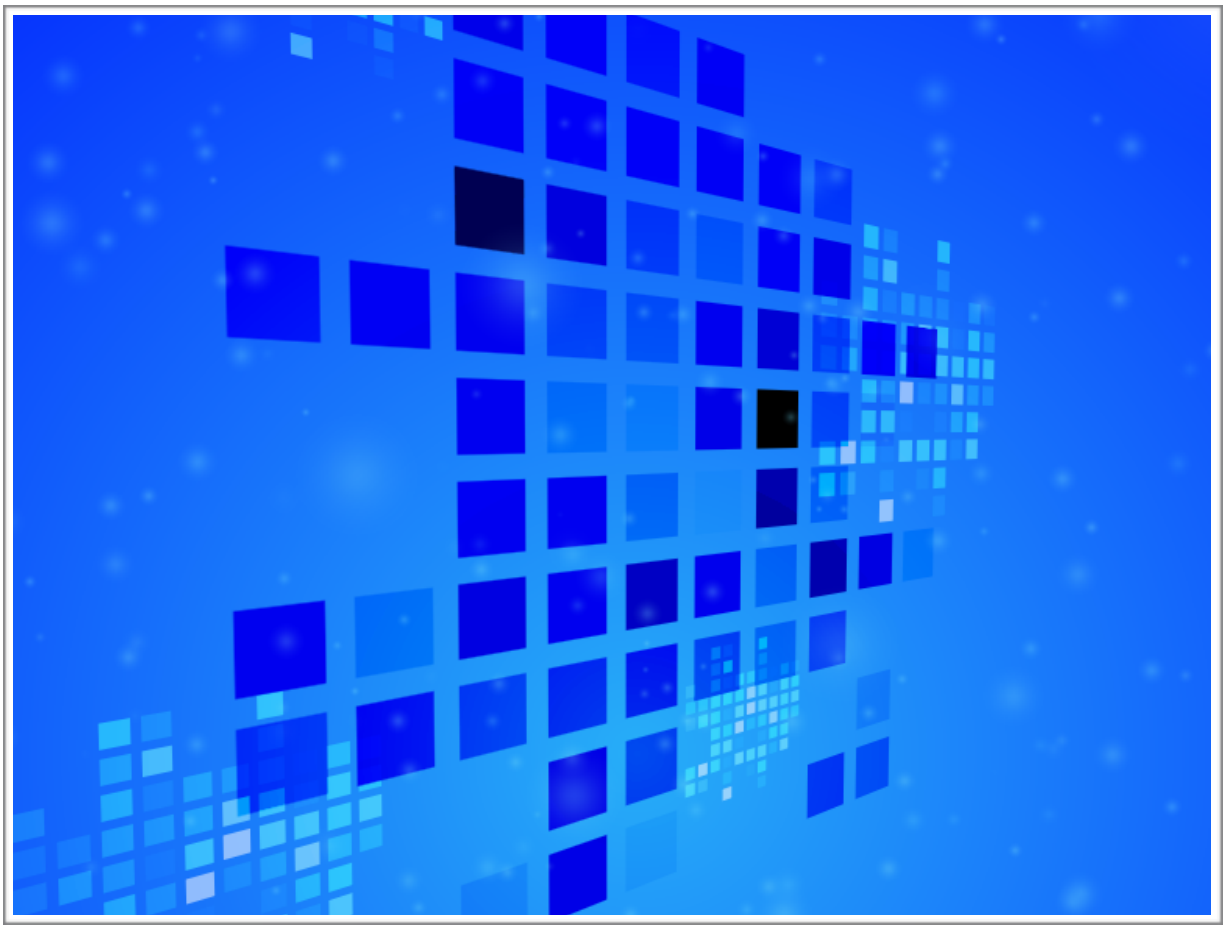
GCSE Computing Theory

For the OCR exam board



S A Longley

Chapter 1

Fundamentals of computer systems

Section 1 - Input, Processing & Output

Computer systems all have **input**, **processing** and **output**. When you think of a computer system you may picture a desktop computer or laptop, but it could easily be a mobile phone, games console or your DVD player. They are all computer systems that have inputs, processing and outputs.

Input

Computer systems allow input so we can interact with them. We can input data in all sorts of ways. A keyboard and mouse are often used to interact with desktop computers, but there are others ways too. Games often have special controllers to input actions into a console. Artists may use special touch sensitive tablets to draw their pictures. You can use a microphone to record a song. Webcams can take pictures. All these are different ways of getting data into a computer. DVD players respond to the buttons or a remote control.

Computer systems often do not understand the type of data coming into them, so often the first thing that is done is the data is converted into a form the computer system can understand. Computers only understand **binary,** which is a series of 0's and 1's (see chapter 4 for more on binary). All input must be converted into binary which is the computer's internal representation of the data.

Processing

Processing is the manipulation of the data by the computer system's **processor**. This can be anything from converting the data into a more useful form or performing a calculation or transformation on the data. So this can be anything from adding numbers together in a spreadsheet or changing colours in an image to checking to see if you have shot the bad guy in a game or pausing your movie on your DVD player.

During processing, the computer manipulates the internal representation of the data. So the computer is processing the binary data inside the computer.

Output

Computer systems output in a variety of ways so we can see what they are doing. This is commonly on a display, whether it is seeing a letter appear on a screen after a key is pressed, your car steers in a computer game or a sound is heard to indicate a task has completed. Some outputs are the finished product from a series of processes, like printing a document, while others help to give the user feedback as to what is going on.

When a computer system needs to output, it has to convert the computer representation of the data into a form the user can understand.

Storage

Sometimes the computer system will need to store the data it has, for use on a later date. This is commonly a hard disc for internal storage or possibly a memory card or a writable DVD. This can also mean transmitting to another computer over a network. When data is stored or transmitted, it is in the computer's own internal representation (**binary**).

Feedback

The user of a computer system may be monitoring what has been outputted as a result of their input and this may effect future inputs. This happens all the time when playing games, but also happens when using other software like word processors or art packages. When using a word processor the user types in using a keyboard. The computer converts the keystroke into binary for processing and storage. It also uses the data to display the keystroke on the screen of the word processor. The user may get extra output from the computer such as underlining spelling mistakes and the user would then correct them (hopefully anyway).

Section 2 - Different types of computer system

When you think of a computer system a desktop PC or laptop may be the image that immediately springs to mind. However the microchip has enabled computer systems to spread further just these tools.

laptop

Games consoles are computer systems. Many consoles have a complex array of input devices, from the game controller with its buttons and joysticks to guitar shaped controllers and even drum controllers. We now also have cameras that can detect you and the movements you make, so that you do not have to touch a device to input data.

The console takes these inputs and processes them in the game and shows you on screen the output of the input. So you may press the X button on a game pad, the console processes this to decide this means shoot. It calculates whether you have shot the baddie or missed and shows you the result on the screen. Some controllers have output in them as well as inputs. The Wii remote for example has a mini speaker and a vibration unit. These are outputs.

games console

Mobile phones are computer systems, even the non-smart type mobile phones. Smart phones are basically mini computers. Many smart phones have a touch sensitive screen instead of a keyboard. This makes the screen an input and an output device.

smartphone

Burglar alarms are a much simpler example of a computer system. They have sensors as input devices. These are commonly passive infrared sensors that detect people moving. You can have pressure plates and magnetic sensors on doors and windows. Theses sensors are constantly sending information to the processor when they are triggered. The processor decides to trigger

the alarm if the sensor is triggered and if the alarm has been set. The processor can also show which sensor triggered the alarm. These are also examples of embedded systems.

An embedded system is a computer system that has the programs saved into a chip. These programs are fixed and so cannot be changed. Embedded systems usually have a limited function that often needs to be performed quickly. A wireless router is an example.

wireless router

Traffic lights are another computer system that you come across regularly. Traffic lights have induction coils embedded into the road on the approach to the lights. When a car moves over these coils the processor knows that there is a car waiting to go through the lights. The processor will make the traffic light change more often if all the roads are busy or keep one road on green for longer, if the side road is less busy.

Reliability

It is important for computer systems to be reliable. Some systems need to be more reliable than others however.

When we say something is reliable, we mean that it does what we expect. Many computers are complex pieces of hardware running complex software. Occasionally something will go wrong, such as a program may crash or a piece of hardware such as a hard drive may fail.

Some computer systems could put people in danger if they fail or even lead to death. If your games console fails and a game crashes, whilst it is frustrating it does not compare with a computer system on an air liner. If a computer system on an aeroplane fails, it could lead to the plane crashing and people being killed.

If traffic lights fail, it could cause serious traffic congestion which could effect the emergency services for example. If traffic lights fail regularly at one spot then it could affect people travelling

on that route and using a more reliable one. This could in turn effect shops as people may avoid that road.

There are different ways of measuring reliability. One way is to measure how often a system is down for in a given time. So for example if a Superstore's computer systems were run for 100 hours and in that time, computer failure resulted in the computers not working for 3 hours then we could say that the system has a 97% reliability. The Superstore company would have to decide whether this was acceptable or not.

If you need to have a computer system that is close to being 100% reliable then extra steps would need to be taken to make sure that things do not fail.

One step is to have backup systems. This can be as simple as making a backup of your photographs and music from your computer to an external drive. It can mean having a whole duplicate computer system, so that if one fails then the back up will immediately take over. This duplication of resources will mean more expense.

It is recommended to have a disaster recovery plan. The greater the importance of the computer system, the more detailed a plan for recovering it if the worst happens.

Software is often the most complex part of any computer system and is very difficult to write error free. This is why you often need to update your software as errors become known. Some computer systems, failure is not an option. In computer systems such as medical life support, aircraft fly by wire software and software on board satellites, failure can either result in death or in the loss of a very expensive piece of equipment. In these cases the system often runs three different versions of the software created by three different software teams. When the programs run and make a decision, the decisions are compared and the most common result is the one that gets executed. It is thought that it is unlikely that three different teams of software engineers will make the same errors in the same places. So if one of the three programs makes a mistake, then the other two should make the correct decision and it will be that outcome that happens.

Section 3 - Computers & the Law Introduction

While computers are undoubtably a useful tool, there will always be those that will misuse them and use them as a tool to break the law. There are people called **hackers** who use computers to access and gain control of other people's computer systems. This is often done remotely over the Internet and they may use a variety of software to help them do this along with some **social engineering**. To give an example of how this is done I will outline a high profile hack that took place in 2011.

HBGary Hack

In 2010-11 the organisation ***Wikileaks*** was in the news for leaking secret US military communications which showed that they may have acted illegally in some of their actions in Iraq. As a result of the pressure organisations like ***MasterCard*** and ***Amazon*** that provided payment methods and Internet hosting respectively were forced to withdraw their services to Wikileaks. In response the hacker group known as ***Anonymous*** tried to disrupt the websites of MasterCard and Amazon.

HBGary, an Internet security firm we hired to track down members of Anonymous. Anonymous responded by hacking HBGary's computer systems, leading to an embarrassing situation where a firm advising about Internet security gets hacked itself.

HBGary had a **content management system** (CMS) to run their website. It had some rather big security holes allowing the hackers to get access to the usernames, email addresses and passwords of the people who could edit their website. While the passwords were encrypted they were done so badly and were easily unscrambled.

If the users of the CMS at HBGary had used passwords that were unique then it wouldn't have been such a big problem. However the CEO of the company used the same password for his Twitter account, email and a whole lot of other places.

They used these passwords to hack the Gmail account of someone at HBGary. He controlled all the Gmail accounts of all the company's employees and found out the email addresses of other key members of the company. They found out some other passwords for a secure website HBGary ran. They needed a setting on that computer changing to allow access. This is where the social engineering came in. They emailed a HBGary technician, from the email account of a senior HBGary's email account and asked him to change the setting. They provided him with the password, so the technician thought everything was authorised. He made the changes, allowing the hackers from Anonymous to take control of HBGary's website, taking it down and replacing it with their own message.

If you want to read more detail about how this was done, please read http://arstechnica.com/tech-policy/news/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack.ars/1.

Mark Zuckerberg, the founder of Facebook, started by hacking the different student websites of Harvard University to create a website that allowed you to rate the looks of students on campus.

Other hacks that have been done have resulted in more criminal activity such as stealing credit card details or blackmail.

There are many laws that can be applied to computer systems but there are two laws which deal with computer systems directly.

Computer Misuse Act

In the 1980's a number of people were caught hacking into other people's computer systems but the law at the time was not able to deal with these new crimes.

The Computer Misuse Act 1990 was designed to protect the integrity of computer systems by deterring the activities of hackers. It created three new offences:

- Unauthorised access to computer programs or data
- Unauthorised access with a further criminal intent (known as the 'ulterior intent' offence)
- Unauthorised modification of computer material.

So what does all this mean? Well to start with it makes hacking illegal. Hacking is accessing a computer either physically or remotely, without the user's permission. So you are not allowed to access a person's computer or work area or look at the data stored there without the permission of the person whose data it is. This means hacking into some's email account is illegal. This could lead to a fine of up to £5000 or 6 months in prison.

If the hacker then makes a change to any data on the computer system or they use their access to commit a further crime, like fraud or blackmail, they could face up to 5 years in prison. This is in addition to the sentence given for that additional crime.

Some people write computer viruses. These pieces of software can travel from computer to computer and often change data on an infected machine. This means that virus writers can be sent to prison for up to 5 years.

Government sponsored hacking

There have been an increasing number of accounts that governments are involved in hacking the computer systems of foreign powers. One example could be found in the **Stuxnet virus**. This virus was reported from Iran where it infected computers running centrifuges that were used in their nuclear programme. They made the centrifuges not work properly and so stopped Iran from refining uranium for use in nuclear weapons. People analysing the virus have commented on its complexity and the fact it targets a particular computer system, lead them to believe that only a government sponsored team could have created it and Israel and the USA have been named as probable creators, though no-one has officially claimed responsibility.

Data Protection Act

Many companies collect personal data which they use for their business. Some of this data is vital to the day to day running of these companies, such as being able to insure a car or house, or deliver goods. However this data can be valuable in other ways. If a car insurance company knew the names and addresses of who bought cars and when, then they could target them for mail shots at the right time for insurance renewal. The chances of getting a response would be higher.

It was decided that the company who collects the data must look after it and can only pass it on with the owner's permission. The Data Protection Act has eight statements about how a company must look after personal data they store. Failing to uphold these statements could lead to criminal prosecution.

- Personal data must be obtained and processed fairly and lawfully.
- Personal data must be held only for the purposes which the data user has declared. Personal data must not be used for purposes other than those which have been declared.
- The personal data kept for the declared purpose must be relevant, adequate, and not excessive in relation to that purpose.
- Personal data should be accurate and kept up to date (where appropriate).
- Personal data must not be kept longer than is necessary for the purpose for which it is held.
- An individual is entitled to find out whether and what data is held on him or her and, where appropriate, to have such data corrected or erased.
- Appropriate measures must be taken to prevent unauthorised access, or modification of personal data.
- Data should not transferred outside the UK without adequate protection.

So this means that companies and others who collect personal data as part of the work they do must keep that data secure and cannot pass it on to anyone without consent. This is often an opt out tick box on forms that you need to fill in. Organisations can make a lot of money selling personal data and lists of possible customers can help a company target their marketing more effectively, making them more money.

Organisations cannot collect data which is not relevant to their work. So a car sales person cannot collect data about a person's medical records.

Copyright & Patents Act

With digital data, making copies has never been easier. Even though it is easy however, does not make it legal. Many things are covered under the copyright law such as music, film, photographs and software. Making copies of such items and sharing them with friends is often against copyright law and as such can be prosecuted.

When buying a software pack it usually comes with a license that covers its use. This licence will inform you how many computers you are allowed to install the software onto. Single user licences allow the installation of software on one and sometimes two computers used by one user. A site licence will allow an organisation to install software on all the computers it owns on a site for however many users it has. A licence may come with a key to unlock the software or even a dongle, a physical device that must be plugged into the computer in order for the software to work.

Some files like films, music and audiobooks may come with Digital Rights Management (DRM). This is software that tries to make sure only licensed users can use the file. This however often locks you in to a particular platform or device. Books bought on the iBooks store cannot be transferred to another ebook reader for example.

Parts of software and hardware can be patented to protect the inventions of people and companies. These inventions may sometimes be used by other companies in their software and hardware under licence.

ETHICS

Laws are there to prevent and punish illegal behaviour but there are some situations that are not covered by the law. These are often ethical issues.

Imagine a company could buy a computer system that would do the work of 50 people and could work 24 hours a day, 365 days a year. This could save the company a great deal of money and so would make sense commercially. However just making 50 people unemployed would be unethical. The company has an obligation to look after its work force. This may mean retraining them so they can work in areas that computer systems are not so well suited for.

Employers need to make sure they respect health and safety regulations for their computer users. This may mean paying for eye tests and providing equipment to make it easier for people with disabilities to work effectively.

Due to the international nature of much of today's business, it may mean working for companies and governments who have different political rights for their people than we enjoy. The Chinese government censor what websites its people can look at and even censor and delete messages placed on social networking sites. Google has been criticised in the past for working with the government in helping them censor searches made in China. This is helping the government suppress freedom of speech and political expression that would be frowned upon in western nations.

Other examples of the international nature of some ethical issues arose in the Arab Spring revolutions that swept many middle eastern nations in 2011. Some governments controlled access to the Internet and mobile telecommunications, even switching them off to prevent demonstrators from organising themselves. Facebook and Twitter were used in those countries to organise demonstrations and videos taken by citizens on their mobile phones enabled reporting to get out of the country even though international reporters inside the country were banned.

There are a number of professional bodies that people can belong to. These often have a code of conduct the professional signs up to. One such organisation is the British Computer Society. It has a code of conduct to make sure its members work to a standard set of ethics.

Section 4 - Professional Standards

The need for standards

Most computer programs are very complex and require teams of people working together to complete the task. This requires some organisation if the program is going to work and perform the tasks required of it. There are many different standards that have been created for the design, development and testing of software.

When creating a new computer system a number of stages need to be gone through in order to implement a solution effectively.

- Analysis of the current system. Understand what the current system does (if there is one), including what works well and what doesn't work well.
- Designing a new solution. Creating a new computer system to replace the old one, hopefully taking note of the things that don't work well with the old system so they can be avoided.
- Implementation the new system according to the design.
- Testing the solution. This may have a number of stages. Alpha testing is usually done by the programmers to test against their designs. Beta testing often involves the end users so they can test it doing the types of jobs they do in real life situations.
- Installing and training. This means putting in the new system and make sure the users have been trained sufficiently to do their job.

DEVELOPMENT STANDARDS

The programmers need to adhere to standards when they work in teams. Programs are usually broken down into chunks that can be implemented independently. In this way many programmers can work on a different parts of the program. For example if a new word processor is being written, one programmer may work on the code to format the text, while another programmer may work on making tables work and yet another may implement a spell checker.

These separate tasks need to come together to work as a whole. The programmers need to make sure the interaction of the various parts work well together. This will include having standardised names for variables and functions. The code should be well commented so that another programmer can make sense of the work that has been done.

They need to agree on how the separate modules interact with one another and what data needs to be transferred between modules.

This may include sticking to recognised standards in the industry. Programmers who are working on a video game may agree to use a particular image and sound format. They may agree a particular programming language.

The programmers need to make sure that their code will work as part of the whole and need to make sure they are professional in their work. If one programmer does a sloppy job then the program could fail or perform differently to what was intended. They need to follow agreed testing and documenting procedures.

There are a number of other types of standards that a professional may have to consider in their work.

Open Standards

These are standards that have been agreed by a body of professionals, often from a number of different companies in the industry. These standards are available for anyone to look at and use. These standards are often vital to make a core technology work. Standards that govern the World Wide Web and the Internet are open, so that any company can develop software such as web browsers, that can use the Internet to send and receive data. HTML 5 is the language of modern web browsers and is an open standard.

HTML 5 - an open standard

Proprietary standards

These are standards that are created by a company and kept secure. These standards may be licensed to another company for a fee. These standards may give a company an advantage over its competitors. Microsoft Windows is a proprietary standard, as many companies need to work with it, even though they have no access to some parts of the code that makes it up.

De-facto Standards

Sometimes a proprietary standard becomes so popular that people have to use it because it is so common. Microsoft Office applications such as Word, Excel and PowerPoint are so widely used that they become a standard for those types of files and a common way to transfer data. Many people will create Word documents and email them to people outside the company with a knowledge that either most people will be able to open them without a problem. Competitors had to reverse engineer how Word saved its files as the file format was still a secret held by Microsoft.

Another de-facto standard is the QWERTY keyboard. The QWERTY keyboard was designed to slow down early typists who used mechanical typewriters. Early keyboard layouts would jam, so commonly used letters were spaced out more, slowing the typist and preventing jams. Even though the reasons for the keyboard layout are part of the history of mechanical typewriters and the modern computer does not suffer from the same problem, we are still stuck with the same keyboard layout.

Industry Standards

These are standards that are set to make sure a technology works and items produced meet a particular standard. These industry standards may be backed by law. All electrical devices released in the EU must meet certain health and safety standards to make sure components don't over heat or create electric shocks or interfere with other electrical devices. The USB specification and WiFi protocols are industry standards so that all manufacturers can create products that work together.

Section 5 - Computers and the Environment

Recycling

The electronic circuit boards and components used in making modern computer devices often contain valuable materials. Much of the wiring is made of copper, which is a good electrical conductor. Small amounts of gold may be used instead, which is an even better conductor. There are also other rare metals such as tantalum. When computer systems such as PCs, games consoles and mobile phones come to the end of their usefulness they can often be thrown away. This is extremely wasteful.

unused mobile phones

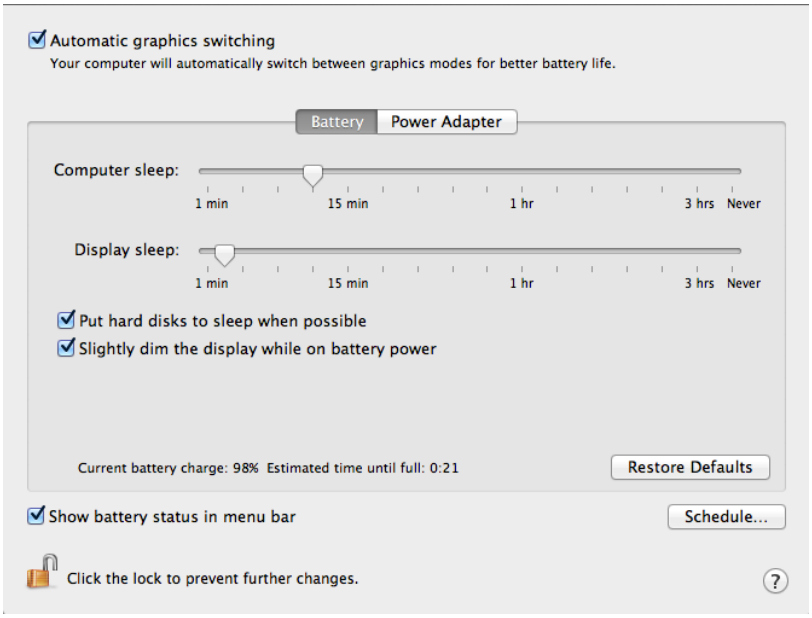
Laws have been passed in many countries to ensure that computer systems can be recycled easily or their life extended. Many businesses and organisations run powerful computers that when they become too slow to do the work they were purchased for, are often useful for schools and charities as well as in developing countries. There are companies that will even buy you old mobile phone and refurbish it before selling it to developing countries. In developing countries mobile phones can often be used in areas where it would not be economical to have fixed line telephones, and so there is big demand.

Manufacturers also are being forced to make sure that the electronics they create can be more easily recycled to get the precious metals and other materials back so they can be used to make the next generation of gadgets.

Energy Efficiency

We use computer systems so much in todays lifestyle that they are now making up a significant role in our usage of electricity. The electronics in your house hold could contribute

around 6% of your electricity bill. Modern computer systems are designed to save energy when not in use. There are energy saving settings in most modern operating systems that allow you to configure your computer so that it can save energy when you are not using it. There are a number of things that can be set including:



energy saving settings on a mac

- Dimming your screen after a period of inactivity.
- Turning the display of after a longer period of inactivity.
- Powering down the hard drive when not in use.
- Sending the computer into sleep or hibernate mode. The computer is still on, and can often be woken by a simple key press, but in this mode most of the systems in a computer are powered down, so it uses less power.

In many businesses there can often be hundreds of computers and monitors. Settings in the computer can arrange for it to switch off completely at a particular time. So when the workers go home the computers automatically turn themselves off.

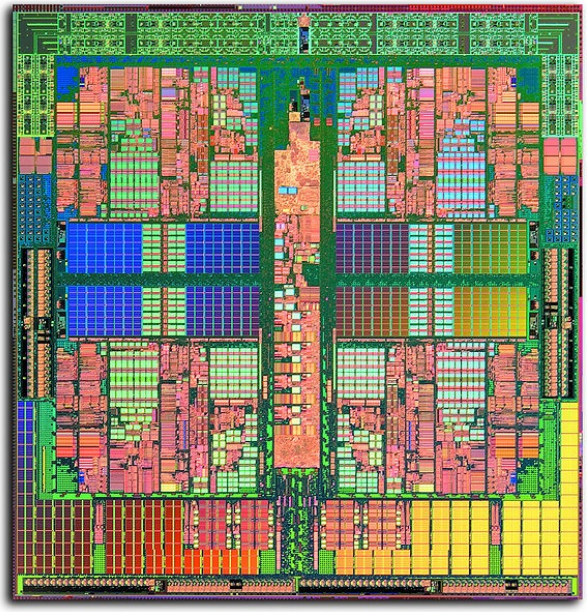
Laptops tend to use less power than desktop computers and tablets often use less power than laptops. This is often because the computers are less powerful. The need by customers for mobile computing with a longer lasting battery has forced computer chip manufacturers to develop new low powered processors that draw less power and so conserve the battery for longer.

Chapter 2

Computer Hardware

Section 1 - The Central Processing Unit

The Central Processing Unit (**CPU**) is the brain of the computer. It is the part of the computer that carries out the instructions of the computer system.



AMD Quad-core Opteron

The CPU has understands a number of instructions. The CPU has two parts. The ***Arithmetic Logic Unit*** which performs arithmetic and logical operations such as addition, subtraction and comparing values. The ***Control Unit*** gets instructions from memory (**RAM**), decodes them and executes the instructions, using the ALU when needed. Putting it simply, the CPU fetches instructions and then executes them. There are three basic types of instructions.

- Get some data from RAM and perform some arithmetic or logic on it such as add them together, multiply them together or compare them (such as decide which number is the largest).
- Move data from one part of memory to another.
- Make a decision and fetch a different instruction from memory based on that decision.

FACTORS WHICH GOVERN THE SPEED OF A CPU

Clock Speed

CPU's all have a clock speed. This is one measure of how fast they are. Clock speeds are measured in cycles per second (hertz or Hz). Todays CPUs are very fast and clock speeds are measured in million cycles per second (mega-hertz or MHz) or billion cycles per second (giga-hertz or GHz). CPU's need a certain number of clock cycles to complete each instruction. The faster the clock speed the higher the number of instructions that can be completed in one second.

Number of Cores

CPU manufacturers such as Intel and AMD in the 1990's and first part of the 00's were competing to make their chips faster by increasing the clock speed. There was a limit to this however, as it became increasingly difficult to continue to get faster. A different strategy is now being used. Having a CPU with more than one core. This is like embedding two processors in one chip.

In single core CPU's only one program can run on them at once. Computers often seem that they are running multiple programs at once, but in reality they are switching from one program to another rapidly, so they appear to be running multiple programs at once.

Multiple cores enable the CPU to run a different task on each core. So a 2 core processor can run two tasks at once, while a quad core processor can run four tasks.

While it is possible to write a program that runs across multiple cores, this requires some planning and special programming techniques to make this happen. Most of the time multi-core CPUs run one program on each core.

Cache

When a CPU fetches an instruction from RAM this may take a number of cycles to perform. The faster RAM can help speed up this process, but faster RAM is expensive to make. In order to keep the cost of the computer down many manufacturers using caching to speed up the computer.

Without caching the CPU may get some data (or an instruction) from RAM, execute an instruction and then send the data back to RAM. What if they next instruction needed that data? Well the CPU would have to go back to RAM and fetch it again. There are also some memory addresses that are commonly used and need to be accessed frequently.

A cache is a small amount of fast RAM located either near to the CPU or actually inside it. It acts as a buffer between the CPU and RAM. When the CPU carries out an instruction on some data it stores the result in the cache as well as sending it back to RAM. When it carries out the next instruction and sees that it needs the result of the previous instruction, it searches the cache for it and then uses that value, instead of getting it from RAM. This is faster than getting it from RAM. If the data needed is not in the cache then the CPU has to go to RAM to get it. As long as the time it takes to search the cache for the data is less than the time to retrieve the data from RAM then the cache will result in a faster CPU.

The larger the cache size, the more data it can store locally and so needs to get data direct from RAM less. CPU's with larger cache sizes are quicker than those with less.

SUMMARY

- So a CPU is like the brain of a computer.
- It is where the programs run.
- The CPU fetches and executes instructions one at a time.
- The speed of a CPU can be measured using its clock speed in megahertz or gigahertz.
- The faster the clock speed the more instructions it can carry out in one second.
- Processors can have more than one core, each of which can run a program, making the computer run faster.
- Cache is a type of fast RAM located near to or in the CPU. The CPU looks in the cache first for data it needs before looking in RAM. This speeds up the processor.

Section 2 - Binary Logic

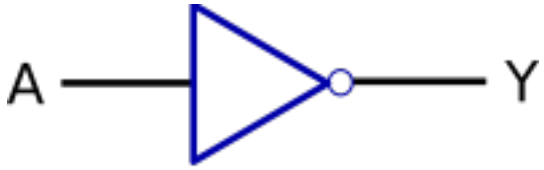
Just On and Off

For all the complexity of a computer system, at their heart is something really simple. The CPU, and the memory are just a series of switches. These switches have two settings, on or off. There are billions of switches that make up a modern computer, and all the data and programs that run on them must be stored on a series of switches that are either on or off. The CPU when it processes instructions, really just turns different switches on and off. It is when the settings of these switches are turned into something we can understand, like a sound or pixels on a screen, that we can interpret what the computer has done.

In order for the computer system to be able to make decisions about which switches to turn on or off, there are a series of circuits called gates. These take a signal which can either be on or off, and then alter it in some way. These gates are called logic gates. By combining them in different ways the computer system can make decisions. Its using logic gates that enable the computer system to work.

THE NOT GATE

This is the simplest of the logic gates. The symbol for a logic gate is shown below.

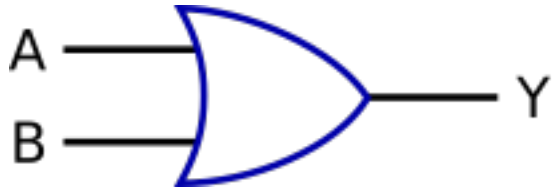


The NOT gate takes a signal A and inverts it. This means that if A is ON then Y will be OFF. Similarly if A is OFF then Y will be on. This can be shown in a table known as a **truth table**.

INPUT A	OUTPUT Y
1	0
0	1

THE OR GATE

The OR gate takes two inputs. Each of these inputs can be ON or OFF. Diagram for an OR gate is shown below.



The OR gate is so called because if input A **or** Input B are on then output Y will be on too. This can be shown in the truth table.

INPUT A	INPUT B	OUTPUT Y
1	1	1
1	0	1
0	1	1
0	0	0

Notice that even if A and B are on then the output Y is on.

The AND Gate

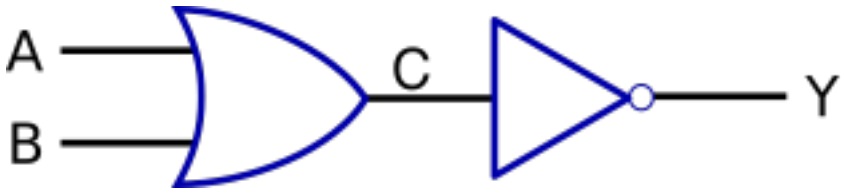
The AND gate has two inputs like the OR gate. It gets its name from the fact that both inputs A and B need to be on for the output Y to be on. The diagram for an AND gate is shown below.

The truth table for an AND gate is:

INPUT A	INPUT B	OUTPUT Y
1	1	1
1	0	0
0	1	0
0	0	0

Combining Gates

In a computer chip these logic gates can be combined to make more complex gates, to make more complex decisions. Take a look at the example below.



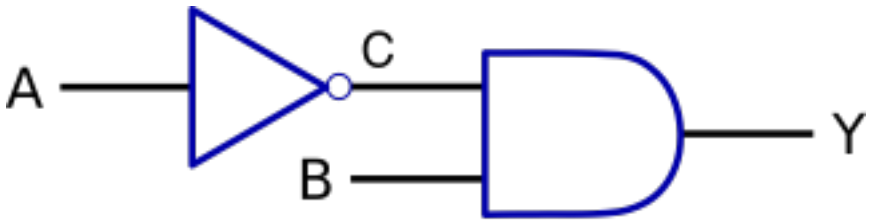
It is an OR gate that flows into a NOT gate. If A or B are on then C will be on. If C is on then the NOT gate will invert that switch and turn it off.

Writing a truth table is often a good idea to see all the different combinations and allows us to work out what is going on more easily.

INPUT A	INPUT B	INPUT C	OUTPUT Y
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

So we can see that as long as there is an input in A or B the final result will be that Y is off. Only when A and B are both off will the output Y be on.

Lets have a look at another example.



Here we have a NOT gate moving into one of the inputs of an AND gate. So lets look at our truth table for this.

INPUT A	INPUT B	INPUT C	OUTPUT Y
1	1	0	0
1	0	1	0
0	1	1	1
0	0	1	0

Here we can see that output Y is only on when input A is off and input B is on.

By combining different gates in more complex ways we can get more complex decisions being made from a variety of inputs.

You will be asked in the exam to interpret logic diagrams and be able to fill in a truth table for it.

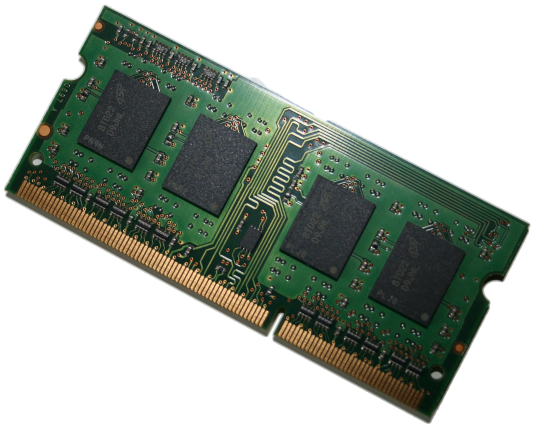
SUMMARY

The microchips that make up computer systems are made of billions of switches that can be either on or off. These switches can be connected together using logic gates. The arrangement of logic gates enables the computer system to make decisions.

Section 3 - Memory

Remember, Remember

Computers need to remember a lot of data. This includes all the programs that are currently running, all the data associated with those programs, along with data to do with the computer's current settings, user input and output. All this data is stored in the computer's memory. There are two types of computer memory, **RAM** and **ROM**.



computer RAM

RAM

RAM stands for **Random Access Memory**. It is called random access memory because the data stored in it can be accessed in any order. Data stored on tape or CD needs to read in a predetermined order. This type of memory is used to store the computer programs and their data. This type of memory is said to be **volatile**. This means it needs power to work. When the computer system is switched off the RAM is wiped clean and any data not saved is lost. This type of memory is constantly be written to and read. It can be changing all the time as the computer system runs the programs.

RAM is used to store the programs and data while the computer system is in use.

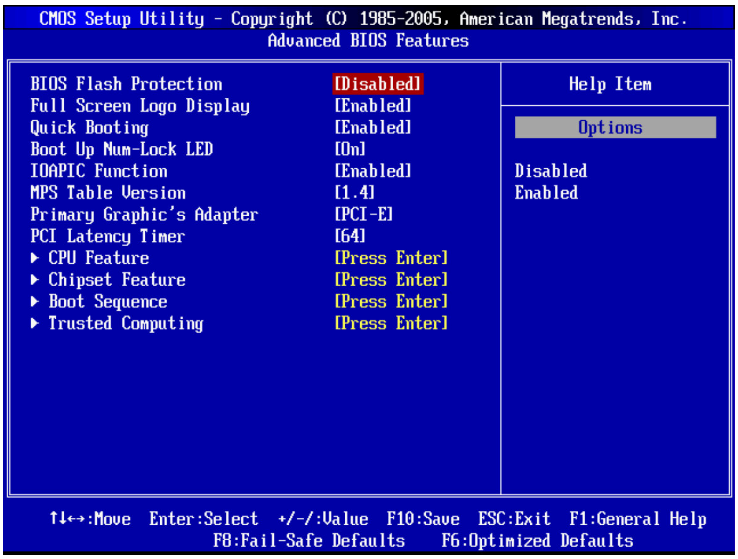
The more RAM your computer system has the more programs that can be running at once, and the more data your computer system can manipulate at once. Some data files can be enormous. It is possible to get some image files that are very large. Image editing software may need to have more than one of these images in memory in order to operate. The more RAM a computer has the faster it can operate as it does not need to read and write from the hard disc as much, which is a much slower operation.

ROM

ROM stands for **Read Only Memory**. This type of memory is non-volatile. The contents of the memory are not erased when the computer is switched off. Some ROM chips have their data hardwired into their circuitry. This means that the contents of these ROM chips cannot be altered. There are other flash ROM chips that can be erased and rewritten, but again the important point is that the data is not erased when the power is switched off.

When a computer is turned on it needs to have some basic instructions loaded, such as how to connect to basic input and output systems and how to load an operating system. ROM chips are used to store these basic settings and instructions so that they can be loaded when the computer system is first switched on.

The ***BIOS*** or Basic Input/Output system is a ROM chip on the motherboard of a PC. This stores the basic settings the CPU needs to connect to RAM, the different drives such as the CD drive or hard disc drive, the keyboard and monitor. When a PC first boots up showing the boot screen, you are seeing the output from the BIOS as the CPU sets up the systems needed to be able to load the operating system.



BIOS settings on a ROM chip

VIRTUAL MEMORY

Modern operating systems have a way of using the hard disc to extend the amount of RAM available to the computer. This virtual memory enables computer systems to act like they have more RAM than they have by using the hard disc.

Reading and writing from the hard disc is much slower than using RAM, so one way to increase the speed of a computer system is to have more RAM and rely less on virtual memory.

Cache Memory

This type of RAM is usually made from faster more expensive chips than the computer systems main memory and is located either in the CPU or close to it. It is used to store data or instructions from commonly used parts of RAM. When a CPU needs to get something from RAM it checks to see if it is in the cache memory first. If it is then it uses this data. Often this is quicker than fetching the data from RAM.

The larger the size of the cache the more data it can store but the longer it takes for the CPU to search it. Because of this cache memory tends to be much smaller than RAM. As long as the cache can be searched in a shorter time than fetching the data from RAM, then it will speed up the computer.

Flash Memory

Flash memory or solid state memory is a type of ROM chip that can be electronically erased and rewritten. It has been used for removable storage for cameras, MP3 players and USB sticks. These chips have become smaller and cheaper over the years with sizes up to 32 gigabytes available. Some flash memory cards are quicker to read and write than others. The fast cards are often used in professional cameras and video cameras. Flash memory does have a limit to how many times the data can be erased. This is getting very large now however leading to some computer systems using solid state flash drives instead of a hard disc.

These drives can enable the computer system to boot up faster allowing almost instant on. They are more resilient that the spinning discs inside a hard drive. This has lead to computer systems breaking away from the familiar desktop PC or laptop into tablets and smartphones. These types of drive also need less power to operate than a hard disc and so the batteries last longer on mobile devices. Some call this the *post-PC era* of computing as it is creating newer types of computers that are small light weight and touch screen as opposed to a keyboard & mouse or trackpad.

iPad has solid state memory

Section 4 - Input & Output Devices

Communicating with computer systems

Input and output devices allow us to communicate with and control computer systems and enable them to communicate with us in ways we can understand. There are a variety of different input and output devices, each suited to a particular job.

COMMON INPUT DEVICES

Input devices take data in a form we can understand and turn it into a binary form the computer understands.

Keyboard

There are many input devices which we see and use a great deal. The humble keyboard and mouse are two such examples. The keyboard is probably the most common type of input device on a computer system. The keyboard is inspired by mechanical type writers, and still adopts the QWERTY layout of keys. Pressing a key on a keyboard to input some data. Sometimes the key press creates a letter on the screen, while a combination of keystrokes can cause a more complex action. In a word processor you can turn bold on and off using a simple keystroke, saving and printing can be done through similar keystroke shortcuts. The keyboard is quite a versatile input device.

Mouse & other pointing device

The mouse is used to control an onscreen pointer. This along with buttons on the mouse enable certain actions to be performed, from launching an application to dragging a file to the trash can. Most mice have two buttons. Clicking one button (left click in most cases) selects items

on the screen. The secondary click is usually used to get a short cut list of options in a menu. This saves the user from moving to the menu bar so often.

Laptops often have a trackpad to control the pointer. This is a touch sensitive pad along with buttons. Apple laptops have done away with separate buttons, the whole pad being one big button that recognises how many fingers are touching it, so it can use gestures to control it.

Many devices now-a-days support multi-touch. This is especially the case on smart phones. This allows the user to use multiple fingers to create gestures. Pinch to zoom and two fingers to scroll are common multitouch gestures.

Another pointing device is a tracker ball. This is a cup with a ball inside it. By moving the ball around a sensor underneath the ball senses the movement and translates this into movement of the pointer on screen. These devices are useful for those who have limited movement.

Microphones

Microphones are increasingly common in computer devices. They are often used to record sound but can also be used to control the computer system through voice recognition software. The electrical signal generated by a microphone is converted into a binary signal. It is this signal that the computer processes.

microphone

Cameras

Digital cameras are often separate devices but again are increasingly common in computer systems such as mobile phones, laptops and games consoles. The camera takes light and focuses it onto a light sensitive sensor using a lens. The sensor is made up of tiny elements which turn the brightness and colour of light hitting it into a binary signal. These tiny picture elements (**pixels**) build up a mosaic of tiny squares that make up the image.

Often the images that are taken become photographs or video. Video can be used to have online video conferences, so that people can have face to face meetings even when they can be in different countries.

Video cameras such as **X-BOX Kinect** and **PlayStation Move** are being used to control the games consoles, providing a new way of interacting with computer systems.

playstation camera

Common Output Devices

Output devices convert the binary data stored in the computer into a form we can understand.

Visual Output

Monitors, screens or displays are the main output device of a computer system. They allow the user to see a visual response to their actions. Many computer systems have a graphical user interface. This show images like buttons, scroll bars and the like to give the user feedback as to what is going on.

Some displays also combine with an input device. Smart phones and tablets have a touch sensitive screen that can act both as an input and an output device.

Sound Output

Speakers and headphones are output devices. As well as play sound when music and videos are being played, sound is also used in feedback. The sound can be used to give an error beep when the user performs an incorrect action.

Vibration

Some game controllers have a vibration module, as do mobile phones. These are used to give the user feedback or alert them to a call or message.

Printers

Another common output device is a printer for hard copies of a document. There are many different types of printers but the most common ones are inkjet and laser printers. Some printers are very large and capable of printing onto large sheets of paper, large enough to wall paper walls.

large format printer

There are now printers that are capable of printing in three dimensions. Taking 3D models from CAD software the printers create physical objects by printing thin layers of plastic or other material, layer by layer.

Laser cutters can be used to cut out or etch materials from card to wood and plastic. The cut out pieces can often be assembled into a 3D item such as a box.

laser cutter

Section 5 - Storage

Why we need to store?

Computers need computer programs to run and data to manipulate. When the computer system is working, these are stored in the memory or RAM. When a computer system is turned off, all the programs and data stored in RAM are lost. If you had to type in a computer program every time you wanted to use it, it would be a very slow and tedious task. Some modern computer programs can be millions of lines of code long. Entering a document over and over again would also be a waste of time.

Computers need a way of storing data so that it can be loaded into RAM quickly when the computer system is turned on again. We also need to be able to transport data from one computer to another.

Computer systems have different ways of storing data. These ways each have their advantages and disadvantages and can hold different amounts of data.

MAGNETIC DISCS

Most computer systems have a hard disc drive. This is used to store the operating system, programs and data and is usually located inside the computer system itself. The hard disc is a platter of metal that can be magnetised. The disc spins rapidly and arm can move across the disc to read or write data. In some hard discs there may be more than one platter to increase the storage capacity.

The read write heads never touch the spinning discs, which often spin at 7200 times per minute. If they did then they can damage the disc. This can happen when a laptop is dropped. If the disc is damaged then often the data stored on it is lost.

The advantage of hard discs is that their capacity can be very large. It is common to have a hard disc of 500 gigabytes and not uncommon to have larger ones. These drives are also quite fast at reading and writing data.

An open hard disc drive with three platters

OPTICAL DISCS

These drives use lasers to read from an optical disc such as a CD, DVD or a Blu-ray disc. The disc is removable and so can be used in a number of computers. Many optical discs are read only. Once the data has been encoded on the disc, it cannot be altered. It is often used to store the installation program for software. So a DVD disc is used to install or copy the important files onto a computer's hard drive. Once this is done the DVD is not needed, unless you need to install the software again. Optical discs can be used to load data the program needs, such as when playing a game or an interactive encyclopaedia.

a Compact Disc

Optical discs are much slower to read than hard discs. Some optical discs are also writeable. They come in different types.

- Writeable - these discs can only be written to once, but read many times.
- Re-writable - these discs can be written to, erased and re-written to a number of times. There is a limit to the number of times they can be erased, and tend to be more expensive.

There are different types of optical disc, each with different storage capacities.

Disc type	Storage
CD	~700Mb
DVD single layer	4.2Gb
DVD dual layer	8.4Gb
Blu-ray single layer	25Gb
Blu-ray dual layer	50Gb

Optical writeable discs are often used to backup data. Photographers will often back up their work to DVD once they have finished working on them. This frees up space on the hard drive, but making the files relatively accessible.

Solid State

These are erasable ROM chips that can be wiped and written to electronically, but when the power is off they retain the data stored on them. These discs were initially used for removable storage for portable devices such as digital cameras. The removable discs have a limit to their capacity due to their size. They do have a limit to how many times they can be erased and re-written to, but advances in technology make the latest drives comparable with hard discs in terms of life span.

various sized flash memory cards

Smart phones and tablet computers use solid state memory as they need less power and are more resistant to shock when compared with hard discs. This is leading to changes in what computer systems people want to use. Newer laptops like the **Macbook Air** are using solid state drives to make the battery last longer in addition to making the laptop slimmer. This seems to have started a trend with the so called **Ultra books** following a similar design.

Chapter 3

Software

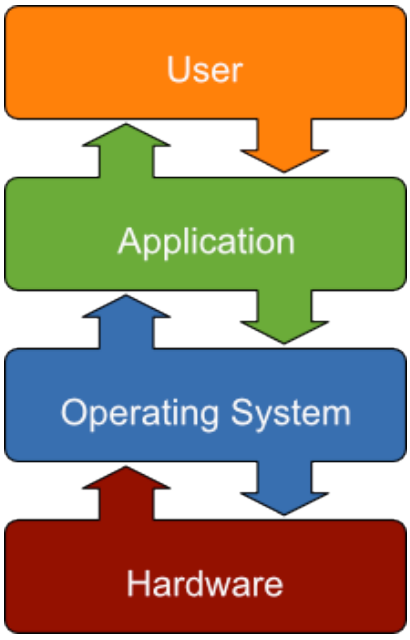
Section 1 - Operating Systems

What is an Operating System

Most modern computers, from Main Frame computers to desktops to tablets to smart phones have operating systems. What are they though and what do they do?

Hardware management

The Operating System (OS) allows the programs that run on the computer to interact with the different hardware in a consistent way. When you save a document to a disc, you don't need to worry about whether it is a hard disc from one manufacturer or a solid state drive from another. The way the data is written to these discs is different, but the OS hides these differences from the user and the programmer who wrote your application.



The OS enables your computing device to communicate with the different hardware that can connect to your computer, from disc drives and printers to monitors and network connections. When you need to use these different devices the OS provides a consistent way to use them.

When you get a new piece of hardware you will get a driver that enables the OS to communicate with it. So when you when you want to print a document, the OS loads the printer driver for the printer you want to use. It provides a similar set of options no matter which printer you use. There may be some options though that are specific to each printer.

Memory management

The OS manages how memory is allocated to applications so they do not clash. When you start a new program or load a file the OS assigns each its own memory space so they do not interfere with each other. The OS is also responsible for enabling the use of cache memory as a way of speeding up RAM access.

Storage management

The operating system deals with how storage devices are formatted and the way in which the files are stored and retrieved.

Hard discs need to be formatted so they can be divided up into sectors. Each has an address which enables the OS to save a file to a part of the disc so it can be retrieved again.

User interface

The OS is also responsible for providing the interface for the user to operate the computer. This is often a graphical user interface in modern devices such as smart phones and laptops. Each OS provides a way for the user to interact with the computer to perform tasks, open programs, create documents etc. This interface includes everything we see on the screen, from the windows, buttons, scrollbars etc, to the way we interact with it, either through a mouse pointer on a desktop or through touch gestures like pinch to zoom on an iPad.

There are different types of user interface, each with their own advantages and disadvantages.

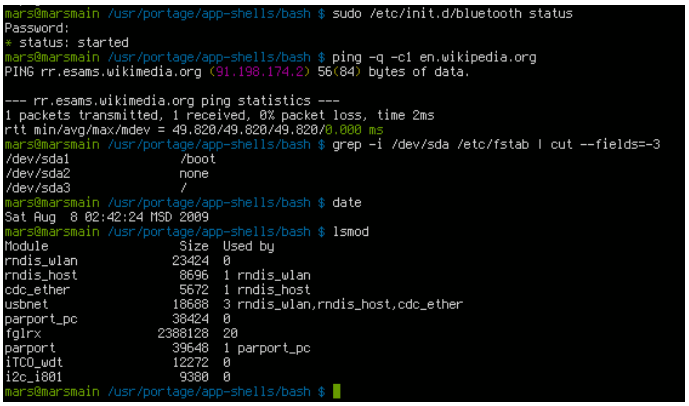
Command-Line Interface

This type of interface used to be the only interface computers could use. Now-a-days it is used mainly by experts and computer technicians to perform tasks.

It comprises of a screen with a prompt where commands are entered. As there is little on screen help, this is quite difficult for novices to use as you need to know the commands you can enter to get the computer to do something.

Entering **ls** for example with list all the files and folders in the current directory.

This type of interface doesn't use up much in the way of computer resources because the screen only contains text. It can be run on a low powered computer.



command line interface

Menu Driven Interface

A menu driven interface is more user-friendly and shows on the screen the options available. These can usually be navigated using the arrow keys on a keyboard or a remote control. Selecting an option either performs a command or takes you to a sub-menu where further options can be seen. This is useful for beginners who can see what the options are, but cannot be configured for expert users who may want to get to some of the more embedded menu items more quickly.

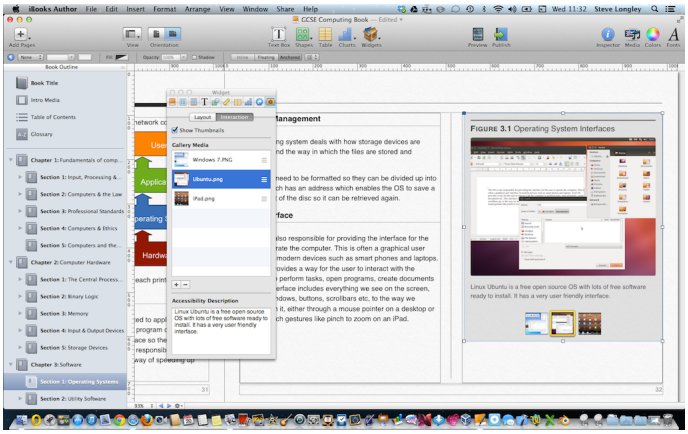
Menu driven interfaces can be seen on BIOS setups and digital TVs.

Sky EPG

Graphical User Interface (GUI)

This type of interface is one you are most used to seeing. This type of user interface is common on modern desktop and laptop computers. Tablets and smartphones may have a touch sensitive version of this interface. This is very graphics intense, giving feedback to the users

actions. GUIs are often have programs working in windows, so they can run more than one program at once. They use pointers to select items. The have menus and icons to represent actions the user can perform. For this reason they are often called ***WIMP*** interfaces (windows, icons, menus and pointers). GUI interfaces however need a powerful computer as they require a great deal of processing to display.



a Graphical User Interface

SECURITY

Security is an important issue with computers, especially those that are attached to the Internet. The operating system is responsible for making sure that only programs that are supposed to run, do so and that unauthorised access is not allowed. Operating Systems are very complex and often security holes can be found that enable hackers to get access to vulnerable computers. For this reason you should always make sure that your computer OS is up to date.

Often users of a computer system will need to have a username and password that they need to use to login to the computer or to make any changes like installing new software. This is done to try and reduce the likelihood of malicious software being installed remotely.

Section 2 - Utility Software

What is Utility Software?

There are some programs that come with the OS or that can be bought separately, that can help keep the computer running smoothly. There are a number of different types of utility software.

SECURITY UTILITIES

Keeping your computer secure is really important, especially as many people use their computers to bank and shop online as well as store private information. Computers on the Internet are especially at risk.

Anti-Virus

Viruses and Worms are malicious programs that copy themselves and spread from one computer to another. As well as spreading they often damage data or even steal passwords without the user's knowledge. Anti-virus software has a database of known viruses and every time a file or program is opened the anti-virus software scans it for any known virus signature. If it finds any then it warns the user so that it can be removed.

New viruses come out all the time so it is important that anti-virus software is kept up to date. Some viruses that have been discovered, have been out in the wild for two years before the anti-virus companies spotted them. This means that even though you have anti-virus software, it may not always be enough.

Firewall

When you connect to a **network** such as the Internet, there is data leaving your computer and data returning. A firewall is software (or sometimes hardware) that checks data travelling along the network connection and blocks it if is not allowed. Firewalls can be used to block access to websites. You may have encountered this in schools. Firewalls however can also block the data from malicious sources, such as hackers trying to control your computer.

Firewalls, like anti-virus software, need to be updated as new attacks become known.

Spyware Protection

Some software that you install monitors what you do without you knowing. It may monitor your Internet use or log your key presses. This software is often installed without you being aware, such as when you download a free game. The information that spyware collects may be of use to advertisers as well as criminals.

Spyware protection software tries to detect these malicious programs and removes them.

File & Disk Managers

We all store our data on discs, whether it is hard discs or solid state drives. There are a number of utilities that manage the discs and the files that are stored on them.

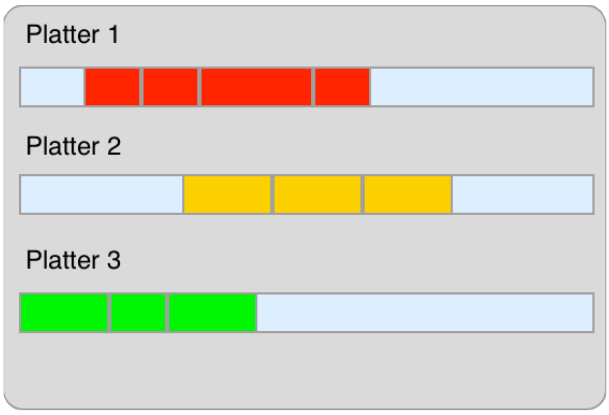
Formatting

Before a disc can be used it needs to be formatted to create sections that can be addressed by the OS. When a file is saved the OS checks a file allocation table to find empty sections and then saves the file in them. Different operating systems have different ways of doing this.

Defragment

When you add files and programs to a new computer it can often slow down. This is often due to the the way some OS's save files. Each sector on the hard disc is capable of holding files of a certain size. If a file is bigger than one sector it may take up a number of sectors. Sometimes when the OS saves a file it starts saving it some sectors but runs out before the file is completely saved. It then breaks up the file and saves the rest in a different part of the hard drive. This means that the file is in fragments. Reading the file can be slow as the mechanical read/write heads on the drive need to move to different parts of the disc. Defragmenting a drive moves the fragments so they are all together on the disc.

fragmented drive

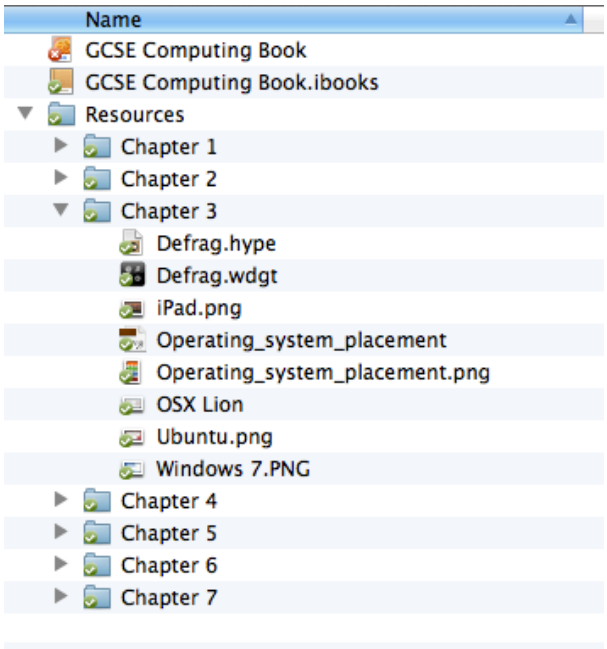


defragmented drive

Nowadays hard discs are so large that this rarely happens as much as it used to. Some OS's actively defragment by themselves with no input from the user.

File Management

Part of the OS is to take care of saving, deleting and moving files. File managers like Windows Explorer or Mac OS's Finder take care of these tasks. These provide a graphical way to select and move files around and provide a way to organise them.



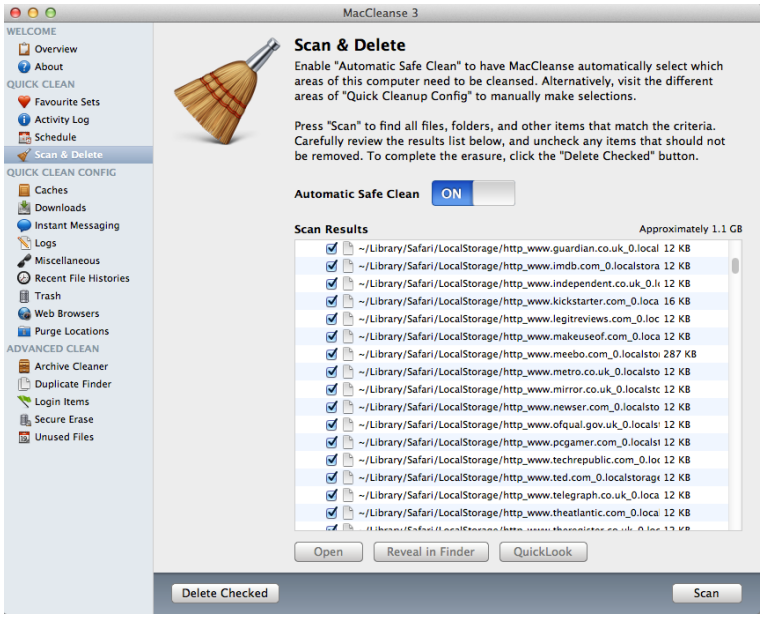
Finder shows folder structure

Most OS's use a folder structure. Folders don't really exist on the hard drive, but the OS shows files and folders as it makes it easier for the user to find them again. Dragging a file from one folder to another tells the file manager software to update the file allocation table. In most cases moving a file around on the same disc drive doesn't actually move the file on the drive, it only updates the file allocation table to show what folder it appears in.

When a file is moved to a different drive such as a USB stick, then the file manager deals with the transfer of the file from one disc to another.

System Maintenance

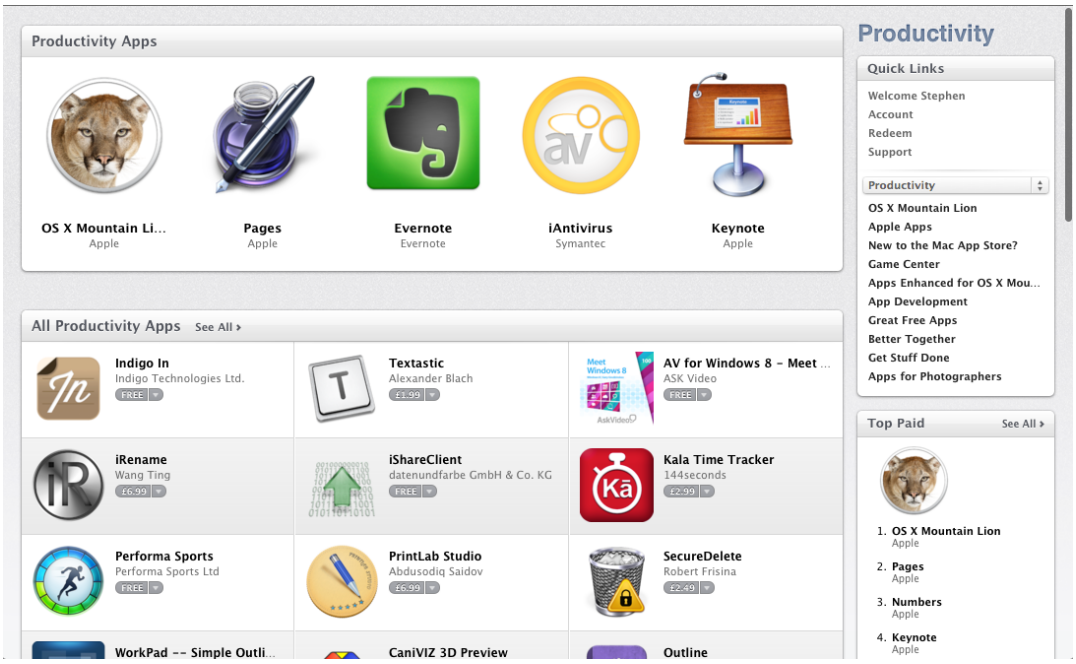
There are many files that can collect on your computer that you no longer need. Old installation files, temporary files and Internet cached files. These can clog up your computer and slow it down. Some tools can clear these files, helping to speed up your computer.



System maintenance

Section 3 - Types of Software

In order to make a computer useful it needs software. Software are the computer programs that make your computer do something useful. While an operating system and various utility software might be needed to make your computer system work, it is these other software programs that actually make a computer system useful. There are different types of software, from games to play to productivity applications which help you to produce documents more efficiently. Software can be put into a number of different categories.



OFF THE SHELF SOFTWARE

Most people start with software that has been written by someone else, usually a software company, that designs software to sell to a mass market. This software includes programs like image editing packages and office applications like word processors, spreadsheets and databases. These programs have mass appeal because they can be used for a variety of tasks. A word processor for example can be used to write a thank you letter to grandma, a school report or your next novel. As result these types of software packages are known as general purpose software for their ability to perform a number of related tasks.

The price and availability of off the shelf software is often low. The more specialised the market for that type of software, the higher the price, due to the fact that fewer people would buy the software. A simple word processor like Apple's Pages may cost a few pounds while MicroSoft Word has more features and so costs more. The 3D modelling and rendering software Autodesk Maya has a price of around £3,300. This software is very complex and involves the work of a

great number of programmers to create. Also the number of copies they can sell each year is lower than something like a word processor. All these factors raise the price of a piece of software.

Examples of of-the-shelf software include:

Software	Usage
Word processor	Reports, letters, screenplays, novels
Spreadsheet	accounts, data analysis, stock control, modelling
Graphics editor	creating and editing images
Desktop publisher	creating brochures and newsletters
Database	storing and retrieving large amounts of data

The advantages of off-the-shelf software are:

- often relatively cheap
- readily available
- will have been thoroughly tested and if bugs are found updated regularly
- will often have support in the form of online help, books and tutorials

The disadvantages are:

- may not match the need exactly. May have features missing.
- it may have features that they don't need. These features are still being paid for.
- the design and layout of the application may not suit the work flow of the user.

BESPOKE SOFTWARE

Sometimes the software available for purchase just doesn't meet the needs of what you require. In these instances you can have the software custom written for you needs. There are considerable challenges and costs involved but the result is that you get a piece of software that does exactly what you want. This custom designed software is known as bespoke software.

The advantages of bespoke software are:

- you get software that does exactly what you want
- the software can be designed to look and work exactly how you want it to.
- it will not contain features that your don't need for your business.

The disadvantages are:

- the software needs to be properly designed, written and tested. This process takes time. The more complex the software needed the more complex this is and so the more time and money are needed to produce it. It can take months or years to go through this process.
- the only people who can produce documentation are the people who built and tested it. There are no Internet forums to go to when you need help.

Open Source v Proprietary Software

Proprietary software is software that you buy from a company. They write the programming code to make the application and this code is then compiled to produce the binary executable program that they sell on disc or as a download. You cannot look at the code that was used to make the program. That code belongs to the software development company that wrote the program. They will often guard this code as it makes sure that the techniques they used cannot be used by other people freely. The company often wants to guard it their code so as to have a competitive advantage over other companies. A lot of the programs that you have bought will be proprietary. If you want a feature adding to the software, or you have found a bug, then you need to wait for the software developer to do that. This may take some time and in the case of new features, may never happen.

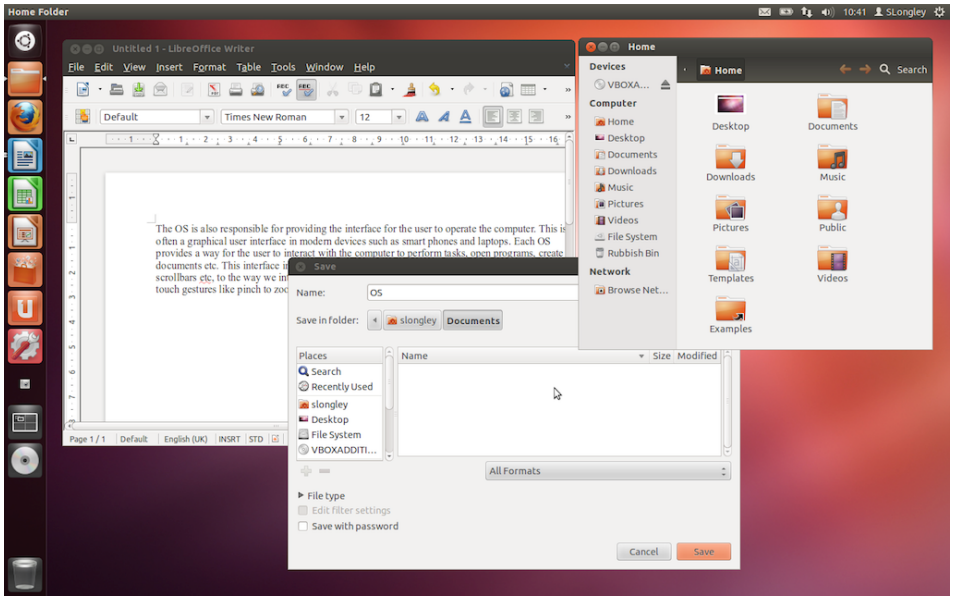
Proprietary software will often be developed behind closed doors, so that customers do not know what features are being developed in new versions of the software and when they may arrive.

Proprietary software can often lock users into using that software, making sure that data cannot be transferred easily to other programs, especially those of a competitor. This makes it difficult to change to a different program.

What is Open Source Software?

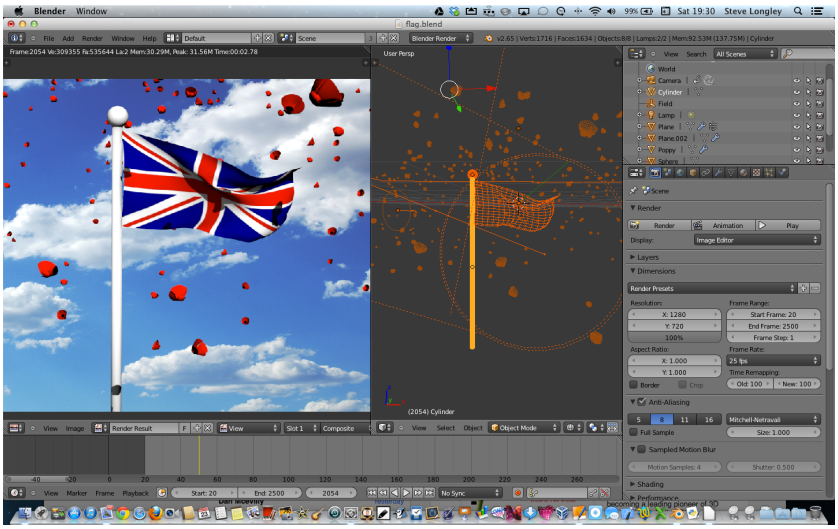
Some programmers create programs and then share the code they used with other people. In this way the source code is open and anyone can see how it works, make changes and improvements, contributing to the continued development of the software. Open source software is often free, and can have hundreds of developers around the world contributing to the development of the software. There are many open sources projects, many trying to mimic proprietary applications.

The advantage of open source software is that you can download the source code and see how it works. Changes can be made to give the software the features you want without having to write a program from scratch.



Linux is an open source operating system

Many, (though not all) open source programs have many developers from all over the world working on them. As there are so many developers looking at the code and checking it, bugs get found and fixed more rapidly than proprietary software. It can be more quickly updated. There is a new version of the 3D software package Blender out every six weeks, with bugs fixed and new features. This is much faster than most proprietary software which may stay the same version for over a year.



Blender is an open source 3D animation program, compositor and video editor.

The web browsers Firefox and Chrome are open source and they are quick to respond to changing standards in the web page language HTML.

Whilst some of the big open source applications like Blender and Linux have books written about them, some of the smaller projects have very little in the way of documentation, let alone a published book or tutorial.

Open source projects often have a roadmap published showing a plan for when features are going to be added. This can make it easy to see when an important feature may be added. Some open source projected are released before being version 1 and so may be lacking some features, and there may still be bugs in the application.

Chapter 4

Representation of data in computer systems

Section 1 -Binary

> here are 10 types of people in the world. Those who understand binary and those who don't.

If that quote doesn't make any sense now then hopefully by the end of this chapter it will.

In primary school when learning about place value you may have put headings above numbers like the table below.

H	T	U
		5
	1	2
2	5	4

Can you remember what do the headings mean? They stand for hundreds, tens and units. The smallest digit that a column can hold is 0 while the largest is 9. If 1 is added to a column then that column goes back to 0 and 1 is added to the next column. So when 1 is added to 9 this becomes 10. So we now have 1 ten and no units.

How data is stored in computers

Computers are made up from millions of switches. These switches can be ON or OFF. Everything in a computer from numbers to text, from sounds to pictures must be converted into a series of ON's and OFF's.

Computer scientists often think of these switches as 0s and 1s.

0 = OFF

1 = ON

Each 0 or 1 is known as a binary digit or **Bit**.

Computers would be pretty useless if the biggest number it could deal with was the number 1 unless it had a way of storing larger numbers from a series of 0s and 1s. This is Binary. It is a way of storing numbers as a series of 0s and 1s.

Introduction to Binary

Computers often group together 8 bits. This makes one **byte**.

Just like in **denary** (the number system we use) we often use column headings to remind us of the place value.

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	1

This is the number 1. Each column heading is a power of 2.

How to convert from denary to binary

You often need to convert a number from denary into binary.

How to convert 37 into a binary number.

What is the biggest number that goes into 37 without going over. Looking at the grid 32 is the highest number, so we put a 1 in that column.

128	64	32	16	8	4	2	1
0	0	1	0	0	0	0	0

Now we take 32 from 37.

This leaves 5. Now we repeat the process.

4 goes into 5 without going over. So we put a 1 in the 4 column.

128	64	32	16	8	4	2	1
0	0	1	0	0	1	0	0

That leaves us with 1. So we can put a 1 in the 1 column and then this is our answer.

128	64	32	16	8	4	2	1
0	0	1	0	0	1	0	1

So 37 in binary is 00100101 or just 100101.

Converting binary numbers into denary

This is much easier. It is just a matter of adding up the columns which contain a 1.

Convert 01010110 into denary.

Put the numbers under the place headings.

128	64	32	16	8	4	2	1
0	1	0	1	0	1	1	0

So the answer is 64+16+4+2 = 86.

Data Sizes in Computing

So we now know that the smallest piece of data that can be stored on a computer is one binary digit or Bit. This can only store two values, 0 and 1. We have already said that in computers bits are usually grouped in 8s to form one byte. What about larger measurements? Look at the table below.

Name	Size
bit	0 or 1
nibble	4 bits

Name	Size
byte	8 bits
1 kilobyte	1024 bytes
1 megabyte	1024 kilobytes
1 gigabyte	1024 megabytes
1 terrabyte	1024 gigabytes

Section 2 - Hexadecimal

What is Hexadecimal?

Hexadecimal is another number system used in computer science. It is often used as a shorthand way of writing a binary number.

Hexadecimal is a base 16 number system. So the column headings are based on powers of 16. This gives us a problem as the number 16 has 2 digits and number systems need to have just a single digit in each column.

16	U
1	0

1 sixteen and no units

16	U
0	10

we have a problem writing 10 units

So the number 10 above in base 16 is the same as 16 in denary, 1 sixteen and zero units. The problem arises when we have more than 9 units.

The example above us impossible. If we were to write down 010 in base 16 we would confuse it with the previous example.

The solution is to use letters instead of numbers. So the values that you can get in a column in base 16 are as follows.

Denary	Hexadecimal
0-9	0-9
10	A
11	B
12	C
13	D

Denary	Hexadecimal
14	E
15	F

So if we want to display the number 12 as a hexadecimal number we would write C. We would write 20 in hexadecimal as 14 and we would write 31 in hexadecimal as 1F (one 16 and 15 units).

How is this used as a short cut for a binary number?

Well look at the following binary number.

128	64	32	16	8	4	2	1
1	0	1	0	1	1	0	0

We can split this into two 4 bit chunks (nibbles) like this.

128	64	32	16	8	4	2	1
1	0	1	0	1	1	0	0

splitting into two nibbles

128	64	32	16
1	0	1	0

The most significant bits

8	4	2	1
1	1	0	0

The least significant bits

Looking at the least significant digits we can see that it is the number 12 in denary. We can write this as the hexadecimal number C.

Now looking at the most significant digits in the same way. We will deal with the column headings just like we did for the right most digits.

128	64	32	16
1	0	1	0

transforming the headings

8	4	2	1
1	0	1	0

So 1010 in binary is ten in denary.

We would write this as A in hexadecimal. So the binary number 10101100 can be written more simply as AC in hexadecimal.

Remember that AC means:

16	U
A	C

So this means there are 10 sixteens and 12 units. This makes 160 + 12 = 172.

Checking this against the binary 10101100 we have 128 + 32 + 8 + 4 = 172

Lets look at another example. The number 01101001 in binary would result in:

128	64	32	16	8	4	2	1
0	1	1	0	1	0	0	1

8	4	2	1
1	0	0	1

The least significant bits would be 9.

128	64	32	16
0	1	1	0

The most significant bits would be the same as

8	4	2	1
0	1	1	0

Which is 6. This means the number 01101001 is 69 in hexadecimal.

Converting from denary to hexadecimal and back again

To convert a number from denary to hexadecimal we first divide the number by 16. So if we look at the denary number 75. If we divide 75 by 16 we get 4 remainder 11(16 * 4 = 64 so 75 - 64 gives us a remainder of 11). So the number in the 16s column is 4 and we have 11 units. Remember we need to use letters for denary numbers above 9. So 11 is B in hex.

So 75 in denary is the same as 4B in hexadecimal.

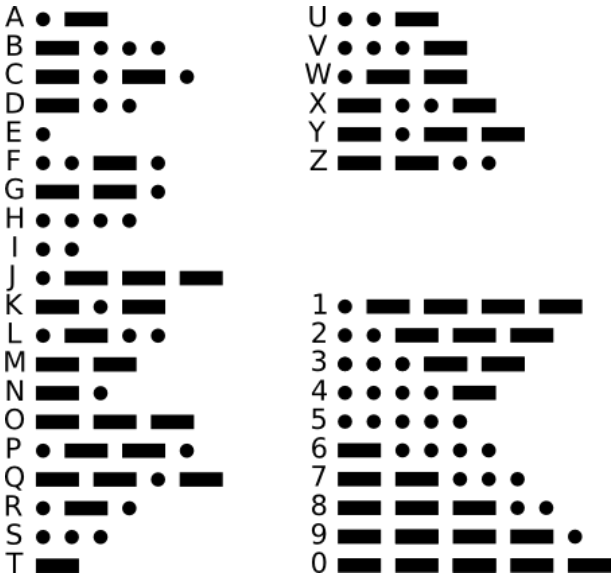
To convert a hexadecimal number into denary we need to first convert any letters into numbers. Then multiply the first digit by 16 and add on the units. So converting E6 into denary would be 14 x 16 + 6 = 230.

Section 3 - Storing Text

Character sets

Just like numbers, text needs to be stored as a binary code too. Each letter has a particular binary code. It is important for transfer of information that the binary codes used for each character are known between Operating Systems, so that data can be accurately transferred between computers no matter what type of computer system is used.

Character sets are nothing new. **Morse code** has a character set to describe the codes used for letters and numbers. As each telegraph operator knew morse code it made it possible to transmit text in a binary form (dots and dashes).



morse code

When computers were developed, they needed a way to store text that could be encoded. The problem with morse code is that each character has a different length. The byte is the unit of data storage in a computer and so it using this unit that the text has to be stored.

ASCII

One of the first character sets created was ASCII. This stands for American Standard Code for Information Interchange. This encoded 128 characters in a 7 bit binary code. The characters encoded included the alphabet (uppercase and lowercase), numbers, punctuation marks and non printing characters such as a line break and a carriage return. The final bit of the eight bit byte

was initially used as a check digit. Check digits are used to make sure that the data has been transferred correctly.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

ASCII table

As computers became more popular and used internationally, problems arose with languages such as French and German which have accents on some of their characters. As 8 bit Operating Systems were becoming the norm then ASCII was extended to include other glyphs such as accented letters and mathematical symbols such as ÷, ≠ and π.

Unicode

In the 2000's it became clear that ASCII did not have enough characters for all the symbols that needed to be used. Some languages such as Japanese and Mandarin have character sets that are much bigger than the latin character sets used by the west. Also some languages such as Arabic are read from right to left.

Unicode was developed to store letters as a 16 bit code. As more bits are used then more characters could be encoded. Unicode was designed to be large enough so that would it encompass all the characters in all the major languages of the world.

Most modern computer systems use Unicode to store text.

Remember that the letter is not actually stored, but a binary code to represent that character. When the computer reads that data, a computer program has to figure out which character to display on the screen for each binary code.

There is also a problem when storing numbers. The binary code for the digits 0-9 is not the binary numbers 0-9. The ASCII value for the character **5** is 53 in binary. In some computer programs data may be stored as text or as numbers. If a calculation needs to be done on

numbers which have been stored as text characters, then the data will need to be converted into the right data type first.

Section 4 - Storing Images

Pixels

Images are broken down into tiny coloured squares called picture elements or **pixels** for short. Each **pixel** represents just one colour.

Photograph made of pixels

Zoomed in showing square pixels

Different images can store different numbers of colours. A simple black an white image only needs to store two colours, black and white, and so an image could store the colours as a simple binary sequence.

Example

The following binary sequence could be used to store this simple picture of an umbrella.

0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	1	0	1	0	0	0
0	0	0	1	1	0	0	0

data used to encode image

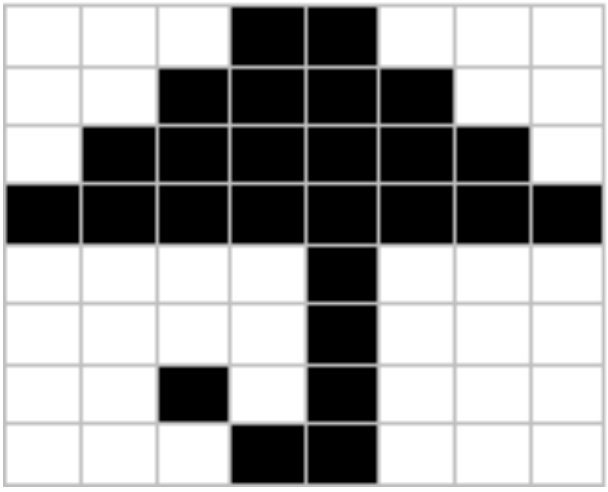
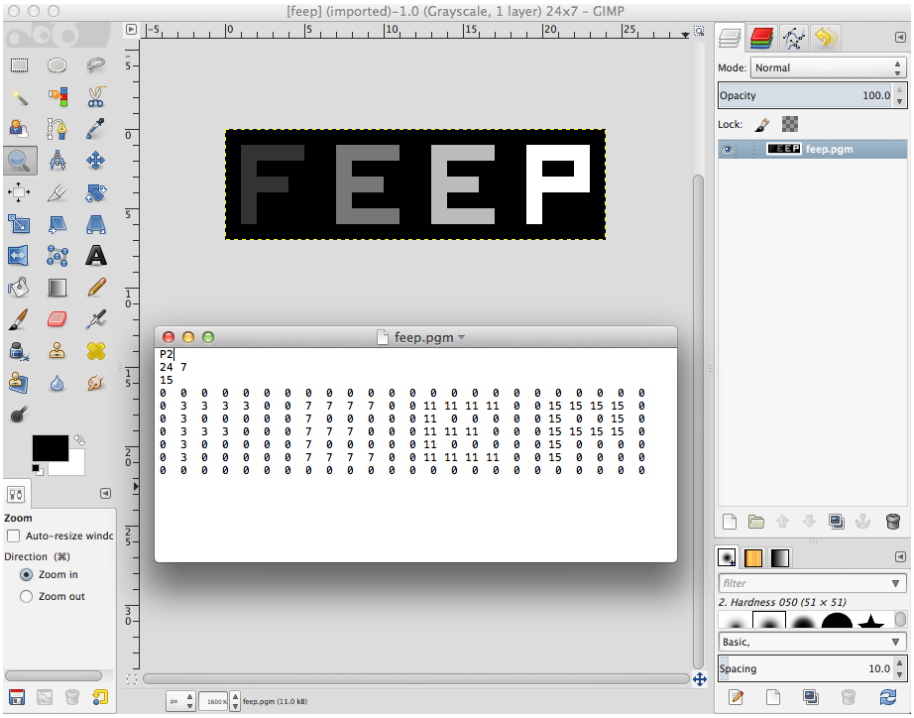


image of umbrella

In the example above a 1 represents black and a 0 represents white.

If we want more colours then we need to use more data to represent more colours. If we use more bits to represent each colour then we can have more colours available to use. This is known as an image's colour depth.



PGM files store data as text that can easily be edited

To display photographic images on the screen 3 whole bytes (24 bits) are used to represent the colour of each pixel. This is split up into one byte for each of the primary colours of light, **red**, **green** and **blue**. This is because computer displays uses these primary colours of light to display all the different colours we see on the screen. By mixing the three colours with different intensities the display is able to show over 16 million different colours.

No of bits per colour	No of possible colours
1	2
4	16
8	256
24	16.7 million

The amount of red, green and blue is usually stored as an 8 bit value, giving 256 levels of brightness for each colour (zero representing no colour and 255 representing the maximum brightness of that colour). So to represent bright yellow the values 255, 255, 0 can be used to represent the brightness of red, green and blue.

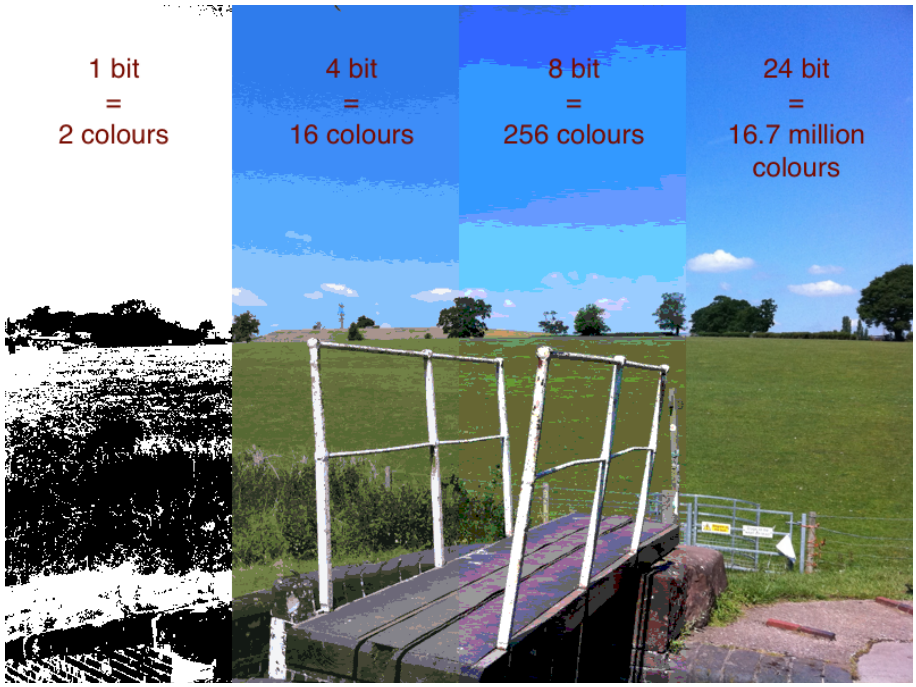


image showing different colour depths

IMAGE RESOLUTION

Image resolution is a measure of how much detail there is in the image. Resolution is defined as the ability to distinguish between two points. In a high resolution image it will be easier to distinguish between two dots close together, while in a low resolution image the two dots may become blurred into one shape. Look at the two images below. The first image has smaller pixels and so can pack in more into a smaller space. This results in a sharper image, however the file will be larger as it requires more pixels and so more data needs to be stored.

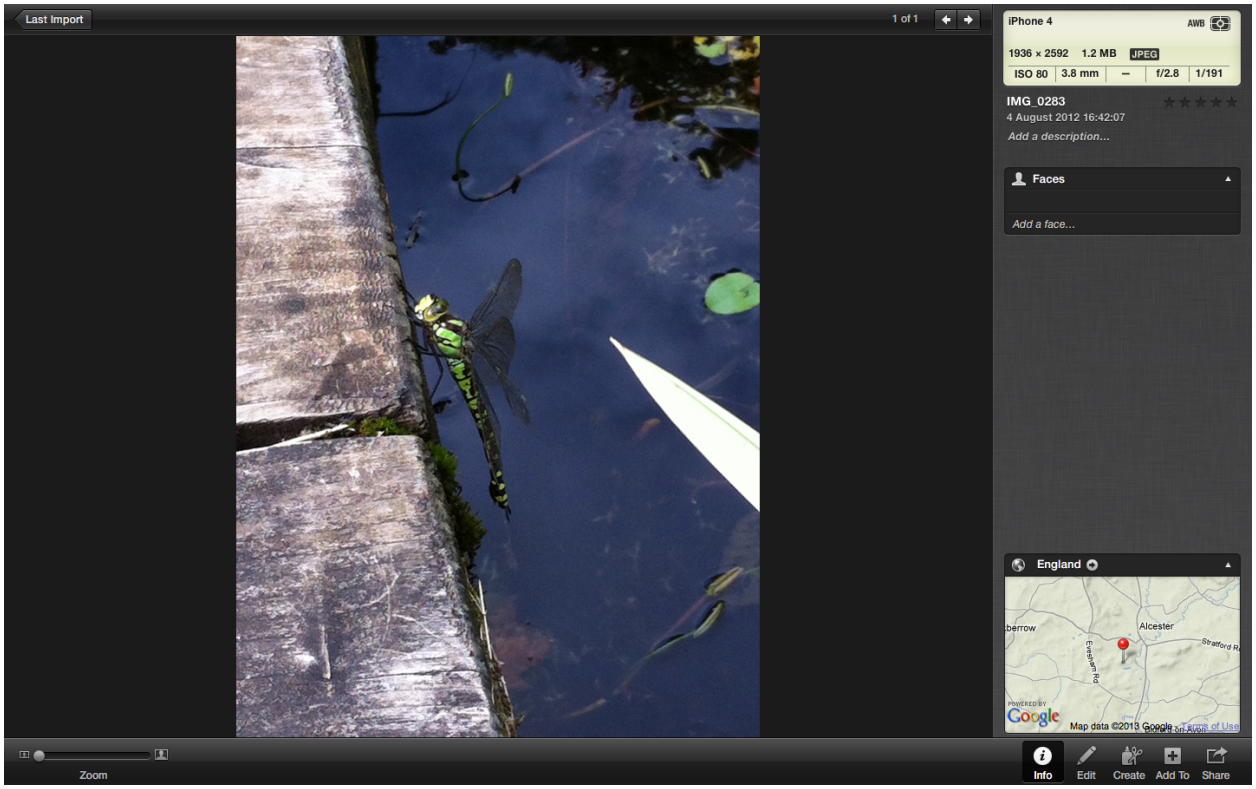
this image has a good resolution

This image has a poor resolution

Metadata

As well as the data about the pixels there is other data that is stored with the image that describes the file. This type of data is called **metadata** as it describes the image data. Most images contain some basic meta data such as the image **height**, **width** or **colour depth**.

Some digital cameras also store metadata in the photographs that are taken. This information may include data about the camera's settings when the image was taken, such as the focal length of the lens and the exposure settings. If the camera has GPS it may also store the data about where the image was taken too.

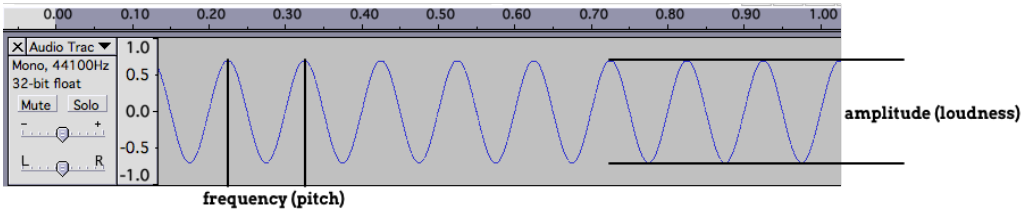


the meta data about this image stores its size, camera settings and GPS coordinates

Section 5 -Storing Sound

Analogue Sound

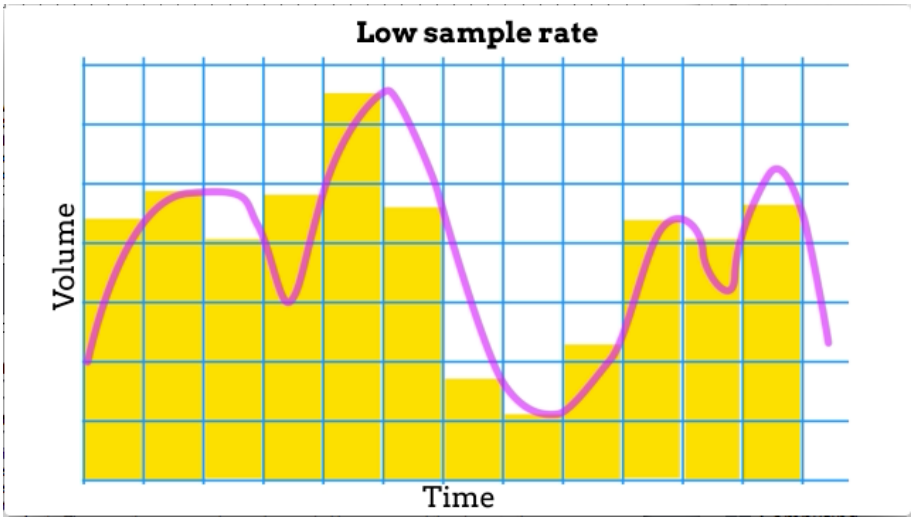
Sounds are analogue signals. This means that the sound wave has a range of values. Look at the blue sound wave below. The height of the wave (amplitude) is the volume of the sound. The distance between two peaks is known as the frequency and corresponds to the pitch of the sound.



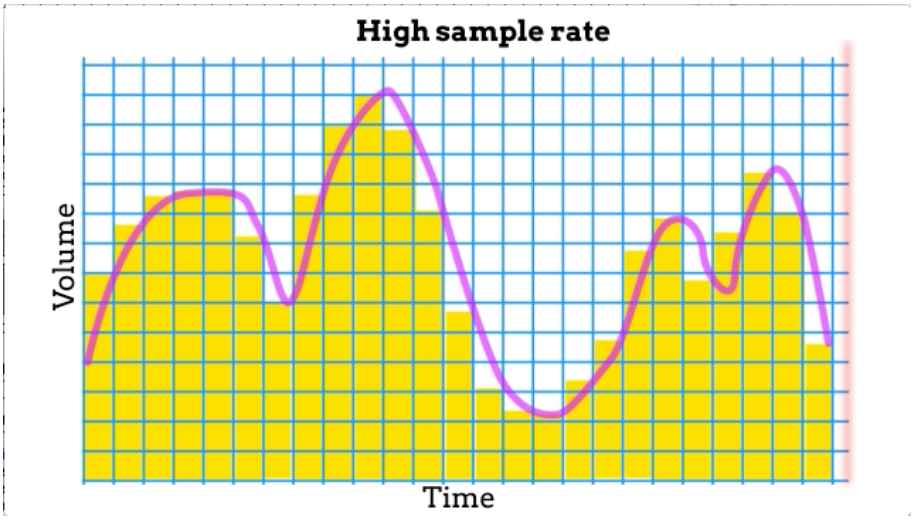
analogue sound wave

Digital Conversion

The sound wave needs to be converted to into numbers that can be stored in binary. To convert an analogue sound into a digital sound, the computer samples the height of the sound wave a number of times a second.



low sample rate



high sample rate

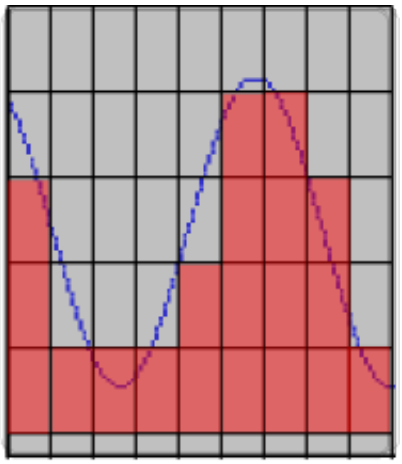
The number of samples in one second is known as the **sample rate**. This is measured in **Hertz** (samples per second) or more commonly Kilo Hertz (1000 Hz). A CD quality sound is sampled at 44kHz. This means the sound wave is checked 44,000 times a second and the height of the wave at that point is stored as a binary number. This is represented by the yellow bars. Notice that this does not follow the pink analogue line perfectly. The more times a sound wave is sampled in one second the closer to the original sound wave and the better it will sound. This however results in a larger file. When music is recorded in a studio it is often sampled at 192kHz.

Saving files such as MP3 and WAV files with a lower sample rate will compress a file. This is a lossy compression as data is lost in the re-sampling and cannot be brought back. MP3 files also saved take advantage of the way our ear works to combine two frequencies to make one without any perceivable loss of quality.

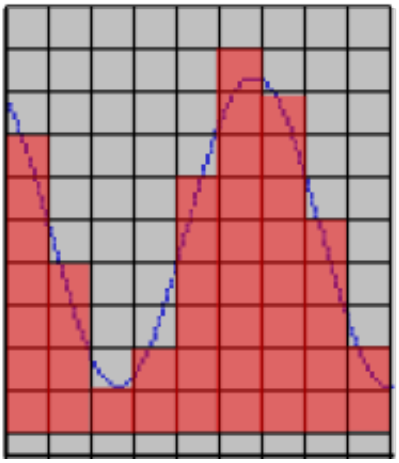
Another factor that determines the quality of the sound as it is digitised is the amount of data used to store the height of the wave. This is know as the **bit rate**.

A 2 bit number can only have 4 possible values. This would not accurately measure the different possible wave heights. If we used 8 bits there would be 255 different levels of volume that could be encoded. This would result in telephone quality sound. For music 16 bits are more commonly used, giving 65536 different possible values for the height of the wave. This can more accurately give a smoother transition between different volumes.

Imagine the diagram at the top shows a sound encoded so the height can either be 0,1,2,3 or 4. This does not give enough detail so the digitised sound does not mimic the sound wave closely enough.



The diagram below shows more detail so that the digitised sound more accurately follows the sound wave, even though the sample rate remains the same.



Section 6 - Storing Instructions

Encoding Instructions

When the first computers were created they were just simple calculating devices. If you wanted to change what the computer did then you needed to rewire the computer.

In World War II an American scientist called John Von Neumann was thinking about this problem. He thought that there was no difference between data and a digital instruction. He thought that both data and the computer programs that work on that data could be stored in the computers memory. The instructions could determine which data to retrieve from which memory location and what to do with it. This is called a store-program computer and is the basis for modern computing.

Each instruction would be fetched from memory to the CPU before being decoded and then the instructions are executed. Then then next instruction is fetched and so on. For this reason it is known as the **Fetch-Execute Cycle**.

Storing Instructions

Each computer processor understands a series of instructions. Each instruction is a binary sequence. Each processor has its own set of instructions that it understands. These instructions carry out actions on the data that is stored in the memory. This may mean moving data from memory to another location, such as a register in the processor, or adding data from two different locations together.

These instructions are located in one part of the computer's memory while the data is located in a different part. Both the data and the instructions can be accessed via the address of the memory location. The instructions have two parts to them. The instruction and the memory address of the data they are operating on. As these instructions and the memory addresses are both binary, this can cause problems for programmers who need to write instructions in this way.

Assembly code is a way of using short text commands and memory addresses instead of binary. This code can then be converted into binary by an assembler.

Example

LDR R1, [R5]

This **LOAD**s the **R**egister R1 with the value from location R5.

STR R1, [R6]

This **STORES** the value in register R1 into memory location R6.

The instructions LDR and STR will have specific binary bit patterns that the computer processor understands. When the computer comes across the bit patterns it knows how to follow

that instruction. Each instruction is followed in turn. Some instructions tell the processor to jump to another location and carry out instructions from there.

Writing in assembly language is difficult and specific to the type of processor. Most computer programs are written in a higher programming language.

Chapter 5

Databases

Section 1 - Introduction to Databases

What is a Database?

A database is a data that is stored in an organised way to make searching through it to find information easier. We have had databases long before computers. Address books are an example. People would have a special book to write the names an addresses of the people who you know stored in alphabetical order. So if you wanted to write to Becky Smith you would open your book to the **S** section and look through until you found the entry for Becky Smith. The problem with address books is that while they have sections for each letter of the alphabet, the entries within each section are probably not in any order. If you have 30 names and addresses under one letter, you would still have to look at each in turn until you found the correct one. This is because you may enter a new name and address into the book at anytime. You may have entered the name and address for Becky Smith and then a week later you make friends with Marla Singer and add her into your address book. Singer is before Smith in the alphabet and so they would not be in order. Also when people move houses updating the book becomes messy as old addresses are crossed out.

an address book is a paper based database

CARD FILES

The some of the problems with data stored in books can be solved by having data on cards. New cards can be added or replaced and they can be resorted when new cards are added. This is how recipe cards are organised so new ones can be added. The cards can be organised in different ways, either by main ingredient, or by type of meal (lunch or main meal or dessert).

In Library's card file systems have been used to help people find the books they want. There are often two card file systems in libraries. One has the cards ordered by book title and another set has the books ordered by author name. This duplication of data is one of the disadvantages of a paper based database. Also the way the data is currently organised may make searching for data problematic.

a library card file system

When I check my home telephone to see if someone has called whilst I was out (using 1471) I am only given the telephone number of the person who has called. If I can only remember a handful of telephone numbers then how could I find out who had called me? I don't want to call back someone who I don't wish to speak to (a cold calling sales person for example) but then again the call may be from someone who I do wish to speak with. As my paper address book is organised by the surnames of the people in it, I cannot easily search through each entry to find the phone number, to see if it is someone I know.

When computers began to be used to store data, searching for data became easier. The computer could sort the data in different ways and I could search through the data more effectively. The address book on my phone allows me to search through the list by either forename, surname, telephone number and even post code.

There is special software that manages and searches through databases. These types of software are know as Database Management Software (**DBMS**).

Flat File Databases

The databases I have discussed so far can be described as flat file databases. The data is stored as a record. Each record has a number of fields to store data. This is like a table made up of rows and columns.

A table column is a database field such as First name, last name and phone number. Each record would have data for each of these fields. Looking at the flat file database below we can we that the column headings are the fields and the rows are the individual records.

Last name	First name	Phone	Email	Total in Party	Attending?	Notes
Ryan	Craig	(123) 456 9688	cryan@noreply.com	2	TRUE	
Field	Diane	(123) 456 8813	dfield@noreply.com	1	TRUE	
Smith	Richard	(123) 456 5667	rsmith@noreply.com	1	FALSE	Vegetarian
Jones	Chris	(123) 456 5088	cjones@noreply.com	1	TRUE	No dairy products
Summers	Christine	(123) 456 4370	csummers@noreply.com	1	TRUE	
Sloan	Robert	(123) 456 5737	rstevents@noreply.com	2	TRUE	
Scott	Marie	(123) 456 7547	mrichards@noreply.com	1	FALSE	
Johnson	Kelly	(123) 456 6048	kjohnson@noreply.com	1	TRUE	

This data can be searched, to find out who is attending for example. It can also be sorted, in order of surname or by how many in the party.

Problems with Flat File Databases

There are problems however with flat file computer databases. These problems occur when the number of fields increases. This flat file database is for a hospital that loans out equipment to out patients. In the database data about both the patient and the item loaned needs to be stored. As items are returned then the item can appear in the database more than once. So you can see that the **Crutch Adult - CA1** gets loaned more than once, as does **Crutch Adult - CA2**. Patients can also return, so their details will need to be entered for every item that is loaned to them. This duplication of data is time consuming to enter and also makes the database very large very quickly, making searching through it slower. Duplicating data is known as **data redundancy**.

Surname	Forename	Address1	Town/City	Post Code	Telephone	Ward	Item Code	Description	price	Date Loaned	Date Returned
Simpson	Homer	768 Evergreen Terrace	Springfield	SP12 3WE	01471 555 897	W4	CA1	Crutch - Adult	£25	2/5/13	5/5/13
Gumble	Barney	7 Beer Gardens	Springfield	SP24 6AW	01471 555 421	W6	CA2	Crutch - Adult	£25	3/5/13	14/5/13
Cartman	Eric	1245 Snowy Cresent	South Park	SP 12 7UY	01247 555 874	W6	NCJ1	Neck Collar - Junior	£6	3/5/13	10/5/13
Simpson	Homer	768 Evergreen Terrace	Springfield	SP12 3WE	01471 555 897	W3	WCL1	WheelChair - Large	£300	7/5/13	16/5/13
Simpson	Homer	768 Evergreen Terrace	Springfield	SP12 3WE	01471 555 897	W4	CA1	Crutch - Adult	£25	7/5/13	9/5/13
Gumble	Barney	7 Beer Gardens	Springfield	SP24 6AW	01471 555 421	W6	CA2	Crutch - Adult	£25	21/5/13	28/5/13
Cartman	Eric	1245 Snowy Cresent	South Park	SP 12 7UY	01247 555 874	W6	WCJ1	WheelChair - Junior	£250	3/6/13	9/6/13
Simpson	Homer	768 Evergreen Terrace	Springfield	SP12 3WE	01471 555 897	W4	CA2	Crutch - Adult	£25	6/6/13	19/6/13

a poor flat file database

Another problem is that items can only be entered if they are loaned out. What happens if there is a new wheel chair WCL2. This would only get entered if when it is loaned out. This is not good record keeping.

In order to fix these problems the data can be split up into smaller tables that can be linked together. This is called a relational database.

Database Management Systems

The database is the file or files that store the data. Often this data is separate from the program that actually manipulates that data. The application used to access, manipulate and

present the data is known as a Database Management System (DBMS). The separation of the data from the DBMS is often desirable for a number of reasons.

- Standard way of Accessing Data - the DBMS provides a uniform way of accessing the data, regardless of how that data is stored. Many DBMS's use a special programming language called SQL (Structured Query Language) to access the data. The DBMS uses these SQL commands to access the data in the databases. This may mean pulling together data from a number of separate files. SQL can be used both to create new databases and tables as well as accessing and querying existing databases.

- Security - The DBMS provides a level of security around the data, allowing you to password protect the data and even giving different users different access.

- Multi User Access - the database can be accessed by different users and even different applications at the same time. This means that the data could be accessed through a website by one user and by an administrator updating records through the DBMS software.

- Data Maintenance - The DBMS makes sure that the data is stored efficiently and also keeps data integrity. If a record exists in one table that is linked to data in another table, the DBMS makes sure that you cannot delete one record without also deleting related records.

Modern databases are often relational databases and so the software used to create and manage such systems are known as Relational Database Management Systems (RDBMS). Examples of which include Microsoft Access, MySQL and Base which is included in the Open Office suite of applications.

Section 2 - Relational Databases

Solving the problems of Flat File Databases

Looking again at our bad flat file database we can see that the first issue is that that we have duplication of a lot of data. This is known as data redundancy. A relational database solves this by separating the data into different tables. To do this we use the concept of **entities**. An entity is a thing. This can be a physical thing like a customer or a item to be loaned, or it can be a concept like the loan itself. Each entity is stored in its own table. However these entities are related to each other and so we need a way of keeping this relationship intact.

Each record in a table is unique and is given a unique identifier. This identifier is used to create a link between tables. An identifier can be any field as long as its unique. Unfortunately unique fields are hard to come by, so often a number is often generated that is unique. This is why we often have ID numbers for this like passports, bank accounts and student IDs in schools.

Looking at our bad flat file database we can see that we only have 3 patients that are in the database multiple times. We can create a Patient table and create an unique ID for each record in the table. The unique ID field is known as a **Primary Key**. A Primary Key is a field that uniquely identifies a record in that table. In table below we can see our new Patient Table with the PatientID as the primary key.

PatientID	Surname	Forename	Address1	Town/City	PostCode	Telephone
1	Simpson	Homer	768 Evergreen Terrace	Springfield	SP12 3WE	01471 555 897
2	Gumble	Barney	7 Beer Gardens	Springfield	SP24 6AW	01471 555 421
3	Cartman	Eric	1245 Snowy Cresent	South Park	SP 12 7UY	01247 555 874

Patient table

Now we need to link this data to the item loan. Looking at figure 5.4 we have removed the patient data from the table. In its place we have the patientID which is the primary key from the Patient table.

LoanID	Ward	Item Code	Description	price	Date Loaned	Date Returned	PatientID
1	W4	CA1	Crutch - Adult	£25	02/05/13	05/05/13	1
2	W6	CA2	Crutch - Adult	£25	03/05/13	14/05/13	2
3	W6	NCJ1	Neck Collar - Junior	£6	03/05/13	10/05/13	3
4	W3	WCL1	WheelChair - Large	£300	07/05/13	16/05/13	1
5	W4	CA1	Crutch - Adult	£25	07/05/13	09/05/13	1
6	W6	CA2	Crutch - Adult	£25	21/05/13	28/05/13	2
7	W6	WCJ1	WheelChair - Junior	£250	03/06/13	09/06/13	3
8	W4	CA2	Crutch - Adult	£25	06/06/13	19/06/13	1

loan table

A field that contains a primary key from another table is known as a **foreign key**. Foreign keys are used to create a link between tables. So in record 5 in the Loans table we can see that Patient 1 loaned the crutch. We can use the patient ID to look up who that is in the Patient table. We can see that the patient with the patientID of 1 is Homer Simpson. Using a foreign key in the loan table means that we do not have to duplicate all the patient data when they loan an item.

We still have the item information duplicated. When we remove this entity from the Loans table we get three tables that can be seen below.

PatientID	Surname	Forename	Address1	Town/City	PostCode	Telephone
1	Simpson	Homer	768 Evergreen Terrace	Springfield	SP12 3WE	01471 555 897
2	Gumble	Barney	7 Beer Gardens	Springfield	SP24 6AW	01471 555 421
3	Cartman	Eric	1245 Snowy Cresent	South Park	SP 12 7UY	01247 555 874

Patient table

ItemCode	Description	price
CA1	Crutch - Adult	£25
CA2	Crutch - Adult	£25
NCJ1	Neck Collar - Junior	£6
WCL1	WheelChair - Large	£300
WCJ1	WheelChair - Junior	£250

item table

LoanID	Ward	Item Code	Date Loaned	Date Returned	PatientID
1	W4	CA1	02/05/13	05/05/13	1
2	W6	CA2	03/05/13	14/05/13	2
3	W6	NCJ1	03/05/13	10/05/13	3
4	W3	WCL1	07/05/13	16/05/13	1
5	W4	CA1	07/05/13	09/05/13	1
6	W6	CA2	21/05/13	28/05/13	2
7	W6	WCJ1	03/06/13	09/06/13	3
8	W4	CA2	06/06/13	19/06/13	1

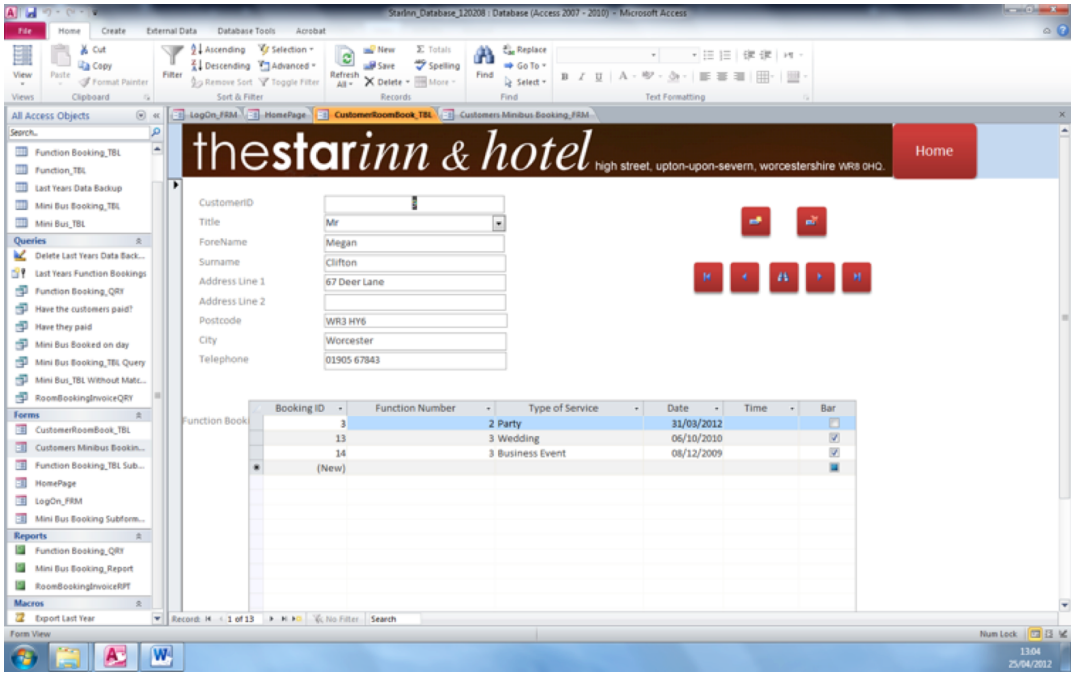
loan table

This type of database is known as a **Relational Databases** due to the relationships between each table. A **RDBMS** has a number of components that are used to access the data stored in the database. Accessing the data directly through the tables is not recommended. Often data is accessed through forms, queries and reports.

Forms

A **form** is used for data entry. They provide a more user friendly way of entering data into a table and even multiple tables in one form. A user form can also provide some data validation checks. Validation checks are used to make sure that the data is in a valid form before it is

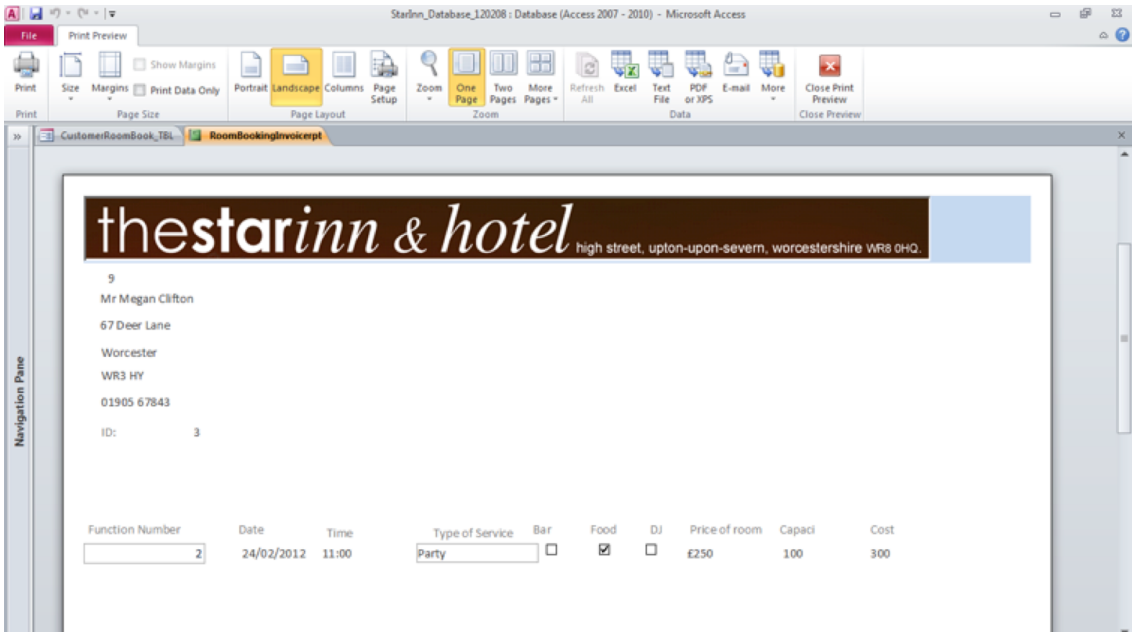
entered. This can anything from making sure a telephone number is the correct length to making sure an item number actually exists. Forms can also make some database fields read only so that some data cannot be altered.



a data entry form

REPORTS

Reports are used to properly format data from queries or when the data needs to be printed out. The reports can calculate results based on the data from fields, adding up the prices to give a grand total for example.

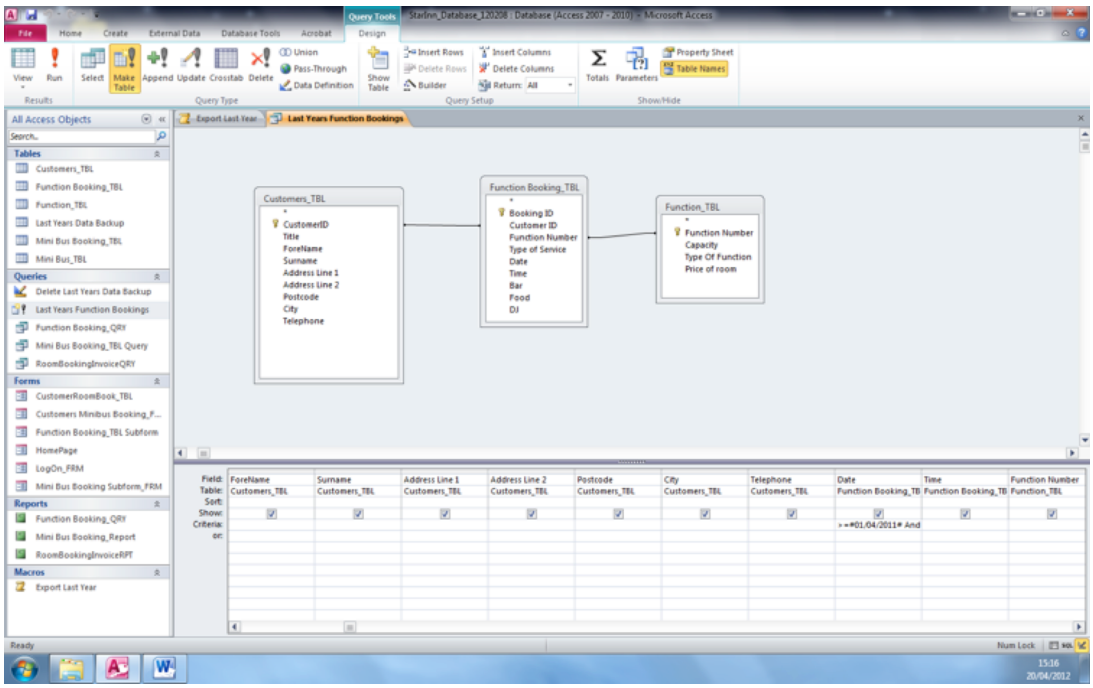


a report showing an invoice

QUERIES

Queries when the DBMS selects and searches for specific data from the database. This can be done using a specialist language like SQL or can be done through a query by example interface which is often easier for beginners.

Queries are really important in databases, allowing us to get to the data we need.



Query by example allows you to create queries interactively

Looking back at our hospital loans example database we have three tables. If we wanted to search for data relating to the loans we would need to start with the loans table.

LoanID	Ward	Item Code	Date Loaned	Date Returned	PatientID
1	W4	CA1	02/05/13	05/05/13	1
2	W6	CA2	03/05/13	14/05/13	2
3	W6	NCJ1	03/05/13	10/05/13	3
4	W3	WCL1	07/05/13	16/05/13	1
5	W4	CA1	07/05/13	09/05/13	1
6	W6	CA2	21/05/13	28/05/13	2
7	W6	WCJ1	03/06/13	09/06/13	3
8	W4	CA2	06/06/13	19/06/13	1

loans table

If we wanted to know what loans patient two had we could query the data using

```
PatientID = 2
```

This would return these rows.

LoanID	Ward	Item Code	Date Loaned	Date Returned	PatientID
2	W6	CA2	03/05/13	14/05/13	2
6	W6	CA2	21/05/13	28/05/13	2

PatientID = 2

We can see that the patient has loaned the same item on two different occasions.

We could find out which which loans were done with patients in Ward W6 by using this query:

```
Ward = "W6"
```

That would return this data.

LoanID	Ward	Item Code	Date Loaned	Date Returned	PatientID
2	W6	CA2	03/05/13	14/05/13	2
3	W6	NCJ1	03/05/13	10/05/13	3
6	W6	CA2	21/05/13	28/05/13	2
7	W6	WCJ1	03/06/13	09/06/13	3

Ward = "W6"

As the field contains text data, the query must put the data in quotes.

More complex queries using Logical Operators

Sometimes we need to select data between particular ranges. If we wanted to know about all the loans in may 2013 we would need all the dates equal to or greater than 1st May and all those below or equal to 31st May.

```
DateLoaned >= 1/05/2013 AND DateLoaned<= 31/05/2013
```

That would return the following data.

LoanID	Ward	Item Code	Date Loaned	Date Returned	PatientID
1	W4	CA1	02/05/13	05/05/13	1
2	W6	CA2	03/05/13	14/05/13	2
3	W6	NCJ1	03/05/13	10/05/13	3
4	W3	WCL1	07/05/13	16/05/13	1
5	W4	CA1	07/05/13	09/05/13	1
6	W6	CA2	21/05/13	28/05/13	2

DateLoaned >= 1/05/2013 AND DateLoaned<= 31/05/2013

The query uses the AND keyword to make sure that both criteria are met. The date must be both greater than or equal to 1/05/2013 AND be less than or equal to 31/05/2013.

If we wanted all the adult crutch loans we could perform this query.

```
ItemCode = "CA1" OR ItemCode = "CA2"
```

That would return these rows.

LoanID	Ward	Item Code	Date Loaned	Date Returned	PatientID
1	W4	CA1	02/05/13	05/05/13	1
2	W6	CA2	03/05/13	14/05/13	2
5	W4	CA1	07/05/13	09/05/13	1
6	W6	CA2	21/05/13	28/05/13	2
8	W4	CA2	06/06/13	19/06/13	1

ItemCode = "CA1" OR ItemCode = "CA2"

We use the OR keyword to make sure we return records that match either "CA1" or "CA2". Each record can only be one or the other, not both so here we need to use the OR operator.

AND & **OR** are know as **logical operators**. **NOT** is another logical operator. It means do not get records that have this value.

Another way of performing the last query is to use a **wildcard**. A wildcard is a symbol (usually an asterisk) to represent any other letters or numbers.

So the query

```
ItemCode = "CA"*
```

is the same as

```
ItemCode = "CA1" OR ItemCode = "CA2".
```

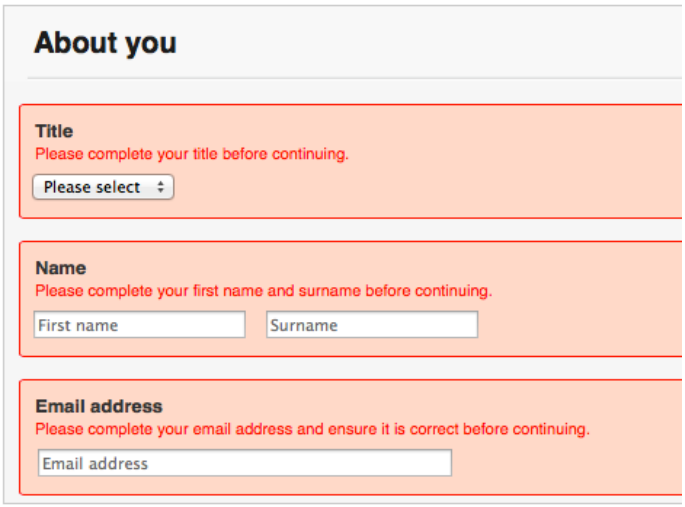
DATA VALIDATION

It is important that the data in a database is sensible and correct. Validation helps in this process by checking to see if the data being entered is sensible and valid and in the correct format.You often see this being performed on web forms which often send their data into a database.

One database may contain a table to store bookings. One field may store the date for the booking. We could enter the data as 21/3/2014 or 21st March 2014. Americans often put the month first so they would enter 3/21/2014. If we are going to be able to search through the data we need to store it consistently. Validation can help us enter data in the correct format. Validation is also used to make sure the data is sensible. 30th Feb 2014 is not a valid date, so we should not allow that to be entered into our database.

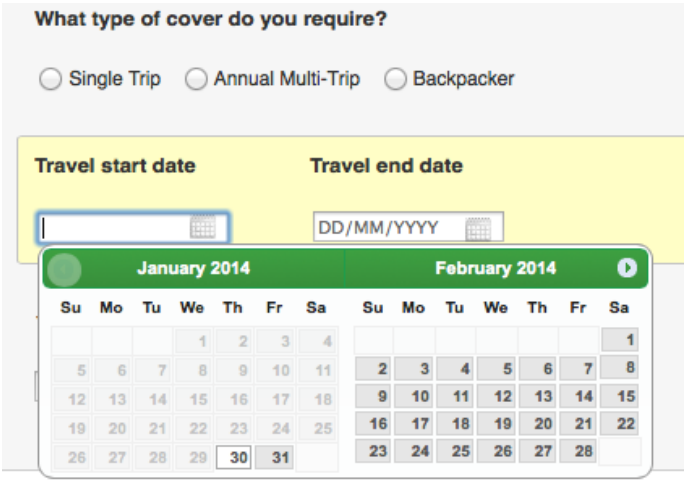
There are different types of validation checks that can be performed on data before it is allowed into our database.

Presence check - This is the simplest of all validation checks. it simply checks to see if data is being entered into a field. Often these fields have an asterisk showing they are required and are highlighted if they are left blank.



presence check

Date Check - This field will often be a calendar to make sure a valid date is selected. The calendar ensures it shows they correct year, allowing the user to pick easily.



date picker

Range Check - This type of validation ensures that numerical data is between a particular range. Shoe sizes may go from 4 to 12 on a particular online shopping site.

Length Check - This will check to see if the length of the data is acceptable. Mobile phone numbers, ISBN numbers and credit card numbers all have particular lengths. If the data was too short of too long then the data would not be valid.

Format check - Some fields require data to be in a particular format. Some product codes may need to have a combination of letters and numbers in a particular order. Car registrations have to follow a particular format, so do Post Codes. National insurance numbers follow a particular format, LL 99 99 99 L where L is a letter and 9 is a number 0-9.

Lookup check - This type of validation looks up the data to make sure it exists. This could be a product number in a catalogue superstore or a car registration number on an insurance website.

Check digit - Some numbers end in a number that is calculated from the previous numbers. Credit card numbers, ISBN numbers and barcodes all have a check digit. If any number is incorrectly entered the check digit calculated will not match the one entered.

Validation **does not check to see if the data is correct**, only if the data is **sensible and valid**. If a field asked me to enter my date of birth I could enter 21/3/1981. This would be a valid date, but it is not by true birthdate.

Some fields are difficult to validate. People's names are very hard to validate. While most people have names that probably appear and a baby naming book, what about foreign names and those parents who give their children made up names. In New Zealand some children were named Violence; Number 16 Bus Shelter; Midnight Chardonnay; Benson and Hedges.(http://news.bbc.co.uk/1/hi/7522952.stm) These names would be hard to validate for.

Separating Data from Applications

Having a database on a single user computer is perhaps OK for a small business but usually the databases needs to be accessed by a number of people, sometimes millions of people.

Often the database will be stored on a server, allowing access via a number of different applications.

A search engine like Google has a database of web-pages on the Internet and keywords that link to them. This database can be accessed through a web-browser, but the same data can be displayed through the GoogleMaps application on a mobile phone.

FaceBook store millions of accounts in their database along with all the data people post online. This data can be accessed via a web browser, or mobile app but can also be accessed via games like Candy Crush or Words with Friends, allowing you to see which of your friends are playing the game.

The separation of the data in the database from the application that uses it gives the users greater flexibility in how the data can be used.

separation of data from applications

Chapter 6

Computer Communications and Networks

Section 1 - Networks

Why Network Computers?

Joining computers together, either with cabling such as ethernet or wirelessly can be complex, so why do we do it? There are many reasons why we would want to **network** computer together. The main reason is to share. This may be to share files and data, or a resources like a printer or a connection to the Internet.

TYPE OF NETWORK

There are different types of computer network. They can be defined by the relationship the computers have with each other and the way they are joined together. We will look first at the relationship between each computer.

Peer to Peer

All stations are joined together in the network have equal status. If you have a number of computers at home on sharing a wireless router then this is the type of network you will probably have. Each device shares the connection to the Internet, whether that is a tablet, games console or computer. You may have other shared resources like a printer or even a media server. You may be able to get access to your media server through your tablet or games console, to stream music or video on to your TV.

So in a peer to peer network, there is no central computer or data store. You may be able to share data from one computer to another. This needs to be set up on each device and is not done centrally.

Applications may be stored on different computers and accessed by all as longer as the owner gives permission. Work is backed up on individual user PCs.

In a peer to peer network you get the benefit of being able to share resources but each computer must be set up individually. This is fine for the home but becomes inefficient the more computers you have.

Client-Server Network

A client-server network has different types of computers. A powerful central computer called a server manages the client computers the users use. The data is stored centrally. This means that it does not matter which client computer the user logs on to they can still access all their work.

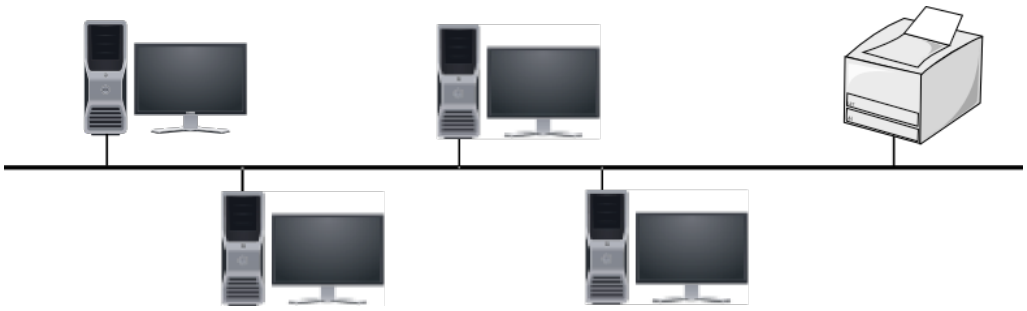
In a client server network, the client computers can be managed centrally from the server. This means that installation of programs, back ups, security and access rights can all be managed from one place. This can save a great deal of time. However this does mean there is a reliance of this central computer. If the server goes down then no-one can access anything. Also it is more technically challenging to set up and maintain such a network. This means there is often someone responsible for managing the computer. For large networks this is the most efficient and flexible despite the reliance on the central computers. Effort is taken to make sure this central computer is always backed up and working properly.

TOPOLOGY

Computers can be joined together in different ways. The way they are joined together is known as topology. There are three main network topologies that we need to be aware of.

Bus network

This is one the simplest of wired networks.



Bus network

Workstations are connected to the main central cable/bus. Data can travel in both directions but only one computer can use the bus at a given time. A terminator is connected to the ends of the central bus to stop signals bouncing up and down the network.

Two nodes could attempt to transmit at the same time and a collision will occur. To avoid this, a node waits until no traffic on the bus and pauses slightly before transmitting. If a collision occurs both nodes wait and try again at a random time interval.

This is the cheapest network topology as the smallest amount of cabling is required.

Advantages

- Less cable need than a ring. Relatively cheaper and easier to install and maintain.

- Easy to add new nodes by removing terminator
- If a workstation/terminal goes down the others will continue

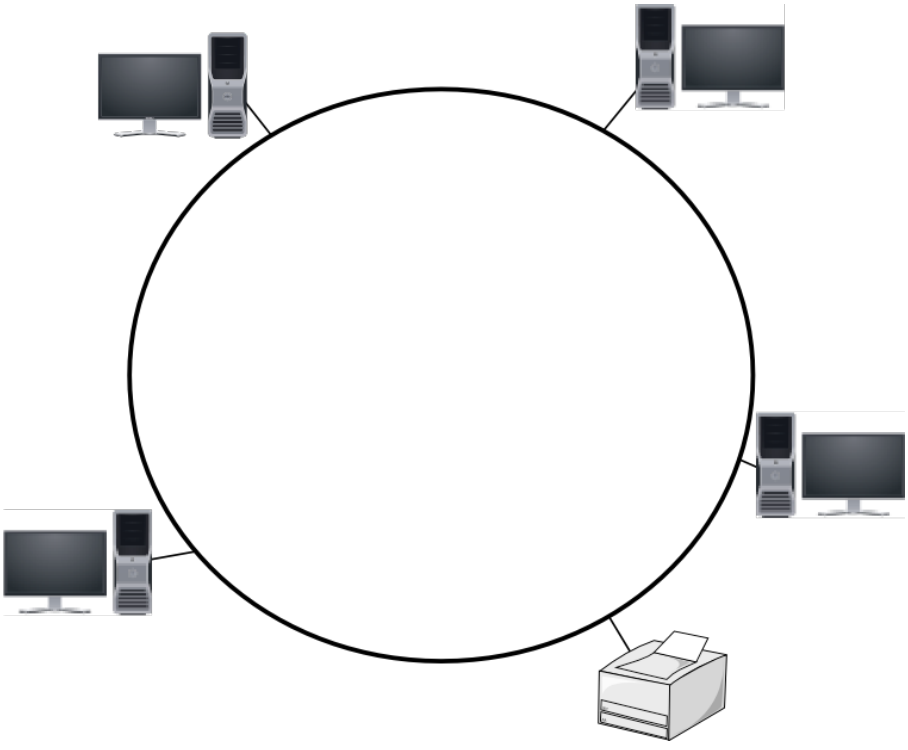
Disadvantages

- Heavy traffic can cause network failure and delays
- Heavily dependent upon the cable backbone and cable damage can cause network failure.

Ring Network

A ring has no central host computer and none of the nodes need to have overall control of access to the network. Messages in the ring flow in one direction from node to node.

An imaginary token is continuously passed around the ring. A token is a small packet that contains bits of data which passes around the ring (Always the same way around).



Ring network

The token is recognised as a unique character sequence.

If a computer is waiting to transmit, it catches a token and with it the authority to send data. It attaches its data. The packet will contain the destination address.

As long as one device has a token no other device can send data.

Once the sending workstation has received acknowledgement that the message has been received the token is free for use by another device.

Advantages

There is no dependence upon a central computer as data transmission is supported by all devices on the ring. Each node has sufficient intelligence to control the transmission of data from and to its own node.

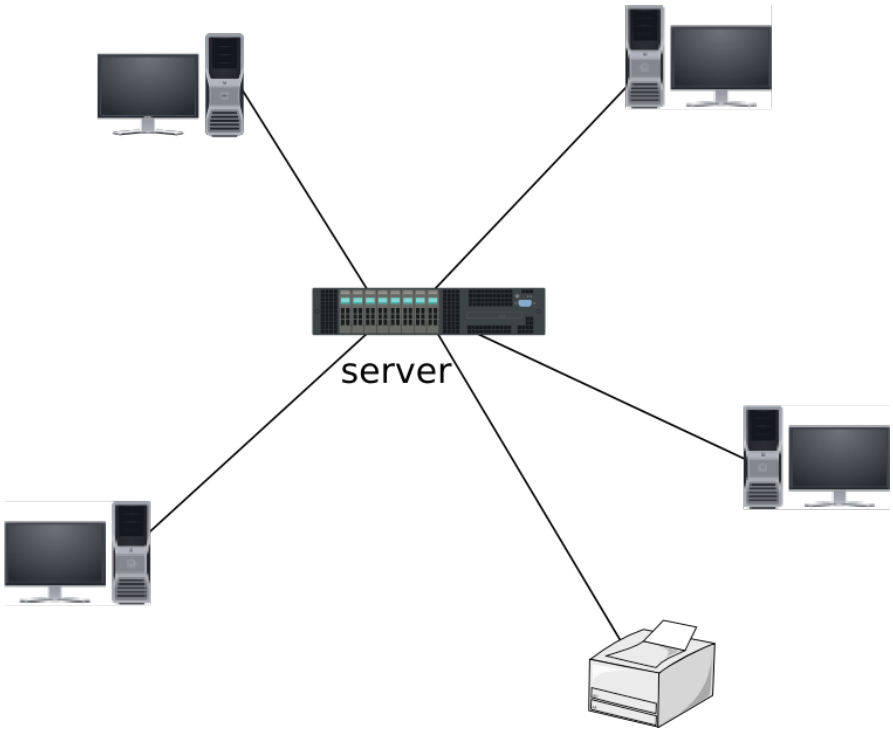
- Very high transmission speeds are possible.
- Routing between devices is simple because messages normally travel in one direction.
- As data is in one direction it can transmit large volumes of data

Disadvantages

- Systems depends upon the reliability of the ring repeater. If one node malfunctions this can affect the operation of the network.
- Cabling costs can be high.
- Difficult to extend the ring.

Star Network

The Star network is the most commonly used network topology in use today. Each device is connected directly to a central computer (usually through a switch or router).



Star network

Advantages

- Suitable for WANs where large organisations rely on a central computer for the bulk of data processing tasks
- Central control of message switching allow a high degree of security
- Each spoke is independent upon the rest. If one spoke breaks down the others can continue and the central CPU can identify any faults.
- Data transmission speeds can vary from spoke to spoke so one can use a high-speed device and another a low speed e.g. disc drive.
- Saves cabling.

Disadvantages

- Network is vulnerable to central hub failures.
- As a distributed system, some processing is still possible at the nodes but internode connection is lost.
- If servicing a WAN a mini or mainframe system is needed for control of messaging. So hardware and software is expensive.

Local Area Networks and Wide Area Networks

As well as being able to describe a network based on its topology, we can also describe a network in terms of geography.

A **Local Area Network** (LAN) is a network that is on one geographical location. This may be just one room of computers or as big as a single site, covering a number of buildings. The computers are all connected together through wiring such as ethernet, fibre optic or wireless LAN.

Some large organisations have more than one site, but still need to communicate between sites. These separate LANs are joined together through the telephone networks to create a **Wide Area Network** or WAN. The Internet is the biggest WAN, joining together millions of computers.

Addressing

Each computer on a network needs to be uniquely identified so that data can be sent and received from it. There are many different ways of doing this.

Protocols are used to help control network communication. Protocols are a set of rules that all the devices on a network must follow if they are to be able to communicate with each other.

When connected to a network or the Internet each computer will have an Internet Protocol address or IP address. This is made up of 4 bytes. We often see this as a series of 4 numbers ranging from 0-255 separated by full stops. So 10.1.128.240 is an example of an IP address. Computers use these addresses to communicate with each other in the same way we use phone numbers.

These numbers are hard for people to remember, so on the Internet each IP address of a website has a domain name, a text based name that is easier to remember. See the next section for more details.

As well as IP addresses, each computer also has a unique Media Access Control address or MAC address. Each device usually has this address hardwired into them when they are manufactured. Again these addresses help a network identify each device, making sure that data travels to the correct destination.

Packets and Protocols

Imagine you are an author and you have written a 500 page manuscript which must be sent by post. The problem is that due to the postal charges, you can only send 50 pages at a time.

How do you get the manuscript to your publishers, so they can assemble it correctly?

Firstly you would split your document into 50 page sections, putting each into an envelope. On the envelope you would put the publisher's address as well as your own address, so they know

who sent it. You would also need to label each envelope so that the publisher know which set of pages were in each. So the first one would be labelled *packet 1/10 pages 1-50*, the second *packet 2/10 pages 51-100* etc. Then the envelopes could be posted and the publisher knows from the envelopes how to reassemble to manuscript. It does not matter if the envelopes do not arrive in order. If one goes missing, the publisher can figure out which one it is and ask for a replacement.

This is how data gets sent around networks.

When data is sent between computers it is broken down into manageable chunks called packets. The size of these packets is usually between 1000 - 1500 bytes, so an image of 1Mb would be broken up into around 10 packets.

Each **packet** has extra information attached to it. The IP address of the host computer as well as the IP address of the destination computer. It also has data referring to how many packets there are in total and which packet this is.

When the packets are sent the computers follow a set of rules or protocols for sending the data. HTTP is the protocol used to transfer data between web servers & web browsers.

The packets are sent over the Internet. It does not matter which route each packet takes, nor does it matter which packet arrives first. Once all the packets have arrived they can be used to reassemble the original file.

If one packet gets lost the protocol asks for a replacement.

Security on Networks

When we have so many users on a network, security becomes a big issue. In big companies and organisations, while they often want to be able to access data from any computer on their network, it does not mean that every employee should have the same access to all the data and programs.

In a bank for example, there is data kept on record about people's bank accounts, including account numbers, names and addresses and PIN numbers for bank cards. The networks are designed so that no one employee can gain access to all the data they need to steal any money from people's accounts. So one employee may be able to access a customers account, but be unable to see their password or other secure details. Another employee may be able to see and reset customer passwords, but not be able to access the account numbers.

Each user on a computer network is given an account. This will consist of a username and password. Their account will give them access to particular resources on a network, such as shared drives, applications, printers etc.

Each user will be given different permissions regarding data.

Permission	Capabilities
No Access	The user has no access to the files or folders
Read Only	The user can read but not save new files or alter the ones present.
Read/Write	The user can both read and write files as well as create new ones.
Full Control	The user has full access rights to create, edit and delete files.

Some network permissions

Passwords are designed to make sure only authorised users can gain access to a computer system. Passwords need to be easy to remember but hard to crack by a computer. Longer passwords with a mixture of upper case and lower case letters, Numbers and punctuation marks makes for a better password as the combinations are bigger and so harder to crack. It is best never to use things that can easily be guessed such your birthday or dictionary words. When hackers Break into websites they often show the passwords people have chosen. An Internet search would easily find the most common passwords used by people. Make sure your password isn't on the list.

Some data needs to be kept secure. One way of doing this is to do this is to encrypt the data. This scrambles the data in a way that can be unscrambled later, usually by providing the correct authentication details. If the data is accessed without permission, it would not be in an understandable form.

Managing Networks

A computer network needs a great deal of effort to manage and keep it running smoothly. Computer networks often have A network manager who has a number of responsibilities.

Each user often has just signed an acceptable use policy. This document in forms the users have a network can be used in an acceptable way. This may be different for each organisation. For example some companies may prohibit the use of USB memory sticks, as they may be a source of viruses and other malware as well as be a device to steal sensitive documents. An acceptable use policy would also cover the use of the Internet. Internet use in a company is often monitored and any unacceptable use could result in the employee being disciplined or even sacked.

Another task needs to be completed is that of archiving and backing up of data. Computer hardware like hard disks can fail. This would result in the loss of all the data that was on the disk. Backing up means copying that data onto another storage medium such as another hard disk, a

tape or the cloud. If I hard disk was to fail and data could be restored from the backup onto a new hard disk.

As companies produce a lot of data backing is very time-consuming. There are ways to manage this. Often at the weekend when time is available A full backup is created. On the next working day only the changes are stored. Is type of backup is much quicker to carry out. The next day, again only the changes are saved. At the end of the week a full backup is created again. This type of backup is known as grandfather, father, son backup.

The backups should be stored away from the computer, preferably on another the site in a fire proof safe. Another way of backing up is to use a RAID system. The server would have two hard disks. The main disk and another one that is a copy. When changes are made to the main disk these are copied onto the second disk. If the first disk fails the second disk immediately takes over. But failed to drive can be replaced while the computer is switched on. This now becomes the the copy disk, copying the data so that it mirrors the other disk.

Archiving is copying data that is no longer being used onto storage discs. This might be last years financial data. The company needs to keep a copy but does not need to have it taking up space on the network. Archiving data helps to reclaim space on the hard disk.

Network managers have to prepare for disaster recovery. This often means doing things to prevent a disaster. Backing up is one example. Another example is using antivirus software and firewalls to prevent attacks by viruses and unauthorised access from outside. Computer servers are often connected to an uninterrupted power supply (UPS). This is like a large battery that will keep the server running in case of a power cut, giving the server enough time to make sure all the computer systems are shut down properly.

Many computer networks have a main server and a backup server. If the main server stops working then backup server will take over. This is known as failover.

A lot of the systems and procedures are time-consuming and expensive, but generally the loss of data would be more damaging to the company.

Section 2 - The Internet

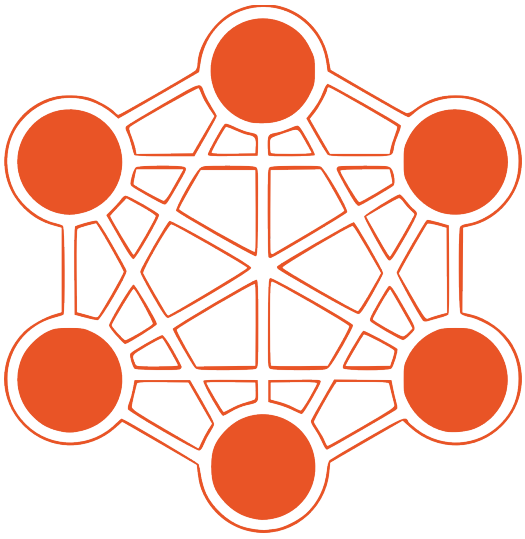
A BRIEF HISTORY OF THE INTERNET

The Internet was created out of the fear of a nuclear war in the USA. In the 1960's the Cold War gripped the super powers and they feared that they would be bombed by their enemies. In the USA they had networks of computers in different cities communicating with different military centres. If one of these centres was bombed then it could disrupt the entire country's communications, severing links to other cities.

The solution was a mesh network. Each communication centre was connected to multiple other centres. If one centre was destroyed the connections to the others would just find another route. This was called ARPANET.

Universities in the USA found ARPANET useful to share data. They asked to be linked in. Gradually over the years more and more networks joined this networks of networks to get the Internet we have today.

The Internet uses a number of different technologies. Email was one of the first tools on the Internet and was a big hit. In was invented in the early 1970's.



the Internet is a mesh network

The Internet today

So the Internet is an enormous network of networks. Today almost a third of the entire Earth's population uses the Internet. We get access to the Internet through Internet Service Providers or ISPs. These companies attach to the Internet proper. We connect with them through

fixed line or mobile phone networks. The Internet is uses a number of technologies to transfer data. Email, World Wide Web, File Transfer, Messaging all use the Internet.

The services that make up the Internet, such as the World Wide Web, are hosted on thousands of computers all over the world. Special servers allow access to data and transfer this from server to computer. The world wide web for example is hosted on web servers by a variety of web hosting companies and private companies. These web servers host the webpages and other files that people want to view.

When you browse the Internet using a web browser, a number of things happen when you request a page. Normally when we request a page we type in a web address or click on a hyperlink to an address. For example if we clicked on a link to **http://www.bbc.co.uk** your web browser is making a request to that computer. Computers however don't have names, they are identified by numbers known as IP addresses. These addresses are stored on special servers called **Domain Name Service** servers. These are a little like telephone directories. The name of the website is looked up in one of the **DNS** servers and finds out the IP address of the web host. The message then gets passed on to that server at the BBC. The web server understands that a web page has been asked for. It then sends this page back to the computer that asked for it.

How do websites get hosted

Websites are hosted on computers which have web server software installed (such as Apache). A folder on the web site contains the pages and images that make up the website. When a request is made, the web server finds the file and send it back to the computer that made the request. Web sites need to be hosted on computers that have static (non changing) IP addresses so that the DNS servers can always find them. Most people's home Internet connection have dynamic IP addresses. This means they change every time they connect to the web. Companies can pay to have static IP addresses and can host their websites. Other people can have companies host their website for them. Web hosts will rent space out on their web servers. A web host could host thousands of different websites. You need to copy the files that make up your website on to the web server. People often use an FTP client to copy the files to the web server from their computer.

To get a web address (domain name) for your website you need to buy it from a domain registry. This is a company that will update the DNS servers with your domain name and the static IP address off your website. **Domain names** are the names given to web servers.

To connect to the Internet you need a modem. This may be combined with a wireless router at home. A modem allows data to travel along the phone lines. Routers are devices that route the data to the correct computer or other router on its way. When you click on a link or enter a web address in your web browser, that makes what is known as a request. That request is carried in a data packet that travels the from your modem to your **ISP**. From there it travels to a DNS server to look up the IP address of the computer where the website is. From there it can travel to the web server itself, again going through a number of routers. When it gets to the web server, it finds the page you requested and sends that back as a data packet to your computer's IP address.

Web Pages

There are billions of web pages on the Internet. These pages are all written in a special language called Hypertext Markup Language (**HTML**).

HTML is the language of the World Wide Web. Every web page that you have looked at is written in this language. HTML is a text based language that tells the web browser (Internet Explorer, Google Chrome and the like) how to display the page.

HTML is made up of **tags** that describe to the web browser how to display the web page. Tags are surrounded by **less than** and **greater than** signs, **< tag >**.

Tags generally come in pairs. One tag opens the formatting and the closing tag ends the formatting. A closing tag has a slash (**/**) to show that it is a closing tag.

Tags

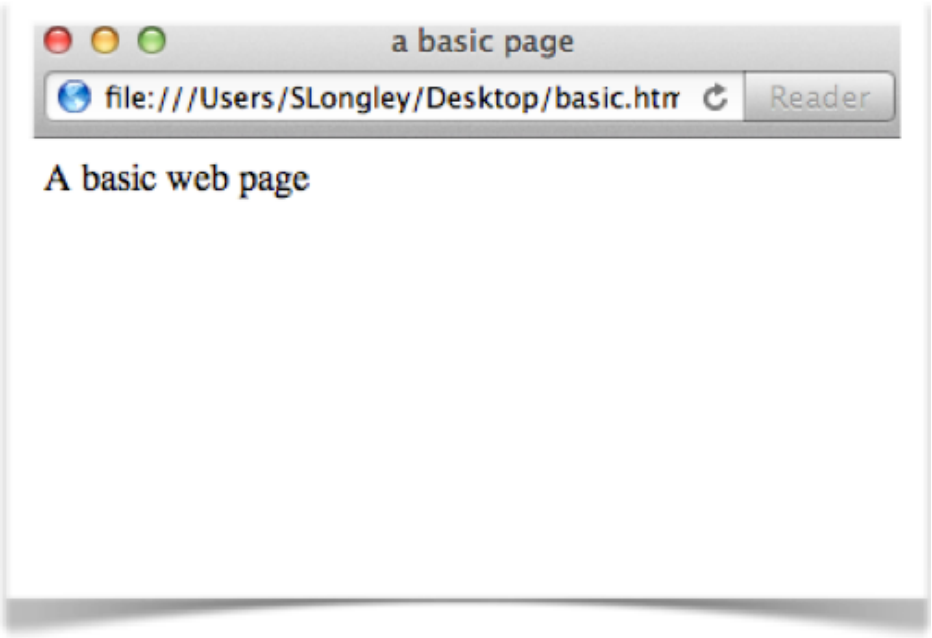
All web pages start with a **<html> tag** and end with the closing **</html>** tag. This tells the web browser that the page is a web page.

Web pages are divided into two sections. The **head** section contains information that is not displayed in the main browser window. This is often JavaScript code that can make the page interactive or style information that can make a page look more colourful. The next section is the **body** section. Everything in this section is displayed inside the browser window.

A basic web page.

```
<html>
<head>
<title>a basic page</title>
</head>
<body>
A basic web page
</body>
</html>
```

The title tag in the head section displays the title of the page on the browser window.



web page

Tags are used to explain to the browser how the content should be laid out. There are tags that create headings, links and lists, there are tags to create tables and insert images and other media elements into the page.

Tag Attributes

Many tags have attributes. These are keywords followed by a value in quotes.

Eg:

```
<table width="70%">
```

The table tag has a width attribute that is set to 70%. This means the table is 70% of the browser window. A common attribute is a tag's id. This can be used by JavaScript to select a tag to change it.

Tag	Description
<h1> .. <h6>	Headings 1 - 6. This tag make text look bold. <h1> is a large heading while <h6> is a very small heading
<a>	An anchor. This tag creates a hyperlink, usually to another page
<table>	Begins a table. These are used to help layout content into rows and columns.

Tag	Description
<tr>	This begins a new table row. Found inside a <table> tag.
<td>	This starts a table data cell. Cells in rows make the table have columns. If a row has two <td> tags inside then it will have two columns. This is found inside a <tr> tag.
<form>	This tells the browser that a data entry form has started. This form will contain tags that help the user input data such as text boxes, buttons and radio buttons.
<input>	This tag is found inside a <form> tag. These create controls for the user to input data. They have attributes that determine the type of control.
	This begins a bullet list (unordered list).
	This begins a new list item. This tag is used inside a .
<img>	This is the image tag and is used to link to an image that will be displayed on the page.

some common html tags

CSS

Another web technology is cascading style sheets or **CSS**. These help provide more style information as to how the browser displays the content. CSS cover things like colours and borders and helps with organising the layout of the content. Often one style sheet can be used to style a series of pages giving them a similar appearance.

Changing a style sheet can be used to dramatically change the appearance of a web page without actually changing the underlying HTML code. This means that web designers can keep their websites looking fresh without having to rewrite all their web pages and often by only altering one style sheet.

The Beauty of CSS Design

A demonstration of what can be accomplished visually through CSS-based design. Select any style sheet from the list to load it into this page.

Download the sample html file and css file

The Road to Enlightenment

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.

Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.

So What is This About?

There is clearly a need for CSS to be taken seriously by graphic artists. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The code remains the same, the only thing that has changed is the external .css file. Yes, really.

CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way that gets people excited is by demonstrating what it can truly be, once the reins are placed in the hands of those able to create beauty from structure. To date, most examples of neat tricks and hacks have been demonstrated by structurists and coders. Designers have yet to make their mark. This needs to change.

Participation

Graphic artists only please. You are modifying this page, so strong CSS skills are necessary, but the example files are commented well enough that even CSS novices can use them as starting points. Please see the CSS Resource Guide for advanced tutorials and tips on working with CSS.

You may modify the style sheet in any way you wish, but not the HTML. This may seem daunting at first if you've never worked this way before, but follow the listed links to learn more, and use the sample files as a guide.

A web page without CSS styling

The same page with CSS styling

The same page with different CSS styling

JavaScript

To make web pages interactive a programming language must be used. **JavaScript** is a script based interpreted language that can be embedded in webpages to allow for interactivity and animation. A combination of HTML, CSS and JavaScript can be used to create complex interactive websites and even games. A version of Angry Birds and Cut the Rope is playable online and all programmed using HTML, CSS and JavaScript.

angry birds online

FILES OF THE INTERNET

Web pages are pure text files. The HTML, CSS and JavaScript that make up web pages can be viewed using any simple text editor. It is the web browser than converts the tags into the

formatting we see when we view a page. So what about things like images, sounds and videos? These files are linked to inside the HTML code. When the web browser finds these links it requests them from the web server and when they have been downloaded, inserts them into the page.

There are some special file formats that have been developed for use on the Internet. Usually these formats are compressed in some way to speed up download times.

File Format	Use	Comments
JPG	Photographic images	Lossy compression. Does not support transparency
GIF	Simple image like logos	Lossless compression. Can have transparent pixels and animations. Can only have maximum of 256 colours.
PNG	Photographic images and logos	Lossless compression. Can have 256 levels of transparency and support true colour images.
PDF	Documents	Used to share documents that keeps the page layout.
MP3	Sound, eg music and podcasts	Lossy compression.
MPEG	Video	Lossy compression.

files used on the Internet

COMPRESSION

Sending large files over the Internet takes time. Back when most people had dial up modems to connect to the Internet, it was very slow to download a web page, 30 seconds or more for a page with lots of images. Making sure that the image file sizes were small helped to make sure that these download times were as low as possible.

Nowadays most people have broadband and pages generally load pretty fast. However there is an increasing number of people access the Internet on a mobile phone and file sizes not only affect speed of download but also affect peoples data limit.

Compressing a file means finding a way to save the same data into a smaller space. One way to do this is to use an algorithm to search for repeating patterns. In many files there is data that is repeated. Instead of saving all the data multiple times in a file it is just saved once. When the data is repeated a code referring to the data is used instead.

a close up of an uncompressed image

the same image compressed

When reducing the size of images or sound files, the algorithms rely on what we can perceive. A digital photograph will have many different shades throughout the image. By looking at pixels in close proximity the algorithm can decide to make two pixels of similar shade into a larger area of the same shade. This saves space. The same goes for sound. If we cannot perceive particular sounds then that data will not be saved.

When videos are saved for the Internet, instead of saving each frame of video the file will store a keyframe every second or so. In the frames between each key frame only the differences are stored. This saves a great deal of space.

Types of Compression

There are two types of compression lossless and lossy. These words describe what happens to the data when a file is compressed.

Lossless compression means that no data is lost in the process and when the file is decompressed, all the data that was in the original file is present. Image files such as GIFs and PNGs use lossless compression.

When **lossy compression** is used data is lost in the process and cannot be retrieved. When a JPG image is saved data is lost as the compression algorithm throws out colour data when pixels of similar shades are combined. When audio files are saved as MP3s sound data is lost, making the file smaller. Lossy compression relies on our inability to notice when some of the data is lost. If files are compressed again and again, each time data is lost and we start to notice that data is missing. Sound files can sound muffled as high frequencies are lost or images seem pixelated.

Chapter 7

Programming

Section 1 - Programming Languages

What are Programming languages?

Programming languages enable users to create programs that run on computers to perform a task. The problem is that processors only understand instructions in binary. This is very difficult for us to understand. For example:

10110000 01100001

This binary instruction tells an Intel chip to Move the value to memory address 97. Its quite difficult for humans to read and understand. It is very easy to make a mistake programming in this way.

Assembly language was developed to be readable by humans but has a direct binary equivalent. So each instruction can be easily converted into a single binary instruction.

MOV AL, 61h

This is the same instruction in assembly language.

While more readable by humans, it is still quite hard for the non-technical person to understand and is specific for the computer chip in the computer. A different computer chip has a different assembly language. Machine code binary and Assembly languages are **low level languages**. This means that they are close to the instructions the computer needs to operate. Low level languages deal with registers and specific memory address. These are specific to particular processors and computer systems. Instructions written for one processor may be different for a different processor.

High Level Computer Languages

Higher level languages have been designed to be easier to understand by humans and easier to write programs. Instead of using memory addresses, they use variables as a way of easily remembering where data is stored. They can also be used to write programs that run on different processors. The high level language needs to be translated into machine code before it can run. This is done by a special program called a **compiler** or an **interpreter**.

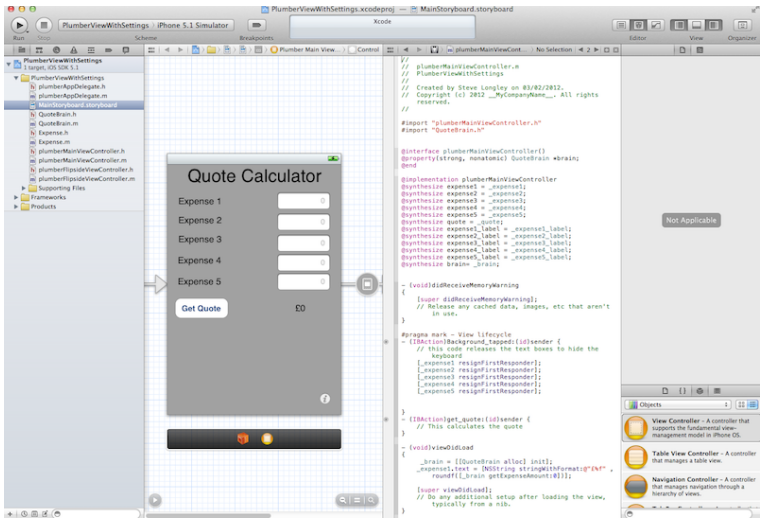
An interpreter turns the instructions from the high level language into instructions a processor can understand one at a time then executes it. The code is then discarded and needs to

be interpreted again if it is to run. **Javascript**, a language used to make webpages dynamic is an example of an interpreted language.

A compiler turns the instructions from the high level language into instructions a processor can understand just like an interpreter, but it stores all these instructions to create a compiled program. **C** is an example of code that is compiled. Many programs that you use every day are written in C, like Windows or your web browser. As the code is only translated into machine code once, compiled code runs faster than interpreted code.

With both compiled and interpreted code, as there is no direct link between the computer language and the machine code. As a result the code often can be slower than machine code or assembly code. As processors are very fast this tends to be less of an issue.

Special tools have been written to help the programmer to write and test the programs they write. These are **Integrated Development Environments** or IDE's. An **IDE** often is a group of tools that the programmer can use to write computer programs. It contains an editor to write computer code. The keywords of the language are often highlighted to make the code more readable. They often also enable the user to design graphical user interfaces (GUI's) for their programs. The programs can be tested and debugged (checked to remove errors). When the program is completed the IDE allows the user to compile and distribute their finished application.



X-CODE is an IDE for making OS and iOS apps

PYTHON

This book is going to use the programming language called **Python** to illustrate examples and so you can follow along. There are a number of reasons for choosing this language.

- Its free.
- It runs on a wide variety of computer systems including Windows, Mac OS X and Linux.

- Its simple to learn but very powerful.

Python is a free interpreted language. You can download it from http://www.python.org/download/. It is easy to install, and comes with a simple IDE called IDLE. We will use this to write, test and debug computer programs. We are going to use version 3.3 of Python for this book.

Just because it is easy to get started does not mean that Python is not a serious language and that you cannot do serious work with it. Many people use Python in their work and is capable of running websites, standalone applications and even games.

You should be a able to do all your computer programming assignments in Python.

Most of the programs we will write will be programs that do not have a graphical user interface. We will look at the end how we can create a GUI for our programs if you need to.

Section 2 - Algorithms

What is an Algorithm?

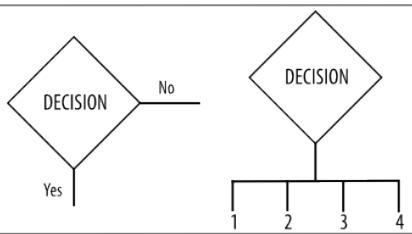
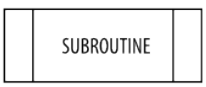
Computers work by following sets of clearly defined rules to process data. These rules are called algorithms. Devising a set of algorithms to solve a problem is the most important part of developing a solution to a problem. Well-defined algorithms will lead to well constructed and effective programs.

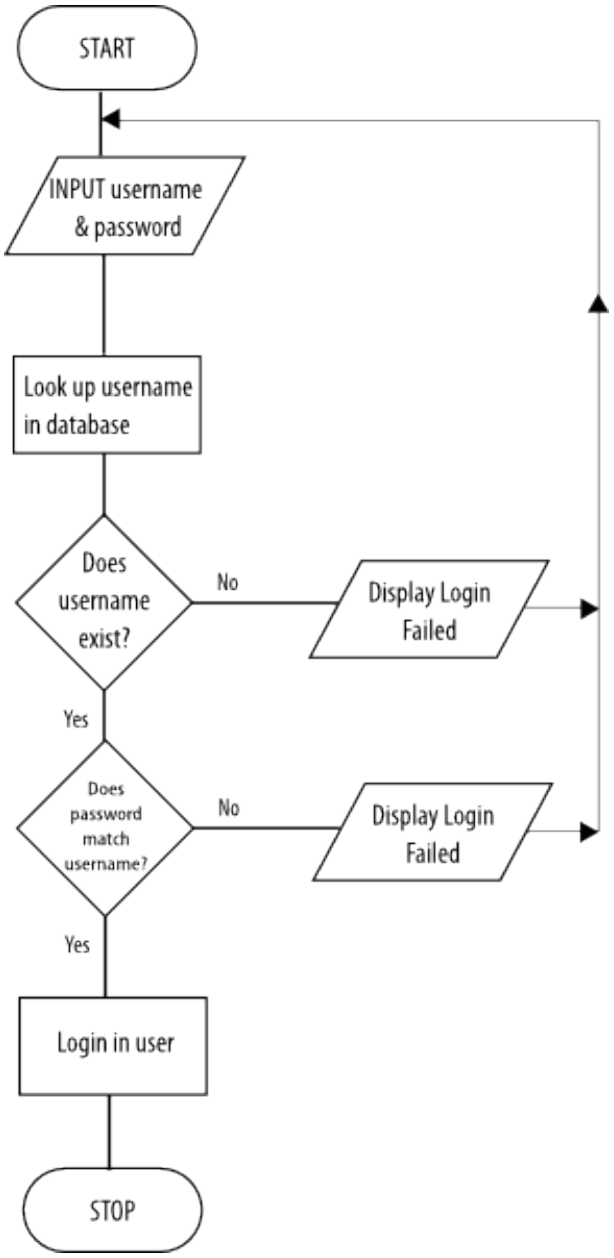
Algorithms are simply sets of rules and can be defined in many ways, including program flowcharts or pseudocode. We often apply simple sets of rules when making decisions, for example:

- Enter username
- Enter password
- Does username exist in database?
- If Yes then does password match the one for the username in the database?
- If Yes login computer.
- If No display error
- If No display error

Flow Charts

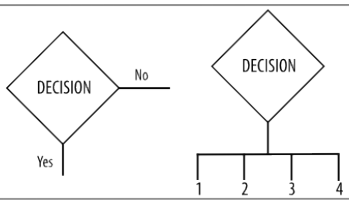
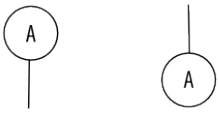
This algorithm can be displayed as a flowchart.

	This begins and ends the algorithm
	Gathers input from the user or provides them with an output such as a message box.
	Describes a process that happens such as a calculation. Eg Calculate VAT, Shuffle cards, Load file.
	Allows a decision to be made. Either a yes/no question (an IF statement) or multiple choice (CASE statement).
	A subroutine that can be called to perform a particular task. Eg sorting a list.
	A continuation showing that a flow chart follows on a different page or continues from a different page.



flow chart

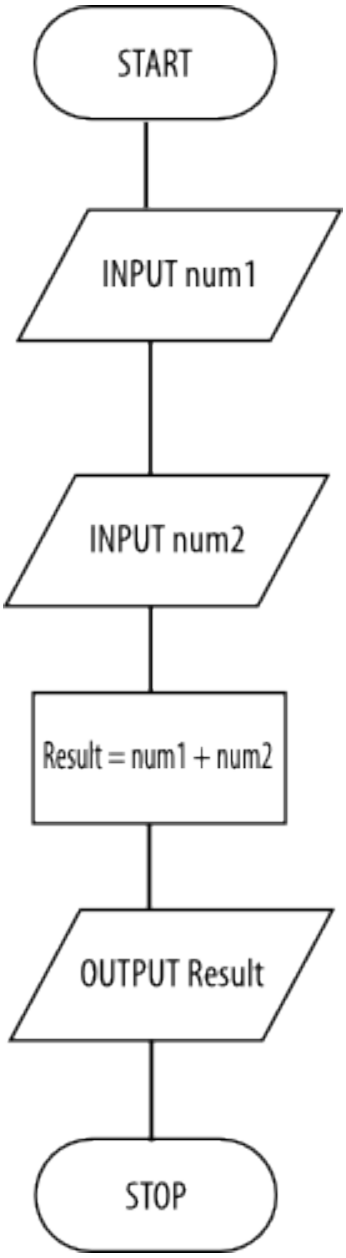
Flow charts can make it easier to follow the algorithm. Checking an algorithm works is a key process in designing a good computer program and flow charts can help in this process.

	This begins and ends the algorithm
	Gathers input from the user or provides them with an output such as a message box.
	Describes a process that happens such as a calculation. Eg Calculate VAT, Shuffle cards, Load file.
	Allows a decision to be made. Either a yes/no question (an IF statement) or multiple choice (CASE statement).
	A subroutine that can be called to perform a particular task. Eg sorting a list.
	A continuation showing that a flow chart follows on a different page or continues from a different page.

flow chart symbols

Examples:

Adding two number together. This simple flowchart asks the user for two number which are identified as num1 and num 2. These numbers are added together and store the answer in Result. The Result is output to the user.



adding two numbers flow chart

PSEUDOCODE

Pseudocode is instructions written in English that are formatted and structured like computer code. This pseudocode isn't computer code however but can often help in converting an algorithm into computer code as the structures are often similar.

Using the last example we could write the algorithm as:

```
INPUT num1
INPUT num2
Result = num1 + num 2
OUTPUT Result.
```

Developing Algorithms

Solving problems is the key to being able to program. Learning a programming language is fairly straight forward, but learning how to use it to solve a problem is more difficult. We shall look at some strategies for looking at a problem, breaking it down into more manageable chunks and what techniques we can use to solve a problem in an efficient manner.

Computers can only follow one instruction at a time, so when we look at a problem we need to be able to break it down into simple steps, so each step is a simple instruction. This sound straight forward but can be difficult as we often perform many instructions in our head without realising it. We also make assumptions that a computer cannot do.

- When looking at a problem there are some questions need answering.
- What data do I need to solve the problem? What type of data is this?
- What data do I know already and what data do I need to get from the user?
- What decisions need to be made in solving the problem?
- Can I write the questions as True / False questions?
- What happens if the question is True?
- What happens (if anything) if the answer is False?
- Do any of the steps need repeating?
- If there are steps that need repeating, do I know before hand or can I calculate how many times it needs repeating?
- Do the steps need repeating as long as some condition is true?
- Can I write the algorithm so it can be reused in different circumstances?
- What outputs are needed?

Simple Example

We will look at a simple problem first. We will go through the questions and then use the answers to help develop an algorithm.

Here is the problem.

> A dog that is 5 years old is equivalent to a 42 year old human. Ashok is writing a program which converts the age of a dog to the equivalent age for a human.
>
> The program uses the following method:
>
> The user inputs age of the dog in years
>
> If the age is 2 or less, the human equivalent is 12 times the age
>
> If the age is more than 2, the human equivalent is 24 for the first 2 years, plus 6 for every additional year.
>
> Write an algorithm to calculate and output the human equivalent of the age of a dog using the method described.

So using this we will go through and answer our questions before developing our algorithm.

What data do I need to solve the problem? What type of data is this?

We need two know two pieces of data, the dogs age in human years and the age in dog years. Both these pieces of data are whole numbers.

What data do I know already and what data do I need to get from the user?

The dog's age in human years we need to get from the user. the dog's age in dog years we need to calculate.

What decisions need to be made in solving the problem?

Can I write the questions as True / False questions?

What happens if the question is True?

What happens (if anything) if the answer is False?

We need to be able to ask if the dog's age is less than 2. If it is True then the dog's age in dog years is 12 times the age entered.

If it is false then the dog's age is 24 + ((age - 2) * 6).

Another question that needs asking, is did the user actually enter numerical data when asked. If they didn't this would cause problems in our calculations.

Do any of the steps need repeating?

If there are steps that need repeating, do I know before hand or can I calculate how many times it needs repeating?

Do the steps need repeating as long as some condition is true?

In this simple example we don't need to repeat any code. However when we check to see if they user has entered numerical or not, if they have not we need to keep asking until they do.

Can I write the algorithm so it can be reused in different circumstances?

It is possible to write the part of the program that asks the user to input a number so that we could reuse that in other programs.

What outputs are needed?

We need to know the dog's age in dog years.

Now we have answered those questions we need to put them together in a logical order to solve the problem.

We need to:

Ask the user for the dog's age

We need to check it is a number. If not we display an error. If it is we continue.

If the dog's age is less than or equal to 2 years then dog_age = 2 * age

Else dog_age = 24 + ((age - 2) * 6)

Print dog_age

As pseudocode it would be:

```
SET age = 0
SET dog_age = 0
INPUT age
IF age IS NUMERIC
     IF age <=2 THEN
          dog_age = 2 * age
     ELSE
          dog_age = 24 + ((age-2 * 6)
     PRINT " Your dog is " + dog_age + "years old"
ELSE
     PRINT "You need to enter a numeric age"
```

Complex Example

We will use one of the OCR sample problems as an example. This problem is complex enough to be able to illustrate all the steps listed above.

Here is the problem.

Design, code test and evaluate a system to accept and test a password for certain characteristics.

It should be at least 6, and no more than 12 characters long

The system must indicate that the password has failed and why, asking the user to re enter their choice until a successful password is entered.

A message to indicate that the password is acceptable must be displayed.

Password strength can be assessed against simple criteria to assess its suitability; for example a password system using only upper and lower case alphabetical characters and numeric characters could assess the password strength as:

WEAK if only one type used, eg all lower case or all numeric

MEDIUM if two types are used

STRONG if all three types are used.

For example
hilltop, 123471324, HAHGFD are all WEAK,
catman3 and 123456t are MEDIUM
and RTH34gd is STRONG

A message to indicate the password strength should be displayed after an acceptable password is chosen.

So using this we will go through and answer our questions before developing our algorithm.

What data do I need to solve the problem?

I need to know the password that I am going to test. I also think I need to know the message I need to tell the user at the end. Both these pieces of data can contain text and so they must be strings.

I also want to have some data that stores whether the password contains lowercase letters, uppercase letters and numerical characters. These only need to store yes or no, so a boolean is best here. These types of data are often known as flags.

What data do I know already and what data do I need to get from the user?

I obviously need to get the password data from the user. The message will be constructed in the program.

The various flags need to be set to False in the program and then set to True later.

What decisions need to be made in solving the problem.

Can I write the questions as True / False questions?

What happens if the question is True?

What happens (if anything) if the answer is False?

Looking at the issue of decisions. The whole program is about making a decision based on the composition of the password.

To be able to program the algorithm we need to be able to construct the questions as True/ False questions.

One question that ought to be asked first is did the user enter a password at all. If not then the program should ask the user to enter one.

The questions we need to ask are:

Is the password length greater than or equal to 6 characters and less than or equal to 12 characters? If not then show an error message.

Does the password contain uppercase letters?

Does the password contain lowercase letters?

Does the password contain numerical characters?

By answering the last three questions we can set the various flags. These are then used to answer the following questions. We will check for the strong password first. In this case we need to see if **all** the flags have been set to true. As we want all the flags to be on we need to use the AND logical operator.

If lowercase = True AND uppercase = TRUE AND numerical = TRUE Then password is STRONG

Next we need to check for medium passwords. This is a bit tricker as **any** pairs of flags can be true to be a medium strength password. We only really need to ask this question if the answer to the last question was False. We need to check to see if any combination of flags are True. We do this using a combination of AND and OR operators. Only one of the bracketed **AND** statements needs to be True for the password to be medium strength.

If (lowercase = True AND uppercase = True) OR (lowercase = True AND numerical = True) OR (uppercase = True AND numerical = True) Then password is MEDIUM

Finally we need to check for the weak password. Again, we really only need to answer this question if the answer to the other questions was False. As this is the last question we could just display the answer weak.

Do any of the steps need repeating?

If there are steps that need repeating, do I know before hand or can I calculate how many times it needs repeating?

Do the steps need repeating as long as some condition is true?

There are perhaps no obvious steps that need repeating. One thing that does spring to mind is how exactly do we find out if the characters in the password are indeed upper or lower case or numerical. Some computer languages have built in methods to check these sort of things. If not we need to take each letter in the password in turn and then check to see if the letter fits into fits the criteria we are looking for. To do this we need to loop through each letter. As we can calculate the length of the password we need a counting loop to go through each letter in the password.

We also would need to keep asking for an input until the user actually enters a value.

Can I write the algorithm so it can be reused in different circumstances.

Nothing immediately springs to mind here, though again we could write the algorithm as a function that could be reused.

Putting it all together

Taking all the answers we have we need to put together an algorithm. The algorithm needs to run in a logical order. We need to deal first with getting input from the user and making sure they have actually entered a value. We then need to ask our various questions, first to set the flags and next to check the strength of our password. Finally we need to display the message to the user.

```
SET password = ""
SET strength = ""
SET lowercase = False
SET uppercase = False
SET numerical = False
WHILE password = ""
   PRINT "Enter password"
   INPUT password
   IF password IS EQUAL TO "" THEN
        PRINT "You need to enter a password"
IF password LENGTH >=6 and password LENGTH <=12 THEN

IF password CONTAINS number THEN numerical = TRUE
IF password CONTAINS lowercase_letter THEN lowercase = TRUE
IF password CONTAINS uppercase_letter THEN uppercase = TRUE
IF lowercase = True AND uppercase = TRUE AND numerical = TRUE THEN
strength = "Strong"
```

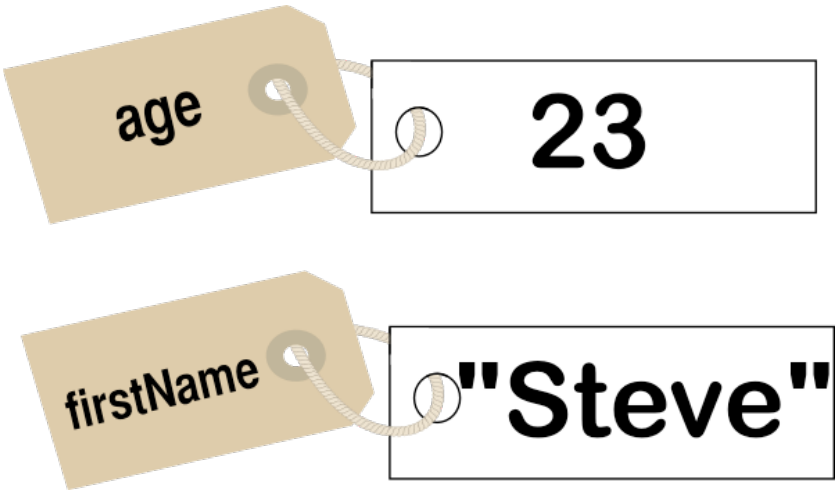
```
   ELSE IF (lowercase = True AND uppercase = True) OR (lowercase = True
AND numerical = True) OR (uppercase = True AND numerical = True) THEN
strength = “medium”
   ELSE strength = “weak”


   PRINT “your password is ” + strength
   ELSE
      PRINT “Your password needs to be between 6 and 12 characters.”
```

Section 3 - Variables and Data types

WHAT IS A VARIABLE?

When writing computer programs we often need to store data. This is done by using variables. A variable is a name we give to a piece of data and we can use that name in our program as a way of getting that data. You can think of variables as a label for our data. When we use the label, the computer program gets the data with that label.



variables are labels for data

Look at the computer code below.

We have four variables. Each one has a name such as FirstName, Age and Height. These can be used again in our program. This code can be typed in the Python Shell. This is an interactive window that allows us to test code out. The equals sign is used to assign a value to the variable. So in the example above the value 44 is stored in the Age variable. When we use the name Age in our program, the value that is stored in it will be used.

Variables can be named anything according to the following rules.

- They should start with a letter
- They should not contain spaces
- They should not be keywords used in the language (so words like IF, ELSE and OR should not be used.
- They should not have any punctuation except an underscore (_)

Data Types

Variables store different types of data. There are four different types of data variables can store.

- **Integer** - whole numbers, including negative ones. Python calls these **int**.
- **Real** - numbers with decimals including negative ones. Python calls them **float**, short for floating point.
- **String** - string of characters. These can be letters, numbers and punctuation marks.
- **Boolean** - True or False

So for example:

- `Shoesize = 7` : Shoesize is an integer
- `smoker = False` : smoker is a boolean
- `AverageWeight = 65.4` : AverageWeight is a real (float)
- `myPosition = "Advisor"` : myPosition is a string

The reason there are different data types is because each one requires a different amount of RAM to store. The programming language needs to know what data that variable is going to accept and so how much RAM to set aside for it.

As well as the amount of RAM required the programming language needs to know what type of data a variable is holding because different things can be done to different types of data. Numbers (integers and real numbers) can be used in calculations where as strings are not. Strings however can be manipulated in different ways. They can be stuck together to create longer strings of text. They can be split up or searched through. The data type helps tell the programming language how to deal with it.

Lists

Sometimes we have similar data that we want to deal with as a group. We could for example want to have a set of variables to hold a set of ice cream flavours.

We could create a set of flavour variables like this:

```
flavour1 = "vanilla"
flavour2= "strawberry"
flavour3= "chocolate"
```

Another way of storing them is as a list. In other programming languages Python lists are known as **arrays**. To create a list we place the items in square brackets with commas separating each item in the list.

```
flavours = ["vanilla","strawberry","chocolate"]
```

We can access each item in the list using its index. Each item has an index starting at zero. So the first item is:

```
>>> flavours = ["vanilla","strawberry","chocolate"]
>>> flavours[0]
'vanilla'
```

and that would return the value **"vanilla"**.

Lists and arrays are often used in conjunction with loops, both to fill the list and to access the data. The following code creates a deck of cards. It uses a list for the suit and another for the value and then creates a list called deck to store the cards.

Section 4 - Variables and expressions

Assignment and Mathematical Expressions

In programming variables are used when you want to store data for use later on. Before a variable can be used it must be assigned a value. We use the equals sign to assign a value. You can enter the code in Python Shell in IDLE to try these out.

In the example above the variables are assigned values. The value assigned to the variable determines the data type stored. So celsius is an integer data type because a whole number was assigned to it. As the variable pi has been assigned a number with a decimal it will be a float.

Variables can be used in expressions.

For example:

```
celsius = 37
pi = 3.14
firstAttempt = True
studentName = "Sophie"
print (celsius)
print (pi)
print(firstAttempt)
print(studentName)
```

When assigning values to variables, the expression to the right of the equals sign is evaluated and the result is assigned to the variable. The variable area is assigned the result of height multiplied by width (10 x 2). In programming the asterisk (*) is used to signify multiply and the forward slash (/) is used for divide. So in the example above area is equal to 20.

```
height = 10
width = 2
area = height * width
print(area)
```

Variables will store their data until the end of the program or function. They will only change when their value is reassigned.

Look at this example.

```
cost = 10
total = cost
cost = 12
print(total)
print(cost)
```

The variable **cost** is assigned the value 10. The variable **total** is assigned the value of the **cost** variable. So **total** will be 10. The variable **cost** is now assigned the value of 12. What do you think the value of **total** is? Remember variables will only change when their value is reassigned. The variable **total** has not had its value reassigned so it is still equal to 10.

As we've said when assigning values to variables, the expression to the right of the equals sign is evaluated and the result is assigned to the variable. Python understand basic arithmetic. Lets take a look at some examples.

```
baseFare = 2.5
myFare = baseFare + 10
print(myFare)
```

The variable **myFare** will be 12.5.

```
total = 200
numberOfStudents = 10
AverageScore = total / numberOfStudents
print(AverageScore)
```

The variable **AverageScore** will be 20.

Python understands the order of mathematical operators. Look at the following example.

```
MyScore = 5 + 2 * 10
print(MyScore)
```

`MyScore` will be 25 and not 70. This is because multiplication and division happen before addition and subtraction. Remember **BIDMAS** from maths lessons?

B

Brackets first

I

Indices (ie Powers and Square Roots, etc.)

DM

Division and **M**ultiplication (left-to-right)

AS

Addition and **S**ubtraction (left-to-right)

If we need to do more complex maths we can use brackets to make things happen in the order we want. For example:

```
fahrenheit = 212
celsius = (fahrenheit - 32) / 1.8
print(celsius)
```

So this results in the value 100 being assigned to the variable **`celsius`**.

In Python to use powers we use ** . So to calculate 8^2 we would write that as 8**2. To calculate 10^5 we would write 10**5.

There is another operator used in Python that you may not have come across in maths and that is modulus. Modulus is used to calculate the remainder when dividing. In Python, the symbol used for modulus is %. For example:

```
7 % 2
```

would evaluate to 1. It means 7 divided by 2 is 3 remainder 1.

A variable can be used to in an expression to reassign its own value. Look at the following:

```
counter = 1
counter = counter + 1
print(counter)
```

This calculates **counter + 1** and then assigns the result to **counter**, so that **counter** now becomes 2.

There is a shortcut if we want to add a value onto a variable. We can use **+=** to add a value.

So counter += 1 is the same as counter = counter + 1 and counter += 10 is the same as counter = counter + 10.

Comparison Expressions

Often in programming we want to do something depending on the outcome of a comparison (*see section 6 for some examples*). When we compare two values Python checks to see if the comparison is **true** or whether it is **false**. We can then use this true or false to alter the flow of our program. Before we get into that however we are going to look at how we can compare values in Python. Most of the time you will be comparing a variable to a known value or comparing two variables. You can write the code in the Python Shell in the IDLE interface.

Checking for equality

In Python a single equals sign is used to assign a value to a variable. So if we were to write **myHeight = 1.8**, will assign the value of **1.8** to the variable **myHeight** and whatever myHeight was before will be overwritten. So how do we check if two values are equal? We use two equal signs together. So the statement **myHeight == 1.8** is actual comparing myHeight with 1.8 and checking to see if they are equal. If they are equal then Python returns the value **True** and if they are not equal Python returns the value **False**.

Try the following in the Python Shell.

```
area = 25
area == 25
area == 30
```

The first line sets the area variable. The second line compares the value stored in the **area** variable and compares it with the value 25. So Python finds the value in the variable, in this case 25, and compares it with the value 25 to check if they are equal. They are so Python returns the value **True**. The last line compares the area variable with the value 30 to see if they are equal. As the value stored in area is 25 and not 30 Python returns the value **False**.

Try the following.

```
bottles = 5
maxBottles = 6
bottles == maxBottles
bottles = bottles + 1
bottles == maxBottles
```

At the start bottles is 5 and so when bottles is compared with maxBottles they are not equal. After the number of bottles is increased by 1 the second comparison shows the two are now equal.

CHECKING FOR INEQUALITY

To check to see if two values are NOT equal we don't use the ≠ symbol as it is too hard to type in normally. Instead we use **!=** to mean not equal. Try the following.

```
myTeacher = "Mrs Smith"
myTeacher != "Ms Jones"
myTeacher != "Mrs Smith"
```

So when we first check to see if myTeacher is not equal to Ms Jones we get True and the second comparison is False.

CHECKING GREATER THAN AND LESS THAN

We can also check to see if a value is greater than or less than a value. We use the maths symbol **<** to mean less than and the symbol **>** to mean greater than. Try these examples:

So when we see if bottles is less than maxBottles we find that this is true. That means the next statement, bottles greater than maxBottles is false. When we add 1 to bottles the comparisons are both false as bottles is equal to maxBottles. To check to see if something is **less than or equal** to something we use **<=**. Similarly we use **>=** to check if a value is **greater than or equal** to another value.

```
bottles = 5
maxBottles = 6
bottles < maxBottles
bottles > maxBottles
bottles = bottles + 1
bottles < maxBottles
bottles > maxBottles
```

Section 5 - Writing Simple Programs

PROGRAM 1 - BMI CALCULATOR

All programs begin with a problem that needs to be solved.

Problem

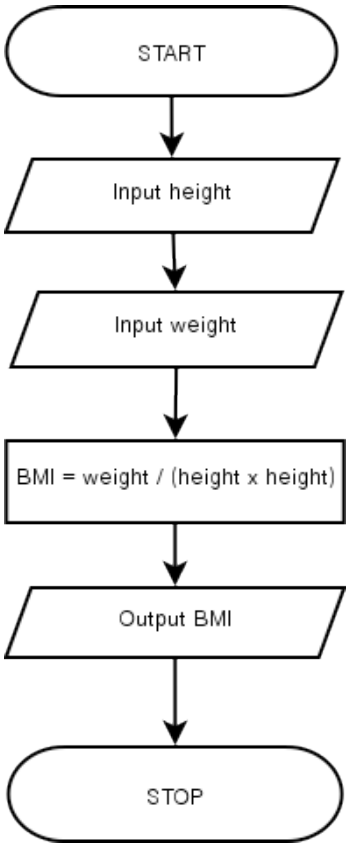
A gym want a BMI calculator to see what a gym member's body mass index is. This will help them to calculate whether the user is overweight or not.

This needs the users height in metres and weight in kg.

To calculate the BMI is weight divided by the height squared *(weight / (height x height)*.

The Algorithm

We need to design our algorithm to calculate the BMI. So firstly we need to ask the user to enter their height and then their weight. Once that has been done then we can calculate the the BMI. We can then create a message to give feedback to the user and then show that message.



Design the Code

Use the Python IDE called IDLE and start a new window (File > New Window). Save the file as **BMI_Calculator.py**

1. Setting up the variables

We need a variable to store the height and the weight. We also need one to store the BMI. We will also build a message to display to the user. The height, weight and BMI can have decimals and so the variables need to be able to store **real** numbers. In Python a **float** stores numbers with decimals. The message will contain words and numbers and so needs to be a **string** data type.

Code

Write this code to declare the variables they need.

```
myHeight = 0.0
myWeight = 0
myBMI = 0.0
myMessage = ""
```

2. Getting the user input

Now we need to get user input. We can use **`input`** to get data from the user. This will return data in string form. When we want data that is a number we must first convert it. We can use **`float()`** or **`int()`** to convert the string to the appropriate data type. We need to use real number so we will use **`float()`**.

Code

Copy this under the variable declarations.

```
myHeight = float(input("please enter your height in metres: "))
myWeight = int(input("Please enter your weight in kg: "))
```

The code **`input("Please enter your height in metres:")`** asks the user for their height and stores the result as a string. Using **`float()`** around this converts the data into a float data type.

The code **`input("Please enter your weight in kg: ")`** asks the user for their weight and stores the result as a string. Using **`int()`** around this converts the data into a float data type.

3. Calculating the BMI

Now we need to calculate the BMI. I will use brackets to make sure the height is squared first before we divide the weight.

Code

```
myBMI = myWeight/(myHeight * myHeight)
```

4. Displaying the result

We will construct a message we want to display then print the message to the screen.

We can use % as a place holder for our variable. We know our BMI is a float data type and so we can use %f to insert the BMI into our message.

Code

```
myMessage = "Your BMI is %f" % myBMI
```

So the %f in our code will be replaced by the myBMI variable.

This code prints out the message on the screen for the user.

```
print (myMessage)
```

Finished BMI Calculator Code

```
myHeight = 0.0
myWeight = 0
myBMI = 0.0
myMessage = ""

myHeight = float(input("please enter your height in metres: "))
myWeight = int(input("Please enter your weight in kg: "))

myBMI = myWeight/(myHeight * myHeight)

myMessage = "Your BMI is %f" % myBMI

print (myMessage)
```

Testing

We can test our program with the following data.

Test Data	Expected BMI
Height 1.6 Weight 70	27.34375
Height 1.8 Weight 80	24.691358025

Program 2 - Celsius converter

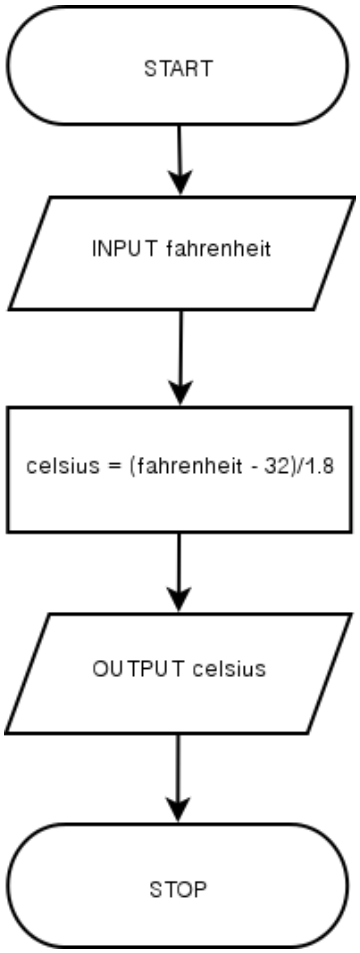
The problem

We want a program that converts a temperature from fahrenheit and converts it into celsius. The formula to make the conversion is:

$$\text{Celsius} = (\text{fahrenheit} - 32) \div 1.8$$

The Algorithm

We need to break down the problem so that it makes easier to program.



Writing the program

We need two variables to store the temperatures, one for celsius and one for fahrenheit. The fahrenheit variable will take a whole number from the user and so will be an integer. The celsius variable needs to store real numbers as our formula will probably calculate a decimal. So we need to make sure our celsius variable is a float. We will also create a message variable so we can feedback properly to our user.

```
fahrenheit = 0
celsius = 0.0
message = ""
```

We then need to ask the user to input the temperature in fahrenheit. We can use **`input()`** to do this. Remember that the result of an input is always a string type. We need to convert it into a integer type. We can use **`int()`** to do this.

```
fahrenheit = int(input("Please enter the temperature in fahrenheit: "))
```

We can now use this in a formula to calculate the celsius.

```
celsius = (fahrenheit - 32) / 1.8
```

Now that we have our temperature in degrees celsius we can construct a message to display. We will use **`%`** as a place holder to put in the variables. As fahrenheit is an integer we need to use **`%i`** while celsius is a float data type and so we must use **`%f`**. This will output with six decimal places. We can write **`%0.1f`** to display the number.

```
message = "The temperature %i degrees fahrenheit is %0.1f degrees celsius" % (fahrenheit, celsius)
```

We now can show the user the message using the **`print()`** command.

```
print(message)
```

So the whole program is as follows:

```
fahrenheit = 0
celsius = 0.0
message = ""

fahrenheit = int(input("Please enter the temperature in fahrenheit: "))

celsius = (fahrenheit - 32) / 1.8

message = "The temperature %i degrees fahrenheit is %0.1f degrees celsius" % (fahrenheit, celsius)

print(message)
```

Save the file as **celsius.py** and run.

Test Data

We can test the program by using the following test data.

Temperature in °F	Expected outcome
212	The temperature 212 degrees fahrenheit is 100.0 degrees celsius
0	The temperature 0 degrees fahrenheit is -17.8 degrees celsius
100	The temperature 100 degrees fahrenheit is 37.8 degrees celsius
-40	The temperature -40 degrees fahrenheit is -40.0 degrees celsius

Section 6 - Writing Complex Programs

SELECTION STATEMENTS

All the programs we have written so far have begun at the start and then followed each line of code in turn and then finished at the end.

BMI Again

We are going to return to our BMI program and alter it to make it more informative. As a reminder the code we wrote was as follows.

```
myHeight = 0.0
myWeight = 0
myBMI = 0.0
myMessage = ""

myHeight = float(input("please enter your height in metres: "))
myWeight = int(input("Please enter your weight in kg: "))

myBMI = myWeight/(myHeight * myHeight)

myMessage = "Your BMI is %f" % myBMI

print (myMessage)
```

The program inputs your height and weight and the calculates your BMI and then tells you what it is. This is all well and good, but most of us don't know if a particular Body Mass Index is good or bad. When need some more information to tell us. If an adult BMI is greater than or equal to 25 then you are overweight. If it is below 18.5 then the person is underweight. If the BMI is between these numbers then the person is a healthy weight.

IF STATEMENT

An IF statement allows us to run a piece of code only when a condition is true. If the condition is false then the code will be skipped. Lets have a look at the structure of an IF statement.

```
IF condition statement :
    code that should run if the condition is true
    goes here
```

An IF statement has a condition statement that must evaluate to either true or false. If the statement is true, then all the code that is indented under the IF statement is executed. If the condition is false then Python skips that code until it gets to the next line which is not indented. Notice the colon at the end of the line. This tells Python that it is the end of the statement.

We want to add some text to our message so that it tells the user if they are overweight if the BMI is greater than or equal to 25. The condition statement would be

```
myBMI >= 25
```

This code would be true when myBMI is 25 or greater. If myBMI is less than 25 then the condition would be false. So we want to write a statement that checks this condition and if it is true adds some text to our message to tell the user that the they are overweight.

```
if myBMI >= 25:
    myMessage = myMessage + ". You are overweight."
```

Remember that the line **`myMessage = myMessage + ". You are overweight."`** is indented. This tells Python to run this line when the statement **`myBMI >= 25`** is **`True`**. This code needs to go just before the **`print(myMessage)`** statement.

```
myHeight = 0.0
myWeight = 0
myBMI = 0.0
myMessage = ""

myHeight = float(input("please enter your height in metres: "))
myWeight = int(input("Please enter your weight in kg: "))

myBMI = myWeight/(myHeight * myHeight)
myMessage = "Your BMI is %f" % myBMI

if myBMI >= 25:
    myMessage = myMessage + ". You are overweight."
print (myMessage)
```

What about those who have a low BMI and are under weight. You are underweight if your BMI is lower than 18.5. We can add these lines after our first IF statement:

```
if myBMI < 18.5:
    myMessage = myMessage + ". You are underweight."
```

We can combine these two IF statements together to make one block like this.

```
if myBMI >= 25:
    myMessage = myMessage + ". You are overweight."

elif myBMI < 18.5:
    myMessage = myMessage + ". You are underweight."
```

What's this **elif** word? In Python its means **else if**. This means that when the first IF statement is false it next line means else check this IF statement. As well as elif there is another option that we can have in an IF statement and that is the else statement. The else statement runs when the if and any elif statements are false. We need to have a final part of our IF statement that takes care of the people who are not overweight or underweight. So if they are not overweight and they are not underweight then they must be healthy. We can complete our IF statement as shown below.

```
if myBMI >= 25:
    myMessage = myMessage + ". You are overweight."
elif myBMI < 18.5:
    myMessage = myMessage + ". You are underweight."
else:
    myMessage = myMessage + ". You are a healthy weight."
```

The code under the else statement will only run when the if condition and the elif condition are both false. The **else** is like a catch all statement for everything that does not correspond to an if condition or an elif condition.

Now our alterations to our BMI program are complete. Save your work and run the program.

```
myHeight = 0.0
myWeight = 0
myBMI = 0.0
myMessage = ""

myHeight = float(input("please enter your height in metres: "))
myWeight = int(input("Please enter your weight in kg: "))

myBMI = myWeight/(myHeight * myHeight)
myMessage = "Your BMI is %f" % myBMI

if myBMI >= 25:
    myMessage = myMessage + ". You are overweight."
elif myBMI < 18.5:
    myMessage = myMessage + ". You are underweight."
else:
    myMessage = myMessage + ". You are a healthy weight."


print (myMessage)
```

LOOPS

Sometimes you want to run the same code (or similar) again and again. For example if we wanted to write a program that printed out a times table we could use the following code.

```
multiplier = int(input("Which times table do you want?: "))
print (multiplier , " x 1 = ", multiplier * 1)
print (multiplier , " x 2 = ", multiplier * 2)
print (multiplier , " x 3 = ", multiplier * 3)
print (multiplier , " x 4 = ", multiplier * 4)
print (multiplier , " x 5 = ", multiplier * 5)
print (multiplier , " x 6 = ", multiplier * 6)
print (multiplier , " x 7 = ", multiplier * 7)
print (multiplier , " x 8 = ", multiplier * 8)
print (multiplier , " x 9 = ", multiplier * 9)
print (multiplier , " x 10 = ", multiplier * 10)
```

If you look at the code most of the lines are pretty similar. In fact the only thing that changes between each line is the number we use to multiply by. This number changes in a predictable way too. It just counts up in 1s. We can use this idea to repeat that line using a counter.

There are two types of loops in Python. One loop is used when we know how many times we need to repeat some code. This is the FOR...LOOP. The other loop is used when we don't know how many times we need to repeat some code, but we do know that it needs to be repeated as long a condition is true. This is the WHILE...LOOP. Loop structures are known as **iteration**. Going through a loop once is know as one iteration.

FOR Loop

The For loop is used when we know how many times we need to repeat some code. For loops have the following structure:

```
For variable in range():
```

Code to be repeated.

The variable is used to keep track of how many times the loop needs to be repeated. The range is a list of numbers that determine how many times the loop is to be repeated.

We can re-write out times table code using a For loop.

```
multiplier = int(input("Which times table do you want?: "))
for counter in range(1, 11):
   print (multiplier , " x ", counter , " = ", multiplier * counter)
```

So what does this all mean? Lets look at the line

```
for counter in range(1, 11):
```

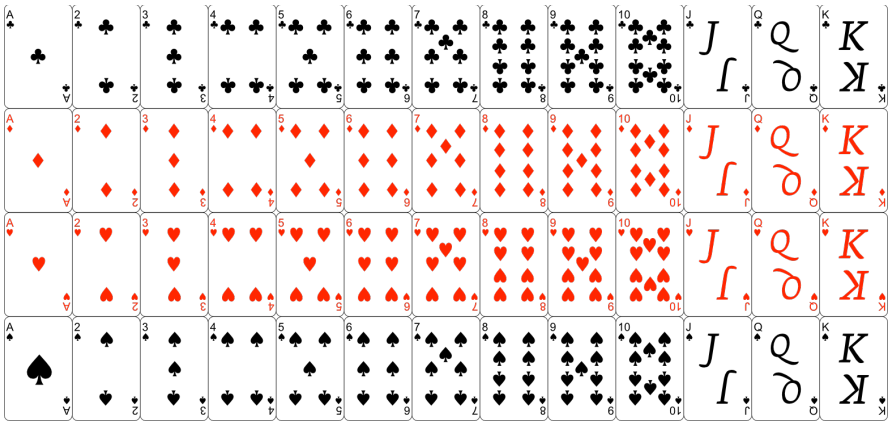
This begins the **for loop** block. It uses a variable called **counter**. This counter will store the values declared in the range function each time it goes through the loop. The loop will begin at the first number in the range function. Each time it goes through the loop the counter increases by 1 until it gets to the number before the last number declared in the range function. So to go from 1 to 10 we need to use **`range(1,11)`**. If we want to count from 1 to 100 we would use **`range(1,101)`**. We always need to have 1 more that we want to go to.

The next line is indented. This is the line that will be repeated each time we go around the loop. This line prints the line of our times table, using the counter and the multiplier variables.

A for loop can also be use to go through items in a list. Look at the code below.

```
deck=[]
for suit in ["Clubs","Diamonds","Hearts","Spades"]:
     for value in ["A" 2,3,4,5,6,7,8,9,"J","Q","K"]:
         deck.append(str(value)+ suit)
```

Here we have two loops, one loops through the list of suits and the nested loop cycles through the values. The nested loop will run for each suit, so will run four times in this example. So when the suit is **hearts** in the first loop, that nested loop all cycle through all they values before the first loop will move onto **diamonds**.



A string is a list of characters and so we can loop through each character in a string. This code is part of a hangman game. It takes the secret word and loops through each letter checking to see if it is a letter or not. If the character is a letter then a "-" is added to the guessing word otherwise "/" is added.

```
secret = "computer science"
guessingWord = ""
for character in secret:
```

```
    if character.isalpha():
        guessingWord = guessingWord + "-"
    else:
        guessingWord = guessingWord + "/"
print(guessWord)
```

Counting down

We all know the lyrics to 99 Green Bottles, but it would be a pain to have to write out the entire song. Lets have a look at one verse.

> 99 Green bottle sitting on a wall,
>
> 99 Green bottle sitting on a wall,
>
> If one green bottle should accidentally fall,
>
> there'd be 98 Green bottles sitting on the wall.

The next verse is pretty much the same, only the numbers of bottles changes. We can use a for loop to help create the lyrics. The problem with this is we wish to count down and not up. The range function has an optional third argument known as the step.

```
range (start, end, step)
```

If the step is not defined then it is set to 1. If we wish to count in 10s we would set the step to 10. If we wish to count down we set the step to be -1.

```
for bottles in range (99, 0, -1):
    print bottles, " Green bottle sitting on a wall,"
    print bottles, " Green bottle sitting on a wall,"
    print "If one green bottle should accidentally fall,"
    print "they'd be", bottles - 1, "sitting on the wall
```

Notice how we use the variable **bottles** to store how many bottles we have and the fact we can use that in our print statements to print the correct amount of bottles in each line.

WHILE LOOPS

We do not always know how many times we need to perform a loop. Sometimes we need to perform an action as long as something is true. A While Loop is a conditional loop. Just like an IF statement has a condition, a While Loop does too. The contents of the loop only run when the condition is true. The structure of a while loop is like this:

```
While condition:
   Code to be repeated.
```

The code will only be repeated if the condition is true.

Guessing game

This simple guessing game the computer will create a random number between 1 and 100 and the player has six guesses to get it right. In this game you do not know how many guesses the player will need in order to get right. The loop will be repeated when the guess is not equal to the secret and the players still has tries is left.

```
import random
secret = random.randint(1,100)
tries = 6
guess = 0

print("I've though of a number between 1 and 100.")
print("Try and guess it. You have 6 tries.")

while guess != secret and tries >0:
    guess = int(input("Enter a number between 1 and 100: "))

    if guess < secret:
        print("Your guess is too low.")
```

```
        elif guess > secret:
            print("Your guess is too high")
       tries = tries - 1
    if guess == secret:
        print("Well done. you got it.")
    else:
        print("You ran out of tries. It was:" , secret)
```

Sometimes when we are in a loop you may want to break out of the loop early. We can do this by using the break keyword. In this example we have a very simple Magic 8 ball program that allows us to ask the computer question and it will tell the user its opinion.

```
    import random  #need to generate random numbers
    #setting up variables
    answer = ""
    question = ""
    randomnumber = 0
    eightball = ""

    while True: #create a loop

        #get the use to ask the magic 8 ball a question
        question = input("What do you want to ask the magic eight
ball?: ")

        #generate a random number between 1 and 3 and assign the
it to a variable
        randomnumber = random.randint(1,3)

        if randomnumber == 1: #if the randomnumber is equal to 1
            answer = "Yes" #set answer to Yes
```

```
    elif randomnumber ==2: #if the randomnumber is equal to 3
        answer = "Possibly" #set answer to Possibly
    else: #else the randomnumber must be equal to 3
        answer = "No" #set answer to No

    print(answer) #Print answer on screen

    #ask if the user wants another go
    anotherGo = input("Do you want another go (y/n)? ")

    if anotherGo != "y": #if the anotherGo is NOT equal to y
        break #break out of the loop

print ("Bye...") #print ending message
```

In this code we set the while condition to be true. This means the code will continue for ever and unless we break out of the loop. At the end the player is asked if they would like another go. If they do not enter "y" then the loop will break.

Notice that this code has comments. The computer ignores anything after a hash mark. So we can write comments in English explaining what the code does. Comments are useful because they explain to other people what your code does. It also reminds you. When you look at code in a few months later you may not remember why you wrote the code you did.

FUNCTIONS AND PROCEDURES.

Functions & Procedures are small snippets of code that are written so they can be reused. You have been using some of the built in functions in Python already, such as print(), input(), range() and int(). A **procedure** is a snippet of code that carries out some task but does not return any data. The print() command is an example. A **function** is a snippet of code that returns a value. This value is often stored into a variable. The input() function gets a value inputted by the user and returns a value as a string. You can tell if something is a function or a procedure as it has brackets at the end. Sometimes one or more values go inside the brackets. These are known as **parameters**.

function_name (*parameter*)

In order to create your own function or procedure we need to define it. A function or procedure needs to be defined before it can be used.

Lets look at writing a simple procedure. This procedure will print out some text on the screen for a hangman game.

```
def help():
    print("Welcome to Hangman")
    print("A word will be chosen at random and you need to
guess the letters.")
    print("For each wrong guess, you become close to being
hanged!")
    print("Careful, you can only make 5 incorrect guesses
before you are hanged.")
```

When the command help() is called then all the print commands will run one after the other. The program will then continue from where the help() command was called.

A function is like a procedure but it returns a value. Lets look at a simple function that takes a number in degrees celsius as a parameter and returns the degrees fahrenheit.

```
def celsiusToFahrenheit(celsius):
    fahrenheit = (celsius * 1.8) + 32
    return fahrenheit
```

We could use the function in the following way.

```
temperature = int(input("Enter the temperature in degrees
celsius: "))
```

```
print("The temperature in fahrenheit is" ,
celsiusToFahrenheit(temperature) )
```

In python the input() function always returns a string.

We could write a modified version of this function so that it returns an integer instead.

```
def inputInt():
   while True:
        try:
             number = int(input("Please enter an Integer: "))
             return number
        except:
             print("You didn't enter a number. Try again".)
```

This code will keep asking for an input until the user enters a number. The While loop condition is set to True so the loop will continue until it can be broken out off. The **try...except** block is used when we want to try something that could cause our program to crash. If we try and use the function int() on a value that cannot be converted into an integer the program would crash. As it is in a try...except, if the commands in the try block fail then the code under the except block will run, printing out an error in this case. If the user inputs a number then the number is returned to the line that called it using then return command. This breaks out of the loop.

We could make this more usable by sending the message we want displayed in the input prompt as a parameter.

```
def inputInt(message):
   while True:
        try:
             number = int(input(message))
             return number
        except:
             print("You didn't enter a number. Try again".)
```

Reading and Writing Files

Python can be used to read and write data to a file. The simplest way to do this is with simple text files.

When we deal with files, there are three ways in which we can deal with them. Firstly we can open a file for reading. This allows us to read the lines of data in the file into our program. We can open a file for writing. As you would expect this means that data can be written into the file. If the file already exists however, any data in the file will be written over. There is a third option. If a file is opened for appending, then the data written into the file will be tagged on at the end of the file.

We use the **open()** function to open a file and assign the file to a variable.

It takes two parameters, the first is the file name, and the second is the operation we wish to carry out on the file.

```
open(filename, operation)
```

Once we have a file open. We can either read or write to it, depending on the operating.

Writing to a file

The following program will open a file for writing and then write the days of the week. We have a variable called **myfile** that stores the file days.txt. The variable myfile is an object that can perform a variety of tasks. The **"w"** means the file is being opened for writing.

```
myfile = open("days.txt","w")

myfile.write("Sun\n")
myfile.write("Mon\n")
myfile.write("Tue\n")
myfile.write("Wed\n")
myfile.write("Thu\n")
myfile.write("Fri\n")
myfile.write("Sat\n")

myfile.close()
```

We can write a line into the file using the .write() command. So **myfile.write("Sun\n")** tells the myfile object to write a line to the file it contains. The **\n** means start a new line.

We must always close the file when we are finished, which is why the **myfile.close()** command is present when we have finished all file operations.

When writing to a file in Python we must make sure that all data is of String data type. If there are any variables that are numeric then they need converting to strings first.

This example shows how variables can be added to text. The variable age is converted to a string using the **str()** function. This converts non string datatypes to strings.

```
name = "Steve"
age = 46


#This opens a file for appending and assigns the file to a variable called myfile
myfile = open("mydata.txt", "a")

#This writes a line of data. Text must be in quotes.  \n means new line
myfile.write( "Name: " + name + "\n")


#Any variables that are not strings must be converted to a string using str()
myfile.write( "Age:" + str(age) + "\n")

#the file is closed. This must happen to save the data.

myfile.close()

print("finished..")
```

When we write to a file it will write over all the contents of the file, if any exists. If we want to add to a file we can user the **append** operation **"a"** instead of **"w"**.

Reading from a File

Reading data from a file can be done in a few different ways. We can use a loop to go through each line in the file until it reaches the end. The example reads back in the days of the week and prints them. Notice we use the read operation "r" when we open the file.

```
myfile = open("days.txt","r")

for line in myfile:
    print(line)

myfile.close()
```

We can read all the files in one go and store them in a list.

In the following example from a hangman game, we see some code that reads from a list of secret words. We use the **readlines()** function of our file object. All the lines are added to the

words list, each line being an item in our list. We only want one from the list and so we use the **random.choice**() function to pick one from our list at random. The **strip**() function gets rid of new line characters and spaces at the ends of the text.

```
import random
hangmanWords = open("secretwords.txt","r")
words = hangmanWords.readlines()
hangmanWords.close()
secretword = random.choice(words).strip()
```

String Manipulation

Strings are basically an array of characters and can be processed in loops like a list.

We can access each letter in a string using it's index. So if we had a string variable called **myString** with the value "*computing*", the characters would have the following index, just like a list or array.

0	1	2	3	4	5	6	7	8
c	o	m	p	u	t	i	n	g

Remember than in computing lists and arrays begin from 0 and not 1.

So we could write myString[2] and that would refer to the character in position 2 which is m.

By using a variable in a loop we can get each letter and process it individually. The following function is from a hangman game. This function takes a string one character at a time using a for loop. This function creates the blanks word from the secret word.

```
def GuessWord(secret):
```

```
guessingWord = ""
for character in secret:
    if character.isalpha():
        guessingWord = guessingWord + "-"
    else:
        guessingWord = guessingWord + "/"
return guessingWord
```

When each character in the secret word is looped through, we ask if the character is a letter from A-Z using the **`isalpha()`** function. If the character is a letter then we add "-" on to the guessword, otherwise we add a "/". So if the secret word was "*monkey bread*" then the guessword would be "**------/-----**" and if the secret word was "*bagel*" the guessword would be "——".

We can get parts of a string using built in functions. Most programming languages have similar string functions.

String function	Description
`string.lower()`	returns the string with all the letters lowercase
`string.upper()`	returns the string with all the letters UPPERCASE
`string.isalpha()`	returns True is all the characters are letters otherwise returns false
`string.isalnum()`	returns True is all the characters are letters and number and there is at least 1 letter otherwise returns false
`string.strip()`	returns a string that has the spaces and non-printing characters on either side of the string removed

We can get length of a string by using the **`len(string)`** function. This also works with lists.

We can get parts of a string by using the index. You have to use **`string[start:finish:step]`**, where start is the first character you want, finish is the first

character you **don't** want, and step is the step of the slice (step = 2 will take one character every two chars). You can omit finish and step if you don't want them.

Also, you can use negative numbers to count from the end of the string, e.g. while string[0] is the first character of the string, string[-1] is the last one.

Here are a few examples:

```
 >>> word = "Hello World!"
 >>> word[0:5]
'Hello'
 >>> word[1:5]
'ello'
 >>> word[5:]
' World!'
 >>> word[:5]
'Hello'
 >>> word[:-1]
'Hello World'
 >>> word[:-2]
'Hello Worl'
 >>> word[1]
'e'
 >>> word[-1]
'!'
 >>> word[::2]
'HloWrd'
```

Section 7 - Testing

WHY TEST?

When we make something we need to test it to ensure it is working as expected. Sometimes we test things without even realising it. Every single time I replace a blown light bulb I can't help but turn the switch on to make sure the bulb lights. Most of the time it does, but when it doesn't that tells me that there may be another problem such as a blown fuse. When I am testing my bulb I have an expected outcome, the light turning on. If it does, then everything is fine. If not then I need to find out what the problem is. I will keep trying different solutions until I turn the switch on and the light turns on.

How to Test

We can only test something if we know what should happen when an action is performed. If we do not know what should happen, then we have no idea whether the test was successful or not.

When designing programs, the developers often write a series of tests even before the program has been written. Each section of code will be tested along with the program as a whole.

Some tests need data, and program may work differently depending on the data inputted. In our BMI program we would need to test it with data that produced the results for an underweight person, a person that was normal weight and one that was overweight to make sure the whole program worked correctly.

We would want to calculate the BMI by hand first to make sure we hadn't programmed our calculation incorrectly.

When we test our program we also want to select a range of data, not just one. We should test with acceptable expected data, unacceptable data and data at the boundaries of what is acceptable. If for example we wanted the user to enter a number between 1 and 12 then these

should be part of our tests. Sometimes in programming it is easy to write > instead of >= and so get an incorrect answer.

What to do if a test fails?

If a test fails this is good. What would be bad is to give your program to someone and it didn't perform as expected. If a test fails then you have found the error in time. Now you need to find out why?

Types of error

There are two types of error that can occur in a program.

- Syntax errors
- Logical errors

Syntax errors

Syntax errors are problems with the language that the compiler or interpreter do not understand. This is usually things like spelling key word incorrectly or missing out brackets etc. These are usually quite easy to find.

Logical errors

These errors are errors in your logic. Something happens that you do expect. You may have gotten your formulae in an equation wrong. These types of errors are much more difficult to spot.

- Logical errors can arise when
- You have a division by zero
- An infinite loop
- Inputting non numerical characters when numbers are expected
- having an index larger than a string or array. This will result in an out of bounds error.
- Saving a file where you don't have permission or trying to open a file that does not exist.