computer B, the switch will drop the packet as computers A and B are already on the same network.

On the other hand, if the destination MAC address is that of computer C or D, the switch will forward the packet to the other network and on towards its intended destination.

Did you know?

A network switch that connects exactly two networks together is also called a network bridge. Most large networks use switches and bridges instead of hubs as these devices are more “intelligent” and will send packets through a connection only if the switch or bridge determines that the intended recipient is on the other end, hence avoiding unnecessary bottlenecks. This makes using switches and bridges more efficient than using hubs.

11.5.4 Router

A **router** forwards packets between separate networks. While a switch combines multiple similar networks that use the same protocol into a single network, a router keeps the connected networks (which may use fundamentally different protocols) separate and forwards packets between them using Internet protocols.

Key Term

Router

Device that forwards packets between separate networks

For instance, Siti's Internet service provider (ISP) has supplied her with a modem (described in section 11.5.5) that connects to the ISP's network using a special protocol that works over telephone lines. For Siti to send packets through the ISP's network using the Ethernet connection on her computer, she will need a router to forward packets between her computer, which uses Ethernet, and the ISP's network, which uses a different protocol.

In order for a router to forward packets between different networks using Internet protocols, both the device *sending* the packet and the device *receiving* the packet must be identified using IP addresses.

Note that switches (described in section 11.5.3) forward packets based on permanent MAC addresses, while routers forward packets based on IP addresses that may change dynamically.

Did you know?

The name "Internet" actually comes from the term "internetworking", which refers to the practice of forwarding packets between different and separate networks. The Internet Protocol achieves this by serving as an additional "overlay" protocol on top of the different Ethernet, Wi-Fi and other protocols used by the separate networks.

11.5.5 Modem

While Ethernet cables are useful for LANs and some MANs, they are not suitable for connecting devices over a long distance (such as from Siti's ISP to her home) as the number of transmission errors increases with cable length. Instead, ISPs typically use special protocols and long-range transmission media such as telephone lines or fibre optic cables.

Long-range transmission media, however, are typically not designed for transferring digital data (i.e., 0 and 1 bits) that are used by computers. For instance, telephone lines are designed for transmitting analogue sounds and not digital data. Hence, it becomes necessary to convert digital data into a form suitable for the transmission medium before it can be transmitted.

The process of converting digital data into a form suitable for transmission is known as **modulation**. The reverse process is known as **demodulation**. On either end of the transmission, the device responsible for modulation and demodulation is known as a **modem** (short for “modulator-demodulator”).

When setting up Internet access in a home network, an ISP will typically provide a modem that sets up a long-distance connection to the ISP's network.

In Figure 11.8, a wireless router and a modem are installed to provide Internet access to multiple devices in the home.

Key Terms

Demodulation

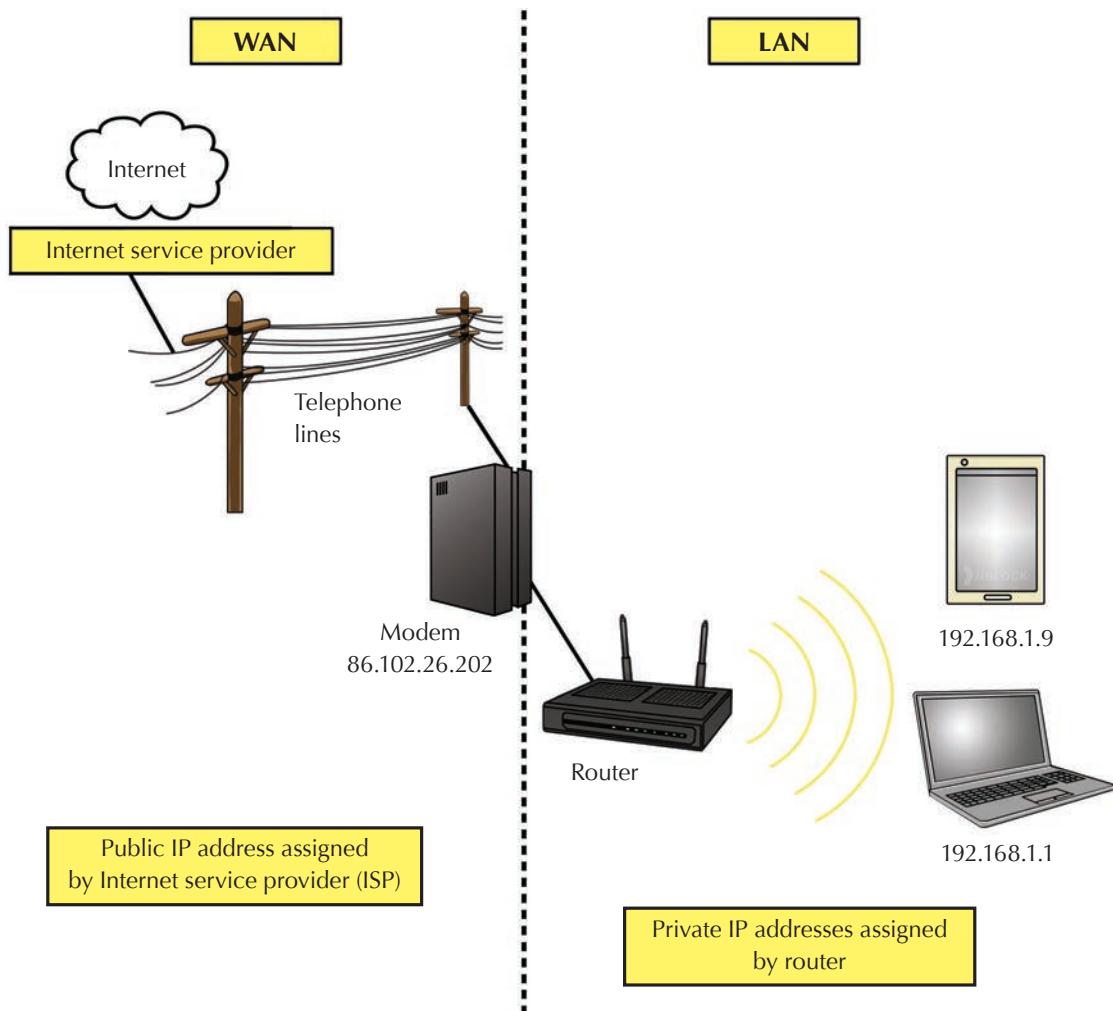
Conversion of transmitted signals into digital data

Modem

Device responsible for modulation and demodulation

Modulation

Conversion of digital data into a form suitable for transmission



▲ **Figure 11.8** How a LAN is connected to the public network using a modem



Quick Check 11.5

1. Which of the following network hardware allows separate networks that use different network protocols to be connected together?
A Firewall
B Hub
C Switch
D Router
2. Give two examples of network hardware used in your home and school.

11.6 Network Topologies

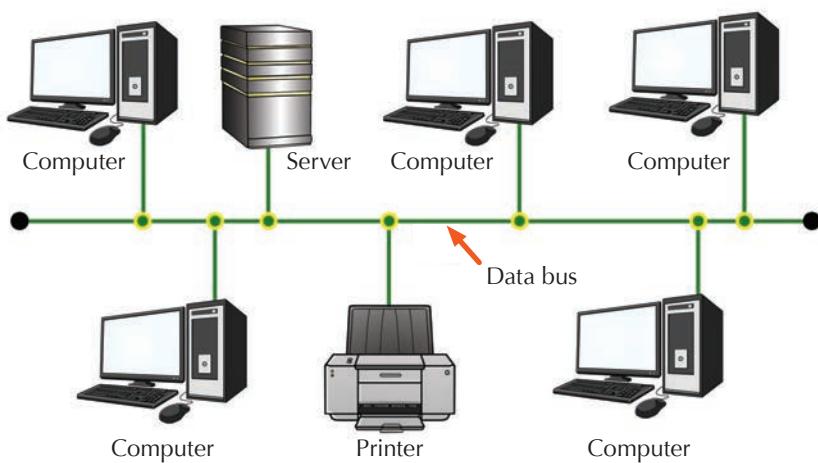
A topology describes the physical layout of a network. Understanding the topology is essential to designing a network.

11.6.1 Bus Topology

In the bus topology, a common cable or backbone known as the bus connects all the devices. The bus is a medium that allows the transmission of data. It also allows the devices to communicate with the server, with each other and with devices such as a shared printer.

During communication, a sender will transmit data along the bus and all the devices connected to the bus can detect that data is being transmitted. However, only the intended recipient will accept and process the data.

Figure 11.9 shows how the devices are connected in a bus topology.



▲ Figure 11.9 A bus topology network

Table 11.6 shows the advantages and disadvantages of a bus topology.

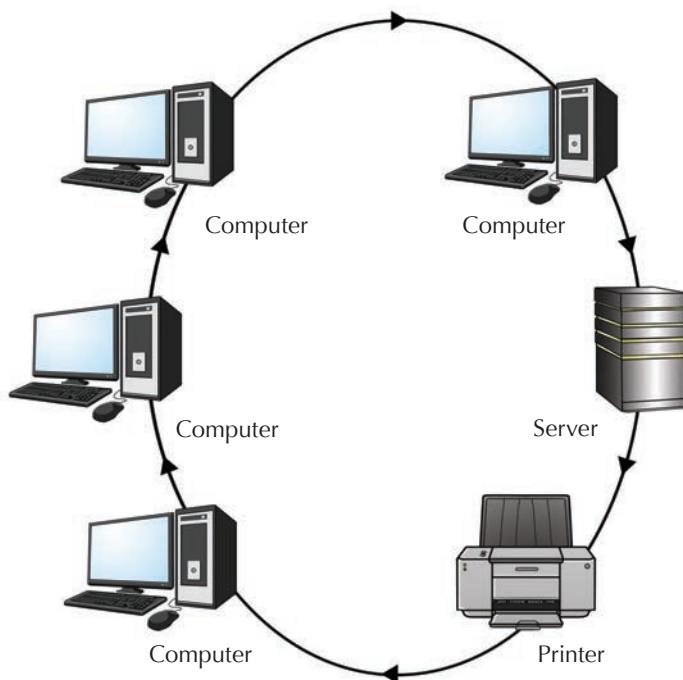
▼ **Table 11.6** Advantages and disadvantages of a bus topology

Advantages	Disadvantages
<ul style="list-style-type: none"> • Easy and cheap to install as it uses less cabling than other network designs • Scalable as new computers can be easily added • Can continue to operate even when one of the computers breaks down • Works well for small networks 	<ul style="list-style-type: none"> • A break anywhere along the bus may disable the entire network • The size of the network is limited by the capacity and length of the bus • A single bus is unsuitable for networks with many computers; performance slows down as the number of computers increases

11.6.2 Ring Topology

In the ring topology, each computer is connected to two other computers in a ring formation. All the data is passed around in the same direction. If a failure occurs in the cable or if a computer breaks down, the entire network will fail to function.

Figure 11.10 shows how the devices are connected in a ring network.



▲ **Figure 11.10** A ring topology network

Table 11.7 shows the advantages and disadvantages of a ring topology.

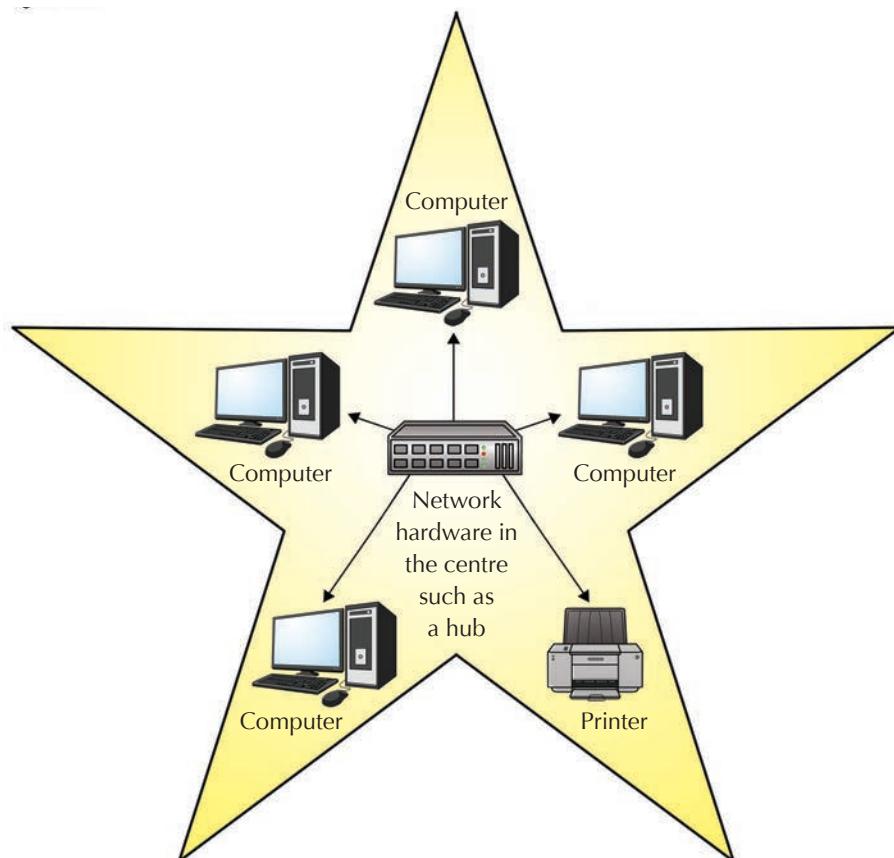
▼ **Table 11.7** Advantages and disadvantages of a ring topology

Advantages	Disadvantages
<ul style="list-style-type: none"> • Can operate over larger distances and handle more data than a bus topology • Data packets that are sent between two computers will pass through intermediate computers, hence a central server is not required to manage the network 	<ul style="list-style-type: none"> • If a computer or cable in the network fails, the entire network may fail as the data cannot be passed on • Adding a new computer to the ring network would mean that the whole communication ring needs to be temporarily interrupted

11.6.3 Star Topology

In the star topology, network hardware such as a hub or switch is at the centre of the network with connections to all the other computers. The computers will send data to the central network hardware and the hardware forwards the data to the intended destination.

Figure 11.11 shows how the devices are connected in a star topology.



▲ **Figure 11.11** A star topology network

Table 11.8 shows the advantages and disadvantages of a star topology.

▼ **Table 11.8** Advantages and disadvantages of a star topology

Advantages	Disadvantages
<ul style="list-style-type: none"> The load on each section of cabling is reduced as each computer uses a separate cable from the rest If a fault occurs at a computer or cable, it is easy to isolate the fault and do a replacement without affecting the rest of the network 	<ul style="list-style-type: none"> Uses more cabling than other topologies and hence costs more If the central network hardware fails, the entire network fails



Quick Check 11.6

- The _____ describes how the network is arranged and how resources are shared.
 - communication channel
 - network topology
 - network hub
 - local area network
- Which of the following network topologies is the least reliable in the event of a breakdown of a computer in the network?
 - Bus
 - Client-server
 - Ring
 - Star
- Jake plans to set up an accountancy consulting business in town. He wants to have a LAN in his office. He has about 100 staff working in the same office while about 20 other staff work from offsite locations such as his clients' offices. The staff need to share files among themselves and be able to connect to printers. Most of the data being handled is confidential and dates back from 10 years ago to the present. He also has plans to expand his business in the next five years.
 - Which network topology is most suitable in this case? Why?
 - Explain why a client-server network is preferred over a peer-to-peer network by considering bandwidth, security and storage issues.

11.7 Error-Checking Methods in Data Transmission

Data can become lost or corrupted during transmission. As mentioned in section 11.5, data is seldom sent in a single stream, but is instead broken up and sent in smaller packets. This allows us to focus on detecting transmission errors one packet at a time.

There are two basic error-checking methods to ensure that the data received at the destination is the same as that at the source:

1. Parity check
2. Checksum

Each packet illustrated in the following examples is eight bits long. Note that, in reality, the packets are usually much longer.

11.7.1 Parity Check

In **parity check**, an additional bit is either appended or prepended to a string of binary data for transmission. This **parity bit** can be either 0 or 1, depending on whether an odd or even parity system is used. Parity bits are the simplest method of error detection.

In the odd parity system, the total number of 1 bits in each packet (including the parity bit) should be odd. In the even parity system, the total number of 1 bits in each packet (including the parity bit) should be even.

For example, the packet in Figure 11.12 has the original data bits 1010001. There are three 1 bits in the original data string. In an even parity system, the parity bit of 1 would be added to increase the total number of 1 bits in the packet to four, which is an even number.

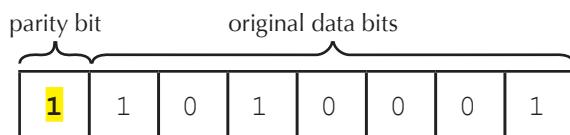
Key Terms

Parity bit

Additional bit, which can be either 0 or 1 depending on whether an odd or even parity system is used, that is appended or prepended to a string of binary data before transmission

Parity check

Error-checking technique which uses a parity bit to detect errors



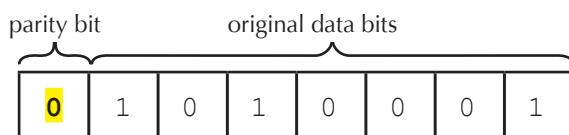
▲ **Figure 11.12** Each packet in an even parity system should have an even number of 1 bits

Table 11.9 describes the two possible scenarios that may occur when the packet arrives at the destination. If the data transmission is successful, the total number of 1 bits detected would be even, which is the same as that at the source. This means that the data is error-free. However, if there is a simple error in the data transmission, the total number of 1 bits detected would be odd, which differs from that at the source. This means that the data is corrupted.

▼ **Table 11.9** Successful and failed data transmission in an even parity system

Original data bits	Parity system – even	Data transmission	
		Successful	Unsuccessful
1010001	11010001	11010001 Total number of 1 bits is even	11110001 Total number of 1 bits is odd

If the same packet is used in an odd parity system instead, then the parity bit of 0 is added, as there is already an odd number of 1 bits in the original data string.



▲ **Figure 11.13** Each packet in an odd parity system should have an odd number of 1 bits

Table 11.10 describes the two possible scenarios that may occur when the packet arrives at the destination. If the data transmission is successful, the total number of 1 bits detected would be odd, which is the same as that at the source. This means that the data is error-free. However, if there is a simple error in the data transmission, the total number of 1 bits detected would be even, which differs from that at the source. This means that the data is corrupted.

▼ **Table 11.10** Successful and failed data transmission in an odd parity system

Original data bits	Parity system – odd	Data transmission	
		Successful	Unsuccessful
1010001	01010001	01010001 Total number of 1 bits is odd	11010001 Total number of 1 bits is even

Parity checking has certain limitations. For instance, it is able to detect that an error has occurred but cannot determine where the error occurred.

It is also unable to detect all the possible types of errors. It can detect simple errors where only a single bit is changed, but not serious errors where multiple bits are changed. In particular, if two different errors occur in a single packet, they can effectively cancel each other out and result in the correct parity even though the data is actually corrupted.

In general, parity checking can only detect errors when an odd number of bits have been corrupted but not when an even number of bits have been corrupted.

11.7.2 Checksum

A **checksum** is a calculated value that is used to determine the integrity of transmitted data. A checksum serves as a unique identifier for the data and is sent together with the data.

Before transmission, the checksum for the data is first calculated. Assuming that the checksum is one byte in length, it can be calculated in two ways.

If the sum of all the bytes in the data is less than or equal to 255, the checksum is this value. If the sum is greater than 255, the checksum is the modulus of 256.

For example, if the sum is 1,350, the checksum is 1,350 modulo 256, which gives 70.

The data and the checksum are then sent together. At the destination, the checksum is recalculated and compared to the sent checksum value. If the checksum value of the received data matches the sent checksum value, the data was transmitted correctly. If they differ, an error has occurred.

Key Term

Checksum

Calculated value that is used to determine the integrity of transmitted data



Quick Check 11.7

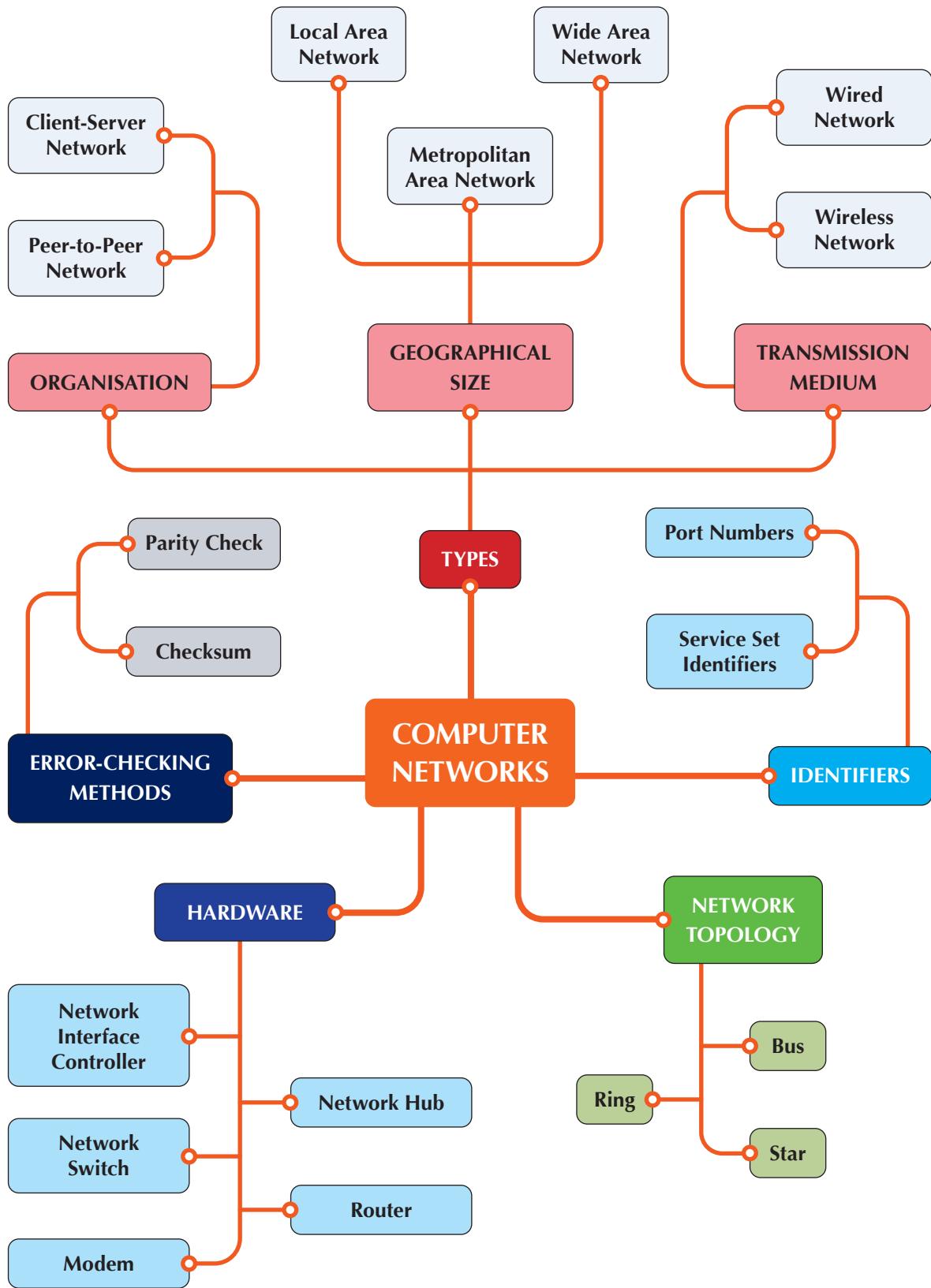
1. Determine the parity bit for the data string in Figure 11.14 if an odd parity system is used.

	1	1	0	0	0	0	1
--	---	---	---	---	---	---	---

▲ **Figure 11.14** A 7-bit data string in an odd parity system

2. Determine the parity bit of the following data strings if an even parity system is used:
 - a) 1001000
 - b) 1100110
 - c) 1001111
3. Determine whether each of the following data packets has been corrupted. Use the parity check method, given that the last bit is the parity bit and an odd parity system is used.
 - a) 11100111
 - b) 11011010

Chapter Summary



Review Questions

- State three advantages of setting up a home or office network.
- Match statements A–L below to the correct network architecture in Table 11.11.

- A. Access to shared resources on the network such as printers, data files and web access is managed by servers
- B. Resources and files located on one computer are accessible by all other computers within the network
- C. All computers are independent of each other and are of equal status when communicating with each other
- D. The network consists of computers operated by users and centralised computers which control the resources of all computers
- E. Regular backups can be easily implemented with the use of a centralised backup storage
- F. Backups are difficult to organise since there is no centralised backup storage
- G. Data is stored independently in each computer
- H. Data is stored in a centralised storage device monitored by the server
- I. More commonly found in businesses and organisations
- J. More commonly found in homes
- K. There is often no mechanism to manage access to the network, hence it is less secure
- L. Access to the network is managed using a database of usernames, passwords and user profile and permission settings, hence the network tends to be more secure

▼ Table 11.11 Network architecture

Design factors	Peer-to-peer	Client-server
a) Organisation of components		
b) Resource management		
c) Storage		
d) Mode of backup		
e) Security		
f) Type of environment		

3. A doctor wishes to set up a LAN in his new clinic. He has three staff located at different counters within the clinic. During operating hours, he and his staff need to retrieve records from a patient history database using their individual desktop computers. They will also need to access a shared network printer to print receipts and medical certificates.

For security reasons, the patient history database and printer must be isolated from the Internet and connected using a wired network. However, he would still like to provide his patients with free wireless Internet access while they wait for their consultation.

- a) The doctor purchases a network switch to help set up the LAN. Name the network topology that is formed when multiple devices are connected to the network switch.
- b) The patient history database needs to be accessed and modified by the doctor and his staff multiple times throughout the day. Recommend, with explanation, whether the patient history database should be stored on a single server or distributed among the doctors and his staff using a peer-to-peer network.
- c) Draw a labelled diagram of the clinic's network(s). The diagram should include at least four laptops, one printer, one fibre optic modem (for Internet access), one network switch and one combined wireless router/access point. You may include additional devices as needed.

ANSWERS

Chapter 1 How Do Computers Work?

Quick Check 1.1

1. True. However, “kilobyte” is still often confused with “kibibyte”, which is 1,024 bytes.
2. a) $2,017 \text{ kB} = 2,017 \times (10^3 \text{ bytes}) = 2,017,000 \text{ bytes}$
 b) $19 \text{ GB} = 19 \times (10^9 \text{ bytes}) = 19,000,000,000 \text{ bytes}$
 c) $65 \text{ MB} = 65 \times (10^6 \text{ bytes}) = 65,000,000 \text{ bytes}$

Quick Check 1.3

1. The main differences between RAM and ROM are summarised below:

RAM	ROM
Read and write: stored data can be easily changed	Read only: stored data cannot be easily changed
Volatile: loses data once power supply to the computer is interrupted	Non-volatile: retains data regardless of whether the power supply is switched on or off
Purpose: to store data and instructions temporarily so that they can be quickly accessed by the processor when needed	Purpose: to store data and instructions that would be needed for a computer to start up or before data can be loaded into RAM

Quick Check 1.4

1. True. Memory addresses are just numbers so they can also be treated as data.

Quick Check 1.6

1. a) A hard disk. It is the most common device with storage capacity available in terabytes.
 b) A memory card. It is the only device small and flat enough to fit in a wallet. A memory card would also be less vulnerable to mechanical shocks that could occur in a wallet.

Review Questions

- 1.**
 - a) Control unit: follows instructions and decides when data should be stored, received or transmitted by different parts of the computer
 - b) Arithmetic logic unit (ALU): processes data by performing basic mathematical and logical operations
 - c) Main memory: stores data and instructions temporarily so they can be quickly accessed by the processor when needed
 - d) Secondary storage: way of storing large amounts of data that will not be lost when power supply is interrupted

- 2.**
 - a) Volatile
 - b) Volatile
 - c) Non-volatile
 - d) Non-volatile

- 3.** Processor register, DVD, hard disk

- 4.** A possible tabulation is as follows:

Computer name	Processor	Memory	Storage media options
VR7 Gaming PC	High-End CPU (8-Core, 16 MB Cache, up to 4.8 GHz)	16 GB Dual Channel DDR4 XMP at 2933 MHz	<ul style="list-style-type: none"> • 1 TB M.2 PCIe NVMe SSD
CMD Multimedia Laptop	Mid-Range CPU (6 MB Cache, 3.6 GHz)	8 GB DDR4 2666 MHz RAM	<ul style="list-style-type: none"> • 1 TB 5400 RPM 2.5-inch SATA Hard Drive • Optical Disk Drive
Mini EMDX PC	3.6 GHz Quad-Core Low-End CPU 6 MB shared L3 Cache	8 GB of 2666 MHz DDR4 SO-DIMM memory	<ul style="list-style-type: none"> • 256 GB PCIe-based SSD

5. a) $17 \times 1,000 = 17,000$ bytes
b) $18 \times 1,024 = 18,432$ bytes
6. a) $8 \times 1,000 \times 1,000 \times 1,000 \times 1,000 \times 8 = 64,000,000,000,000$ bits
b) $0.125 \times 1,000 \times 1,000 \times 8 = 1,000,000$ bits
c) $0.125 \times 1,024 \times 1,024 \times 8 = 1,048,576$ bits
7. a) $64 \times 1,024 = 65,536$ bytes, which need 65,536 addresses
b) Each address represents one byte, so if there are only 256 possible addresses, there are only 256 bytes of memory. Hence, total size of memory = $256 \times 8 = 2,048$ bits
8. a) R is located in the computer's processor.
b) In Step 1, the address bus is used to transport A from the processor to the memory, and the data bus is used to transport data from the memory to the processor.
In Step 2, the address bus is used to transport B from the processor to the memory, and the data bus is used to transport data from the processor to the memory.
c) R and S are both located in the computer's processor. The data and address buses are only needed to manipulate data in the computer's main memory.
9. a) False. The data bus connects the processor to memory only and not to secondary storage such as hard disks.
b) Two improvements (accept any possible answer):
1. Faster reading and writing of files
2. Less noise when laptop is on
3. Lighter laptop
4. Longer battery life due to reduced power use

Chapter 2 How Can Algorithms Be Used to Solve Problems?

Quick Check 2.1

1. A. By itself, the map does not have any step-by-step instructions and therefore is not an algorithm.
2. A modified addition algorithm that works for three positive whole numbers is provided below with the required changes indicated in italics:

- Step 1:** Write the first number, then write the second number below it *and the third number below the second number* such that the digits in the ones place are aligned vertically. Draw a horizontal line below the *third* number.
- Step 2:** Let the current column be the right-most column of digits in the ones place.
- Step 3:** Add all the digits (including any “carry-over” digits) in the current column. If a digit is missing, treat it as 0. There is no “carry-over” digit for the right-most column.
- Step 4:** If the result is greater than 9, write the *first digit of the result* at the top of the column immediately to the left of the current column as a “carry-over” digit.
- Step 5:** Write the last digit of the result at the bottom of the current column, below the line.
- Step 6:** If there are still digits (including “carry-over” digits) to the left of the current column, redefine that column on the left as the new current column and go back to Step 3. Otherwise, the final answer is formed by the row of digits below the line.

The only changes needed are in Step 1 and Step 4. The most significant difference is that when adding three numbers, “carry-over” digits can be either 1 or 2.

3. A possible algorithm to find the top score of a class test is provided below:

- Step 1:** Let the current score be the first score on the list of scores. Write down the current score.
- Step 2:** If the current score is the last score in the list, proceed to Step 5, otherwise proceed to Step 3.
- Step 3:** Redefine the current score to be next score in the list.
- Step 4:** If the current score is greater than the score that was previously written down, erase the previous score and write down the current score. Proceed to Step 2.
- Step 5:** The final answer is the written score.

Quick Check 2.2

1. The modified input and output specifications for addition problem are as follows:

Input	Output
<ul style="list-style-type: none"> • x: a whole number • y: a whole number 	<ul style="list-style-type: none"> • The sum of x and y

The addition algorithm provided previously is *not* a solution to this modified problem. See the counter-example below of using the algorithm to add -2017 to 1965 :

A handwritten addition problem is shown. It consists of two rows of digits separated by a minus sign. The top row has a single digit '1' above it. The bottom row has four digits: '-2017'. Below these is a row of four digits: '1965'. A horizontal line with vertical dashed grid lines passes through all these digits. Below the bottom row, the word 'Incorrect answer' is written in red. To the right of the grid, the result '3982' is written.

Incorrect answer

1
 -2017
 1965
 —
 3982

The counter-example shows that the addition algorithm produces an incorrect answer of 3982 while the correct answer should be -52 .

2. The input and output requirements for the problem are as follows:

Input	Output
<ul style="list-style-type: none"> • <i>Weights</i>: list of weights of different apples 	<ul style="list-style-type: none"> • Weight of the heaviest apple

The following input and output requirements are equivalent:

Input	Output
<ul style="list-style-type: none"> • <i>Number</i>: number of apples; must be a positive whole number • <i>Weight</i>: weight of apple (provided <i>number</i> times) 	<ul style="list-style-type: none"> • Weight of the heaviest apple

3. The title of the class test should be excluded as it is irrelevant and does not affect what we require for the output.

The improved input and output requirements are as follows:

Input	Output
<ul style="list-style-type: none"> Scores: list of class test scores with each student's score on a separate row 	<ul style="list-style-type: none"> The average score of the class test

Quick Check 2.3

1. The correct passage is as follows:

If a complex problem is not broken down into parts, the time taken to solve the problem may be too long. The technique of breaking down the problem into parts, called decomposition, allows the parts to be evaluated separately. By making the parts more manageable, the solutions to these parts can then be combined, and the original problem can be solved.

2. A.

3. a) The input and output requirements of the problem are as follows:

Input	Output
<ul style="list-style-type: none"> x: a positive whole number y: a positive whole number 	<ul style="list-style-type: none"> The absolute difference between x and y

- b) A possible algorithm is as follows:

Step 1: Compare 2017 and 1965.

Step 2: 2017 is bigger, so subtract 1965 from 2017. The final answer is the absolute difference.

- c) A possible algorithm is as follows:

Step 1: Compare 2017 and 2020.

Step 2: 2020 is bigger, so subtract 2017 from 2020. The final answer is the absolute difference.

- d) Let the first number be x and the second number be y . A possible general solution is as follows:

Step 1: Compare x and y .

Step 2: If x is bigger, subtract y from x , otherwise subtract x from y . The final answer is the absolute difference.

Review Questions

1. An appropriate order is as follows:

Step 1: Add tap water to the kettle.

Step 2: Boil the contents of the kettle.

Step 3: Place a teabag in a cup and pour some water from the kettle into the cup.

Step 4: Wait for a few minutes to let the tea steep.

Step 5: Drink the contents of the cup.

2. The input and output requirements for sub-problem #1 are as follows:

Sub-problem	Input	Output
#1	<ul style="list-style-type: none"> <i>Map:</i> a map of roads in Siti's neighbourhood <i>Home:</i> the position of Siti's home on <i>map</i> <i>School:</i> the position of Pines Primary School on <i>map</i> 	<ul style="list-style-type: none"> List of all possible routes from <i>home</i> to <i>school</i> using the roads shown on <i>map</i>

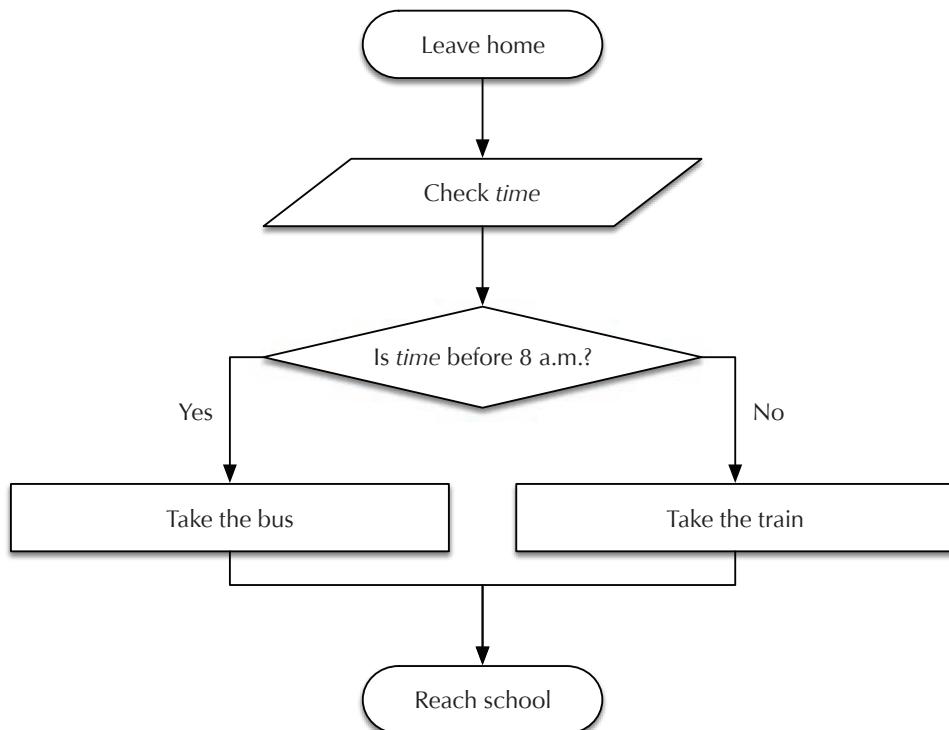
Chapter 3 How Do I Use Flowcharts and Pseudo-Code?

Quick Check 3.2

1. a) The input and output requirements of the problem are as follows:

Input	Output
<ul style="list-style-type: none"> • <i>time</i>: time just after student leaves home 	<ul style="list-style-type: none"> • Student arrives in school

- b) A flowchart that represents the algorithm is as follows:



Quick Check 3.3

1. a) The trace table for test data of 3 is as follows:

<i>result</i>	<i>x</i>	OUTPUT
1		
	3	
3		
	2	
6		
	1	
6		
	0	
		6

The output of 6 is the result of $3 \times 2 \times 1 = 6$.

- b) The trace table for test data of 5 is as follows:

<i>result</i>	<i>x</i>	OUTPUT
1		
	5	
5		
	4	
20		
	3	
60		
	2	
120		
	1	
120		
	0	
		120

The output of 120 is the result of $5 \times 4 \times 3 \times 2 \times 1 = 120$.

- c) From the pattern of output in a) and b), we can deduce that the algorithm uses the test data to calculate the following:

$$\text{Result} = n \times (n-1) \times (n-2) \dots 2 \times 1$$

where n is the input.

This is the formula for the factorial operation you may have learnt in mathematics to count the number of ways that n distinct objects can be arranged. Thus, the purpose of this algorithm is to calculate the factorial of the input, or $n!$ for short.

Quick Check 3.4

- Using the IF...THEN...ELSEIF...ELSE...ENDIF selection construct, the required algorithm can be represented in pseudo-code as follows:

```

INPUT number
IF number == 1 THEN
    OUTPUT "one"
ELSEIF number == 2 THEN
    OUTPUT "two"
ELSEIF number == 3 THEN
    OUTPUT "three"
ELSEIF number == 4 THEN
    OUTPUT "four"
ELSEIF number == 5 THEN
    OUTPUT "five"
ELSEIF number == 6 THEN
    OUTPUT "six"
ELSEIF number == 7 THEN
    OUTPUT "seven"
ELSE
    OUTPUT "Invalid input"
ENDIF

```

The input and output requirements of the problem are as follows:

Input	Output
<ul style="list-style-type: none"> • $number$: a whole number between 1 and 7 inclusive 	<ul style="list-style-type: none"> • The spelling of $number$ in English or “Invalid input” if $number$ is invalid

2. Using the FOR...NEXT iteration construct, the required algorithm can be represented in pseudo-code as follows:

```
sum = 0
FOR i = 1 to 10
    INPUT number
    sum = sum + number
NEXT
average = sum / 10
OUTPUT average
```

The input and output requirements of the problem are as follows:

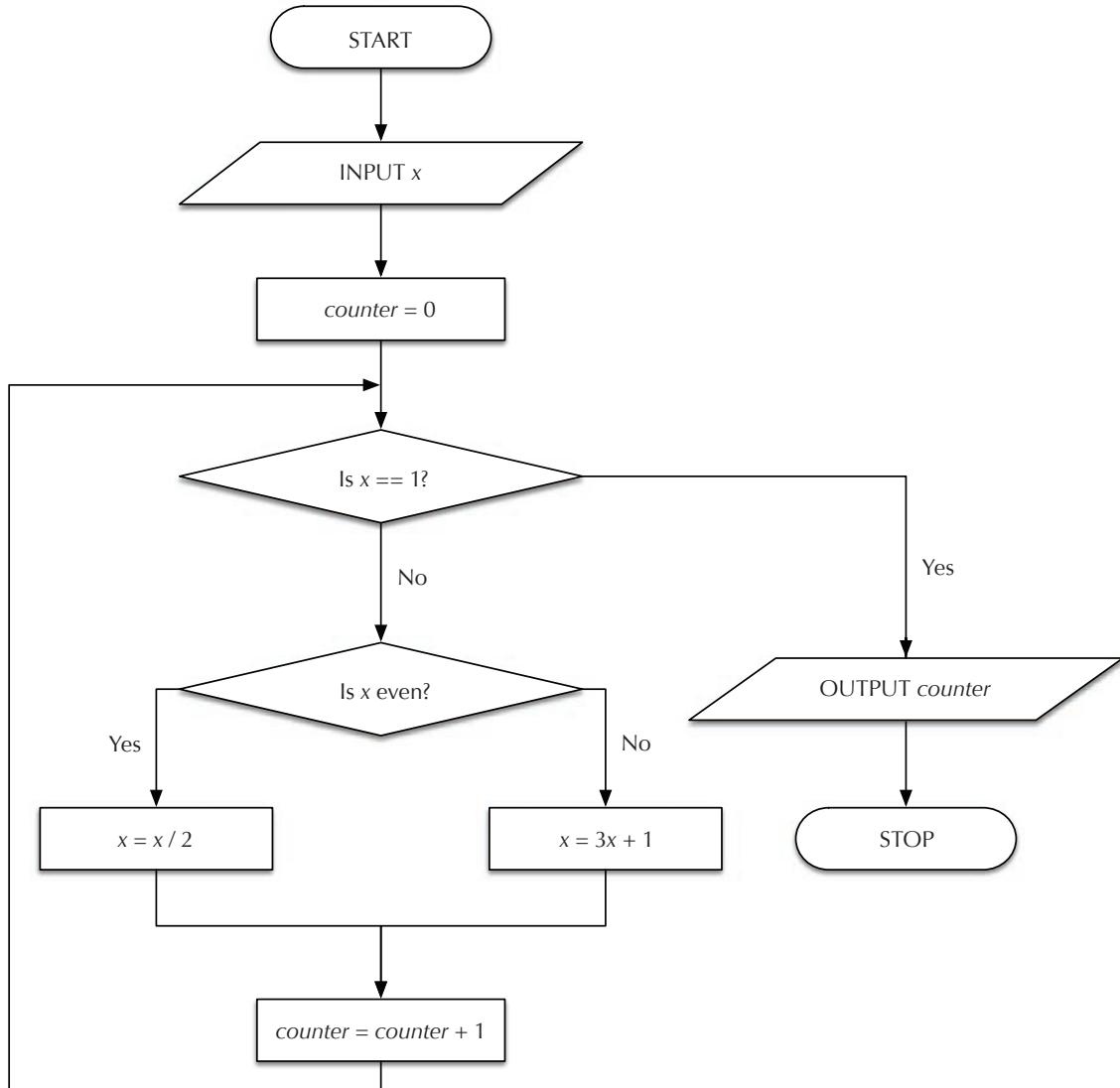
Input	Output
<ul style="list-style-type: none">• <i>number</i>: a number (provided 10 times)	<ul style="list-style-type: none">• The average of all 10 <i>number</i> values provided

A trace table for the provided test data is as follows:

<i>sum</i>	<i>number</i>	<i>average</i>	OUTPUT
0			
	20		
20			
	17		
37			
	19		
56			
	65		
121			
	20		
141			
	18		
159			
	19		
178			
	66		
244			
	20		
264			
	19		
283			
		28.3	
			28.3

Review Questions

1. a) A flowchart of this algorithm is as follows:



b) The trace table for test data of 5:

x	counter	OUTPUT
5		
	0	
16		
	1	
8		
	2	
4		
	3	
2		
	4	
1		
	5	
		5

- c) The trace table for test data of 6:

x	counter	OUTPUT
6		
	0	
3		
	1	
10		
	2	
5		
	3	
16		
	4	
8		
	5	
4		
	6	
2		
	7	
1		
	8	
		8

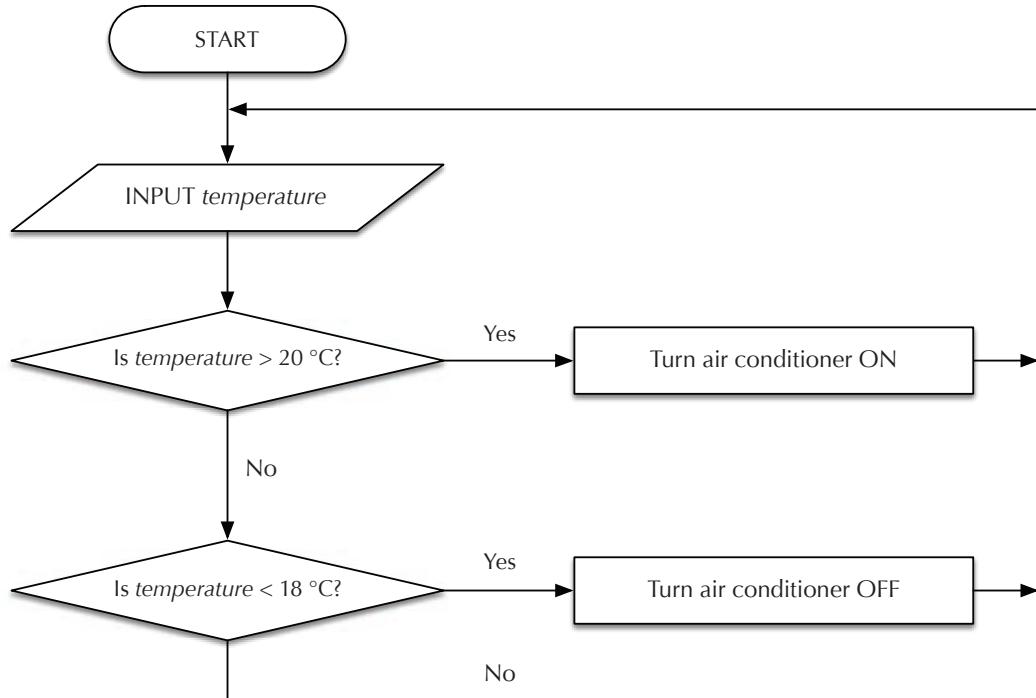
- d) A pseudo-code of this algorithm is as follows:

```

INPUT x
counter = 0
WHILE x != 1
    IF x is even THEN
        x = x / 2
    ELSE
        x = 3x + 1
    ENDIF
    counter = counter + 1
ENDWHILE
OUTPUT counter

```

2. A possible flowchart of this algorithm is as follows:



3. A possible solution is as follows:

```
INPUT Choice  
INPUT Amount  
  
IF Choice == "Ringgit" THEN  
    OUTPUT Amount * 3  
ELSE  
    OUTPUT Amount * 4.66  
ENDIF
```

4. The three errors are located at:

- 1) Line 6: The WHILE condition should be "WHILE Count < 5" instead of "WHILE Count <= 5" so that the algorithm reads in the correct number of inputs.
- 2) Line 12: Sport_Count should be incremented by 1, not 2.
- 3) Line 24: The last variable should be UniformedGroup_Count, not Sport_Count.

The corrected pseudo-code is as follows:

```

1  Count = 0
2  ClubSociety_Count = 0
3  Sport_Count = 0
4  UniformedGroup_Count = 0
5
6  WHILE Count < 5
7      INPUT Type
8      IF Type == "Club/Society" THEN
9          ClubSociety_Count = ClubSociety_Count + 1
10         Count = Count + 1
11     ELSEIF Type == "Sport" THEN
12         Sport_Count = Sport_Count + 1
13         Count = Count + 1
14     ELSEIF Type == "Uniformed Group" THEN
15         UniformedGroup_Count = UniformedGroup_Count + 1
16         Count = Count + 1
17     ELSE
18         OUTPUT "Invalid Option"
19     ENDIF
20 ENDWHILE
21
22 OUTPUT "Clubs and Societies: " + ClubSociety_Count
23 OUTPUT "Sports: " + Sport_Count
24 OUTPUT "Uniformed Groups: " + UniformedGroup_Count

```

5. The required trace table is as follows:

Count	Sum	Num	Average	OUTPUT
0				
	0			
		7		
		7		
1				
		15		
	22			
2				
		11		
		33		
3				
		18		
		51		
4				
		5		
		56		
5				
		17		
		73		
6				
		1		
		74		
7			10.57	
				The sum is 74
				The average is 10.57

6. a) A pseudo-code of the flowchart is as follows:

```

i = 0
INPUT numbers[i]
WHILE numbers[i] != 0
    i = i + 1
    INPUT numbers[i]
ENDWHILE
WHILE i >= 0
    OUTPUT numbers[i]
    i = i - 1
ENDWHILE

```

- b) The required trace table is:

<i>i</i>	<i>numbers</i>	OUTPUT
0		
	20	
1		
	20, 17	
2		
	20, 17, -19	
3		
	20, 17, -19, 65	
4		
	20, 17, -19, 65, 0	
		0
3		
		65
2		
		-19
1		
		17
0		
		20
-1		

- c) The flowchart's purpose is to output the given sequence of integers (including the last integer 0) in reverse order.

Chapter 4 How Do I Write Programs?

Quick Check 4.4

1.	#	Statement	Name of variable that is assigned a value	Final value being assigned
	1	ratio = 0.75	ratio	0.75
	2	greeting = "Hi!"	greeting	"Hi!"
	3	to_hit = 17 + 3	to_hit	20

2.	#	Source code	Result	Explanation
	1	name= "Computing"	(nothing displayed)	The assignment statement just assigns the value on the right to the variable on the left. It usually does not provide a result or display anything as output.
	2	name name	Error: SyntaxError	name name is not a valid Python command.
	3	Name = "Science"	(nothing displayed)	The assignment statement just assigns the value on the right to the variable on the left. It usually does not provide a result or display anything as output.
	4	print(name)	Computing	The print() function outputs the value assigned to name.

#	Source code	Result	Explanation
5	print (Name)	Science	Python treats name with a small letter n as different from Name with a capital N, so the two variables can naturally store different values.
6	Real_Name = name	(nothing displayed)	The assignment statement just assigns the value on the right to the variable on the left. It usually does not provide a result or display anything as output.
7	Real_Name	'Computing'	In interactive mode, the result for each line is automatically displayed (with additional quotes for strs). Normally, the print() function will need to be used.
8	print (Real_Name)	Computing	The print() function is needed to output the value of Real_Name outside of the interactive mode. Within the interactive mode, however, lines 7 and 8 appear to give similar results except that line 7 has extra quotes around the output.
9	Fake_Name =	Error: SyntaxError	This assignment statement is invalid as the value being assigned is missing.

3.

```
original_value = 99
copy_1 = original_value
copy_2 = original_value
```

4. The fixed values of "Hello, ", "Ms. " and "Mr. " are repeated multiple times in the source code. To make the source code more readable and easier to maintain, we should use constants with descriptive names, that is, GREETING, FEMALE_TITLE and MALE_TITLE respectively instead.

A possible modified program is as follows:

#	Program greetings_with_constants.py
1	# Constants
2	GREETING = "Hello, "
3	FEMALE_TITLE = "Ms. "
4	MALE_TITLE = "Mr. "
5	
6	# Prints out a series of greetings, with title based on gender.
7	print(GREETING + FEMALE_TITLE + "Aishah")
8	print(GREETING + MALE_TITLE + "Paul")
9	print(GREETING + MALE_TITLE + "Xiaoming")
10	print(GREETING + FEMALE_TITLE + "Jessica")

Quick Check 4.5

1.

	Literal	Data type
a)	-2017	int
b)	'''	str
c)	'Computing'	str
d)	3.14159	float
e)	"\\"	str
f)	"\"[]\""	str
g)	True	bool
h)	6.02e23	float
i)	'\\'	str
j)	"2017"	str
k)	"""	str
l)	["]"]	list
m)	'False'	str

2.

#	Source code	What will be displayed	Explanation
1	print(1e100)	1e+100	
2	print(103E1)	1030.0	
3	print(1EE1)	Error: SyntaxError	1EE1 is not a valid literal.
4	print(3.141)	3.141	
5	print(3.1.3)	Error: SyntaxError	3.1.3 is not a valid literal.
6	print("Hello,\nWorld")	Hello,World	The line break in the source code is ignored because it was preceded by a backslash (\).
7	print("Hello,\nWorld")	Hello, World	The \n escape code represents a new line.
8	print("Hello,\tWorld")	Hello, World	The \t escape code represents a tab.
9	print('Hello,\n\tWorld')	Hello, World	
10	print('Hello,\t\nWorld')	Hello, World	
11	print("\\\\\"")	\"	Python reads the str literal as two escape codes: \\ (a backslash) and \" (a double quote).
12	print("\\\\\")	Error: SyntaxError	The second quote is part of an escape code and does not close the str literal.
13	print(True)	True	

#	Source code	What will be displayed	Explanation
14	<code>print(true)</code>	Error: NameError	true, unlike True, is not a key word, so Python assumes it is a variable name. The error occurs because a variable named true does not exist.
15	<code>print(False)</code>	False	
16	<code>print(false)</code>	Error: NameError	false, unlike False, is not a key word, so Python assumes it is a variable name. The error occurs because a variable named false does not exist.
17	<code>print(None)</code>	None	
18	<code>print(none)</code>	Error: NameError	none, unlike None, is not a key word, so Python assumes it is a variable name. The error occurs because a variable named none does not exist.
19	<code>print([])</code>	[]	
20	<code>print([2, 3, 4])</code>	[2, 3, 4]	
21	<code>print([[[]]])</code>	[[[]]]	

3.

#	Source code	What will be displayed	Explanation
1	<code>print(int("-3"))</code>	-3	
2	<code>print(float(' -3'))</code>	-3.0	
3	<code>print(float("minus three"))</code>	Error: ValueError	"minus three" is not a valid float literal and cannot be converted by <code>float()</code> .
4	<code>print(str(-3))</code>	-3	
5	<code>print(str(-3.0))</code>	-3.0	
6	<code>print(str(-3e3))</code>	-3000.0	

Quick Check 4.6

1.

#	Source code	What will be displayed	Explanation
1	<code>print(len(range(3)))</code>	3	
2	<code>print(len(3))</code>	Error: TypeError	<code>len()</code> can only be used on sequences such as lists and strs, not ints.
3	<code>phrase = "Hello, World!"</code>	(nothing displayed)	The assignment statement just assigns the value on the right to the variable on the left. It usually does not provide a result or display anything as output.
4	<code>print(len(phrase))</code>	13	phrase has 13 characters: 5 for the word "Hello", 1 for the comma, 1 for the space, 5 for the word "World" and 1 for the exclamation mark.
5	<code>print(len([phrase]))</code>	1	[phrase] is just a list made up of 1 item (i.e., phrase).
6	<code>print(list(range(7)))</code>	[0, 1, 2, 3, 4, 5, 6]	

#	Source code	What will be displayed	Explanation
7	print(list(range(len("2017")))))	[0, 1, 2, 3]	The result of <code>(len("2017"))</code> is 4, so this is equivalent to printing <code>list(range(4))</code> .
8	x = len(range(-2, 3))	(nothing displayed)	<code>range(-2, 3)</code> has 5 items: -2, -1, 0, 1 and 2.
9	print(x, x)	5 5	

2. A possible solution is as follows:

#	Program count_phrase.py
1	# Input phrase = input("Enter phrase: ") # Process count = len(phrase) # Output print(count)

3.

#	Source code	What will be displayed	Explanation
1	<code>x = 2</code>	(nothing displayed)	
2	<code>x **= 3</code>	(nothing displayed)	
3	<code>x / 10</code>	0.8	This is the result of calculating $2^3 / 10$.
4	<code>print(x)</code>	8	The calculation on line 3 does not change the value assigned to <code>x</code> .
5	<code>print(1 + 1 <= "2")</code>	Error: TypeError	The <code><=</code> operator cannot be used to compare a number with a string.
6	<code>print(3 * "2")</code>	222	
7	<code>quotient = 55 // 7</code>	(nothing displayed)	
8	<code>remainder = 55 % 7</code>	(nothing displayed)	
9	<code>print(quotient * 7 + remainder == 55)</code>	True	
10	<code>print(remainder + quotient * 7)</code>	55	Following the standard order of operations, the multiplication is performed first before the addition.
11	<code>print((remainder + quotient) * 7)</code>	91	As <code>(remainder + quotient)</code> is enclosed in brackets, the addition is performed first before the multiplication.

#	Source code	What will be displayed	Explanation
12	55 = x	Error: SyntaxError	
13	2.0 ** 3 == x	True	The == operator can be used to compare any two numbers even if one is a float and the other is an int.
14	print(-55 // 10)	-6	The result of floor division is always rounded down so the answer is -6, not -5.

4. A possible solution is as follows:

```
def area_of_triangle(base, height):
    area = 0.5 * base * height
    return area
```

5. A possible program is as follows:

```
phrase = input("Enter phrase: ")
print([len(phrase) % 3 == 0])
```

6. A possible program is as follows:

```
name = input("Enter name: ")
print([name[0] == name[-1] and not " " in name])
```

7.

#	Source code	What will be displayed	Explanation
1	<code>data = [7, 9, 3, 5]</code>	(nothing displayed)	
2	<code>print(data[3:])</code>	[5]	The slice operator is used with a start index of 3 and a stop index of 4 (omitted). This results in the list [5], which is a list with just the item in index 3.
3	<code>print(data[:1])</code>	[7]	The slice operator is used with a start index of 0 (omitted) and a stop index of 1. This results in the list [7], which is a list with just the item in index 0.
4	<code>print(data + [2])</code>	[7, 9, 3, 5, 2]	The argument to <code>print()</code> is made by joining <code>data</code> with a list containing just the number 2. This results in the list [7, 9, 3, 5, 2]. Note that the original contents of <code>data</code> are unchanged.
5	<code>print(data + 2)</code>	Error: TypeError	The "+" operator can only be used to join two lists, not a list and an int.

#	Source code	What will be displayed	Explanation
6	<code>print(data[-3:-2])</code>	[9]	The slice operator is used with a start index of 1 (3rd last item) and a stop index of 2 (2nd last item). This results in the list [9], which is a list with just the item in index 1.
7	<code>print(data[len(data)- 1])</code>	5	The index of the last item of a list is always the length of the list minus 1. Hence, the result is just the last item of data.
8	<code>print(data[len(data)])</code>	Error: IndexError	As list indexes start from 0 and not 1, in general, no item in a list has the same index as the length of the list. Attempting to retrieve such an item results in an error.
9	<code>print(['A', 'B', 'C'][1])</code>	B	The index operator is used with the index 1, which refers to the second item of the list.
10	<code>['F', 'G'][0] = 'A'</code>	(nothing displayed)	['F', 'G'] is a list and is therefore mutable.
11	<code>"FG"[0] = 'A'</code>	Error: TypeError	Unlike lists, strs are immutable and their contents cannot be modified.

8.

#	Source code	What will be displayed	Explanation
1	<code>x = 5</code>	(nothing displayed)	
2	<code>print(x > -5 and x < 5)</code>	False	
3	<code>print(x >= -5 and x <= 5)</code>	True	
4	<code>print(x > -5 or x < 5)</code>	True	
5	<code>print(not not True)</code>	True	
6	<code>print(not True and False)</code>	False	Without parentheses, the order of operations may not be clear. In this case, the <code>not</code> operation is performed before the <code>and</code> operation.
7	<code>print(not (True and False))</code>	True	As <code>(True and False)</code> is enclosed in parentheses, the <code>and</code> operation is performed before the <code>not</code> operation.

9.

	Variable/Constant	Local or Global
a)	<code>SYMBOL</code> on line 4	Global
b)	<code>length</code> on line 4	Local
c)	<code>border</code> on line 5	Local
d)	<code>SYMBOL</code> on line 8	Global
e)	<code>left_side</code> on line 10	Local
f)	<code>message</code> on line 10	Local
g)	<code>right_side</code> on line 10	Local
h)	<code>middle_line</code> on line 12	Local
i)	<code>message</code> on line 17	Global

Quick Check 4.7

1. A possible program is as follows:

#	Program list_input.py
1	<code># Input and Process</code>
2	<code>result = []</code>
3	<code>while True:</code>
4	<code> input_str = input("Enter item, blank to end: ")</code>
5	<code> if input_str == "":</code>
6	<code> break</code>
7	<code> result += [input_str]</code>
8	
9	<code># Output</code>
10	<code>print(result)</code>

Using the modular approach, we can break down the task into two parts. The first part repeatedly asks the user for an item and adds it to our output list. We can think of this as the “default behaviour” of the program. The second part detects when a blank item is received and ends the program. We can think of this as an “exceptional case” to the default behaviour of the program.

The first part of the task can be accomplished using a loop that keeps repeating without end. We do this using a `while` condition that is always `True` on line 3. As the loop keeps repeating, we keep getting the next item from the user (line 4) and adding it to our output list, which we call `result` (line 7). Of course, before we start the loop we must initialise `result` to an empty list first (line 2).

The second part of the task requires us to detect an exception to the default case. We do this using an `if` statement immediately after getting the user’s input (line 5). If we detect that the input is blank, we break out of the loop (line 6). The use of `break` helps to explain why our program does not run forever as long as a blank input is eventually supplied by the user.

2. A possible program is as follows:

#	Program password_check.py
1	<code># Input, Process and Output</code>
2	<code>while True:</code>
3	<code> password = input("Enter password: ")</code>
4	<code> password_again = input("Enter password again: ")</code>
5	<code> if password == password_again:</code>
6	<code> break</code>
7	<code> print("Invalid")</code>
8	<code>print("Valid")</code>

Using generalisation, we see that, like the previous task, there is again a default behaviour that is required (i.e., repeatedly asking for two passwords and printing “Invalid”) as well as an exceptional case (i.e., stopping when the two passwords match and printing “Valid”).

Thus we can adapt our program from the solution to the previous task. Instead of prompting for an item, we now prompt for two passwords (lines 3 and 4). Instead of adding an item to a list, we print “Invalid” (line 7). Finally, instead of detecting a blank input, we detect when the two passwords match (line 5) before breaking out of the loop (line 6) and printing “Valid” as the final output (line 8).

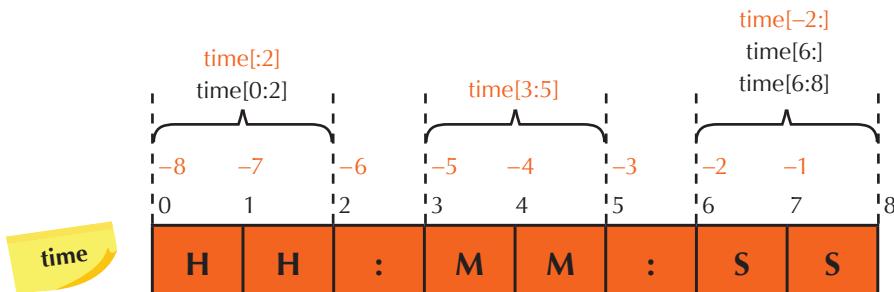
Review Questions

1. a) Alex's method will simplify his program as the `int` values increase in the same order as the corresponding dates. He can then easily compare whether one date is earlier or later than another date by using the less than (`<`) and greater than (`>`) operators.

- b) A possible program is as follows:

#	Program overdue.py
1	<code># This program outputs the titles of all overdue books.</code>
2	
3	<code># Input</code>
4	<code>today = int(input("Enter today's date in YYYYMMDD: "))</code>
5	<code>titles = [</code>
6	<code> # The following data is copied from the given text file.</code>
7	<code> "How to Solve a Mystery", "Vacant Memories",</code>
8	<code> "The Cybersnake Chronicles", "Music History",</code>
9	<code> "Like Tears in Rain", "Out of the Abyss"</code>
10	<code>]</code>
11	<code>duedates = [</code>
12	<code> # The following data is copied from the given text file.</code>
13	<code> 20170727, 20170704, 20170711, 20170630, 20170703, 20170707</code>
14	<code>]</code>
15	
16	<code># Process and Output</code>
17	<code>def is_overdue(duedate, today):</code>
18	<code> return duedate < today</code>
19	
20	<code>for index in range(len(duedates)):</code>
21	<code> if is_overdue(duedates[index], today):</code>
22	<code> print(titles[index])</code>

2. The diagram below shows the different ways we can extract the hours, minutes and seconds components from the time string using the slice operator:



Using this analysis, a possible solution is as follows:

#	Program time_string.py
1	# This program extracts the hour, minute and second values (as ints)
2	# from a time string that is provided in the format "HH:MM:SS".
3	# It is assumed that input data will always be valid.
4	
5	# Input
6	time = input("Enter time string: ")
7	
8	# Process
9	hours = int(time[:2])
10	minutes = int(time[3:5])
11	seconds = int(time[-2:])
12	
13	# Output
14	print(hours)
15	print(minutes)
16	print(seconds)

3. A possible solution is as follows:

#	Program time_interval.py
1	<code># This program calculates the number of seconds between a start time</code>
2	<code># and an end time, both provided in the format "HH:MM:SS". It is</code>
3	<code># assumed that input data will always be valid.</code>
4	
5	<code># Input</code>
6	<code>start_time = input("Enter start time string: ")</code>
7	<code>end_time = input("Enter end time string: ")</code>
8	
9	<code># Process</code>
10	<code>hours = int(start_time[:2])</code>
11	<code>minutes = int(start_time[3:5])</code>
12	<code>seconds = int(start_time[-2:])</code>
13	<code>start_total = hours * 60 * 60 + minutes * 60 + seconds</code>
14	
15	<code>hours = int(end_time[:2])</code>
16	<code>minutes = int(end_time[3:5])</code>
17	<code>seconds = int(end_time[-2:])</code>
18	<code>end_total = hours * 60 * 60 + minutes * 60 + seconds</code>
19	
20	<code>interval = end_total - start_total</code>
21	
22	<code># Output</code>
23	<code>print(interval)</code>

Chapter 5 How Can Programs Be Used to Solve Problems?

Quick Check 5.2

1.

x	-20.17	-19.65	-0.1	0.0	0.1	19.65	20.17
math.ceil(x)	-20	-19	0	0	1	20	21
math.floor(x)	-21	-20	-1	0	0	19	20
round(x)	-20	-20	0	0	0	20	20
math.trunc(x)	-20	-19	0	0	0	19	20

2. A possible answer is as follows:

```
import random

counts = [0] * 13
for i in range(1000):
    die1 = random.randint(1, 6)
    die2 = random.randint(1, 6)
    counts[die1 + die2] += 1
for i in range(2, 13):
    print('{}: {}'.format(i, counts[i]))
```

We want to use counts[2], counts[3] and so on up to counts[12] to track the results. However, list indices always start from 0, so counts[0] and counts[1] must also exist such that counts has 13 entries in total from index 0 to index 12.

Quick Check 5.3

1. A possible answer is as follows:

```
name = input("Enter name: ")
print(name[0].isupper())
```

2. A possible answer is as follows:

```
title = input("Enter title: ")

valid = True
for char in title:
    if not (char.isupper() or char.isspace()):
        valid = False
        break
print(valid)
```

Review Questions

1. a) Siti's new solution stops searching through the list once the availability of the book is determined, but the solution in `availability_refined.py` will always search through the entire list, which makes it slower.
- b) A possible solution is as follows:

#	Program <code>availability_revision.py</code>
	<pre> 1 # Input 2 titles = [3 # The following data is copied from the given text file. 4 "How to Solve a Mystery", "Vacant Memories", 5 "The Cybersnake Chronicles", "Music History", 6 "Like Tears in Rain", "Out of the Abyss" 7] 8 availability = [9 # The following data is copied from the given text file. 10 "Not Available", "Available", "Available", "Not Available", 11 "Not Available", "Available" 12] 13 search = input("Enter title of book to check: ") 14 15 # Process 16 result = "Book Not Found" 17 index = 0 18 while index < len(titles): 19 if titles[index] == search: 20 result = availability[index] 21 break 22 index = index + 1 23 24 # Output 25 print(result) </pre>

2. A possible answer is as follows:

#	Program Q2.py
	<pre> 1 # Input 2 values = [] 3 while True: 4 item = input("Enter item, blank to end: ") 5 if item == "": 6 break 7 values += [int(item)] 8 9 # Process 10 if len(values) >= 2: 11 # For the basic sub-problem, only consider the first 12 # two items in values. 13 if values[0] < values[1]: 14 minimum_value = values[0] 15 second_minimum_value = values[1] 16 else: 17 minimum_value = values[1] 18 second_minimum_value = values[0] 19 # Start considering beyond the first two items. 20 for value in values[2:]: 21 if value < minimum_value: 22 # If there will be a new minimum value, the second 23 # minimum value will be the old minimum value. 24 second_minimum_value = minimum_value 25 minimum_value = value 26 else: 27 minimum_value = None 28 second_minimum_value = None 29 30 # Output 31 if minimum_value == None: 32 print("None") 33 else: 34 print(minimum_value, second_minimum_value) </pre>

3. A possible answer is as follows:

#	Program Q3.py
1	<code># Input</code>
2	<code>length = int(input("Enter length: "))</code>
3	<code>values = []</code>
4	<code>for i in range(length):</code>
5	<code> value = float(input("Enter item: "))</code>
6	<code> values += [value]</code>
7	
8	<code># Process</code>
9	<code>average = None</code>
10	<code>sum_values = 0</code>
11	<code>num_positive = 0</code>
12	<code>for value in values:</code>
13	<code> if value > 0:</code>
14	<code> sum_values += value</code>
15	<code> num_positive += 1</code>
16	<code>if num_positive > 0:</code>
17	<code> average = round(sum_values / num_positive, 2)</code>
18	
19	<code># Output</code>
20	<code>print(average)</code>

Chapter 6 How Do I Ensure That a Program Works as Intended?

Quick Check 6.1

1. a) Format check.

A combination of the `len()` function, the `in` operator and the `str.isdigit()` method is used to check that the input `str` matches the pattern of an O-Level grade (i.e., a capital letter from A to F, followed by a digit).

b) Presence check.

The `first` and `last` `strs` are required inputs and must be provided (i.e., they must not be equal to empty `strs`).

c) Range check.

The `<` and `>` operators are used to ensure that the input `float p` is strictly between 0.0 and 1.0.

d) Length check.

The `len()` function is used to check that the length of input `list integers` is equal to the input `int n`.

2. A possible program is as follows:

#	Program postal_code_validation.py
1	<code># Input and input validation</code>
2	<code>while True:</code>
3	<code> postal_code = input("Enter postal code: ")</code>
4	<code> if postal_code.isdigit() and len(postal_code) == 6:</code>
5	<code> break</code>
6	<code> print("Data validation failed!")</code>
7	<code> print("Postal code should be exactly 6 digits")</code>

Line 3, which asks the user to enter a postal code, is included inside an infinite `while` loop which will repeat until the condition in line 4 is `True`, i.e., the user enters a `str` that is six characters long and consists only of numbers. When that happens, the loop will be exited and we will continue with the rest of the program.

If the `str` entered by the user does not fulfil the conditions in line 4, the program will go on to print the error messages in lines 6 and 7. The loop then restarts and asks the user once again to enter a postal code.

Quick Check 6.2

1. a) Test case that covers normal conditions (accept any possible answer):

→ Input	Expected Output →
• n: 7	***** ***** ***** ***** ***** ***** *****

- b) Test case for the boundary condition when n is at its lower limit:

→ Input	Expected Output →
• n: 1	*

This is a boundary condition as the input of 1 is the smallest possible value for n that is valid based on the problem definition.

Test case for the boundary condition when n is at its upper limit:

This is a boundary condition as the input of 20 is the largest possible value for n that is valid based on the problem definition.

- c) As n is defined as an integer between 1 and 20 inclusive, an input of 0 is not within the valid range of values of n and should be considered as an error condition:

➔ Input	Expected Output ➔
• n: 0	Data validation failed! n should be an integer between 1 and 20 inclusive

Similarly, any non-integer input should also be considered as an error condition:

➔ Input for Test Case 7	Expected Output ➔
<ul style="list-style-type: none">n: "a"	<p>Data validation failed!</p> <p>n should be an integer between 1 and 20 inclusive</p>

2. a) The three errors are located at:

- 1) Line 4: The `if` condition needs to end with a colon (:).
- 2) Line 12: A `str` that is opened with a double quote ("") must be closed with double quote, not a single quote ('').
- 3) Line 19: There is an extra right bracket at the end of the line.

b) The corrected program is as follows:

#	Program diamond_with_syntax_errors_fixed.py
	<pre> 1 # Input and input validation 2 while True: 3 input_text = input("Enter n: ") 4 if input_text.isdigit(): 5 n = int(input_text) 6 if n >= 1 and n <= 19 and n % 2 == 1: 7 break 8 else: 9 print("Data validation failed!") 10 print("n should be odd and between 1 and 19 inclusive") 11 else: 12 print("Data validation failed!") 13 print("n should be a positive integer") 14 15 # Process and output 16 for index in range(n // 2): 17 print(' ' * (n // 2 - index) + '*' * (2 * index + 1)) 18 for index in range(n // 2 + 1): 19 print(' ' * index + '*' * (n - index * 2)) </pre>

3. Although the run-time error occurs on line 25, it is actually caused by a logic error on line 18. This program uses `stars` to store each line of the diamond in a `list` but leaves out the widest line of the diamond due to a logic error. The most straightforward way to fix this error is to change the condition on line 18 from “`index < n`” to “`index <= n`”.

The corrected program is as follows:

#	Program diamond_crash_fixed.py
	<pre> 1 # Input and input validation 2 while True: 3 n_str = input("Enter n: ") 4 if not n_str.isdigit(): 5 print("Data validation failed!") 6 print("n should be a positive integer") 7 continue 8 n = int(n_str) 9 if n < 1 or n > 19 or n % 2 == 0: 10 print("Data validation failed!") 11 print("n should be odd and between 1 and 19 inclusive") 12 continue 13 break 14 15 # Process and output 16 stars = [] 17 index = 1 18 while index <= n: 19 stars += [" " * ((n - index) // 2) + "*" * index] 20 index += 2 21 22 for index in range(n // 2): 23 print(stars[index]) 24 for index in range(n // 2 + 1): 25 print(stars[-index - 1]) </pre>

Quick Check 6.3

1. The results from running `acronym_alternative.py` on all six test cases are as follows:

→ Input for Test Case 1	Expected Output →	
• <code>input_text: "Computer Science"</code>	CS	
Running <code>acronym_alternative.py</code> on Test Case 1 →		Result
Enter the input text: Computer Science CS		Passed

→ Input for Test Case 2	Expected Output →	
• <code>input_text: "Meetings on etiquette"</code>	MOE	
Running <code>acronym_alternative.py</code> on Test Case 2 →		Result
Enter the input text: Meetings on etiquette MOE		Passed

→ Input for Test Case 3	Expected Output →	
• <code>input_text: ""</code>	(blank output)	
Running <code>acronym_alternative.py</code> on Test Case 3 →		Result
Enter the input text:		Passed

→ Input for Test Case 4	Expected Output →	
• <code>input_text: " Computer Science "</code>	CS	
Running <code>acronym_alternative.py</code> on Test Case 4 →		Result
Enter the input text: Computer Science CS		Passed

➔ Input for Test Case 5	Expected Output ➔	
• <code>input_text: "Lead, Care, Inspire"</code>	Data validation failed! Input text should only contain letters and spaces	
Running <code>acronym_alternative.py</code> on Test Case 5 ➔		Result
Enter the input text: <code>Lead, Care, Inspire</code> <code>LCI</code>		Failed

➔ Input for Test Case 6	Expected Output ➔	
• <code>input_text: "Computing 2017"</code>	Data validation failed! Input text should only contain letters and spaces	
Running <code>acronym_alternative.py</code> on Test Case 6 ➔		Result
Enter the input text: <code>Computing 2017</code> <code>C2</code>		Failed

This shows that `acronym_alternative.py` passes all test cases except Test Cases 5 and 6 as the program does not perform any input validation.

Review Questions

1. a) i) Line 5 splits `input_text` into a list of words (`word_list`) that are separated by spaces.
 ii) Lines 7 to 11 reverse the order of the letters in each word in `word_list`.
 iii) Lines 13 to 16 join the contents of `word_list` together using spaces.

- b) i) Normal condition
 ii) A logic error has occurred. While the program has not crashed, its actual output is different from the expected output.
 iii) The error is located on line 10 and is caused by using the index operator to count from the end of a string incorrectly. (The last character of a string is at index -1, not index -0.)

iv) The corrected program is as follows:

#	Program word_reversal.py
	<pre> 1 # Input 2 input_text = input("Enter input text: ") 3 4 # Process 5 word_list = input_text.split() 6 7 for i in range(len(word_list)): 8 current_word = "" 9 for letter_index in range(len(word_list[i])): 10 current_word += word_list[index][-letter_index - 1] 11 word_list[i] = current_word 12 13 result = "" 14 for word in word_list: 15 result += word + " " 16 result = result[:-1] 17 18 # Output 19 print(result) </pre>

2. a) i) True
ii) False
iii) True
iv) False
- b) i) The return value should be `True` since an empty string, having no characters, trivially satisfies the requirement of not containing any non-letter characters.
- ii) The return value should be `False` since an empty string, having no characters, is not a sensible input for the task of checking whether every character is a letter.
- iii) Since an empty string is considered valid input, we choose to have `is_letters_only('')` return `True` so an empty string will pass data validation. A possible answer is as follows:

```

# Returns True if s has alphabetical letters only
def is_letters_only(s):
    for c in s:
        if not c.isalpha():
            return False
    return True

```

- c) i) This is because `convert_string()` calls and uses the return value of `convert_letter()` in its function body to determine its own return value. An incorrect return value from `convert_letter()` is likely to result in an incorrect return value for `convert_string()`.
- ii) The bug is in `convert_string()`. A possible answer is as follows:

#	Program opposite_case_finished.py
	<pre> 1 # Returns True if s has alphabetical letters only 2 def is_letters_only(s): 3 for c in s: 4 if not c.isalpha(): 5 return False 6 return True 7 8 # Converts single letter to opposite case 9 def convert_letter(l): 10 if l.islower(): 11 return l.upper() 12 return l.lower() 13 14 # Converts string of letters to opposite case 15 def convert_string(s): 16 result = '' 17 for c in s: 18 result = result + convert_letter(c) 19 return result 20 21 while True: 22 s = input('Enter string: ') 23 if is_letters_only(s): 24 break 25 print('String must have alphabetical letters only') 26 27 print(convert_string(s)) </pre>

Chapter 7 How Can I Be a Safe and Responsible Computer User?

Quick Check 7.1

1. How data can become corrupted (accept any possible answer):

- By human error, such as multiple users writing to the same file at the same time
- By hardware failure, such as a hard drive failing due to manufacturing defects or overusage
- By power failure to the computer or storage device
- By malicious software that may be running on the computer

Quick Check 7.2

1. Advantages of using biometric systems (accept any possible answer):

- Ensures accuracy as students have to provide hard-to-fake evidence of their own identities before their attendance can be marked
- Reduces time and effort by teachers and administrative staff in tracking attendance as students are responsible for marking their own attendance
- Provides better protection of the attendance data from unauthorised access compared to paper attendance sheets that can be read and modified by anyone

2. Possible measures (accept any possible answer):

- Use encryption to ensure that customer data is protected from eavesdroppers when submitted over the Internet
- Ensure that customer data is stored on a computer that is protected by a properly configured firewall
- Require multiple authentication factors to verify a customer's identity before providing access to that customer's data
- Ensure that customer data is stored on a computer which has its software updated regularly so that attackers cannot exploit known bugs in the software

3. CAPTCHA stands for Completely Automated Public Turing Test to Tell Computers and Humans Apart. It tries to verify that an actual human and not a machine is using the website by requiring the user to type out the letters and/or numbers in a distorted image.

CAPTCHAs are usually used when access control is needed to deny computer programs and other automated systems access to a website or its services.

Quick Check 7.3

- 1.** C
- 2.** B
- 3.** a) Yes
b) No
c) No
d) Yes
e) Yes
f) No
- 4.** Possible effects on the company (accept any possible answer):
 - Legal punishment due to lawsuits from customers
 - Legal punishment due to incorrect claims made in the company's privacy policy
 - Loss of reputation and customer trustPossible effects on its customers (accept any possible answer):
 - Threats or harassment from strangers
 - Identity theft for accounts where an intruder is able to use the revealed mobile number and address information to overcome authentication
 - Inconvenience in changing mobile phone number or address
- 5.** Siti should verify the authenticity of the source by checking that the sender is legitimate, and if necessary, calling the organiser of the lucky draw and asking for proof of the lucky draw results. Even if the result is legitimate, she should request to receive the cash prize in a manner that will not reveal her personal details to a stranger. If there is any doubt, she should decline to reveal any personal details and ignore the message.

Quick Check 7.4

- 1.** a) Both copyright infringement and plagiarism
b) Copyright infringement
c) Neither
d) Plagiarism

Review Questions

1. a) Phishing uses email hyperlinks that lead to a fake website with a different address from the real website, while pharming uses website redirection to show a fake website that uses the same address as the real website.
- b) Phishing
- c) Two possible ways (accept any possible answer):
- The customers could have verified if the email hyperlink's actual destination matched the real address of the bank's website before clicking on it.
 - The customers could have contacted the bank directly to verify the identity of the sender and the authenticity of the email.

2.

	Threat	Unauthorised access likely to occur? Why?	Preventive measure
a)	Cookies	No. Cookies are files that store user information each time a user visits a website. They are not malicious in nature and cannot store information that the user did not already provide to the website through a web browser.	–
b)	Spam	No. Spam is the sending of mass emails for advertising purposes. Unless any hyperlinks from the spam are clicked, no unauthorised data is likely to be sent back to the spammer.	–
c)	Trojan horses	Yes. Trojan horses are malicious in nature and may result in unauthorised access if run on a computer.	Install anti-virus and anti-spyware programs and download their updates regularly.
d)	Viruses	Yes. Viruses are malicious in nature and may result in unauthorised access if run on a computer.	Install anti-virus and anti-spyware programs and download their updates regularly.

3. a) Two economic effects (accept any possible answer):
- The price of music albums, television programmes and films would go up to compensate for the loss in sales due to piracy.
 - There would be fewer music albums, television programmes and films produced as people would find it increasingly difficult to make money from doing so.
- b) Siti's actions are considered to be copyright infringement as she did not have explicit permission from the copyright owner to copy the television series onto her hard drive.
- c) The icon represents that the photograph is under a Creative Commons licence. It means that users are granted permission to copy, modify and distribute the photograph under certain conditions.
- d) Music may be downloaded legally without copyright infringement as long as the copyright owner has given permission for the downloading to occur. For instance, the copyright owner may allow a music file to be downloaded from a legitimate online music store as long as payment is made. Alternatively, the music file can be placed under a Creative Commons licence, which explicitly gives everyone permission to download the file as long as certain conditions are met.
4. a) What the company should do (accept any possible answer):
- Make regular backups of the submitted entries.
 - Check the server's storage devices regularly and replace them immediately when signs of failure are detected.
 - Ensure that the server is set up with a backup power supply or an uninterruptible power supply (UPS).
- b) Plagiarism.
- c) Two possible responses (accept any possible answer):
- The artwork's creator should win because he has not violated any intellectual property laws. The painting's copyright is not violated since the resulting artwork is significantly different from the painting. The photograph he used is also not protected by copyright as it is under the public domain. Hence, his submission of the artwork is legally allowed.
 - The artwork's creator should not win because his submitted artwork was automatically generated using a computer program he did not write and from images that he did not create. Hence, no matter how impressive the result, he has demonstrated little to no originality or artistic ability.

Chapter 8 How are Number Systems Applied in Real Life?

Quick Check 8.2

1. a) 110110_2

Using the division by 2 method:

Denary	Quotient	Remainder
54	27	0
27	13	1
13	6	1
6	3	0
3	1	1
1	0	1



Direction to read the digits

Using the sum of place values method:

Place value	256	128	64	32	16	8	4	2	1	Denary
Binary digit	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	54 22 6 2 0
				1	1	0	1	1	0	

b) 1011001_2

Using the division by 2 method:

Denary	Quotient	Remainder
89	44	1
44	22	0
22	11	0
11	5	1
5	2	1
2	1	0
1	0	1



Direction to read the digits

Using the sum of place values method:

Place value	256	128	64	32	16	8	4	2	1	Denary
Binary digit	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	89 25 9 4 0
			1	0	1	1	0	0	1	

c) $110\ 0100_2$

Using the division by 2 method:

Denary	Quotient	Remainder
100	50	0
50	25	0
25	12	1
12	6	0
6	3	0
3	1	1
1	0	1



Direction to read the digits

Using the sum of place values method:

Place value	256	128	64	32	16	8	4	2	1	Denary
	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Binary digit			1	1	0	0	1	0	0	100 36 4 0

2. $7 = 111_2$

$8 = 1000_2$

$9 = 1001_2$

$10 = 1010_2$

3. a) 19

b) 96

c) 134

Quick Check 8.3

1.	Denary	Binary	Hexadecimal
32	10 0000 ₂	20 ₁₆	
33	10 0001 ₂	21 ₁₆	
34	10 0010 ₂	22 ₁₆	
35	10 0011 ₂	23 ₁₆	
36	10 0100 ₂	24 ₁₆	
37	10 0101 ₂	25 ₁₆	
38	10 0110 ₂	26 ₁₆	
39	10 0111 ₂	27 ₁₆	
40	10 1000 ₂	28 ₁₆	
41	10 1001 ₂	29 ₁₆	
42	10 1010 ₂	2A ₁₆	
43	10 1011 ₂	2B ₁₆	
44	10 1100 ₂	2C ₁₆	
45	10 1101 ₂	2D ₁₆	
46	10 1110 ₂	2E ₁₆	
47	10 1111 ₂	2F ₁₆	

2. a) 1001001₂
 b) 101010110110₂
 c) 1101101001000111₂
3. a) 157
 b) 2998
 c) 44432
4. a) 3E8₁₆
 b) ECD₁₆
 c) 107C₁₆

Quick Check 8.4

1. A possible RGB colour code for magenta is #FF00FF.
2. A possible program is as follows:

#	Program: valid_ipv4.py
	<pre> 1 # Input 2 input_str = input("Enter string: ") 3 4 # Process 5 is_valid = True # Assume input is valid until proven wrong 6 processed = 0 7 digits = "" 8 index = 0 9 10 # To avoid repeated code, <= instead of < is used so collected 11 # digits are checked one final time at end of input string 12 while index <= len(input_str): 13 # Get current character only if not at end of input string 14 # Otherwise, current character is None 15 current = None 16 if index < len(input_str): 17 current = input_str[index] 18 19 if current == None or current == '.': 20 # If at end of input string or current character is a dot, 21 # check that digits is a valid int between 0 and 255 22 if not digits.isdigit(): 23 is_valid = False 24 break 25 digits_int = int(digits) 26 if not (digits_int >= 0 and digits_int <= 255): 27 is_valid = False 28 break 29 # digits was valid, so increase processed and reset digits 30 processed += 1 31 digits = "" 32 elif current.isdigit(): 33 # If current character is a digit, add it to digits 34 digits += input_str[index] 35 else: 36 # If current character is not a digit or a dot, input is invalid 37 is_valid = False 38 break 39 40 # Update index to process next character 41 index += 1 42 </pre>

#	Program: valid_ipv4.py (continued)
43	if is_valid and processed != 4: # If input may still be valid, check that 4 ints were processed
44	is_valid = False
45	
46	
47	# Output
48	if is_valid:
49	print("Valid")
50	else:
51	print("Invalid")

Review Questions

1. When representing large numbers, the hexadecimal form is much shorter than the binary form, making it more compact and easier to key in.

2. a) 1010_2
 b) $1111\ 1111_2$
 c) $10000\ 1011_2$

3. a) D_{16}
 b) 9_{16}
 c) 5_{16}
 d) ED_{16}
 e) 55_{16}
 f) $16CD_{16}$

4. a) $0110\ 1100_2$
 b) $6C_{16}$
 c) Common uses of the hexadecimal number system (accept any two valid answers):
 - To represent red, green and blue intensities in an RGB colour code
 - To present IPv6 addresses in a compact format
 - To present MAC addresses in a compact format
 - To present ASCII codes in a compact format

5. a) There are faults in the display and motor subsystems for lift number 7.
b) There are faults in the motor and ventilator subsystems for lift number 11.
c) 11001110

6. a) Electrical point number 56 is operational.
b) 00100100
c) $9C_{16}$

Chapter 9 How Do Logic Circuits Make Decisions?

Quick Check 9.3

1. a) True

b) True

c) False

2. A

3. B

Quick Check 9.4

1. a) 3 (Note: Intermediate inputs are not counted as inputs.)

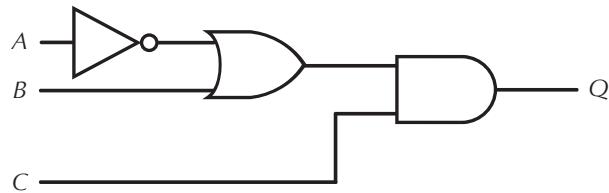
b) 1

Quick Check 9.5

1. a) Truth table:

Input A	Input B	Input C	Intermediate Input (NOT A)	Intermediate Input (NOT A OR B)	Output Q
0	0	0	1	1	0
0	0	1	1	1	1
0	1	0	1	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	1	0
1	1	1	0	1	1

b) Logic circuit:



Review Questions

1. a) $Q = A \text{ OR } (B \text{ AND } C)$

Input A	Input B	Input C	Intermediate Input (B AND C)	Output Q
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(Note: Q is 1 when either A is 1, or both B and C are 1.)

b) $X = (\text{NOT } A) \text{ AND } (\text{NOT } B)$

Input A	Input B	Intermediate Input (NOT A)	Intermediate Input (NOT B)	Output X
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

(Note: X is 1 only when both A and B are 0.)

c) $Y = A \text{ AND } (\text{NOT } B)$

Input A	Input B	Intermediate Input (NOT B)	Output Y
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

(Note: Y is 1 only when A is 1 and B is 0.)

d) $Z = (A \text{ OR } B) \text{ AND } (C \text{ OR } D)$

Input A	Input B	Input C	Input D	Intermediate Input (A OR B)	Intermediate Input (C OR D)	Output Z
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	1	1	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	0	0
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	0	0
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

2. a) Boolean statement: $Q = A \text{ OR } B$

b) Truth table:

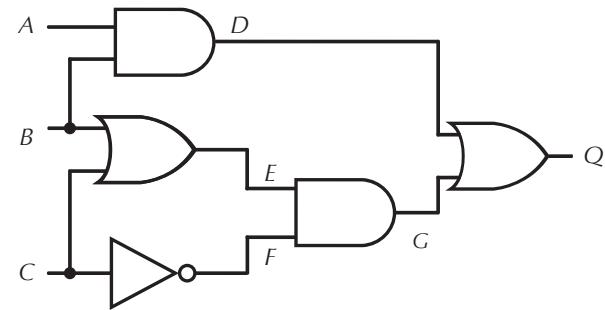
Input A	Input B	Input Q
0	0	0
0	1	1
1	0	1
1	1	1

3. **Step 1:** Total number of rows in truth table $= 2^3 = 8$ as there are three inputs.

Step 2: Draw the truth table and fill in all the input combinations.

Input A	Input B	Input C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Step 3: Work out the intermediate inputs D , E , F and G first.



$$D = A \text{ AND } B$$

$$E = B \text{ OR } C$$

$$F = \text{NOT } C$$

$$G = E \text{ AND } F$$

Input A	Input B	Input C	Intermediate				Output Q
			Input D	Input E	Input F	Input G	
0	0	0	0	0	1	0	
0	0	1	0	1	0	0	
0	1	0	0	1	1	1	
0	1	1	0	1	0	0	
1	0	0	0	0	1	0	
1	0	1	0	1	0	0	
1	1	0	1	1	1	1	
1	1	1	1	1	0	0	

Step 4: Work out the final output, Q .

$$Q = D \text{ OR } G$$

Input A	Input B	Input C	Intermediate				Output Q
			Input D	Input E	Input F	Input G	
0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	0
0	1	0	0	1	1	1	1
0	1	1	0	1	0	0	0
1	0	0	0	0	1	0	0
1	0	1	0	1	0	0	0
1	1	0	1	1	1	1	1
1	1	1	1	1	0	0	1

Chapter 10 How are Spreadsheets Used to Process and Analyse Data?

Quick Check 10.1

- 1.** a) Number
b) Text
c) By entering an apostrophe in front of the equals sign, i.e., '= '
d) C4:C15
e) E4 only
f) C4:C15 and E4:E15

- 2.** =A2/B2

Quick Check 10.2

- 1.** a) 3
b) Text
c) Currency

Quick Check 10.3

- 1.** a) =B2*D2
b) =SUM(E2:E9)
c) i) 6
ii) B

- 2.** Possible formulas (accept any possible answer):
 - =IF(A1>3, "Greater than 3", IF(A1=3, "Equal to 3", "Less than 3"))
 - =IF(A1>3, "Greater than 3", IF(A1<3, "Less than 3", "Equal to 3"))
 - =IF(A1=3, "Equal to 3", IF(A1<3, "Less than 3", "Greater than 3"))
 - =IF(A1=3, "Equal to 3", IF(A1>3, "Greater than 3", "Less than 3"))

- 3.** a) =VLOOKUP(B2, \$E\$2:\$F\$4, 2)
b) =AVERAGE(C2:C6)
c) =MEDIAN(C2:C6)

4. a) =ROUND(B2, 1)
 b) =COUNTIF(C2:C15, "<60")
 c) =MAX(C2:C15)-MIN(C2:C15)

Review Questions

1. a) =G\$1*B2
 b) 0.67%
 c) The required conditional formatting settings for D2:D13 are as follows:

The screenshot shows an Excel spreadsheet titled "Account Answer.xlsx". The spreadsheet contains a table of monthly account activity from January to December, plus a final balance at the end of December. The columns are labeled A through H. The "Interest rate" is set to 0.67%. The "Money Withdrawn" column uses conditional formatting where values greater than 100 are displayed in red text.

	A	B	C	D	E	F	G	H
1	Month	Balance at Start	Interest Earned	Money Withdrawn			Interest rate:	0.67%
2	January	\$5,000.00	\$33.62	\$34.84				
3	February	\$4,998.78	\$33.61	\$140.73				
4	March	\$4,891.66	\$32.89	\$96.34				
5	April	\$4,828.21	\$32.47	\$56.97				
6	May	\$4,803.71	\$32.30	\$102.42				
7	June	\$4,733.59	\$31.83	\$33.31				
8	July	\$4,732.11	\$31.82	\$157.24				
9	August	\$4,606.69	\$30.98	\$151.30				
10	September	\$4,486.36	\$30.17	\$68.14				
11	October	\$4,448.39	\$29.91	\$35.26				
12	November	\$4,443.04	\$29.88	\$168.53				
13	December	\$4,304.39	\$28.94	\$133.33				
14								
15	Amount left at end of December:			\$4,200.00				
16								

Below the table, a "Greater Than" conditional formatting dialog box is open. It shows the formula `100` in the "Format cells that are GREATER THAN:" field, and "Red Text" selected in the "with" dropdown. The dialog box has "OK" and "Cancel" buttons.

Chapter 11 How Do I Create a Simple Network?

Quick Check 11.2

1. C

2. A

3. Advantages of a client-server network (accept any possible answer):

- Centralised control of data and resources
- Easier to schedule backups of all shared files at regular intervals
- Enhances security with the use of built-in security features through the network operating system
- Server is accessible remotely and across multiple platforms

4. a) Peer-to-peer

b) Client-server

c) Client-server

Quick Check 11.3

1. She can install a wireless printer to connect to both her laptop and tablet. Possible advantages (accept any possible answer):

- Siti can move her printer to anywhere within the range of the wireless coverage at home without being restricted to a fixed location.
- It is easy to connect more mobile devices to the wireless printer.
- The physical organisation is neater without the use of cables running across rooms.

2. Why wireless networks are preferred over wired networks at cafes and canteens (accept any possible answer):

- A wireless network allows for user mobility within the range of the wireless coverage without being bound to a fixed position unlike in wired networks.
- The physical organisation is neater without the use of cables.

Quick Check 11.4

1. Accept all valid answers.

2. a) A program running on a network can be uniquely identified by its *port number* and IP address.

b) All *wireless* devices connected to the same WAP must use the same SSID.

Quick Check 11.5

1. D
2. Accept all valid answers.
 - Home network using fibre optic or broadband: the hardware could include a modem (optical network terminal modem or broadband modem) and a router
 - School network: the hardware could include a modem, router and multiple hubs or switches

(Note: Some devices have integrated functions incorporating modem, router and switch capabilities in a single device.)

Quick Check 11.6

1. B
2. C
3. a) Star topology. If a fault occurs at a computer or cable, it is easy to isolate the fault and do a replacement without affecting the rest of the network. This makes the network more fault-tolerant and resilient as compared with the bus and ring topologies.
- b) Why a client-server network is preferred over a peer-to-peer network:
 - Bandwidth: The network needs to support a large number of users (about 120) with the possibility of a business expansion in the future. If a P2P network were to be used, the bandwidth of the network could decrease if the number of computers were to increase.
 - Security: As the accountancy consulting business requires the use and handling of financial data, the nature of the business calls for high confidentiality. A client-server network is best for implementing a secure system where users are required to authenticate to a network, while a P2P network offers little to no security.
 - Storage: The business has a large data store dating back to 10 years ago. A client-server network enables data to be stored centrally and is best for supporting businesses requiring large data storage as storage servers can be added and maintained centrally. In a P2P network, data is decentralised and stored on clients instead of in a server. This limits the amount of data that can be stored as clients have smaller storage space compared to servers.

Quick Check 11.7

- 1.** In an odd parity system, the total number of 1 bits is odd, hence the parity bit has to be 0.

0	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---

- 2. a)** Parity bit: 0

The total number of 1 bits is 2, which is even.

- b)** Parity bit: 0

The total number of 1 bits is 4, which is even.

- c)** Parity bit: 1

The total number of 1 bits should be 6, which will give an even parity system.

- 3. a)** The data packet is corrupted. The total number of 1 bits in the first seven bits is 5, which is an odd number. The parity bit should be 0 and not 1 in an odd parity system.
- b)** The data packet is not corrupted. The total number of 1 bits in the first seven bits is 5, which is an odd number. The parity bit is 0, which tallies with the odd parity system.

Review Questions

- 1.** Three advantages of setting up a home or office network (accept any possible answer):

- Allows users to share data and files
- Allows users to share network resources such as printers or storage
- Allows fast data transmission and communication between users within the network
- Allows data to be backed up and accessed centrally
- Can be configured for added security

2.	Design factors	Peer-to-peer	Client-server
a)	Organisation of components	C. All computers are independent of each other and are of equal status when communicating with each other	D. The network consists of computers operated by users and centralised computers which control the resources of all computers
b)	Resource management	B. Resources and files located on one computer are accessible by all other computers within the network	A. Access to shared resources on the network such as printers, data files and web access is managed by servers
c)	Storage	G. Data is stored independently in each computer	H. Data is stored in a centralised storage device monitored by the server
d)	Mode of backup	F. Backups are difficult to organise since there is no centralised backup storage	E. Regular backups can be easily implemented with the use of a centralised backup storage
e)	Security	K. There is often no mechanism to manage access to the network, hence it is less secure	L. Access to the network is managed using a database of usernames, passwords and user profile and permission settings, hence the network tends to be more secure
f)	Type of environment	J. More commonly found in homes	I. More commonly found in businesses and organisations

3. a) Star topology.
- b) The patient history database should be stored on a single server using a client-server network. As the database contains sensitive data, access to its contents needs to be controlled and monitored. This is easier to achieve on a client-server network compared to a peer-to-peer network. Centralising storage on a single server also ensures that there is only one official copy of the database and that it is always the latest version. On the other hand, distributing the database using a peer-to-peer network may result in multiple conflicting copies as different users make changes.
- c) For the wired network, the following devices should be connected to the network switch: one laptop for the doctor, three laptops for staff, one printer and one dedicated server (optional). For the wireless network, the fibre optic modem should be connected to the wireless router/access point to provide Internet access. Both networks should be isolated from each other. (Note: Detailed drawings of laptops, printers or other devices are not required. Simple shapes such as boxes or ovals can be used to represent any device as long as they are clearly labelled.)

