5 System software

In this chapter, you will learn about

- why computers need an operating system
- key management tasks, such as memory management, file management, security management, hardware management and process management
- the need for utility software, including disk formatters, virus checkers, defragmentation software, disk content analyse and repair software, file compression and back-up software
- program libraries, software under development using program library software and the benefits to software developers, including the use of dynamic link library (DLL) files
- the need for these language translators: assemblers, compilers and interpreters
- the benefits and drawbacks of using compilers or interpreters
- an awareness that high level language programs may be partially compiled and partially interpreted (such as Java^{TM})
- the features of a typical integrated development environment (IDE) for
  - coding (using context-sensitive prompts)
  - initial error detection (including dynamic syntax checks)
  - presentation (including pretty print, expand and collapse code blocks)
  - debugging (for example, single stepping, use of breakpoints, variables/expressions report windows).

5.1 Operating systems

WHAT YOU SHOULD ALREADY KNOW

Try these five questions before you read the first part of this chapter.

1 Microprocessors are commonly used to control microwave ovens, washing machines and many other household items.
Explain why it is **not** necessary for these devices to have an operating system.

2 a) Name **three** of the most common operating systems used in computers and other devices, such as mobile phones and tablets.
b) A manufacturer makes laptop computers, mobile phones and tablets.
Explain why it is necessary for the manufacturer to develop different versions of its operating system for use on its computers, mobile phones and tablets.

3 Most operating systems offer a graphic user interface (GUI) as well as a command line interface (CLI).
a) What are the main differences between the two types of interface?
b) What are the pros and cons of both types of interface?
c) Who would use each type of interface?

4 Before the advent of the operating system, computers relied on considerable human intervention.
Find out the methods used to start up early computers to prepare them for the day's tasks.

5 Describe the role of **buffers** and **interrupts** when a printing job is being sent to an inkjet printer.
Consider the different operational speeds of a processor and a printer, together with size of printing job and interrupt priorities.
Describe potential error scenarios – such as paper jam, out of paper or out of ink – and how these could affect the printing job.

Key terms

CMOS – complementary metal-oxide semiconductor.

Operating system – software that provides an environment in which applications can run and provides an interface between hardware and human operators.

HCI – human–computer interface.

GUI – graphical user interface.

CLI – command line interface.

Icon – small picture or symbol used to represent, for example, an application on a screen.

WIMP – windows, icons, menu and pointing device.

Post-WIMP – interfaces that go beyond WIMP and use touch screen technology rather than a pointing device.

Pinching and rotating – actions by fingers on a touch screen to carry out tasks such as move, enlarge, reduce, and so on.

Memory management – part of the operating system that controls the main memory.

Memory optimisation – function of memory management that determines how memory is allocated and deallocated.

Memory organisation – function of memory management that determines how much memory is allocated to an application.

Security management – part of the operating system that ensures the integrity, confidentiality and availability of data.

Contiguous – items next to each other.

Virtual memory systems – memory management (part of OS) that makes use of hardware and software to enable a computer to compensate for shortage of actual physical memory.

Memory protection – function of memory management that ensures two competing applications cannot use same memory locations at the same time.

Process management – part of the operating system that involves allocation of resources and permits the sharing and exchange of data.

Hardware management – part of the operating system that controls all input/output devices connected to a computer (made up of sub-management systems such as printer management, secondary storage management, and so on).

Device driver – software that communicates with the operating system and translates data into a format understood by the device.

Utility program – parts of the operating system which carry out certain functions, such as virus checking, defragmentation or hard disk formatting.

Disk formatter – utility that prepares a disk to allow data/files to be stored and retrieved.

Bad sector – a faulty sector on an HDD which can be soft or hard.

Antivirus software – software that quarantines and deletes files or programs infected by a virus (or other malware). It can be run in the background or initiated by the user.

Heuristic checking – checking of software for behaviour that could indicate a possible virus.

Quarantine – file or program identified as being infected by a virus which has been isolated by antivirus software before it is deleted at a later stage.

False positive – a file or program identified by a virus checker as being infected but the user knows this cannot be correct.

Disk defragmenter – utility that reorganises the sectors on a hard disk so that files can be stored in contiguous data blocks.

Disk content analysis software – utility that checks disk drives for empty space and disk usage by reviewing files and folders.

Disk compression – software that compresses data before storage on an HDD.

Back-up utility – software that makes copies of files on another portable storage device.

Program library – a library on a computer where programs and routines are stored which can be freely accessed by other software developers for use in their own programs.

Library program – a program stored in a library for future use by other programmers.

Library routine – a tested and ready-to-use routine available in the development system of a programming language that can be incorporated into a program.

Dynamic link file (DLL) – a library routine that can be linked to another program only at the run time stage.

5.1.1 The need for an operating system

Early computers had no operating system at all. Control software had to be loaded each time the computer was started – this was done using either paper tape or punched cards.

In the 1970s, the home computer was becoming increasingly popular. Early examples, such as the Acorn BBC B, used an internal ROM chip to store part of the operating system. A cassette tape machine was also used to load the remainder of the operational software (see Figure 5.1). This was necessary to 'get the computer started' and used a welcome cassette tape which had to be used each time the computer was turned on.

Figure 5.1 An Acorn BBC B (left) and its cassette tape machine (right)

As the hard disk drive (HDD) was developed, operating systems were stored on the hard disk, and start-up of the motherboard was handled by the basic input/output system (BIOS). Initially, the BIOS was stored on a ROM chip but, in modern computers, the BIOS contents are stored on a flash memory chip. The BIOS configuration is stored in **CMOS memory (complementary metal-oxide semiconductor)** which means it can be altered or deleted as required.

The required part of the operating system is copied into RAM – since operating systems are now so large, it would seriously affect a computer's performance if it was all loaded into RAM at once. An **operating system** provides both the environment in which applications can be run, and a useable interface between humans and computer. An operating system also disguises the complexity of computer hardware. Common examples include Microsoft Windows®, Apple Mac OS, Google Android and IOS (Apple mobile phones and tablets).

The **human–computer interface (HCI)** is usually achieved through a **graphical user interface (GUI)**, although it is possible to use a **command line interface (CLI)** if the user wishes to directly communicate with the computer.

A CLI requires a user to type instructions to choose options from menus, open software, and so on. There are often a number of commands that need to be typed; for example, to save or load a file. The user, therefore, has to learn a number of commands (which must be typed exactly with no errors) just to carry out basic operations. Furthermore, it takes time to key in commands every time an operation has to be carried out.

The advantage of CLI is that the user is in direct communication with the computer and is not restricted to a number of pre-determined options.

For example, the following section of CLI imports data from table A into table B. It shows how complex it is just to carry out a straightforward operation.

```
SQLPrepare(hStmt,
?   (SQLCHAR *) "INSERT INTO tableB SELECT * FROM tableA",
?   SQL_NTS):
?   SQLExecute(hStmt);
```

A GUI allows the user to interact with a computer (or MP3 player, gaming device, mobile phone, and so on) using pictures or symbols (**icons**). For example, the whole of the above CLI code could have been replaced by a single icon, like the one on the left.

Table update

Selecting this icon would execute all of the steps shown in the CLI without the need to type them.

GUIs use various technologies and devices to provide the user interface. One of the first commonly used GUI environments was known as **windows, icons, menu and pointing device (WIMP)**, which was developed for use on personal computers (PCs). Here, a mouse is used to control a cursor and icons are selected to open and run windows. Each window contains an application. Modern computer systems allow several windows to be open at the same time. An example is shown in Figure 5.2.

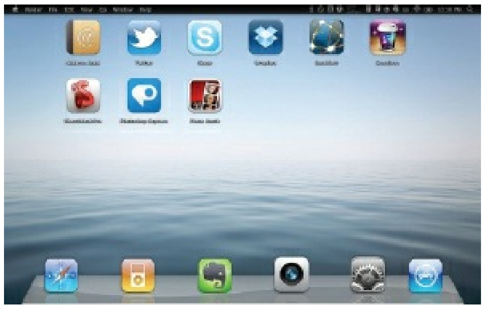


Figure 5.2 An example of WIMP

A windows manager looks after the interaction between windows, the applications and windowing system (which handles the pointing devices and the cursor's position).

However, smart phones, tablets and many computers now use a **post-WIMP** interaction where fingers are in contact with the screen, allowing actions such as **pinching** and **rotating** which are difficult using a single pointer and device such as a mouse. Also, simply tapping the icon with a finger (or stylus) will launch the application. Developments in touch screen technology mean these flexible HCIs are now readily available.

5.1.2 Operating system tasks

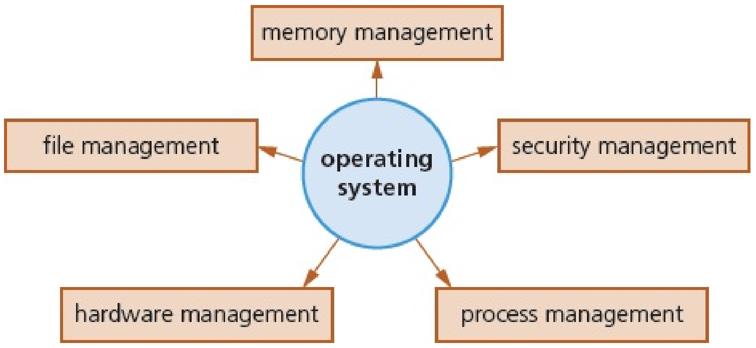


Figure 5.3 Operating system tasks

Memory management

Memory management, as the name suggests, is the management of a computer's main memory. This can be broken down into three parts: **memory optimisation**, **memory organisation** and **memory protection**.

Memory optimisation

Memory optimisation is used to determine how computer memory is allocated and deallocated when a number of applications are running simultaneously. It also determines *where* they are stored in memory. It must, therefore, keep track of all allocated memory and free memory available for use by applications. To maintain optimisation of memory, it will also swap data to and from the HDD or SSD.

Memory organisation

Memory organisation determines how much memory is allocated to an application, and how the memory can be split up in the most appropriate or efficient manner.

This can be done with the use of

- a single (**contiguous**) allocation, where all of the memory is made available to a single application. This is used by MS-DOS and by embedded systems
- partitioned allocation, where the memory is split up into contiguous partitions (or blocks) and memory management then allocates a partition (which can vary in size) to an application
- paged memory, which is similar to partitioned allocation, but each partition is of a fixed size. This is used by **virtual memory systems**
- segmented memory, which is different because memory blocks are not contiguous – each segment of memory will be a logical grouping of data (such as the data which may make up an array).

Memory protection

Memory protection ensures that two competing applications cannot use the same memory locations at the same time. If this was not done, data could be lost, applications could produce incorrect results, there could be security issues, or the computer may crash.

Memory protection and memory organisation are different aspects of an operating system. An operating system may use a typical type of memory organisation (for example, it may use paging or segmentation) but it is always important that no two applications can occupy the same part of memory. Therefore, memory protection must *always* be a part of any type of memory organisation used.

Figure 5.4 shows how different applications can be kept separate from each other.

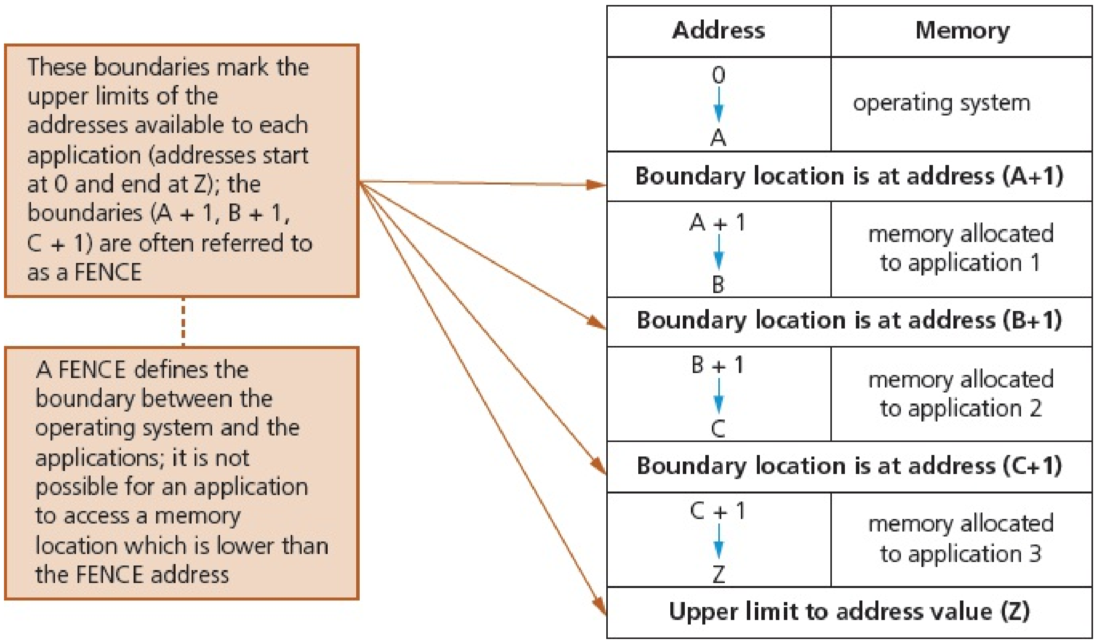

These boundaries mark the upper limits of the addresses available to each application (addresses start at 0 and end at Z); the boundaries (A + 1, B + 1, C + 1) are often referred to as a FENCE
A FENCE defines the boundary between the operating system and the applications; it is not possible for an application to access a memory location which is lower than the FENCE address
Address
Memory
0
A
operating system
Boundary location is at address (A+1)
A + 1
B
memory allocated to application 1
Boundary location is at address (B+1)
B + 1
C
memory allocated to application 2
Boundary location is at address (C+1)
C + 1
Z
memory allocated to application 3
Upper limit to address value (Z)

Figure 5.4 Memory protection

Security management

Security management is another part of a typical operating system. The function of security management is to ensure the integrity, confidentiality and availability of data.

This can be achieved by

- carrying out operating system updates as and when they become available
- ensuring that antivirus software (and other security software) is always up-to-date
- communicating with, for example, a firewall to check all traffic to and from the computer
- making use of privileges to prevent users entering ‘private areas’ on a computer which permits multi-user activity (this is done by setting up user accounts and making use of passwords and user IDs). This helps to ensure the privacy of data
- maintaining access rights for all users
- offering the ability for the recovery of data (and system restore) when it has been lost or corrupted
- helping to prevent illegal intrusion to the computer system (also ensuring the privacy of data).

Note: many of these features are covered in more depth elsewhere in this chapter or in other chapters.

EXTENSION ACTIVITY 5A

While working through the remainder of Chapters 5 and 6, find out all of the methods available to ensure the security, privacy and integrity of data and how these link into the operating system security management. It is important to distinguish between what constitutes security, privacy and integrity of data.

Process management

A process is a program which is being run on a computer. **Process management** involves the allocation of resources and permits the sharing and exchange of data, thus allowing all processes to be fully synchronised (for example, by the scheduling of resources, resolution of software conflicts, use of queues and so on). This is covered in more depth in Chapter 16.

Hardware management

Hardware management involves all input and output peripheral devices.

The functions of hardware management include

- communicating with all input and output devices using **device drivers**
- translating data from a file (defined by the operating system) into a format that the input/output device can understand using device drivers
- ensuring each hardware resource has a priority so that it can be used and released as required.

The management of input/output devices is essentially the control and management of queues and buffers. For example, when printing out a document, the printer management

- locates and loads the printer driver into memory
- sends data to a printer buffer ready for printing
- sends data to a printer queue (if the printer is busy or the print job has a low priority) before sending to the printer buffer
- sends various control commands to the printer throughout the printing process
- receives and handles error messages and interrupts from the printer.

EXTENSION ACTIVITY 5B

Write down the tasks carried out by a keyboard manager when a user types text using a word processor. Consider the use of buffers and queues in your answer.

File management

The main tasks of file management include

- defining the file naming conventions which can be used (filename.docx, where the extension can be .bat, .htm, .dbf, .txt, .xls, and so on)
- performing specific tasks, such as create, open, close, delete, rename, copy, move
- maintaining the directory structures
- ensuring access control mechanisms are maintained, such as access rights to files, password

protection, making files available for editing, locking files, and so on

- specifying the logical file storage format (such as FAT or NTFS if Windows is being used), depending on which type of disk formatter is used (see Section 5.1.3)
- ensuring memory allocation for a file by reading it from the HDD/SSD and loading it into memory.

5.1.3 Utility software

Computer users are provided with a number of **utility programs** that are part of the operating system. However, users can also install their own utility software in addition. This software is usually initiated by the user, but some, such as virus checkers, can be set up to constantly run in the background. Utility software offered by most operating systems includes

- hard disk formatter
- virus checker
- defragmentation software
- disk contents analysis/repair software
- file compression
- back-up software.

Hard disk formatter

A new hard disk drive needs to be initialised ready for formatting. The operating system needs to know how to store files and where the files will be stored on the hard disks. A **disk formatter** will organise storage space by assigning it to data blocks (partitions). A disk surface may have a number of partitions (see Chapter 3 for more details regarding the organisation of data on hard disks). Note that partitions are contiguous blocks of data.

Once the partitions have been created, they must be formatted. This is usually done by writing files which will hold directory data and tables of contents (TOC) at the beginning of each partition. This allows the operating system to recognise a file and know where to find it on the disk surface. Different operating systems will use different filing systems; Windows, for example, uses new technology filing system (NTFS).

When carrying out full formatting using NTFS, all disk sectors are filled with zeros; these zeros are read back, thus testing the sector, but any data already stored there will be lost. So, it is important to remember that reformatting an HDD which has already been used will result in loss of data during the formatting procedure.

Disk formatters also have checking tools, which are non-destructive tests that can be carried out on each sector. If any **bad sector** errors are discovered, the sectors will be flagged as ‘bad’ and the file tracking records will be reorganised – this is done by replacing the bad sectors with new unused sectors, effectively repairing the faulty disk. A damaged file will now contain an ‘empty’ sector, which allows the file to be read but it will be corrupted since the bad sector will have contained important (and now lost) data. It would, therefore, be prudent to delete the damaged file leaving the rest of the HDD effectively repaired.

Bad sectors can be categorised as hard or soft. There are a number of ways that they can be produced, as shown in Table 5.1.

Hard bad sectors (difficult to repair)	Soft bad sectors
• caused by manufacturing errors • damage to disk surface caused by allowing the read-	• sudden loss of power leading to data corruption in some of the sectors

write head to touch the disk surface (for example, by moving HDD without first parking the read-write head) • system crash which could lead to damage to the disk surface(s)	• effect of static electricity leading to corruption of data in some of the sectors on the hard disk surfaces

▲ **Table 5.1** Hard and soft bad sectors

Virus checkers

Any computer (including mobile phones and tablets) can be subject to a virus attack (see Chapter 6).

There are many ways to help prevent viruses, such as being careful when downloading material from the internet, not opening files or links in emails from unknown senders, and by only using certified software. However, virus checkers – which are offered by operating systems – still provide the best defence against malware, as long as they are kept up to date and constantly run in the background.

Running **antivirus software** in the background on a computer will constantly check for virus attacks. Although various types of antivirus software work in different ways, they have some common features. They

- check software or files before they are run or loaded on a computer
- compare possible viruses against a database of known viruses
- carry out **heuristic checking** – this is the checking of software for types of behaviour that could indicate a possible virus, which is useful if software is infected by a virus not yet on the database
- put files or programs which may be infected into **quarantine**, to
  - automatically delete the virus, or
  - allow the user to decide whether to delete the file (it is possible that the user knows that the file or program is not infected by a virus – this is known as a **false positive** and is one of the drawbacks of antivirus software).

Antivirus software needs to be kept up to date since new viruses are constantly being discovered. Full system checks need to be carried out once a week, for example, since some viruses lie dormant and would only be picked up by this full system scan.

Defragmentation software

As an HDD becomes full, blocks used for files will become scattered all over the disk surface (in potentially different sectors and tracks as well as different surfaces). This happens as files are deleted, partially-deleted, extended and so on. The consequence is slower data access time: the HDD read-write head requires several movements just to find and retrieve the data making up the required file. It would be advantageous if files could be stored in contiguous sectors, considerably reducing HDD head movements.

Note that, due to their different operation when accessing data, this is less of a problem with SSDs.

Consider the following example using a disk with 12 sectors per surface.

We have three files (1, 2 and 3) stored on track 8 of the disk surface.

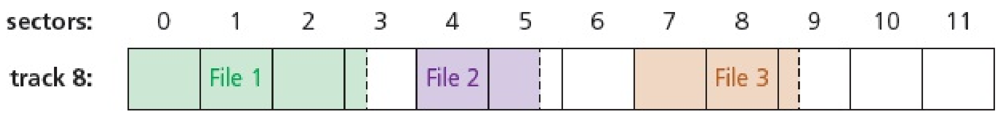


Figure 5.5

File 2 is deleted by the user and file 1 has data added to it. However, the file 2 sectors which become vacant are not filled up straight away by new file 1 data since this would require 'too much effort' for the HDD resources.

We get the following.

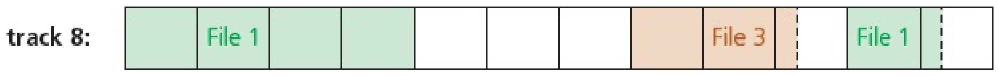


Figure 5.6

File 1 has been extended to write data in sectors 10 and 11.

Now, suppose file 3 is extended with the equivalent of 3.25 blocks of data. This requires filling up sector 9 and then moving to some empty sectors to write the remainder of the data – the next free sectors are on track 11.

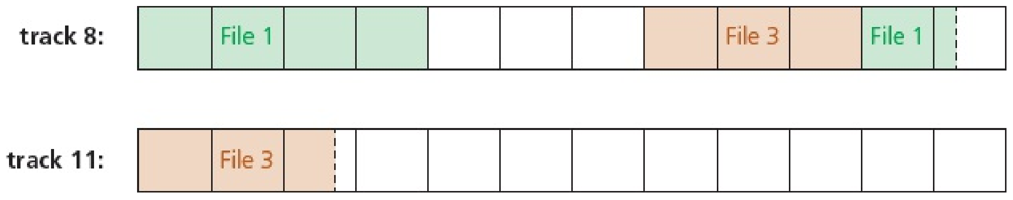


Figure 5.7

If this continues, the files just become more and more scattered throughout the disk surfaces. It is possible for sectors 4, 5 and 6 (on track 8) to eventually become used if the disk starts to fill up and it has to use up whatever space is available. A **disk defragmenter** will rearrange the blocks of data to store files in contiguous sectors wherever possible; however, if the disk drive is almost full, defragmentation may not work. Assuming we can carry out defragmentation, then track 8 now becomes:

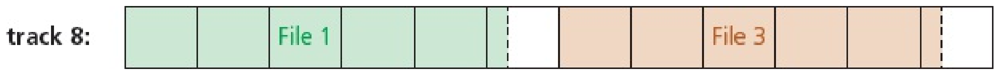


Figure 5.8

This allows for much faster data access and retrieval since the HDD now requires fewer read-write head movements to access and read files 1 and 3. Some defragmenters also carry out clean up operations. Data blocks can become damaged after several read/write operations (this is different to bad sectors). If this happens, they are flagged as 'unusable' and any subsequent write operation will avoid writing data to data blocks which have become affected.

Disk content analysis/repair software

The concept of disk repair software was discussed in the above section. **Disk content analysis software** is used to check disk drives for empty space and disk usage by reviewing files and file folders. This can lead to optimal use of disk space by the removal of unwanted files and downloads (such as the deletion of auto saving files, cookies, download files, and so on).

Disk compression and file compression

File compression is essential to save storage space and make it quicker to download/upload files and quicker to send files via email. It was discussed in Chapter 1.

Disk compression is much less common these days due to the vast size of HDDs (often more than 2 TB). The disk compression utility compresses data before writing it to hard disk (and decompresses it again when reading this data). It is a high priority utility and will essentially override all other operating system routines – this is essential because all applications need to have access to the HDD. It is important not to uninstall disk compression software since this would render any previously saved data to be unreadable.

Back-up software

While it is sensible to take manual back-ups using, for example, a memory stick or portable HDD, it is also good practice to use the operating system **back-up utility**. This utility will

- allow a schedule for backing up files to be made
- only carry out a back-up procedure if there have been any changes made to a file.

For total security, there should be three versions of a file:

1. The current (working) version stored on the internal HDD.
2. A locally backed up copy of the file (stored on a portable SSD, for example).
3. A remote back-up version stored well away from the computer (using cloud storage, for example).

Windows environment offers the following facilities using the back-up utility:

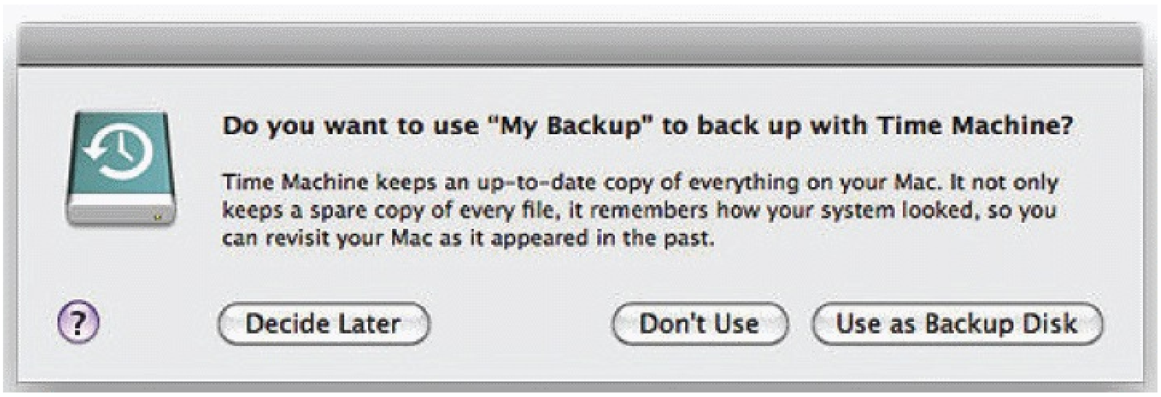
- The ability to restore data, files or the computer from the back-up (useful if there has been a problem and files have been lost and need to be recovered).
- The ability to create a restore point (this restores a computer to its state at some point in the past; this can be very useful if a very important file has been deleted and cannot be recovered by any of the other utilities).
- Options of where to save back-up files; this can be set up from the utility to ensure files are automatically backed up to a chosen device.

Windows uses *File History*, which takes snapshots of files and stores them on an external HDD at regular intervals. Over a period of time, *File History* builds up a vast library of past versions of files – this allows a user to choose which version of the file they want to use. *File History* defaults to backing up every hour and retains past versions of files forever unless the user changes the settings.

Mac OS offers the *Time Machine* back-up utility. This erases the contents of a selected drive and replaces them with the contents from the back-up. To use this facility it is necessary to have an external HDD or SSD (connected via USB port) and ensure that the *Time Machine* utility is installed and activated on the selected computer. *Time machine* will automatically

- back up every hour
- keep daily back-ups for the past month, and
- keep weekly back-ups for all the previous months.

Note that once the back-up HDD or SSD is almost full, the oldest back-ups are deleted and replaced with the newest back-up data. Figure 5.9 shows the *Time Machine* message:


Do you want to use "My Backup" to back up with Time Machine?

Time Machine keeps an up-to-date copy of everything on your Mac. It not only keeps a spare copy of every file, it remembers how your system looked, so you can revisit your Mac as it appeared in the past.

Figure 5.9 Screen shot of *Time Machine* message

5.1.4 Program libraries

Program libraries are used

- when software is under development and the programmer can utilise pre-written subroutines in their own programs, thus saving considerable development time
- to help a software developer who wishes to use dynamic link library (DLL) subroutines in their own program, so these subroutines must be available at run time.

When software routines are written (such as a sort routine), they are frequently saved in a program library for future use by other programmers. A program stored in a program library is known as a **library program**. We also have the term **library routines** to describe subroutines which could be used in another piece of software under development.

Suppose we are writing a game for children with animated graphics (of a friendly panda) using music routines and some scoreboard graphics.

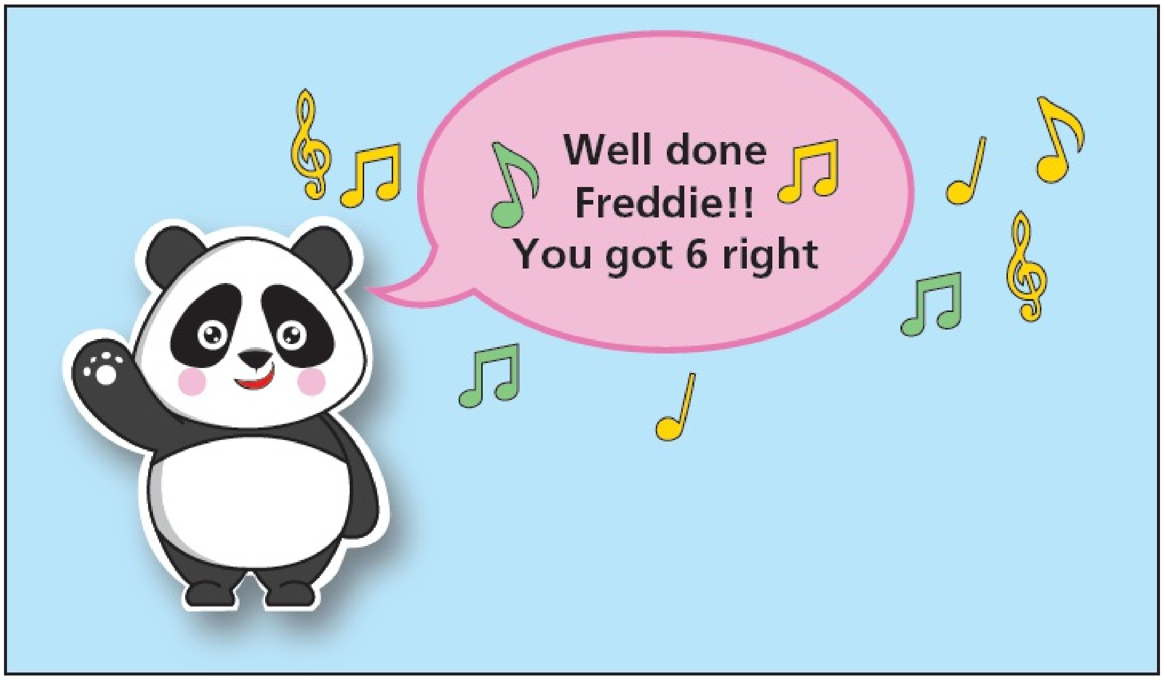


Figure 5.10

This game could be developed using existing routines from a library.

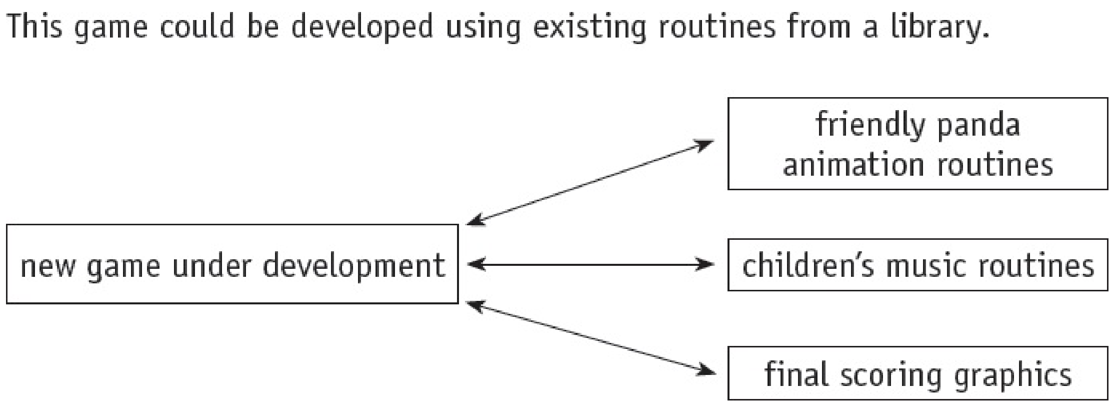

This game could be developed using existing routines from a library.
new game under development
friendly panda animation routines
children's music routines
final scoring graphics

Figure 5.11

Developing software in this way

- removes the need to rewrite the many routines every single time (thus saving considerable time and cost)
- leads to modular programming, which means several programmers can be working on the same piece of software at the same time
- allows continuity with other games that may form part of a whole range (in education, where there may be a whole suite of programs, for example)
- allows the maintenance of a 'corporate image' in all the software being developed by a particular company
- saves considerable development time having to test each routine, since the routines are all fully tested in other software and should be error-free.

All operating systems have two program libraries containing library programs and library routines: static and dynamic.

In static libraries, software being developed is linked to executable code in the library at the time of compilation. So the library routines would be embedded directly into the new program code.

In dynamic libraries, software being developed is not linked to the library routines until actual run time (these are known as **dynamic link library files** or **DLL**). These library routines would be stand-alone files only being accessed as required by the new program – the routines will be available to several applications at the same time.

When using DLL, since the library routines are not loaded into RAM until required, memory is saved, and software runs faster. For example, suppose we are writing new software which allows access to a printer as part of its specification. The main program will be developed and compiled. Once the object code is run, it will only access (and load up) the printer routine from DLL when required by the user of the program. The main program will only contain a link to the printer library routine and will not contain any of the actual printer routine coding in the main body. Table 5.2 summarises the pros and cons of using DLL files.

Pros of using DLL files	**Cons of using DLL files**
the executable code of the main program is much smaller since DLL files are only loaded into memory at run time	the executable code is not self-contained, therefore all DLL files need to be available at run time otherwise error messages (such as

	missing .dll error) will be generated and the software may even crash
it is possible to make changes to DLL files independently of the main program, consequently if any changes are made to the DLL files it will not be necessary to recompile the main program	any DLL linking software in the main program needs to be available at run time to allow links with DLL files to be made
DLL files can be made available to a number of applications at the same time	if any of the DLL files have been changed (either intentionally or through corruption) this could lead to the main program giving unexpected results or even crashing
all of the above save memory and also save execution time	malicious changes to DLL files could be due to the result of malware, thus presenting a risk to the main program following the linking process

Table 5.2 Pros and cons of using DLL files

ACTIVITY 5A

1 a) i) Explain why a computer needs an operating system.

ii) Name **two** management tasks carried out by the operating system.

b) A new program is to be written in a high level language. The developer has decided to use DLL files in the design of the new program.

i) Explain what is meant by a DLL file. How does this differ from a static library routine?

ii) Describe **two** potential drawbacks of using DLL files in the new program.

2 A company produces glossy geography magazines. Each magazine is produced using a network of computers where thousands of photographs and drawings need to be stored. The computers also have an external link to the internet.
Name, and describe the function of, **three** utility programs the company would use on all its computers.

3 A computer user has a number of important issues, listed below.
For each issue, name a utility which could help solve it. Give a reason for each choice.

a) The user wants to send a number of very large attachments by email, but the recipient cannot accept attachments greater than 20 MB.

b) The user has accidentally deleted files in the past. It is essential that this cannot happen in the future.

c) The user has had their computer for a number a years. The time to access and retrieve data from the hard disk drive is increasing.

d) Last week, the user clicked on a link in an email from a friend, since then the user's computer is running slowly, files are being lost, and they are receiving odd messages.

e) Some of the files on the user's HDD have corrupted and will not open and this is affecting the performance of the HDD.

5.2 Language translators

WHAT YOU SHOULD ALREADY KNOW

Try these two questions before you read the second part of this chapter:

1 a) Name **two** types of language translator.
  b) Identify a method, other than using a translator, of executing a program written in a high-level language.

2 Most modern language translators offer an Integrated Development Environment (IDE) for program development.
  a) Which IDE are you using?
  b) Describe **five** features offered by the IDE you use.
  c) Which feature do you find most useful? Why is it useful to you?

Key terms

Translator – the systems software used to translate a source program written in any language other than machine code.

Compiler – a computer program that translates a source program written in a high-level language to machine code or p-code, object code.

Interpreter – a computer program that analyses and executes a program written in a high-level language line by line.

Prettyprinting – the practice of displaying or printing well set out and formatted source code, making it easier to read and understand.

Integrated development environment (IDE) – a suite of programs used to write and test a computer program written in a high-level programming language.

Syntax error – an error in the grammar of a source program.

Logic error – an error in the logic of a program.

Debugging – the process of finding logic errors in a computer program by running or tracing the program.

Single stepping – the practice of running a program one line/instruction at a time.

Breakpoint – a deliberate pause in the execution of a program during testing so that the contents of variables, registers, and so on can be inspected to aid debugging.

Report window – a separate window in the run-time environment of the IDE that shows the contents of variables during the execution of a program.

5.2.1 Translation and execution of programs

Instructions in a program can only be executed when written in machine code and loaded into the main memory of a computer. Programming instructions written in any programming language other than machine code must be translated before they can be used. The systems software used to translate a source program written in any language other than machine code are **translators**. There are three types of translator available, each translator performs a different role.

Assemblers

Programs written in assembly language are translated into machine code by an assembler program. Assemblers either store the program directly in main memory, ready for execution, as it is translated, or they store the translated program on a storage medium to be used later. If stored for later use, then a loader program is also needed to load the stored translated program into main memory before it can be executed. The stored translated program can be executed many times without being re-translated.

Every different type of computer/chip has its own machine code and assembly language. For example, MASM is an assembler that is used for the X86 family of chips, while PIC and GENIE are used for microcontrollers. Assembly language programs are machine dependent; they are not portable from one type of computer/chip to another.

Here is a short sample PIC assembly program:

```
      movlw B’00000000’
      tris  PORTB
      movlw B’00000011’
      movwf PORTB
stop: goto  stop
```

EXTENSION ACTIVITY 5C

Find out what task this very short sample PIC assembly program is performing.

Assembly language programs are often written for tasks that need to be speedily executed, for example, parts of an operating system, central heating system or controlling a robot.

Compilers and interpreters

Programs written in a high-level language can be either translated into machine code by a **compiler** program, or directly executed line-by-line using an **interpreter** program.

Compilers usually store the translated program (object program) on a storage medium ready to be executed later. A loader program is needed to load the stored translated program into main memory before it can be executed. The stored translated program can be executed many times without being retranslated. The program will only need to be retranslated when changes are made to the source code.

With an interpreter, no translated program is generated in main memory or stored for later use. Every line in a program is interpreted then executed each time the program is run.

High-level language programs are machine independent, portable and can be run on any type of computer/chip, provided there is a compiler or interpreter available. For example, Java, Python and Visual Basic® (VB) are high-level languages often used for teaching programming.

EXTENSION ACTIVITY 5D

Find out about three more high-level programming languages that are being used today.

The similarities and differences between assemblers, compilers and interpreters are shown in Table 5.3.

	Assembler	**Compiler**	**Interpreter**
Source program written in	assembly language	high-level language	high-level language
Machine dependent	yes	no	no
Object program generated	yes, stored on disk or in main memory	yes, stored on disk or in main memory	no, instructions are executed under the control of the interpreter
Each line of the source program generates	one machine code instruction, one to one translation	many machine code instructions, instruction explosion	many machine code instructions, instruction explosion

Table 5.3 Similarities and differences between assemblers, compilers and interpreters

5.2.2 Pros and cons of compiling or interpreting a program

Both compilers and interpreters are used for programs written in high-level languages. Some **integrated development environments (IDEs)** have both available for programmers, since interpreters are most useful in the early stages of development and compilers produce a stand-alone program that can be executed many times without needing the compiler.

Table 5.4 shows the pros (in the blue cells) and cons (in the white cells) of compilers and interpreters.

Compiler	Interpreter
The end user only needs the executable code, therefore, the end user benefits as there is no need to purchase a compiler to translate the program before it is used.	The end user will need to purchase a compiler or an interpreter to translate the source code before it is used.
The developer keeps hold of the source code, so it cannot be altered or extended by the end user, therefore, the developer benefits as they can charge for upgrades and alterations.	The developer relinquishes control of the source code, making it more difficult to charge for upgrades and alterations. Since end users can view the source code, they could potentially use the developer's intellectual property.
Compiled programs take a shorter time to execute as translation has already been completed and the machine code generated may have been optimised by the compiler.	An interpreted program can take longer to execute than the same program when compiled, since each line of the source code needs to be translated before it is executed every time the program is run.
Compiled programs have no syntax or semantic errors.	Interpreted programs may still contain syntax or semantic errors if any part of the program has not been fully tested, these errors will need to be debugged.
The source program can be translated on one type of computer then executed on another type of computer.	Interpreted programs cannot be interpreted on one type of computer and run on another type of computer.
A compiler finds all errors in a program. One error detected can mean that the compiler finds other dependent errors later on in the program that will not be errors when the first error is corrected. Therefore, the number of errors found may be more than the actual number of errors.	**It is easier to develop and debug a program using an interpreter as errors can be corrected on each line and the program restarted from that place, enabling the programmer to easily learn from any errors.**

Untested programs with errors may cause the computer to crash.	**Untested programs should not be able to cause the computer to crash.**
The developer needs to write special routines in order to view partial results during development, making it more difficult to assess the quality of particular sections of code.	**Partial results can be viewed during development, enabling the developer to make informed decisions about a section of code, for example whether to continue, modify, or scrap and start again.**
End users do not have access to the source code and the run-time libraries, meaning they are unable to make modifications and are reliant on the developer for updates and alterations.	**If an interpreted program is purchased, end users have all the source code and the run-time libraries, enabling the program to be modified as required without further purchase.**

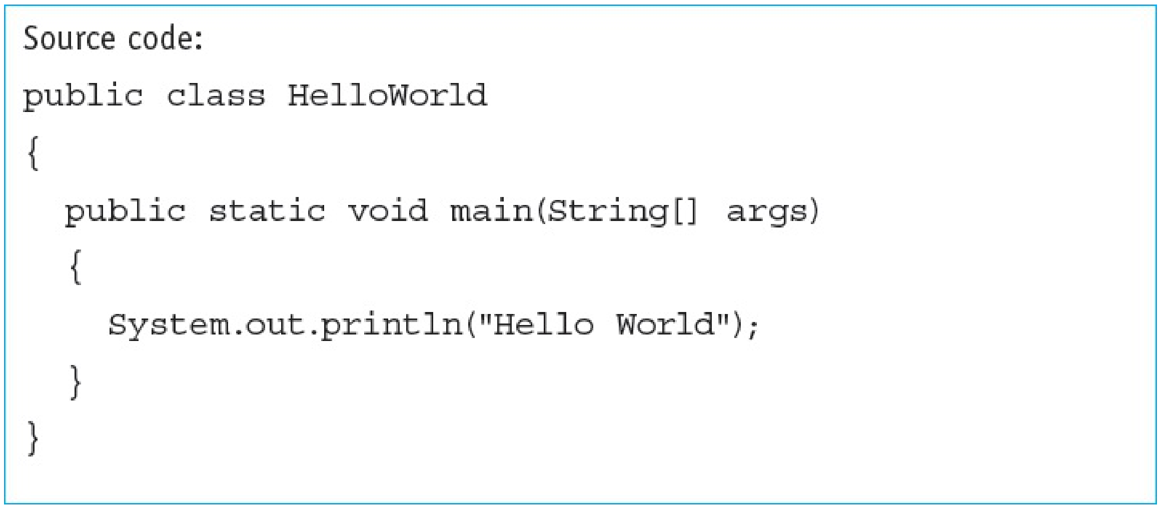
Table 5.4 Pros (blue cells) and cons (white cells) of compilers and interpreters.

5.2.3 Partial compiling and interpreting

In order to achieve shorter execution times, many high-level languages programs use a system that is partially compilation and partially interpretation. The source code is checked and translated by a compiler into object code. The compiled object code is a low-level machine independent code, called intermediate code, p-code or bytecode. To execute the program, the object code can be interpreted by an interpreter or compiled using a compiler.

For example, Java and Python programs can be translated by a compiler into a set of instructions for a virtual machine. These instructions, called bytecode, are then interpreted by an interpreter.

Below are examples of Java and Python intermediate code (bytecode):



Source code:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello World");
  }
}
```

Bytecode:

```
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object{
public HelloWorld();
  Code:
   0: aload _ 0
   1: invokespecial   #1; //Method java/lang/
Object."<init>":()V
   4: return
public static void main(java.lang.String[]);
  Code:
   0: getstatic        #2; //Field java/lang/System.
out:Ljava/io/PrintStream;
   3: ldc         #3; //String Hello World
   5: invokevirtual   #4; //Method java/io/PrintStream.
println:(Ljava/lang/String;)V
   8: return
```

Source code:

```
print ("Hello World")
```

Bytecode:

```
1   0 LOAD _ NAME        0 (print)
    2 LOAD _ CONST       0 ('Hello World')
    4 CALL _ FUNCTION    1 (1 positional, 0 keyword pair)
    6 RETURN _ VALUE
```

EXTENSION ACTIVITY 5E

Visual Basic also has an interpreter for bytecode. Find an example of bytecode for Visual Basic. See if you can find the bytecode for displaying ‘Hello World’ on the screen as in the Python example above.

5.2.4 Integrated development environment (IDE)

An integrated development environment (IDE) is used by programmers to aid the writing and development of programs. There are many different IDEs available; some just support one programming language, others can be used for several different programming languages. NetBeans®, PyCharm®, Visual Studio® and SharpDevelop are all IDEs currently in use.

EXTENSION ACTIVITY 5F

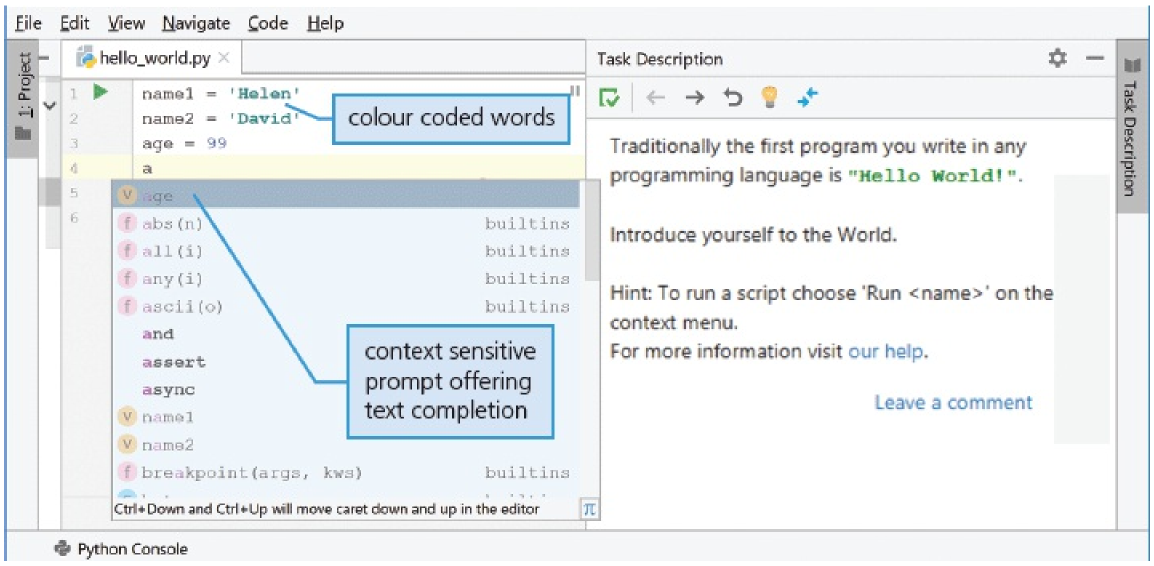
In small groups investigate different IDEs. See how many different features are available for your group’s IDE and identify which programming language(s) are supported. Compare the features of the IDE investigated by your group with the IDEs investigated by other groups in the class.

IDEs usually have

- a source code editor
- a compiler, an interpreter, or both
- a run-time environment with a debugger
- an auto-documenter.

Source code editor

A source code editor allows a program to be written and edited without the need to use a separate text editor. The use of an integrated source code editor speeds up the development process, as editing can be done without changing to a different piece of software each time the program needs correcting or adding to. Most source code editors colour code the words in the program and layout the program in a meaningful way (**prettyprinting**). Some source code editors also offer context sensitive prompts with text completion for variable names and reserved words, and provide dynamic syntax checking. Figures 5.12 and 5.13 show these features in the PyCharm source code editor.



```
name1 = 'Helen'
name2 = 'David'
age = 99
a
```

```
v age
f abs(n)                builtins
f all(i)                builtins
f any(i)                builtins
f ascii(o)              builtins
  and
  assert
  async
v name1
v name2
f breakpoint(args, kws) builtins
Ctrl+Down and Ctrl+Up will move caret down and up in the editor
```

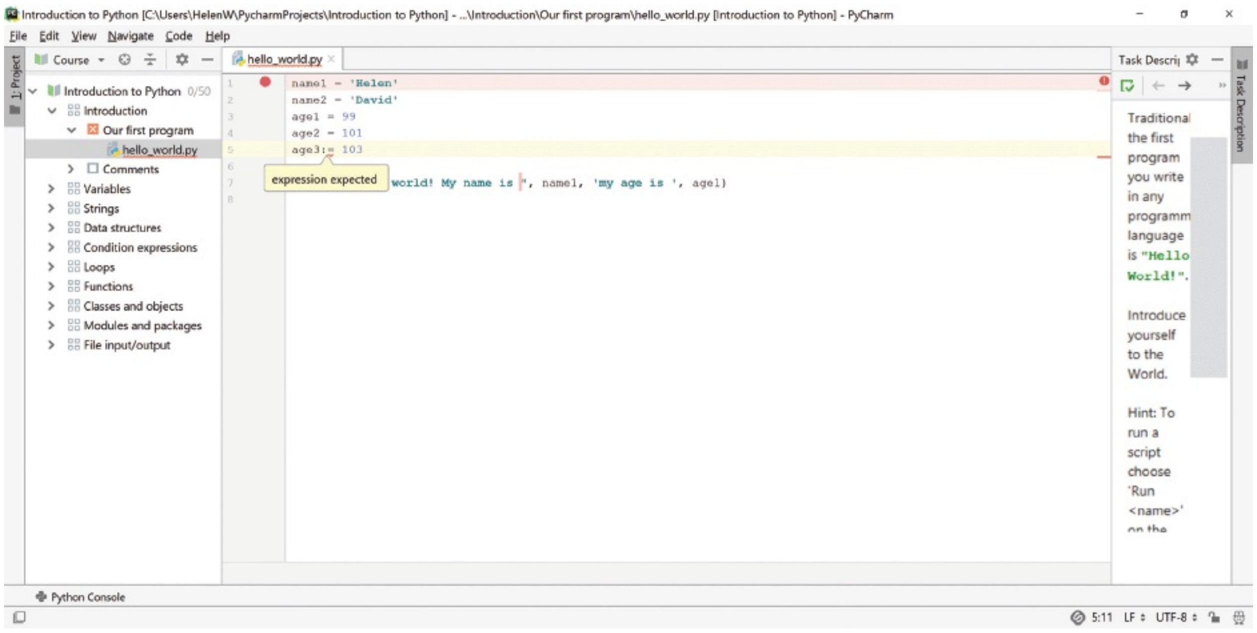
Traditionally the first program you write in any programming language is "Hello World!".

Introduce yourself to the World.

Hint: To run a script choose 'Run <name>' on the context menu.
For more information visit our help.

Leave a comment

Figure 5.12 PyCharm IDE showing source code editor



Introduction to Python [C:\Users\HelenW\PycharmProjects\Introduction to Python] - ...\Introduction\Our first program\hello_world.py [Introduction to Python] - PyCharm

```
name1 = 'Helen'
name2 = 'David'
age1 = 99
age2 = 101
age3:= 103

world! My name is ", name1, 'my age is ', age1)
```

expression expected

Introduction to Python 0/50
- Introduction
  - Our first program
    - hello_world.py
  - Comments
- Variables
- Strings
- Data structures
- Condition expressions
- Loops
- Functions
- Classes and objects
- Modules and packages
- File input/output

Traditional the first program you write in any programm language is "Hello World!".

Introduce yourself to the World.

Hint: To run a script choose 'Run <name>' on the

Figure 5.13 PyCharm IDE showing dynamic syntax checking

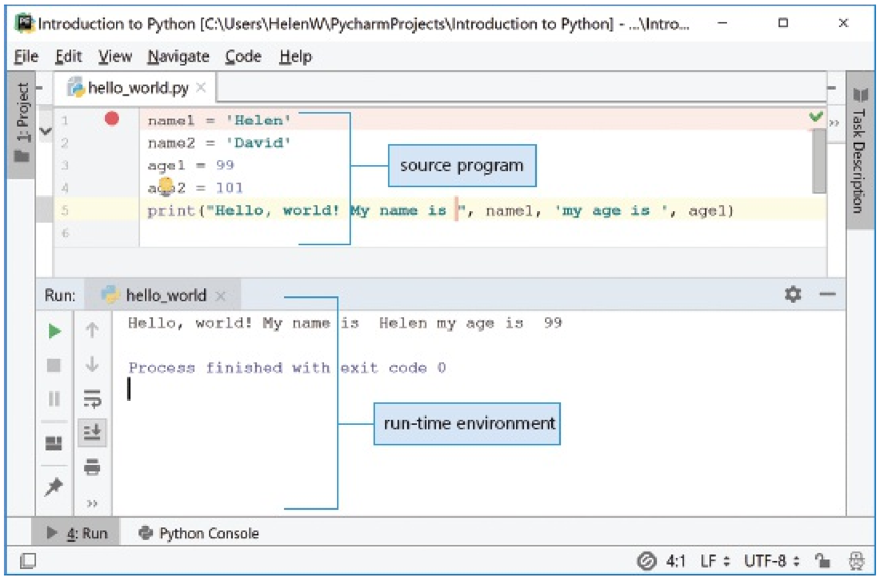
Here, string values are shown coloured green and integer values are shown coloured blue.

Dynamic syntax checking finds possible **syntax errors** as the program code is being typed in to the source code editor and alerts the programmer at the time, before the source code is interpreted. Many errors can therefore be found and corrected during program writing and editing before the program is run. **Logic errors** can only be found when the program is run.

For larger programs that have more than one code block, some code blocks can be collapsed to a single line in the editor allowing the programmer to just see the code blocks that are currently being developed.

Compilers and interpreters

Most IDEs usually provide a compiler and/or an interpreter to run the program. The interpreter is often used for developing the program and the compiler to produce the final version of the object code.


Introduction to Python [C:\Users\HelenW\PycharmProjects\Introduction to Python] - ...\Intro...
File Edit View Navigate Code Help
hello_world.py
source program
Run: hello_world
run-time environment
4: Run
Python Console
4:1 LF UTF-8

```
name1 = 'Helen'
name2 = 'David'
age1 = 99
age2 = 101
print("Hello, world! My name is ", name1, 'my age is ', age1)
```

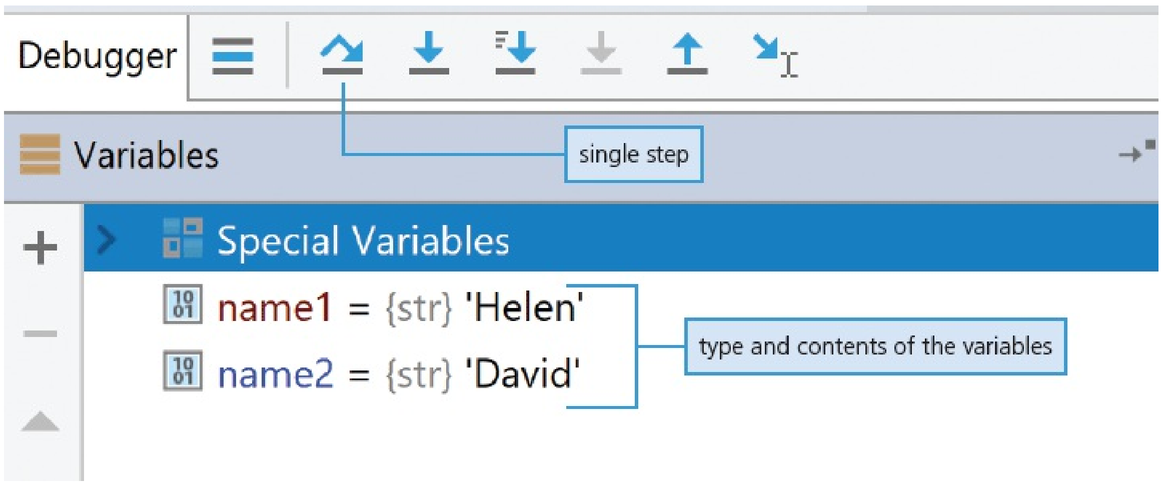
```
Hello, world! My name is  Helen my age is  99

Process finished with exit code 0
```

Figure 5.14 PyCharm IDE showing both program code and program run

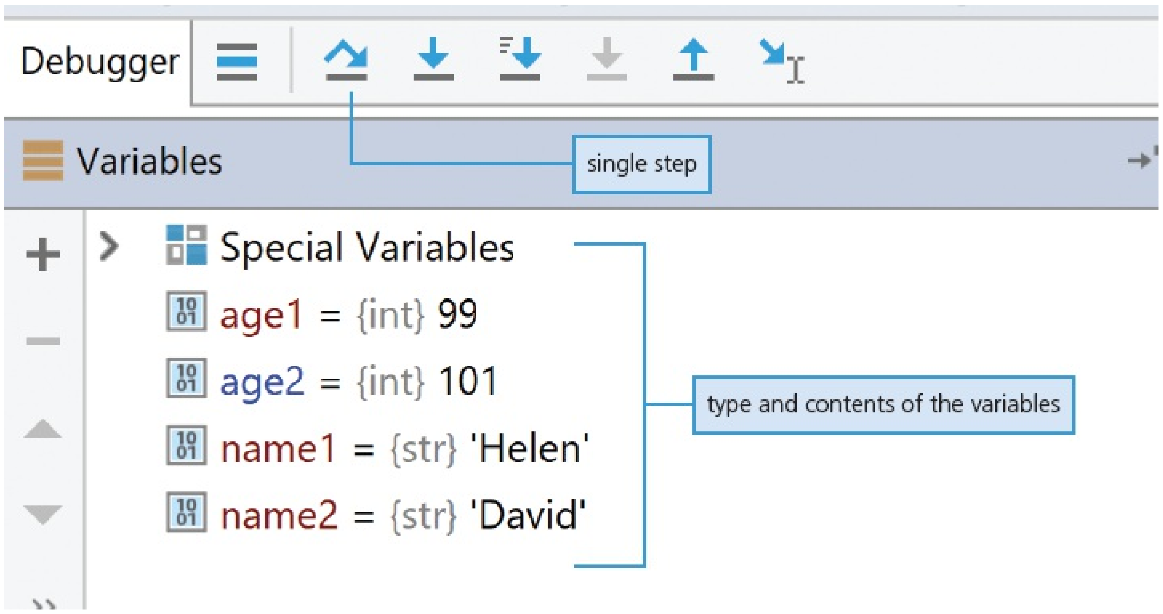
With PyCharm there can be more than one interpreter available for different versions of the Python language. The program results are shown using the run-time environment provided.

A run-time environment with a debugger

A debugger is a program that runs the program under development and aids the process of **debugging**. It allows the programmer to single step through the program a line at a time (**single stepping**) or to set a **breakpoint** to stop the execution of the program at a certain point in the source code. A **report window** then shows the contents of the variables and expressions evaluated at that point in the program. This allows the programmer to see if there are any logic errors in the program and check that the program works as intended.



```
name1 = {str} 'Helen'
name2 = {str} 'David'
```

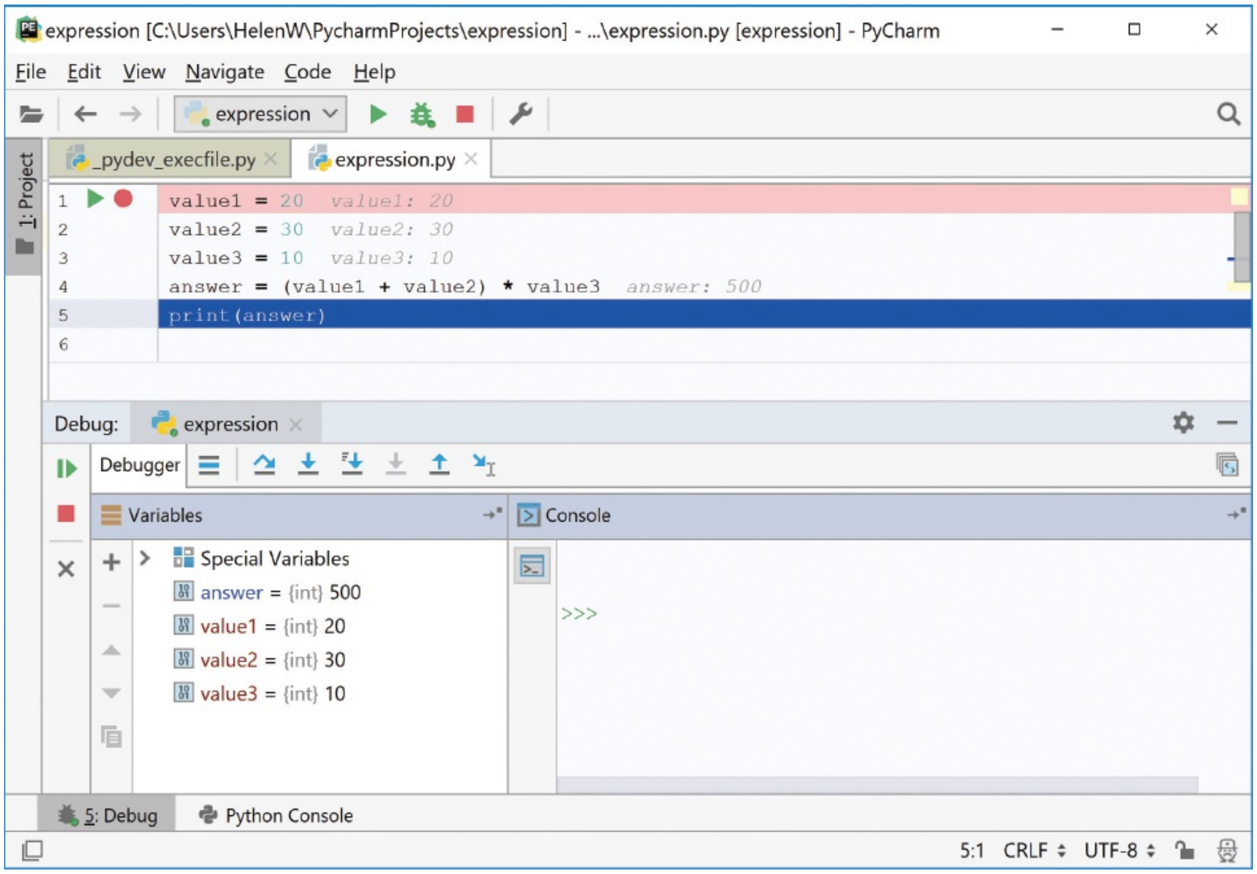


```
age1 = {int} 99
age2 = {int} 101
name1 = {str} 'Helen'
name2 = {str} 'David'
```

Figure 5.15 PyCharm IDE showing the report window after line 2 (page 155) and after line 4 (above)

Each variable used is shown in the report window together with the type and the contents of the variable at that point in the program. The top variable shown is the last one that was used.

Answers to calculations and other expressions can also be shown.



```
value1 = 20   value1: 20
value2 = 30   value2: 30
value3 = 10   value3: 10
answer = (value1 + value2) * value3   answer: 500
print(answer)
```

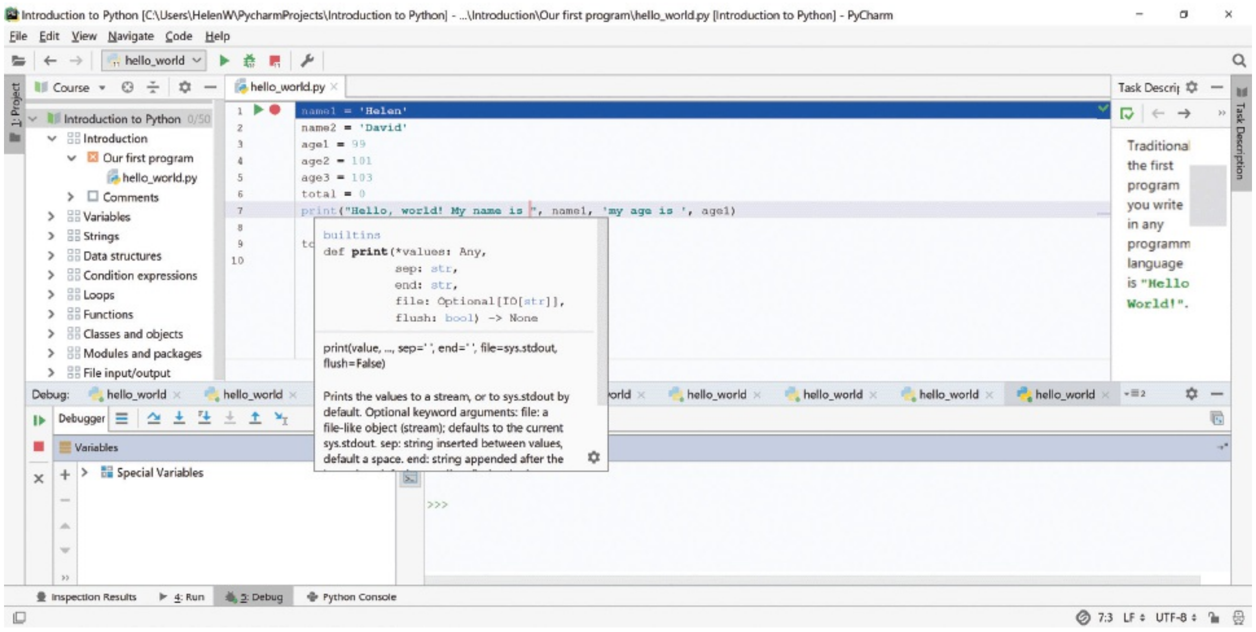
```
answer = {int} 500
value1 = {int} 20
value2 = {int} 30
value3 = {int} 10
```

```
>>>
```

Figure 5.16 PyCharm IDE showing the report window with the answer to an expression

Auto-documenter

Most IDEs usually provide an auto-documenter to explain the function and purpose of programming code.



Introduction to Python [C:\Users\HelenW\PycharmProjects\Introduction to Python] - ...\Introduction\Our first program\hello_world.py [Introduction to Python] - PyCharm

```
name1 = 'Helen'
name2 = 'David'
age1 = 99
age2 = 101
age3 = 103
total = 0
print("Hello, world! My name is ", name1, 'my age is ', age1)

to
```

```
builtins
def print(*values: Any,
          sep: str,
          end: str,
          file: Optional[IO[str]],
          flush: bool) -> None
```

print(value, ..., sep=' ', end=' ', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments: file: a file-like object (stream); defaults to the current sys.stdout. sep: string inserted between values, default a space. end: string appended after the

Traditional the first program you write in any programm language is "Hello World!".

Figure 5.17 PyCharm IDE showing the quick documentation window for print

ACTIVITY 5B

1 a) i) Describe the difference between a compiler and an assembler.

ii) Describe the difference between a compiler and an interpreter.

b) State **two** benefits **and** two drawbacks of using an interpreter.

2 A new program is to be written in a high-level language. The programmer has decided to use an IDE to develop the new program.

a) Explain what is meant by an IDE.

b) Describe **three** features of an IDE.

End of chapter questions

1 A programmer is writing a program that includes code from a program library.

a) Describe **two** benefits to the programmer of using one or more library routines. [4]

b) The programmer decides to use a Dynamic Link Library (DLL) file.

i) Describe **two** benefits of using DLL files. [4]

ii) State **one** drawback of using DLL files. [2]

Cambridge International AS & A Level Computer Science 9608 Paper 12 Q8 November 2016

2 a) The operating system contains code for performing various management tasks. The appropriate code is run when the user performs actions.
Copy the diagram below and connect each OS management task to the appropriate user action.

[3]

OS management task	action
main memory management	The user moves the mouse on the desktop
input/output management	The user closes the spreadsheet program
secondary storage management	The user selects the SAVE command to save their spreadsheet
human–computer interface management	The user selects the PRINT command to output their spreadsheet

b) A user has the following issues with the use of his PC.
State the utility software which should provide a solution.

i) The hard disk stores a large number of video files. The computer frequently runs out of storage space.

[1]

ii) The user is unable to find an important document. He thinks it was deleted in error some weeks ago. This must not happen again.

[1]

iii) The operating system reports 'bad sector' errors.

[1]

iv) There have been some unexplained images and advertisements appearing on the screen. The user suspects it is malware.

[1]

Cambridge International AS & A Level Computer Science 9608 Paper 11 Q6 June 2017

3 *File History* and *Time Machine* are examples of back-up utilities offered as part of two different operating systems.

a) Explain why it is important to back up files on a computer.

[2]

b) One of the features offered by both utilities is the possibility of 'turning back the internal

computer clock'.
Explain why this is an important feature and give **two** occasions when a user may wish to use this feature.

[4]

c) By using diagrams and written explanation, describe how *defragmentation software* works.

[4]

4 Assemblers, compilers and interpreters are all used to translate programs. Discuss the different roles played by each translator.

[6]

5 State **four** features of an IDE that are helpful when coding a program.

[4]