

OCR GCSE COMPUTER SCIENCE

STUDENT REVISION BOOK

A. W. DIMMICK

Version 1.3

OCR GCSE COMPUTER SCIENCE

STUDENT REVISION BOOK

A W DIMMICK

All content in this book has been written by Adam Dimmick. Some of the content of this book forms the basis of the OCR GCSE Computer Science course available at **senecalearning.com** and is used for this purpose with their permission.

Every effort has been made to acknowledge and attribute the sources and authors of all images and video resources used and to adhere to appropriate Creative Commons licensing requirements. Please do make contact if you believe that your intellectual property has been misused so that this can be rectified in future editions.

This eBook is released under the Creative Commons [Attribution-NonCommercial-ShareAlike 4.0 International](https://creativecommons.org/licenses/by-nc-sa/4.0/) license.



This version published 10th October 2018. First published July 2018, this version September 2018

Hello, world

Thank you for downloading and reading my book.

The chances are that if you are reading this then you are probably a student studying GCSE Computer Science or a teacher of this fine subject. Either way, I hope that you find the contents of this book useful.

My intention for this ebook is not to replace a full-blown text book but to offer a little more detail than is found in most revision guides. As much as possible, I have tried to keep explanations to one page per concept so that you, the reader, can focus on learning one thing at a time.

Throughout this eBook are links to various YouTube videos that further explain topics. You should be able to load these by clicking or tapping on the large image on "Video" pages. It is well worth taking the time to watch these to deepen your understanding of topics.

Whilst the content in this eBook is based on the [OCR J276 Computer Science \(9-1\) course](#), this book is **not endorsed by OCR in any way**. In fact, I very much doubt that they even know of its existence! If you are studying a different specification then I hope that you will still find it of some value to you.

Finally, I'd love to receive your thoughts and feedback on any aspect of this book. Please feel free to [email me](#), particularly if you spot any errors or if you think that something is confusingly explained.



Adam Dimmick
Head of Computing Studies
Shiplake College

adimmick@shiplake.org.uk

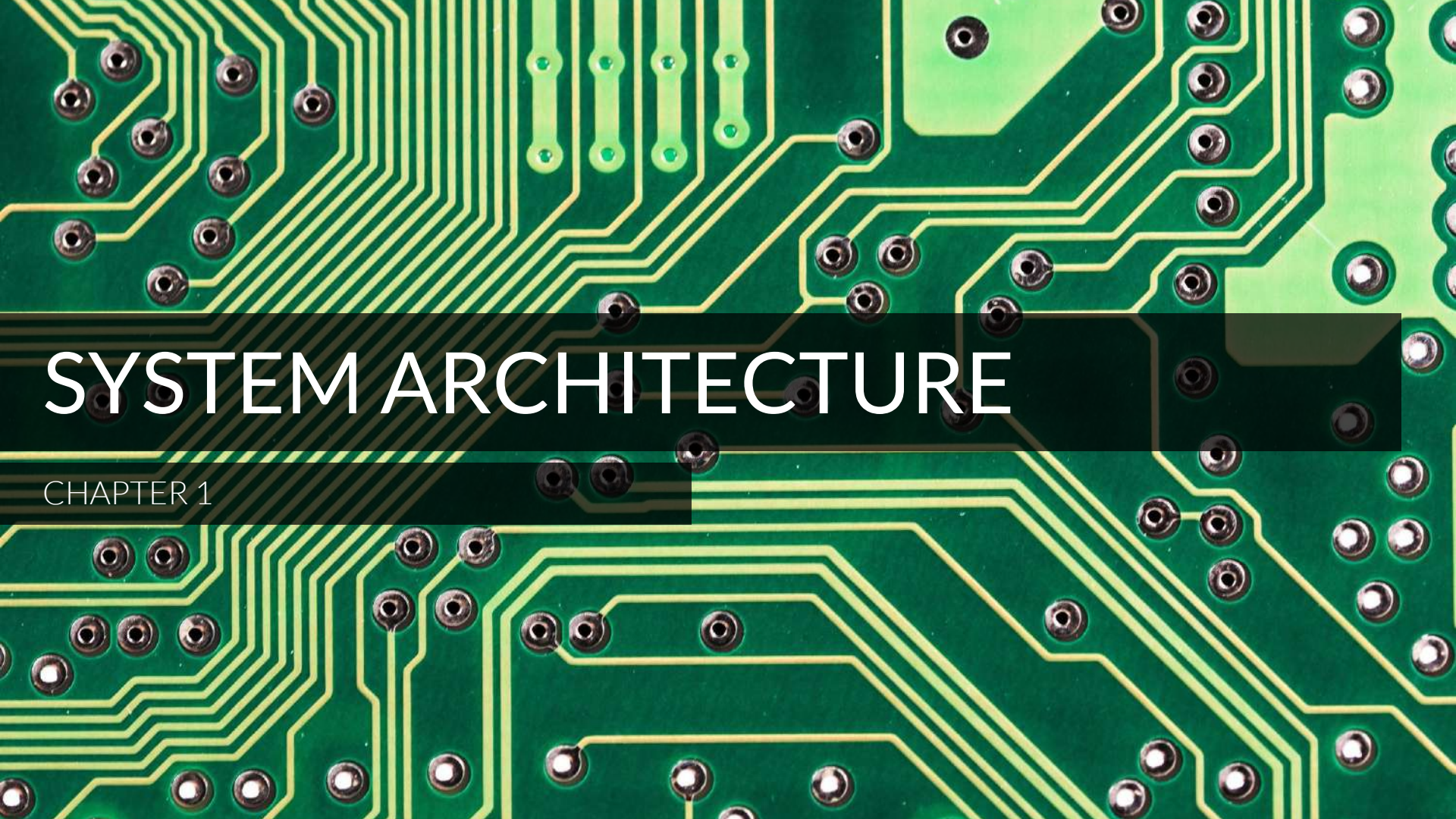
Contents

Component 1

<u>System Architecture</u>	5
<u>Memory</u>	35
<u>Storage</u>	46
<u>Wired and Wireless Networks</u>	76
<u>Network topologies, Layers and Protocols</u>	128
<u>System Security</u>	187
<u>System Software</u>	257
<u>Ethical, Legal, Cultural and Environmental Concerns</u>	315

Component 2

<u>Algorithms</u>	410
<u>Programming Techniques</u>	537
<u>Producing Robust Programs</u>	731
<u>Computational Logic</u>	766
<u>Translators and Facilities of Languages</u>	785
<u>Data Representation</u>	804
<u>Image sources</u>	918



SYSTEM ARCHITECTURE

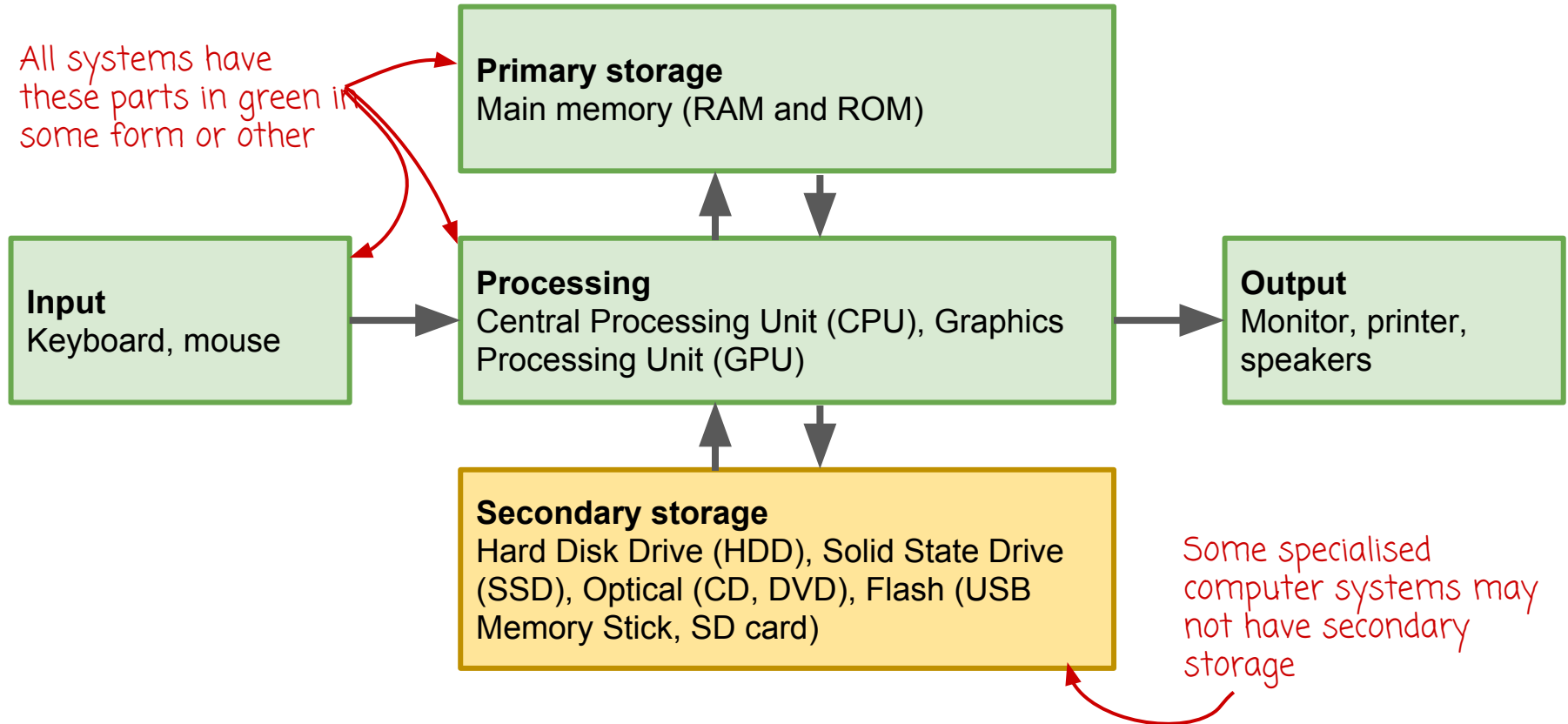
CHAPTER 1

Computer Systems

Computer systems

- The essential purpose of a computer system is to take **in** data, **store** it, **process** it and **output** the result
- Computer systems are made up of **hardware** and **software** working together
- Hardware is the **physical components** that make up a computer system, for example the CPU, monitor, motherboard, etc.
- Software is the **programs that run on a computer system** and control the hardware.

Elements of computer systems



Types of computer system

There are many types of computer systems including

- **General purpose** systems - computer systems that can **perform many tasks**, for example a PC or Mac (desktop or laptop), tablet, or smartphone.
- **Dedicated** systems - computer systems that are designed to **perform one specific function**, for example a satellite TV receiver box.
- **Embedded systems** - computer systems that are **built in to another device** to support their operation.

The purpose of embedded computer systems

- Embedded computer systems are built into larger appliances or devices such as **washing machines, dishwashers** and **microwaves**.
- These devices are not primarily intended to process data but perform another task instead.
- The purpose of the embedded computer system is to **monitor** and **control** various **components** in the host device.
- For example, a washing machine is primarily intended to wash clothes, the embedded computer is there to help it keep track of washing programs, timings, water temperature and so on.

Benefits of embedded computer systems

- Because embedded systems are designed to perform a small number of specific tasks, they are often **cheap to produce** and use **very little power** to run (this makes them **energy efficient**)
- Embedded systems are often very **reliable**. This is due to the fact that:
 - they are only designed to perform specific tasks
 - their software cannot be easily be updated once manufactured
 - their components are relatively simple compared to those in dedicated or general purpose computers.

The Central Processing Unit (CPU)

The purpose of the CPU

- The purpose of a Central Processing Unit (or CPU) is to:
 - **processes data** and **instructions** coming from input and storage devices and
 - **control other components** within the computer
- The CPU does this by manipulating the **flow of electricity** through billions of electrically-operated switches called **transistors**
- The CPU is located on a computer's **motherboard** which connects it to other components such as **RAM** and a **Hard Disk Drive**



Image by [D-Kuru](#)

Video: Inside your computer



Video by [TED-Ed](#)

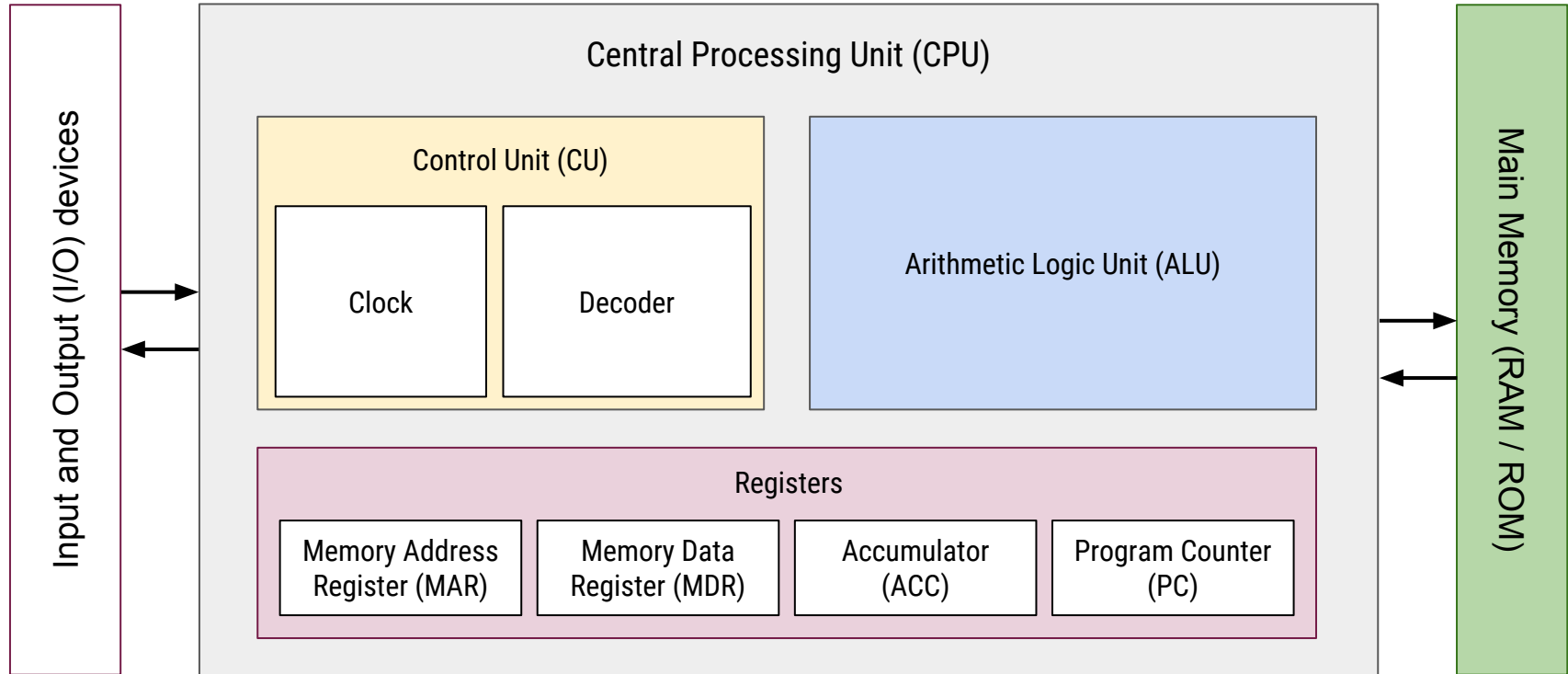
The Von Neumann CPU architecture

- Almost all modern CPUs follow a similar design, formulated by the mathematician John Von Neumann in 1945
- The Von Neumann design for a CPU includes three main elements:
 - A **Control Unit** (CU)
 - An **Algorithmic Logic Unit** (ALU)
 - Small amounts of memory called **registers**

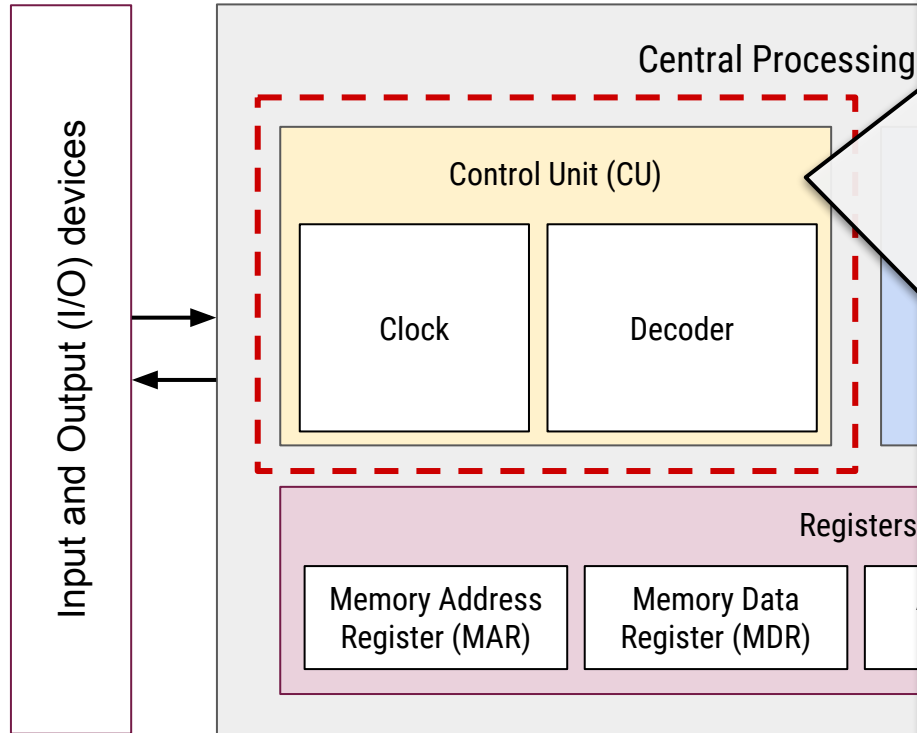


Image by US Department of
Energy

The Von Neumann CPU architecture



The Von Neumann CPU architecture



The Control Unit (CU)

- The purpose of the Control Unit is to **execute instructions** and send **control signals** to other components within the CPU
- The Control Unit contains two main components - the **decoder** and the **clock**
- The **clock** sends an electrical pulse at a fixed interval to **synchronise activity** between components within the CPU
- The decoder **decodes instructions** that are **retrieved** from **memory** during the fetch-execute cycle

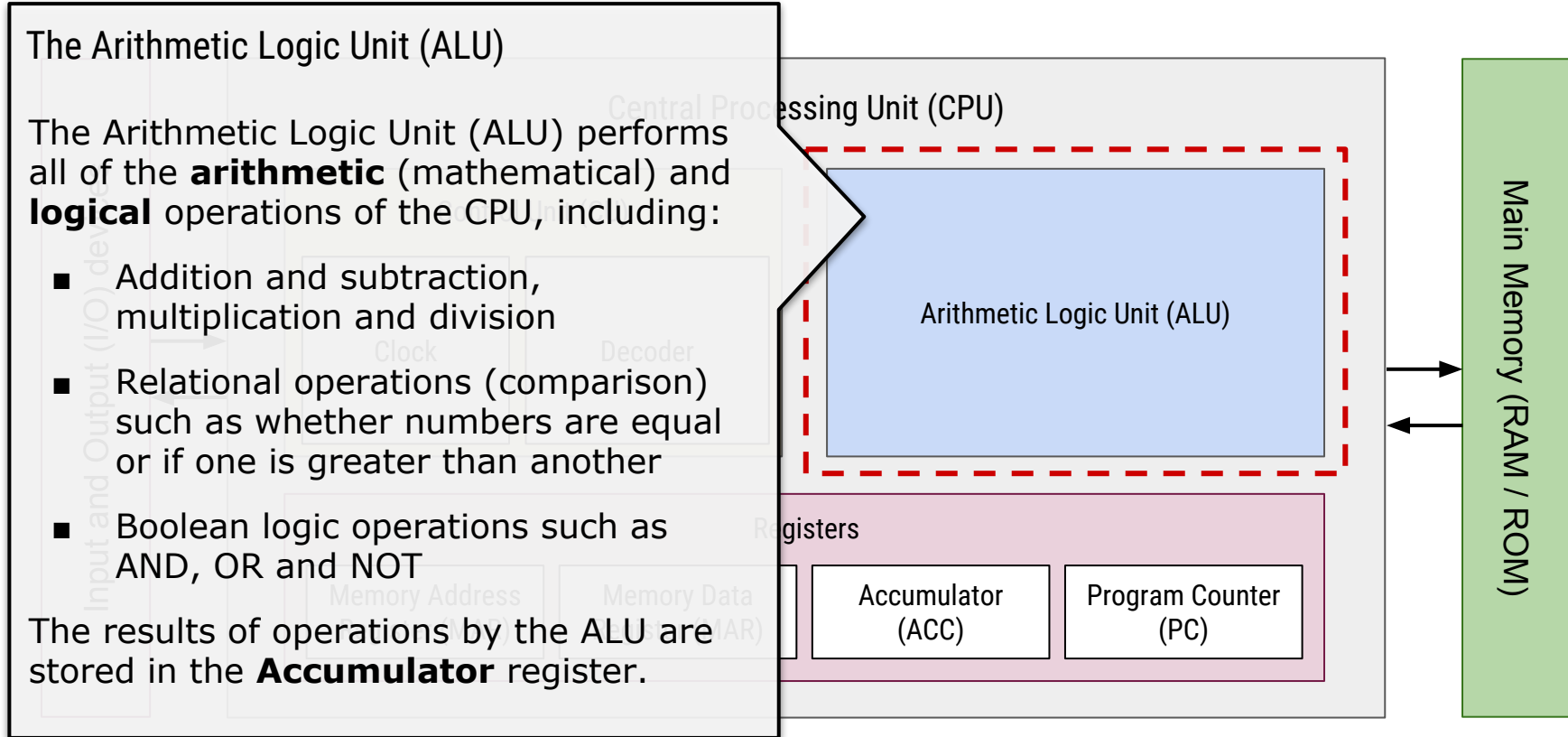
The Von Neumann CPU architecture

The Arithmetic Logic Unit (ALU)

The Arithmetic Logic Unit (ALU) performs all of the **arithmetic** (mathematical) and **logical** operations of the CPU, including:

- Addition and subtraction, multiplication and division
- Relational operations (comparison) such as whether numbers are equal or if one is greater than another
- Boolean logic operations such as AND, OR and NOT

The results of operations by the ALU are stored in the **Accumulator** register.

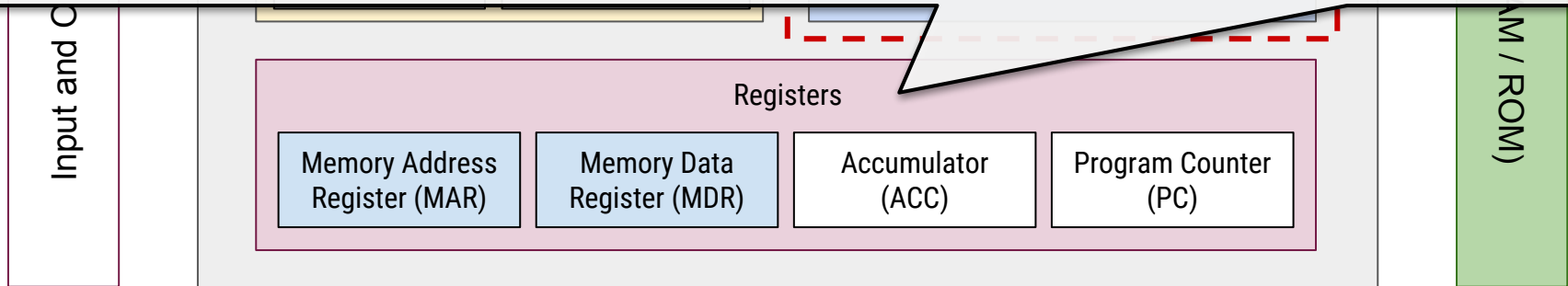


The Von Neumann CPU architecture

The registers (1 of 2)

Registers are small area of memory within the CPU. Each register is used to store a specific item of data:

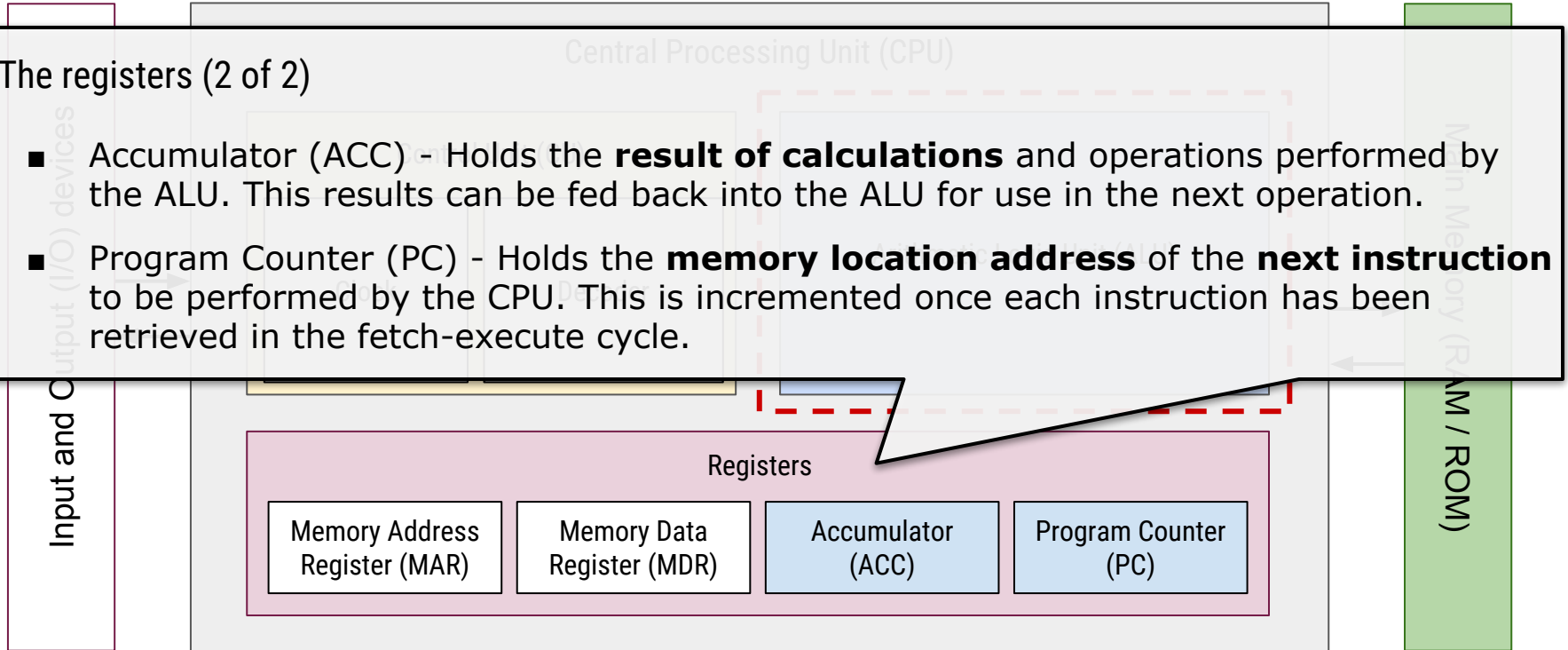
- Memory Address Register (MAR) - Holds the **address** of the **location** in memory where data or an instruction is to be retrieved or stored.
- Memory Data Register (MDR) - Holds the **data or instruction** that has been retrieved from memory or is about to be stored in memory (at the address held by the MAR).



The Von Neumann CPU architecture

The registers (2 of 2)

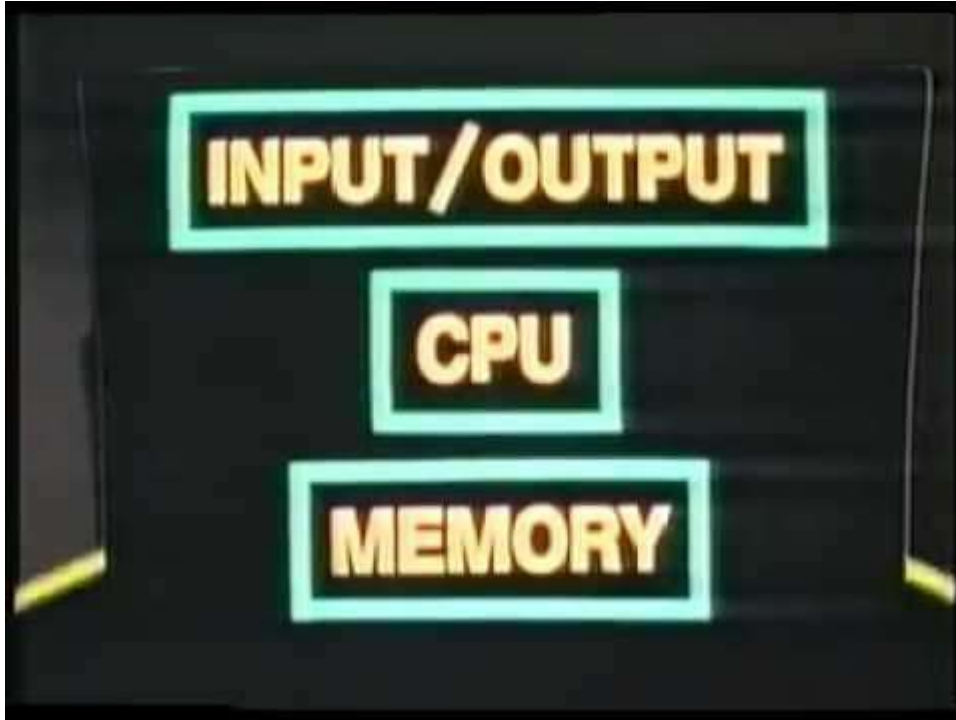
- Accumulator (ACC) - Holds the **result of calculations** and operations performed by the ALU. This results can be fed back into the ALU for use in the next operation.
- Program Counter (PC) - Holds the **memory location address** of the **next instruction** to be performed by the CPU. This is incremented once each instruction has been retrieved in the fetch-execute cycle.



Processing instructions and data

- In order to process instructions and data, the CPU needs to **retrieve** them from **memory**.
- To do this, it follows a process known as the **Fetch-Execute Cycle** (or, sometimes, the Fetch-Decode-Execute cycle).
- This cycle repeats **billions of times each second**, giving the illusion that the CPU is doing many things at the same time when, in reality, it is really only processing one instruction at a time.

Video: How a CPU processes instructions



Video by [Bits and Bytes TVO](#)

The Fetch-Execute Cycle

1. An instruction is **fetched from main memory** (RAM or ROM) at the location held by the Program Counter register
2. The value of the Program Counter is **incremented** so that it points to the next instruction to be processed
3. The fetched instruction is **decoded** by the Control Unit in the CPU
4. The CPU **executes** the instruction by performing an appropriate **operation**, for example the ALU might perform a calculation or data might be loaded from memory
5. The **cycle repeats** until a HALT instruction is retrieved.

The Fetch-Execute Cycle (FCE)

Fetch

- The address of the next instruction to be processed is copied from the Program Counter (PC) to the Memory Address Register (MAR)
- The PC is incremented to point to the next instruction that will be needed when the cycle starts again
- The instruction stored at the location held by the MAR is copied to the MDR

Execute

- The operation indicated by the instruction is performed by the appropriate component, for example:
- The Arithmetic Logic Unit (ALU) performs the operation given by the Control Unit
- The value of stored by the Program Counter or Memory Address Register might be changed

Decode

- The Control Unit decodes the instruction and sends control signals to the component within or outside the CPU that needs to act

Understanding program instructions

Program instructions are stored in memory as binary numbers, for example 1001 1100, and are made up of two parts: an **opcode** and an **operand**:

- **Opcodes** are the first part of the instruction and they determine what operation the CPU should perform
- **Operands** are the second part of the instruction and they provide the values needed for the operation, for example they might represent a memory location for a "Save data" operation, or they may represent a number to include in an addition operation.

Instruction sets

The range of operations that a CPU can perform is determined by its **Instruction Set**. Each operation in the instruction set is represented by its own opcode.

For example, the instruction set for the fictional "Little Man Computer" CPU includes the following operations:

Numeric Code	Binary representation	Instruction	Description
1xx	0001xxxx	ADD	Adds the value on the accumulator to the value stored in memory location xxxx.
3xx	0011xxxx	STORE A	Stores the value of the accumulator to memory location xxxx.
0	0000 0000	HALT	Stops the program.

the first part of the instruction is the **opcode** — the second part of the instruction is the **operand**

Factors affecting CPU performance

The main factors that affect CPU performance are:

- the **clock speed** of the CPU
- the number of **cores** the CPU has
- the amount of **cache memory** on the CPU

These can be easily remembered as the "three Cs".



Factors affecting CPU performance

Clock Speed

- The CPU's clock (part of the Control Unit) sends out a **pulse** at **fixed intervals** to initiate the next action of the CPU
- The **faster the clock**, the **more fetch-execute cycles** can take place (and therefore more instructions processed) in a second
- Clock speed is usually measured in **gigahertz** (GHz)
- 1 GHz is 1 **billion** cycles per second

Factors affecting CPU performance

Number of Cores

- Modern CPUs often have multiple **cores** - dual-core CPUs contain **two** processor cores; quad-core CPUs contains **four** processor cores
- Each CPU core has its **own** Control Unit and ALU but they share cache memory and access to memory
- Multi-core CPUs can therefore process multiple instructions at the same time, leading to a **potential** increase in CPU performance however this will **only** benefit programs that have been written to take advantage of multiple cores.

Factors affecting CPU performance

Number of Cores

- A multi-core CPU with a lower clock speed **may** be quicker at completing tasks than a single-core CPU with a higher clock speed but **only** if the program being run has been written to take advantage of multiple CPU cores and contains **parallelizable** tasks
- Tasks that benefit from multiple cores typically involve a lot of **number crunching**, such as modelling weather patterns, analysing big data sets and cryptocurrency mining.

Factors affecting CPU performance

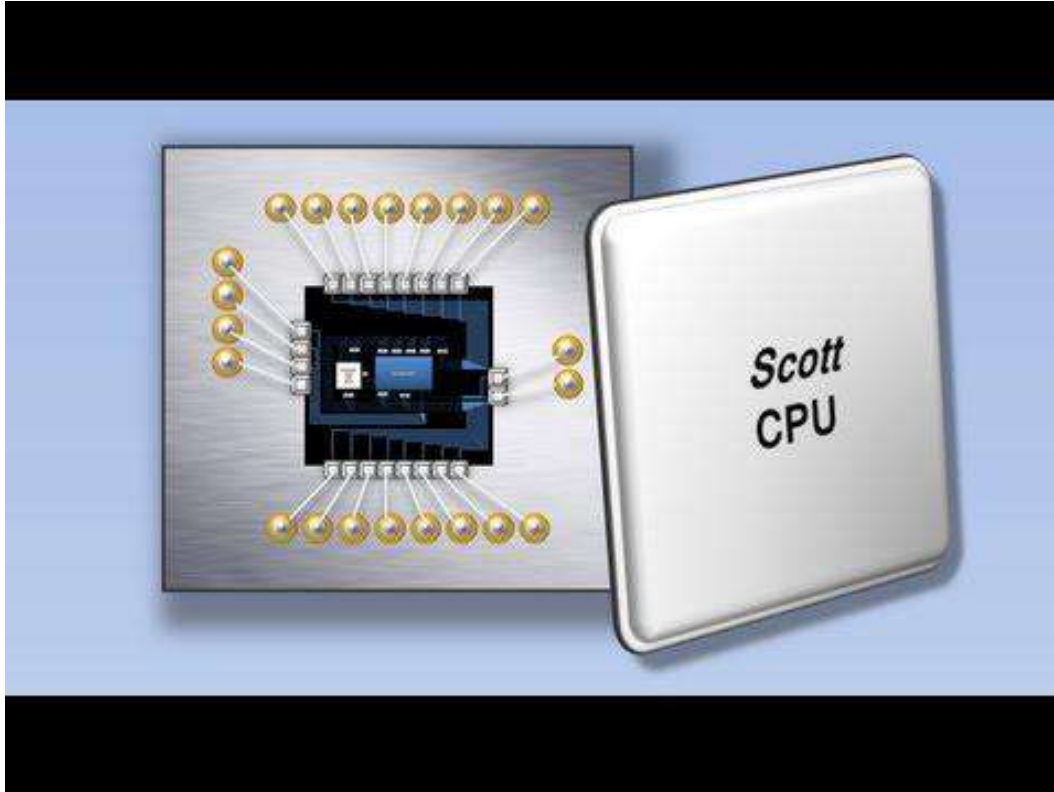
Amount of Cache memory

- Cache memory is **very fast** memory located **within** the CPU
- Cache memory is used to store **frequently used data and instructions** so that the CPU can retrieve them more quickly than fetching them from (comparatively slower) Random Access Memory (RAM)
- Having **more cache memory** allows CPUs to **store more instructions, reducing the amount of time lost** whilst it is waiting to retrieve data from RAM
- Cache is **very expensive**, so a typical CPU might only have 4MB (megabytes) or 8MB of cache. This is 1000 times smaller than 4GB or 8GB of RAM.

Going deeper...

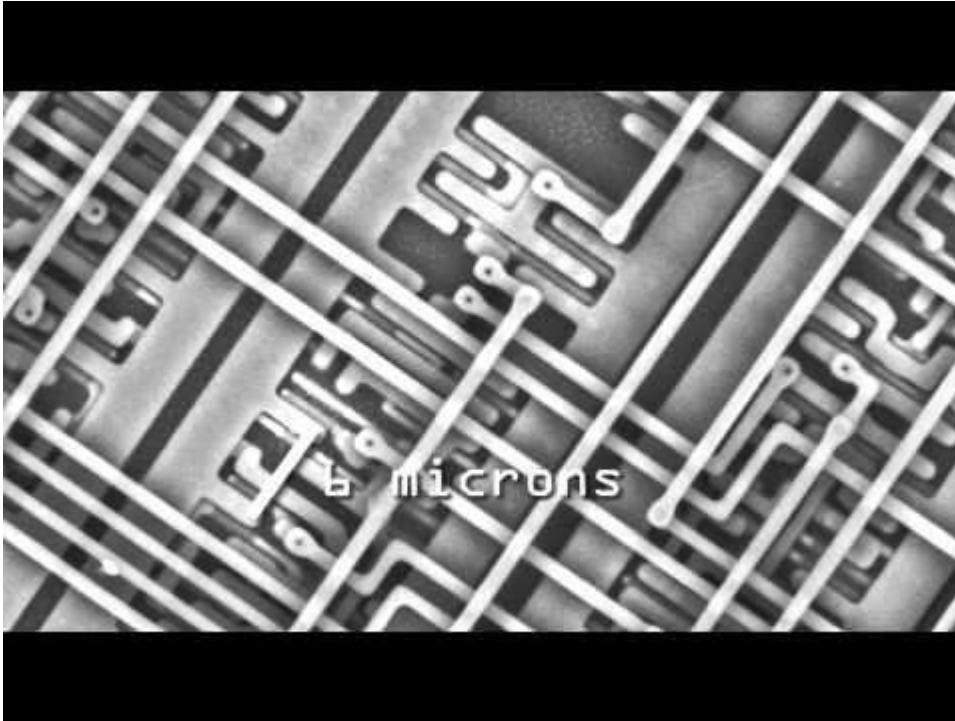
The following videos go beyond the scope of the OCR specification but will help to develop a deeper understanding of the topics covered.

Video: How a CPU works (long but good!)



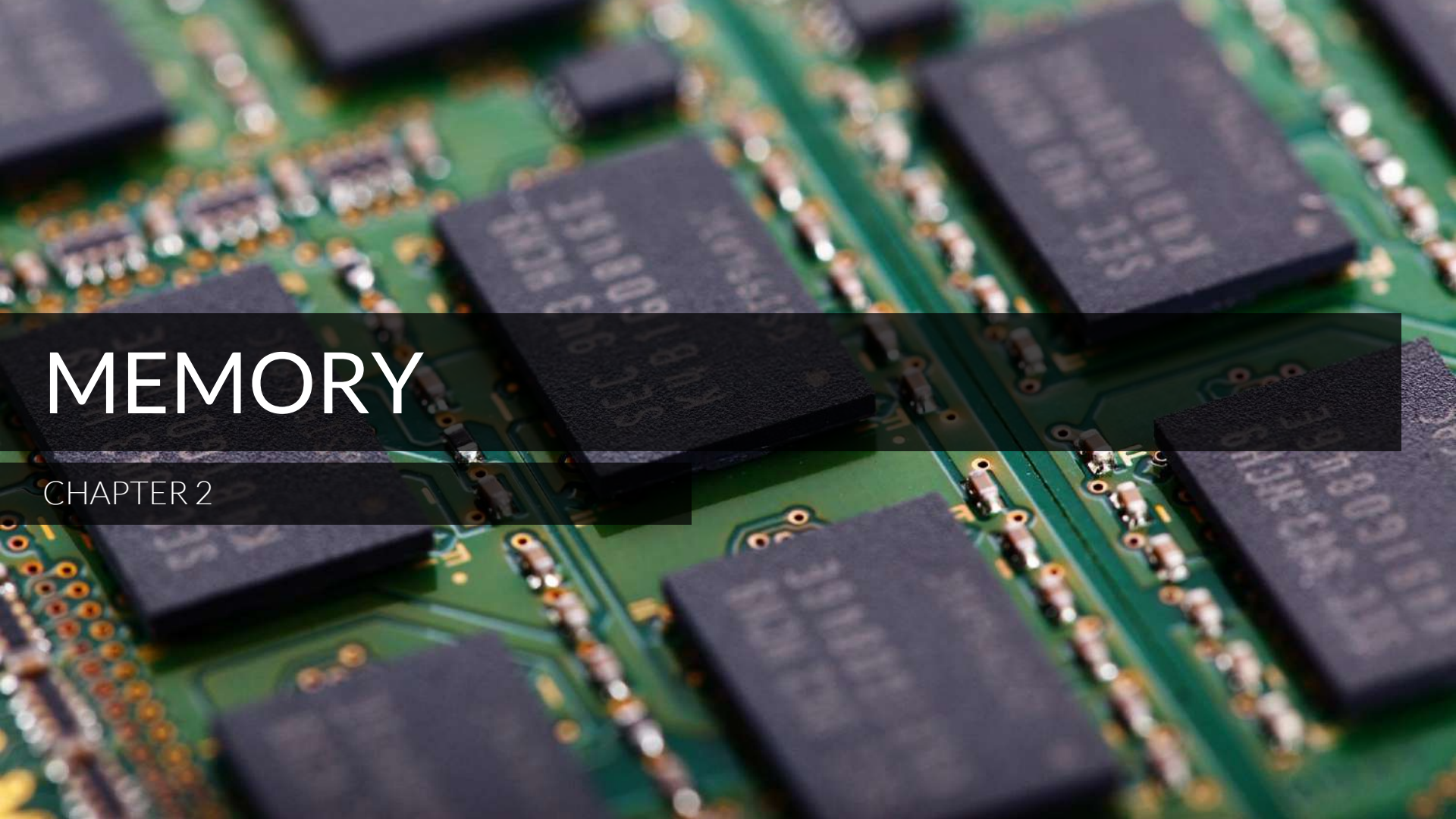
Video by [In One Lesson](#)

Video: Zoom into a microchip



Video by [NISENet](#)

Note, there is no sound for this video.



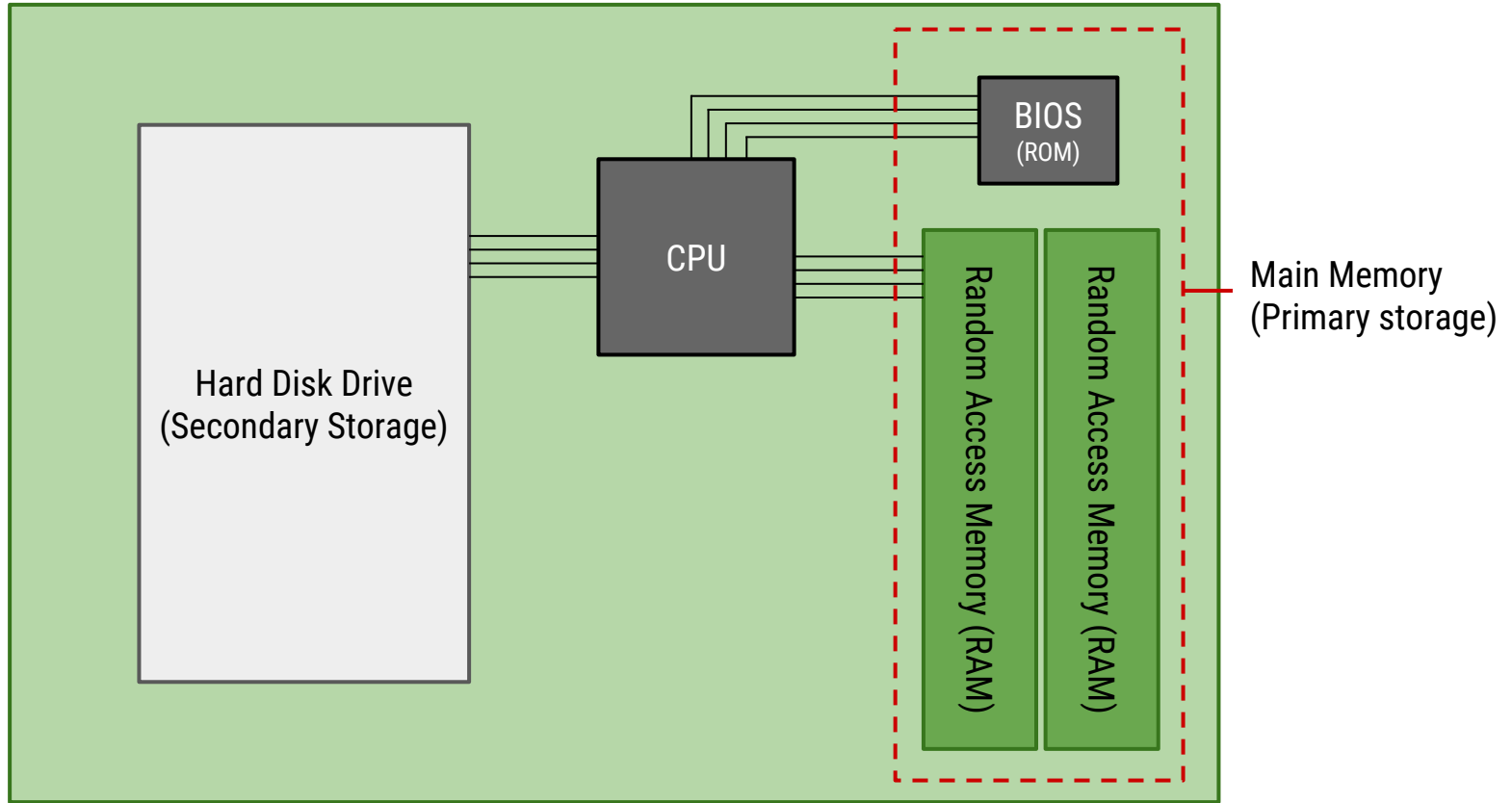
MEMORY

CHAPTER 2

Types of memory

Types of storage in a Computer System

- Computer systems typically have both **primary** and **secondary** storage.
- Primary storage is the computer's **main memory** and is used to store program instructions and data **as they are being processed** by the CPU.
- Secondary storage is the **long-term storage** provided by hard disks, solid state disks, optical media and is used to store data and program instructions until they are required by the CPU.



Main memory: RAM and ROM

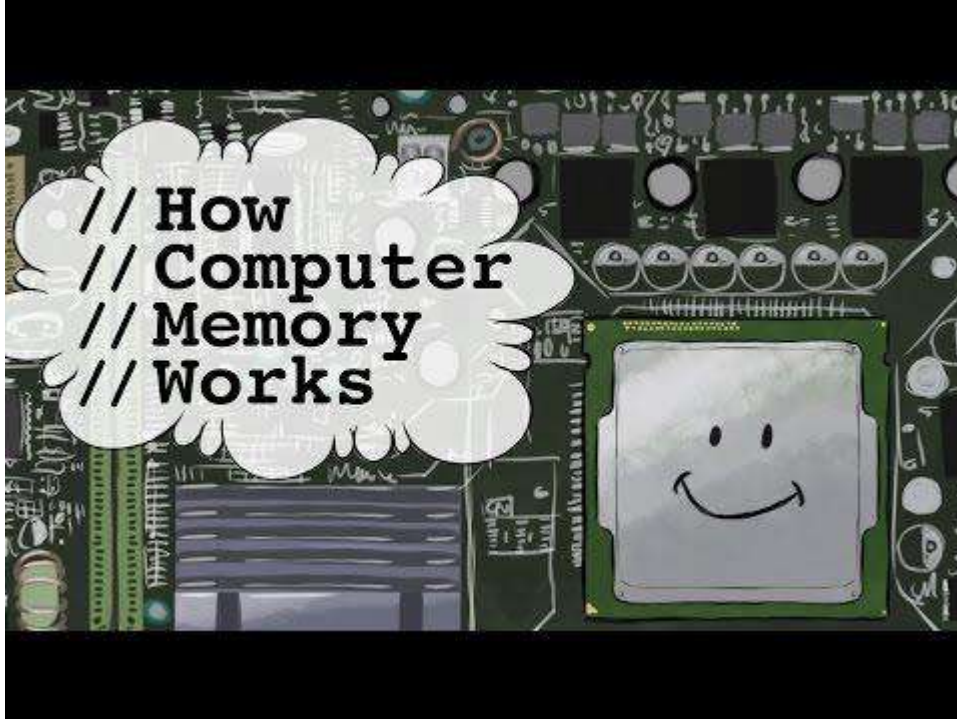
- There are two types of primary storage, or main memory: **Random Access Memory (RAM)** and **Read Only Memory (ROM)**
- Both RAM and ROM can be **accessed directly** by the CPU and may contain the **instructions and data** for the CPU to process



A RAM module for a typical PC

Image by [Laserlicht](#)

Video: Computer memory and the CPU



Video by [TED-Ed](#)

Random Access Memory (RAM)

- Random Access Memory (RAM) **temporarily** stores program instructions and data **while they are being processed** by the CPU.
- RAM is **volatile**, meaning that **its contents is lost** when it loses power (i.e. the computer is switched off).
- RAM is **much faster** than secondary storage devices such as Hard Disk Drives but much **more expensive** on a per-GB basis.
- Because RAM is both volatile and expensive, general-purpose computer systems have secondary storage for long-term storage of programs and data.

Read Only Memory (ROM)

- Read Only Memory (ROM) **permanently** stores program instructions and data but it **cannot have new data written to it**.
- ROM is **non-volatile**, meaning that **its contents is retained** when it loses power (i.e. the computer is switched off).
- ROM is often made from **flash memory** cannot be altered without being "re-flashed". This means that it tends to contain small programs that get the computer up and running, leaving the big stuff to secondary storage.
- ROM is typically used to store the **boot-up sequence** or **Basic Input Output System (BIOS)** that the CPU executes as soon as it turns on. The BIOS contains instructions that load the operating system from secondary storage into RAM for the computer to continue booting up.

Why are both ROM and RAM required?

- When a computer is first switched on, RAM is **empty** and the CPU does not "know" where to access the **Operating System** on secondary storage.
- ROM is used to store a program called the **BIOS** (Basic Input Output System) that provides the CPU with essential information including the **memory addresses** of secondary storage devices so that it can **load the Operating System into RAM** in order to **boot up**.
- ROM is too **small** to store an entire Operating System so secondary storage is required for this. Even if ROM could be used to store the whole OS, the fact that it is **read only** would mean that no changes could be made to the OS, such as installing applications or devices or even customising the desktop wallpaper!

Flash memory

- Flash memory is a form of **non-volatile memory** called EEPROM (electrically erasable programmable read-only memory) that is often used to make ROM chips but also USB memory sticks, Solid State Drives, SD cards and flash storage in mobile devices such as phones and tablets.
- Flash memory will **retain its contents** when it loses power, but can also be **erased and rewritten** through a process called "flashing", which involves passing electrical current through all of its memory "cells".
- Flash memory can **only be flashed a certain number of times** before it starts to degrade, so re-writable devices such as USB memory sticks, SD cards and SSD drives are actually made up of several different blocks of flash memory, meaning that only one block needs to be reflashed at one time thus prolonging the life of the device.

Virtual Memory

- Virtual memory is clever trick used by Operating Systems to use **secondary storage as if it were RAM** when **RAM becomes full**.
- The Operating System stores a **copy** of the data from sections of RAM (called "pages") in a file in secondary storage (the "pagefile"). This page of memory can now be **freed up for another process** to use and its content retrieved from the pagefile when needed again.
- Using virtual memory allows users to have **more programs open at once** than can fit in physical RAM, however secondary storage is much slower than RAM (even if you have an SSD) so **using virtual memory can significantly reduce the performance of a computer system**.



STORAGE

CHAPTER 3

Types of storage in computer systems

General purpose computer systems typically make use of two types of storage:

- **Primary** storage - made up of RAM and ROM, this stores programs and instructions for the CPU to process.
- **Secondary** storage - permanently stores programs and data whilst they are not in use by the CPU. This is often provided by a Hard Disk Drive or Solid State Drive in a computer but can also take the form of removable media such as USB memory sticks and optical (CD/DVD) discs.

The need for secondary storage

Secondary storage is used for:

- **Permanently** storing data and programs when they are not **immediately** needed by the computer.
- Storing **backups** of data in case the originals are damaged or destroyed.

Secondary storage technologies

Types of storage technology used for secondary storage include:

Magnetic storage

- Hard Disks, Magnetic Tapes, old-fashioned floppy disks

Flash memory

- Solid State Drives, USB memory sticks

Optical storage

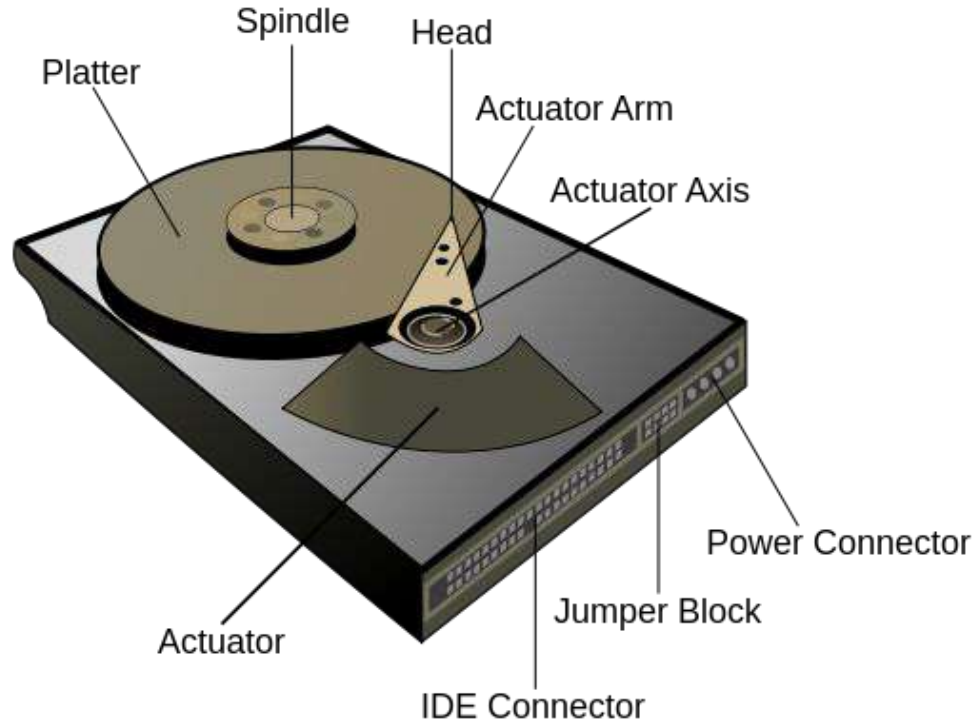
- CD-ROM/CD-R, DVD-ROM/DVD-R, Bluray

Magnetic storage

Magnetic Hard Disk Drives (HDD)

- The value of each bit of data (either 1 or 0) is represented as a positively or negatively charged **magnetic particle** on a magnetic disc (**platter**).
- A magnetic **read/write head** is used to read and write data whilst the platter **spins**, typically at 5400, 7200 or 10000 revolutions per minute.
- Magnetic Hard Disks offer a **large capacity** for a **relatively low cost-per-GB** and they provide reasonably fast transfer rates (around 120MB/sec).
- Magnetic Hard Disks are generally **reliable** - they keep working for a long time, however they will eventually develop faults due to wear and tear of the moving parts. They are also easily damaged if they are dropped or knocked when in operation, meaning that they are **not very robust**.

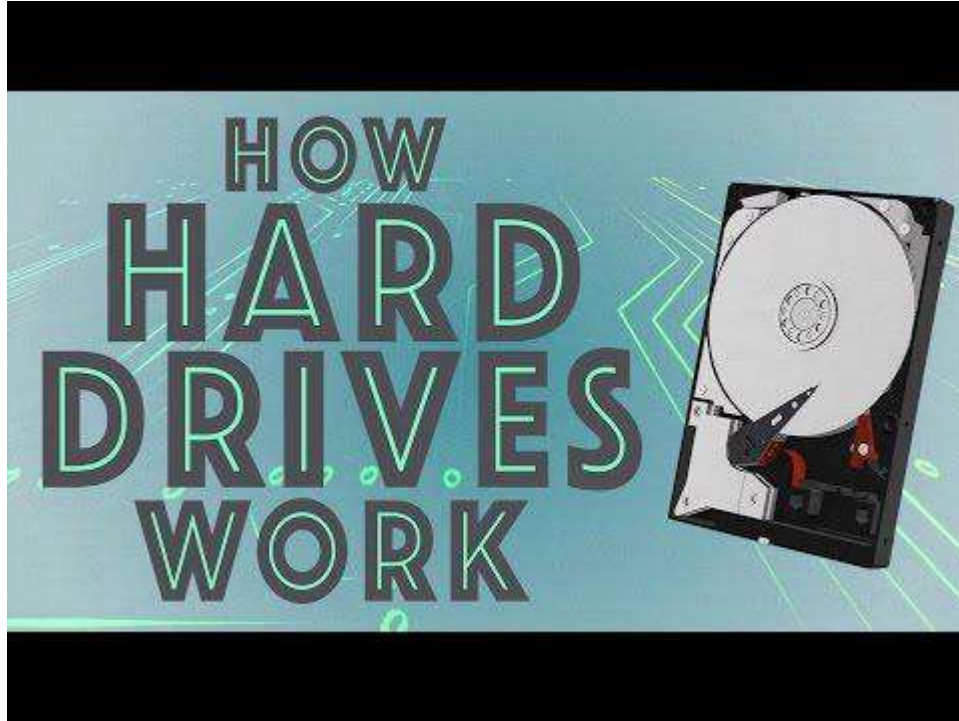
Magnetic Hard Disk Drives (HDD)



Uses of Magnetic Hard Disk Drives (HDD)

- Hard Disks are often used as the **main form** of secondary storage in a computer system to **hold the Operating System, applications and user data**.
- External Hard Disk Drives are often used for home and enterprise **backup** as they offer high capacity and low cost-per-GB whilst being portable and relatively quick for transferring large amounts of data.

Video: How do Hard Disks work?



Video by [TED-Ed](#)

Video: Movement of the read/write head when reading data



Video by [joshuamarius](#)

Magnetic tape

Magnetic tape comes in **cassettes** that contain a very long reel of magnetic tape.

These reels can be up to 1km in length!

Special tape drives are must be installed in a computer or server to read and write magnetic tapes.



A magnetic cassette tape

Image By [Austinmurphy](#)

Magnetic tape

- Magnetic tape offers **very high capacity, good durability** and a **very low cost per GB**, making it ideally suited to long-term backup storage.
- Magnetic tapes can store over 300TB (307,200,000 GB!) of data in a small, portable cartridge and are designed to last in storage for up to 30 years.
- As with magnetic Hard Disk Drives, binary data is stored as positively or negatively charged particles on the surface of a long tape stored within the cartridge.
- The downside of Magnetic Tape is they have **very slow access times** compared to other forms of secondary storage. This is because Magnetic Tape uses **serial access**, where the right part of the tape has to be found before the desired data can be read.

Video: Magnetic tape drives explained



Video by [Techquickie](#)

Solid State and Flash Memory

Solid State Drives (SSDs)

- Solid State Drives are replacing Hard Disk Drives as the main form of secondary storage in computer systems, particularly in laptops due to their **excellent speed, robustness** and **portability**.
- Solid State Drives are made of **flash memory**, contain **no moving parts** and offer a **much faster transfer rate** than magnetic HDDs (around 550MB/s compared to 120MB/s for HDDs).
- Flash memory has a **higher cost-per-GB** than Magnetic Hard Disks and can **only be re-written a limited number of times**, meaning that SSDs will start to degrade sooner than a Hard Disk would typically wear out, **reducing their durability**.
- Mobile devices such as smartphones and tablets use flash memory exclusively for their secondary storage.

USB Flash Drives, Memory Sticks and SD cards

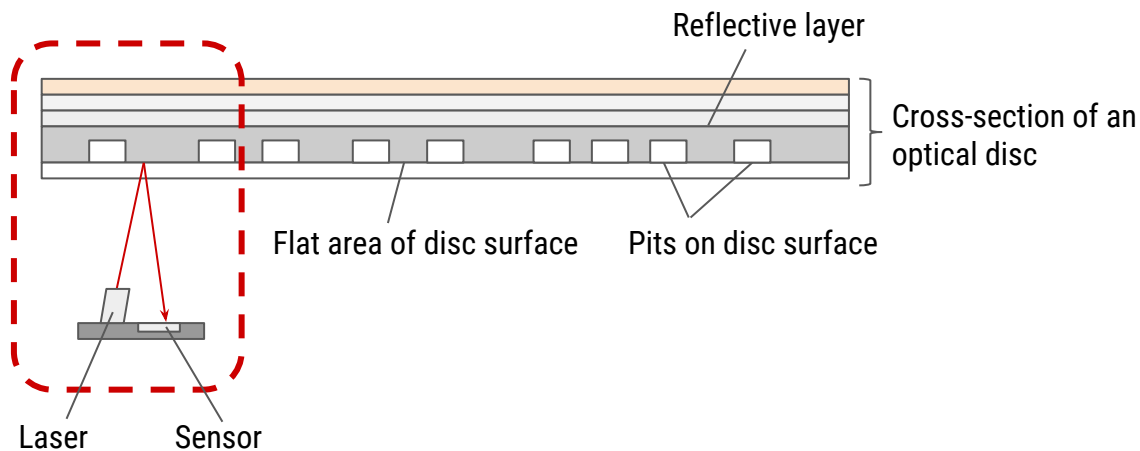
- USB Flash Drives ("memory sticks") and SD cards are made of similar flash memory to Solid State Drives, though their transfer rates are a lot **slower** (typically around 6MB/sec to 30MB/s, but potentially up to 90MB/s).
- They offer **good capacity** (typically around 16GB to 64GB) for a **reasonable cost-per-GB**, though not as cheap-per-GB as Hard Disk Drives or optical media.
- Their good capacity and cost, combined with being **extremely portable** makes them **suitable for transferring documents** and making **short-term backups** of files.
- Good quality USB Flash Drives are also quite **robust**, surviving well in a bag or on a keyring, but they can suffer from damage if they are removed from devices prematurely, making them **less reliable**.

Optical storage

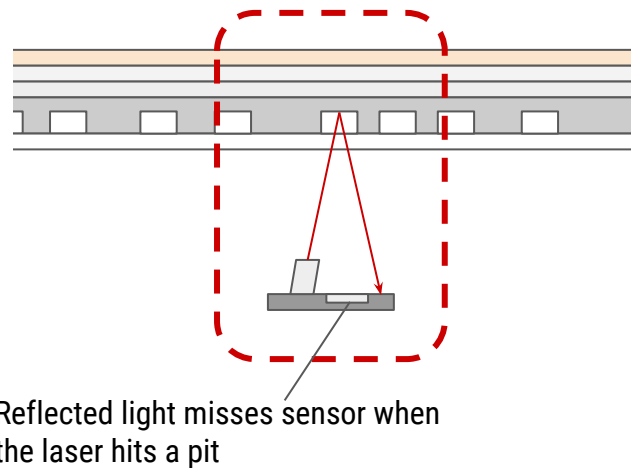
Optical storage

- Optical storage technology uses **laser light** to retrieve data from the surface of optical media such as CDs and DVDs
- Each 1 and 0 of the binary data is represented as microscopic holes (pits) on the surface of the disc.
- A laser is then shone on the disc surface which either bounces back onto a sensor (if it hits a flat part of the surface) or it doesn't (if the laser hits a pit).
- This "hit-or-miss" signal is used to reconstruct the 0s and 1s that make up the binary data.

Reading data from an optical disc

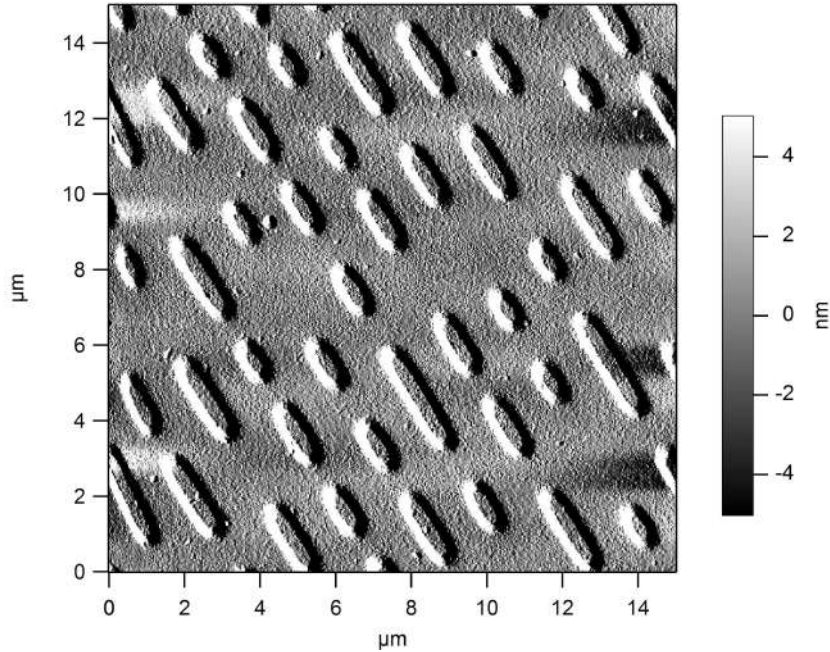


When the laser hits a flat part of the disc surface (a "**land**") its reflection hits the sensor, which is interpreted as binary 1.



When the laser hits a pit on the disc surface its angle of reflection is changed so that it misses the sensor, which is interpreted as binary 0.

Pits



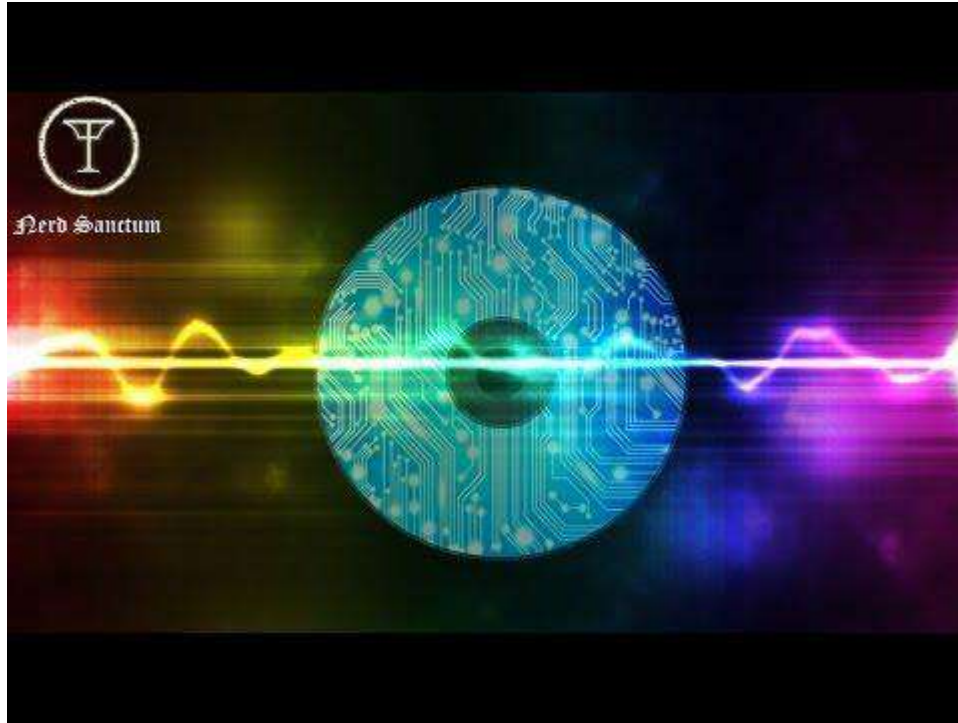
Pits on the surface of a CD-ROM

Image by freiermensch

The **pits** on an optical disc are incredibly small.

Lasers with very short **wavelengths** are used to read them.

Video: How does optical media work?



Video by [Devon Bernard](#)

Different types of optical media

There are three main types of optical media - CDs, DVDs and Blu Ray.

CDs hold the least data (around 700MB), then DVD (4.7GB) and Blu Ray holds the most (around 50GB).

Each type of media (CD, DVD or Blu Ray) comes in different formats:

- ROM (Read only memory) - Data is "pressed" onto the disc in factories and cannot be altered
- R (Recordable) - Data can be "burned" to a disc once only
- RW (Rewritable) - The contents of the disc can be re-written multiple times (though the whole disc is wiped each time, so not suitable for dragging and dropping files as with a memory stick)

Different types of optical media

- CDs, DVDs and Blu Ray are all the same physical size but the pits on the surface of each type of disc gets smaller as the capacity increases.
- Lasers of shorter wavelength are needed to detect smaller pits. Blue light has a very short wavelength, hence the name "Blu Ray" for the format that uses the smallest pits and the shortest wavelength of laser light.

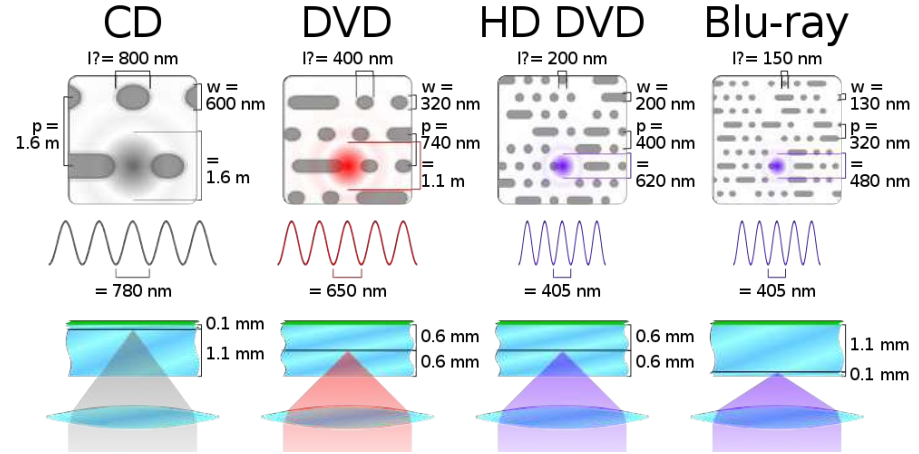


Image by [Cmglee](#)

Uses of optical media

- Optical discs are **cheap** and **quick** to produce in large numbers. They are also small and very portable.
- Recordable optical media is **cheap per GB** although fewer devices have optical drives built in these days and DVDs and CDs have a **smaller capacity** than USB flash drives.
- Optical discs offer faster access times than magnetic tape, but have **slower transfer rates** than Magnetic Hard Disk Drives and Flash Memory devices.
- Optical discs are prone to **scratching** which can cause data loss and rewritable (RW) discs are particularly **unreliable** for long-term storage due to the stability of the chemicals that they are made from.

Data capacity and calculating data requirements

Storage capacity

- The smallest unit of data that a computer can represent is a **bit (b)**, it's value can be either **0** or **1**.
- A group of four bits is a **nibble** and a group of eight bits is a **byte (B)**.
- Because these units are very small, larger units are typically used to describe the size of data, including kilobytes (KB, 1KB = 1024 bytes), megabytes (MB, 1MB = 1024KB), gigabytes (GB, 1GB = 1024MB) and terabytes (1TB = 1024GB).
- Note that each unit is 1024 times larger than the next smallest unit and not 1000 times. This is because 1024 is more easily represented in binary than 1000.

Comparing storage capacities

Unit	Abbreviation	Equivalent to	What sort of things are in this size range?
Bit	b	0 or 1	A single binary value or a True or False boolean value
Byte	B	8 bits	A single ASCII character or a number between 0 and 255
Kilobyte	KB	1,024 bytes	A text file
Megabyte	MB	1,024 kilobytes	Digital photographs and MP3 files
Gigabytes	GB	1,024 megabytes	Videos and software applications
Terabytes	TB	1,024 gigabytes	Backups of whole systems

Calculating storage capacity

- To calculate how much data can be stored within a certain capacity you need to know three things:
 - The size of the data being stored
 - The available space (capacity) for storing the data
 - How to convert between the relevant units

Worked example

If the size of an average digital photograph is 8MB, how many photographs can be stored on a 32GB SD card?

1. Convert the available capacity from GB to MB by multiplying by 1024:

$$32 * 1024 = 32,768 \text{ MB available storage}$$

2. Divide the available space by the space required for one image to find how many can be stored in total:

$$32,768 \text{ MB total storage} / 8 \text{ MB per image} = 4,096 \text{ images}$$

Worked example

If a 1TB Hard Disk Drive costs £49.99, what is the cost-per-GB of the drive?

1. Convert the available capacity from TB to GB by multiplying by 1024:

$$1\text{TB} * 1024 = 1024 \text{ GB available storage}$$

2. Divide the total cost of the drive by the number of GB:

$$£49.99 / 1024 = £0.048 \text{ per GB, roughly equal to 5p per GB}$$



WIRED AND WIRELESS NETWORKS

CHAPTER 4

Types of network

Benefits of connecting computers to a network

Connecting computers in a network has a number of benefits including:

- **Data can be exchanged** between computers without the need for transferring files on memory sticks and other external storage
- **Resources can be shared** such as printers, scanners and Internet connections
- **Managing computers** (updating software, fixing problems) can be done centrally by a network manager

However, there are also risks to networking computers together. For example, **hackers may target a network** to gain access to several computers and **malware** can **spread** from one infected machine to all others on the network if anti-malware software is not used.

Local area networks (LANs)

- A Local Area Network (LAN) is a collection of computers that are **connected to together** over a **small** geographic area
- LANs are typically found in **homes, schools, universities** and **single-site companies** and organisations
- The hardware required to setup a LAN is usually **owned and maintained by the organisation that uses it**
- LANs can be **wired, wireless** or a mixture of both

Wide area networks (WANs)

- A Wide Area Network (WAN) is a collection of computers that are connected together over a **large** geographic area.
- WANs are used by **multinational organisations** that have office locations in countries throughout the world that need to be connected together.
- However, we all make use of one particular WAN: **the Internet is the largest WAN** in the world.
- The hardware required for a WAN is often owned and maintained by large telecommunication companies and is **hired** or **leased** by the people who wish to use it.

Network application models

Client-Server networks

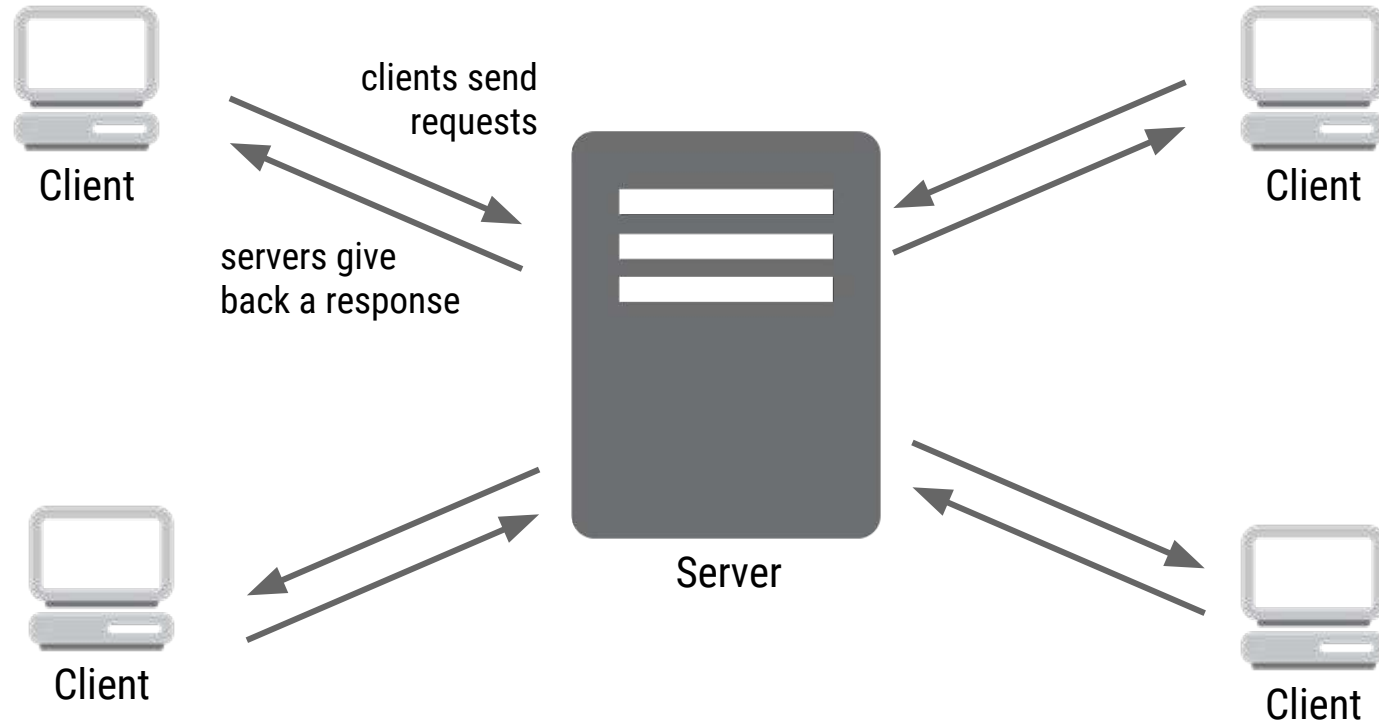
- In the client-server network model, computers take the role of either a **central server** or one of many **clients**.
- The server computer(s) provides **services** to clients such as storing files, email mailboxes or hosting a website.
- A client computer sends a **request** to the server which it **processes** and then returns a **response** back to the client.
- For example, a web browser (client) might **request** a webpage from a web server. The web server will **respond** by sending back a copy of the page to the client.

Client-Server networks

Different types of server provide different services, for example:

- printer servers provide access to printers
- file servers host files
- authentication servers check whether a provided set of username and password matches what is in its database and responds to let a client know whether to allow access to a resource or not.

Roles of computers in a client-server network



Advantages of client-server networks

- Users' files can be **stored centrally** on the server so that they can be **accessed from any connected client computer**.
- **Backups** can be made centrally ensuring that all files will be included.
- Team members can work on the **same version of files**, ensuring that mistakes are made through everyone working on their own copy.
- Software and security **updates** can be managed centrally by IT managers without the need to individually update every client computer.
- Client machines can be monitored centrally.
- Users' accounts can be managed centrally, including changing passwords for users and recovering lost files.

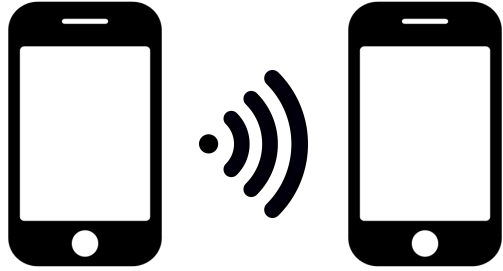
Disadvantages of client-server networks

- If the server becomes unavailable users cannot access their files.
- Server hardware is typically more expensive than ordinary computers, increasing the upfront cost.
- A cyberattack (such as a ransomware attack) needs only to focus on the server rather than each individual client.
- Servers can become overwhelmed by too many requests, preventing clients from accessing their services. This can happen during times of heavy load (such as buying concert tickets the moment they are released) or when a hacker performs a Denial of Service attack.

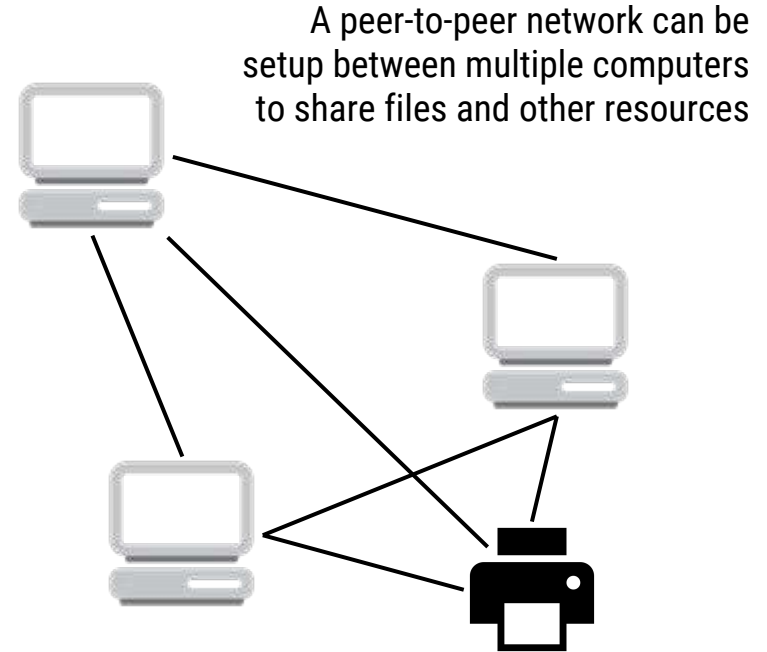
Peer-to-peer networks

- In a peer-to-peer (P2P) network, each computer (called a **peer**) is connected directly together without a central server
- Peer-to-peer networks are **easy to set up** and are often used for:
 - ad-hoc file-sharing between two devices (e.g. AirDrop on iOS)
 - streaming video from a computer or mobile device to a TV
 - wireless printing from a laptop, tablet or phone
 - streaming audio from a device to a Bluetooth speaker
 - sharing mobile Internet connections via "personal hotspots"
- Each peer is responsible for its part of the maintenance and functioning of the network

Examples of peer-to-peer networks



An ad-hoc peer-to-peer network can be set up between two mobile devices to share images, videos or other files



Advantages and disadvantages of peer-to-peer networks

Advantages

- ✓ Easy to set up
- ✓ No expensive or dedicated hardware required
- ✓ No single point of failure makes peer-to-peer networks more robust

Disadvantages

- X Without a central server there is no central management and maintenance of the network meaning that software and security of peers may need to be updated individually
- X No centralised backup of files as files are distributed amongst peers
- X Files are duplicated when transferred between devices, leading to multiple versions that can become out of date
- X Peers may go offline whilst being accessed (for example if a device loses its WiFi connection or is simply switched off)

Network Hardware

Network Interface Controller (NIC)

- In order to connect a computer or device to a network that device must have some form of **network interface**.
- A network interface is what provides the "bridge" between a device and a network.
- A **network interface controller** (NIC) can have different forms depending on the transmission media used, but the most common provide either a RJ45 "ethernet" port to connect to a wired network or a radio transmitter/receiver for connecting to WiFi networks.
- Network Interface Controllers used to be separate hardware components that could be added to computers but they are almost always integrated into the motherboard of a device these days.

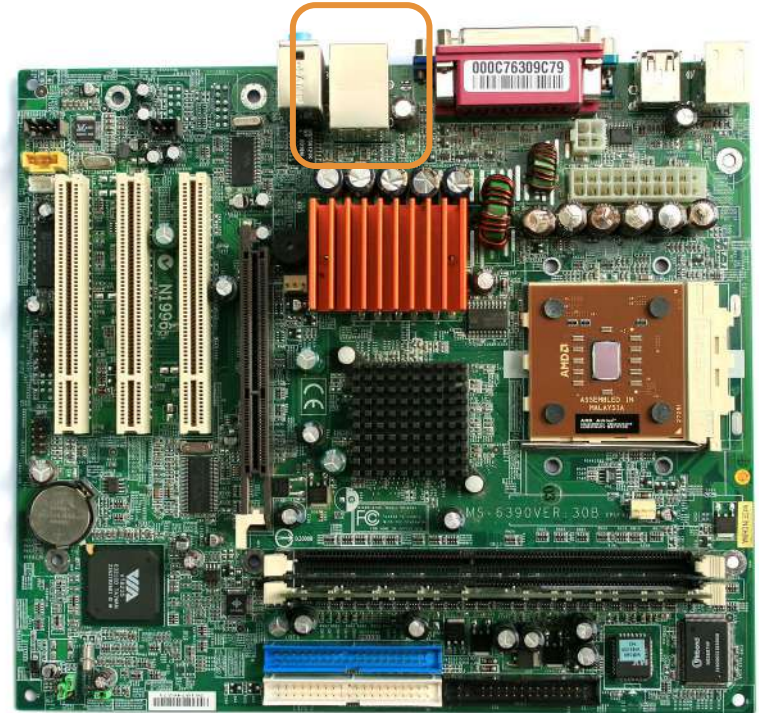
Network Interface Controller (NIC)



A network interface card providing connections to a wired network.

Image by [blickpixel / 1860 images](#)

A network interface integrated within a computer motherboard.
Image by [Jonathan Zander](#)



Switches

- A network switch is a piece of **network hardware** that allows multiple computers and devices to connect together to **form a wired network**.
- As data is sent between devices, the switch **learns the MAC (Media Access Control) addresses** of the devices that are connected to it.
- Switches can then provide **secure communication** between devices by forwarding data packets to the **intended destination device only**.
- Switches do this by using the sender and destination MAC (addresses within the data packet's **header**.
- Multiple switches can be connected together to increase the number of devices on the network.
- Network switches are used to form **star topology** networks.

Switches



Image by Jon 'ShakataGaNaI' Davis



Image by Geek2003

Routers

- A router is a piece of network hardware that **connects networks together**, for example connecting a home LAN to the Internet.
- Routers work by inspecting the destination IP (Internet Protocol) address stated within a data packet and determining whether or not the destination is on the local network or not. If not, the packet is passed onto the connected network.
- Routers form an essential part of the packet switching process that enables data to be sent around the world over the Internet.

Caution! What is often referred to as a "broadband router" is typically a device that combines the features of three network devices - a switch, wireless access point and a router.

Wireless Access Points

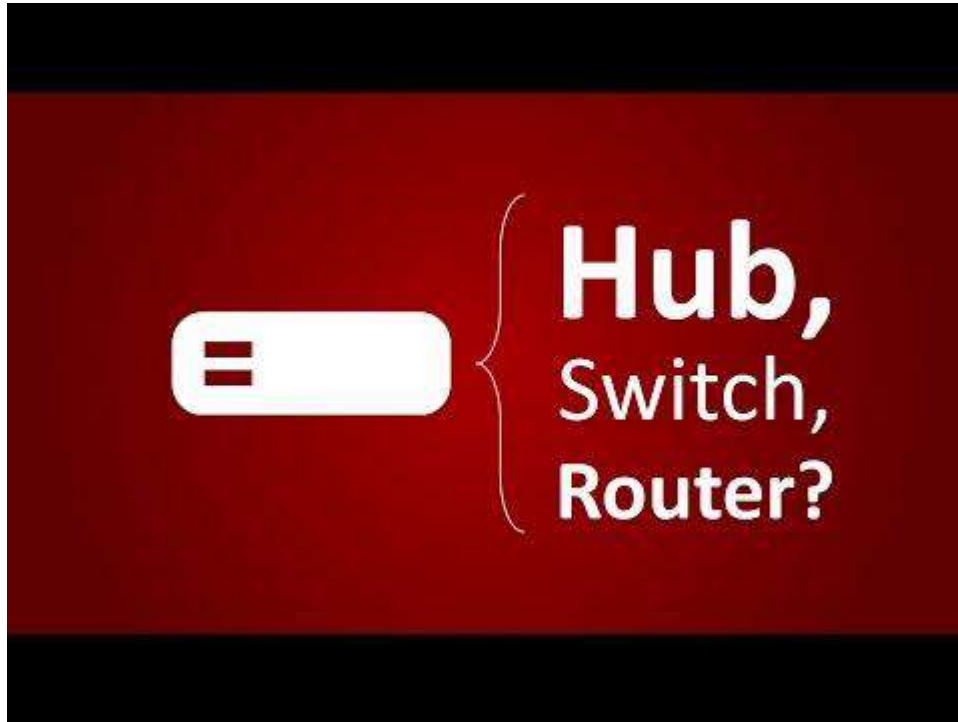
- A Wireless Access Point (WAP) is an item of network hardware that connects to a network switch and **allows wireless (WiFi) devices to connect to the network.**
- Wireless Access Points **broadcast an SSID** (name) for the wireless network so that wireless devices (clients) know which to join.
- Because the radio waves that wireless devices use to communicate are open for anyone to access, Wireless Access Points use **encryption** to prevent unauthorised individuals from reading the transmitted data.
- Wireless Access Points also implement **authentication** to prevent clients from joining the network without providing the correct passphrase or key. WPA2-PSK (WiFi Protected Access 2 - Pre-shared Key) is currently the most secure means of authenticating devices on a wireless network.

Wireless Access Points



Image by [Macic7](#) at [English Wikipedia](#)

Video: Comparing network devices



This video also covers **hubs** which are not on the OCR specification but an understanding of what a hub is helps support understanding what a switch does, which is on the specification.

Video by [PieterExplainsTech](#)

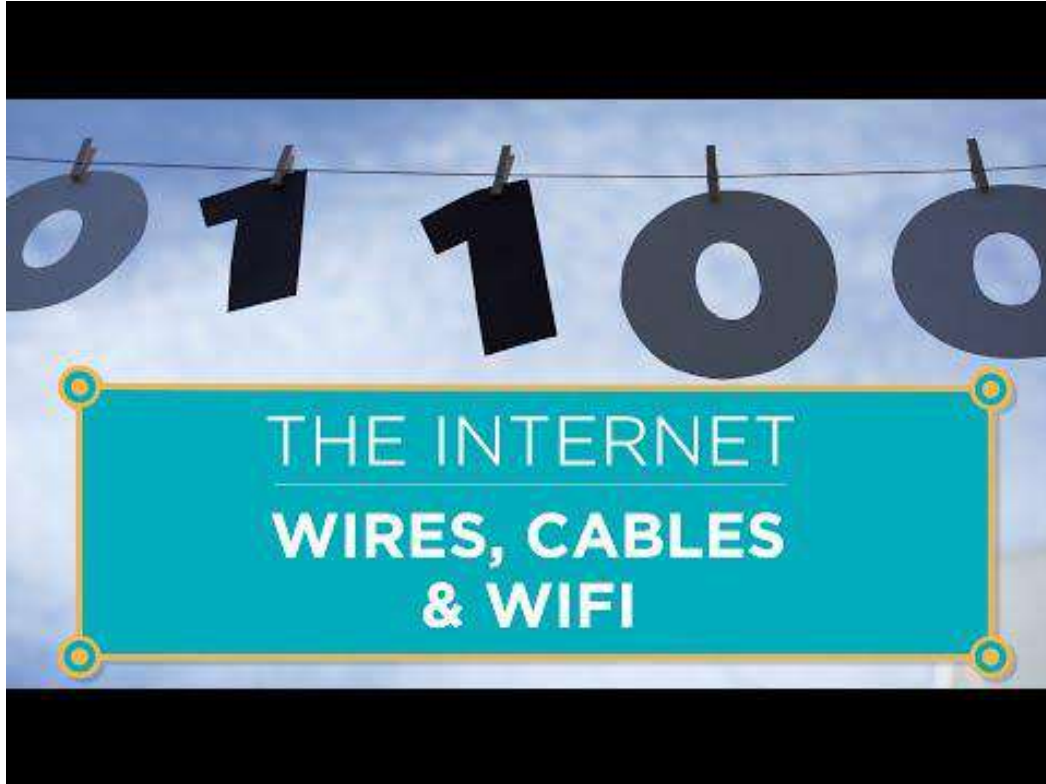
Transmission media

Transmission media are different means of physically transferring data along a network and they include:

- Twisted-pair copper wire cable
- Fibre-optic cable
- Radio waves

Each transmission medium has its own advantages and disadvantages, making it suitable to certain applications.

Video: Comparing transmission media



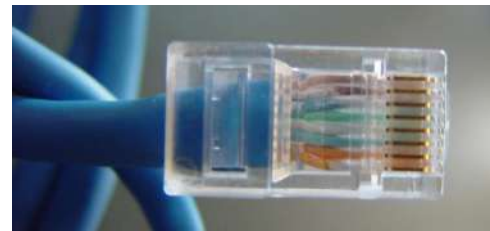
Video by [Code.org](https://code.org)

Twisted-pair copper cable

- This is often referred to as "ethernet cable" and is the **most common networking cables** that you will see in home and small-business LANs.
- Inside the cable are **eight individual wires, arranged into pairs** that are twisted together to **reduce interference** from other signals and therefore improve transmission speed (bandwidth).
- Twisted-pair cable can have different ratings that indicate how quickly it can reliably transmit data and over what range. Category 5e (Cat-5e) and Category 6 (Cat-6) cables are most common and can transmit data upto 10Gbit/sec



Twisted pair cables are typically terminated with RJ45 "ethernet" plugs



Fibre-optic cable

- Fibre-optic cables are made of many **thin glass strands** (fibres) that can **transmit binary data as pulses of light**.
- Fibre-optic cables have a **very high bandwidth**, meaning that they can transmit data very quickly (up to 100Tb/s!) over **long distances**.
- Because fibre-optic cables use light to transmit data they **do not suffer from interference**.
- Fibre-optic cables are used to **connect wide-area networks across large geographic areas**, including the cables that travel under oceans to connect different continents to the Internet.
- Fibre-optic cables are also used for **faster home broadband connections** as it can transmit data faster over long distances without interference.

Fibre-optic cable



Image by [blickpixel](#) / 1860 images

Radio waves

- Radio waves are the transmission media of **wireless networking technologies**, including WiFi and Bluetooth.
- Radio waves form part of the **electromagnetic spectrum**, along with gamma ray, x-rays, visible light and microwaves.
- The strength (amplitude) of a radio wave **decreases as it moves further from its transmission source**, which is why you might need a "WiFi extender" to connect to the Internet from the bottom of your garden.
- Radio waves are also subject to **interference** from other radio signals of the same or similar frequencies. To help reduce this, WiFi hotspots check which frequency channels are least congested when they select one to broadcast on.

Video: How does WiFi work?



Video by [BBC Earth Lab](#)

Comparing transmission media

	Twisted-pair copper cable	Fibre-optic cable	Radio waves
What is it used for?	Connecting devices within a LAN by wires	Connecting wide area networks over long distances	Transmitting data in wireless network technologies such as WiFi and Bluetooth
How is data transmitted?	Electrical current through the copper wire	Light pulses transmitted down glass fibres	Fluctuations in radio signals
Bandwidth	Fast; between 100Mb/s to 1Gb/s	Very fast; potentially up to 100Tb/s	Depends on interference and obstacles, potentially up to 300Mb/s
Interference	Twisted pair cables help protect against interference	Effectively none	Prone to interference from other sources of radio waves including other WiFi networks
Distance	Suitable for distances up to 100m	Over 100km	Depends on obstacles such as walls, but around 60m

Factors that affect network performance

A network's performance can be measured in terms of how much data can be sent in a second (**bandwidth**), the delay between a signal being sent and received (**latency**) and how reliably packets are transmitted (**error rate**).

Factors that affect network performance include:

- **Number of devices and users** - the network's maximum bandwidth is shared between all connected devices, so the more devices, the less everyone gets to use
- **Transmission media** - using wifi will result in slower data transfer speeds and a greater number of lost or corrupted data packets
- **Interference** - wireless and some wired transmission media are prone to electromagnetic interference that can corrupt data as it travels
- **Obstacles** - physical obstacles such as a thick brick or metal walls can prevent radio waves from travelling

The Internet

What is the Internet?

- The Internet is a **worldwide collection of computer networks** (sometimes referred to as a "network of networks").
- The set of rules that ensure that devices can work together on the Internet is called **Internet Protocol** (IP). Every computer on the Internet has an **IP address** that is used to send data from one device to another.
- **Routers** are essential to the Internet as they pass data packets between the interconnected networks that form the Internet via a process called **Packet Switching**.
- The internet is like a major road network connecting places together. Different vehicles can use the road network to send things from one location to another. These vehicles represent the various **applications** that make use of the Internet, such as the World Wide Web (WWW), email, multiplayer games and video streaming services.

Domain Name System (DNS)

- The Domain Name System (DNS) keeps a record of the **IP address** for each web server that is associated with a particular **domain name**. It acts like a worldwide phonebook that relates domain names to IP addresses.
- DNS is the system that allows users to type in a website's URL as a domain name such as <http://www.google.com> rather than needing to enter the destination IP address (<http://216.58.213.67>)
- When an application sends a request to a destination address that includes a domain name, the Internet Service Provider's DNS server is contacted to ask for the matching IP address. If the ISP's DNS server does not have a record for that domain name, it makes a request to the DNS server that it is connected to. This process continues until a DNS server is reached that can return the appropriate IP address. The sending computer can then send their data packets to appropriate destination by using this IP address.

Universal Resource Locators (URLs)

- URLs are unique identifiers (addresses) used to access servers and resources on the Internet
- They are made up of several parts:

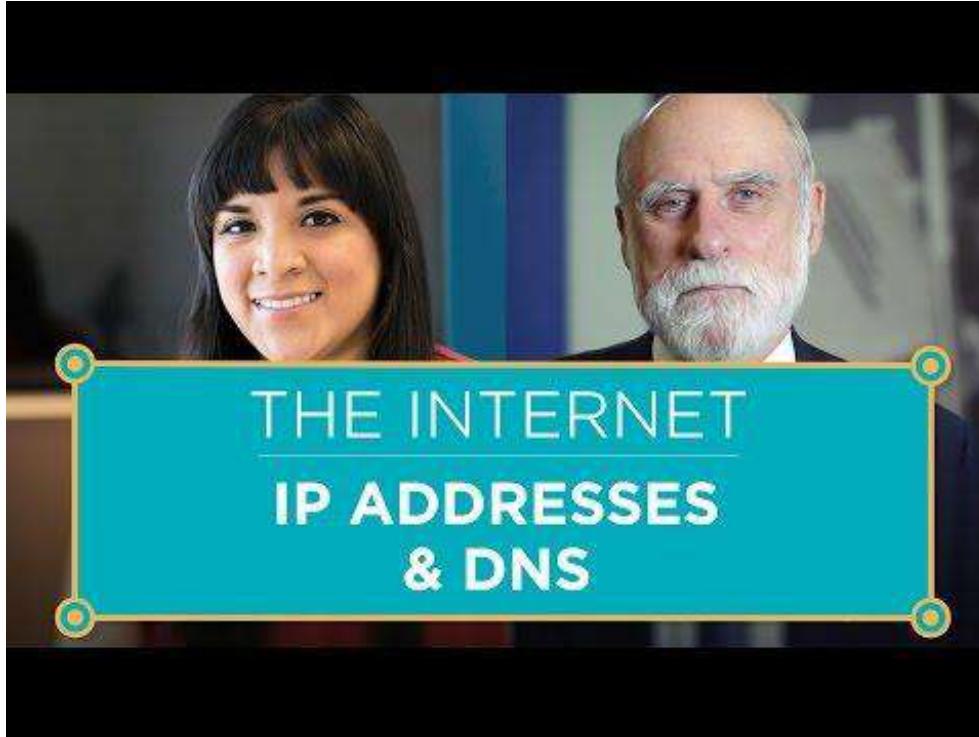
The **protocol** being used to communicate

The **path** the requested file on the server

`http://www.bbc.co.uk/news/technology.html`

The **domain name** of the **web server** that hosts the website or page
- this is replaced with an IP address from DNS

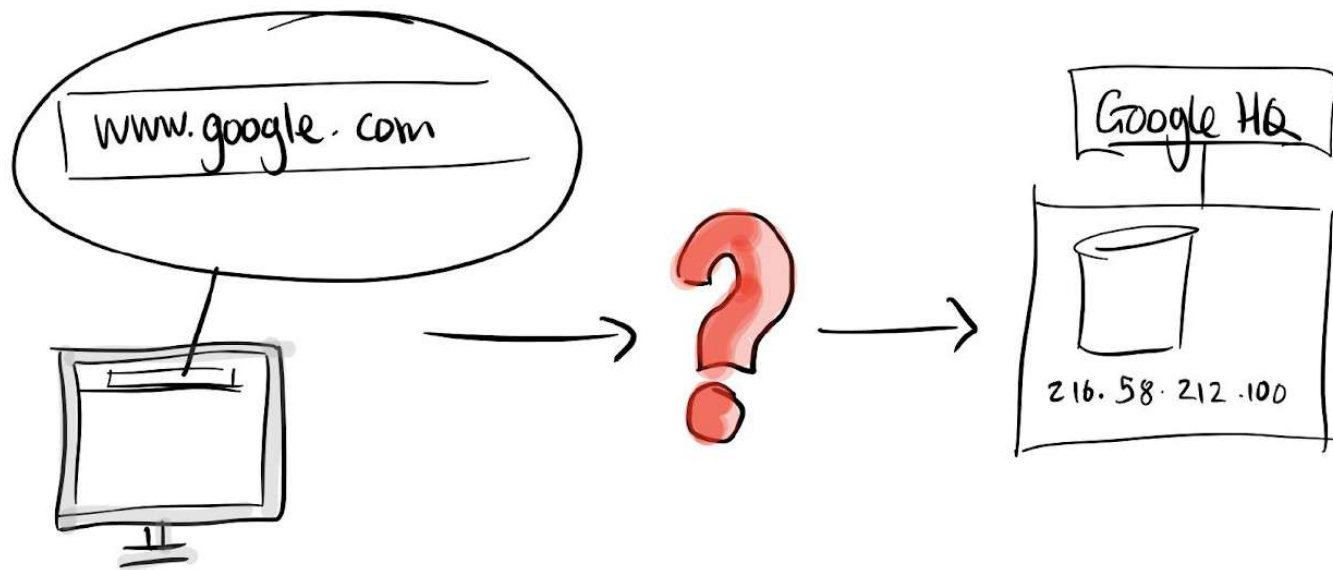
Video: More on IP and DNS



Watch from 4:10 for a great, succinct explanation of DNS

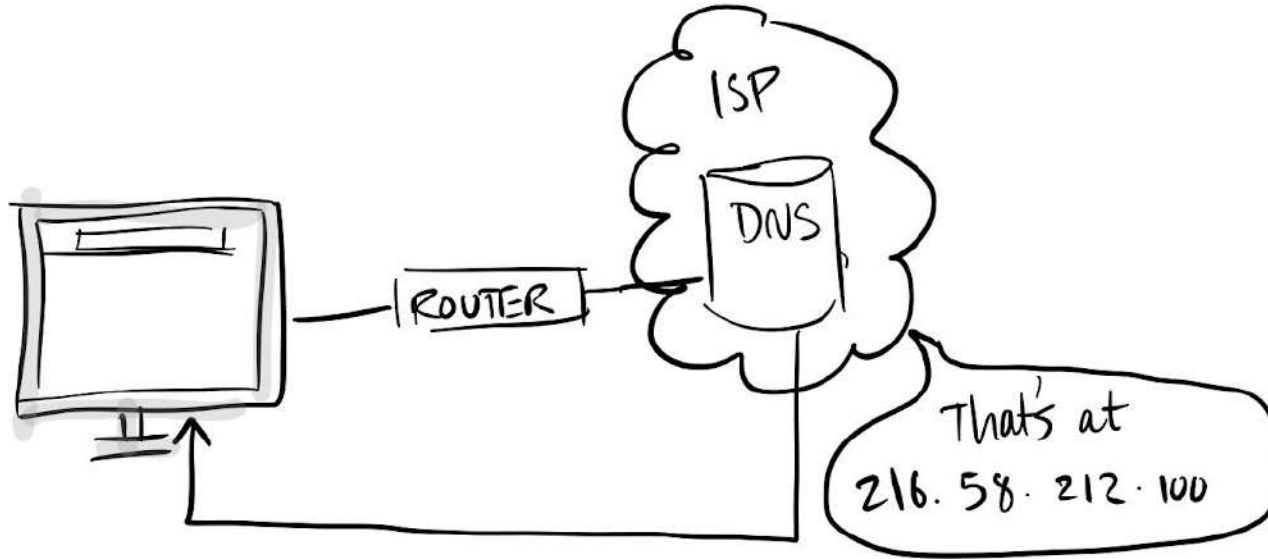
Video by [Code.org](https://code.org)

What happens when you access a website?



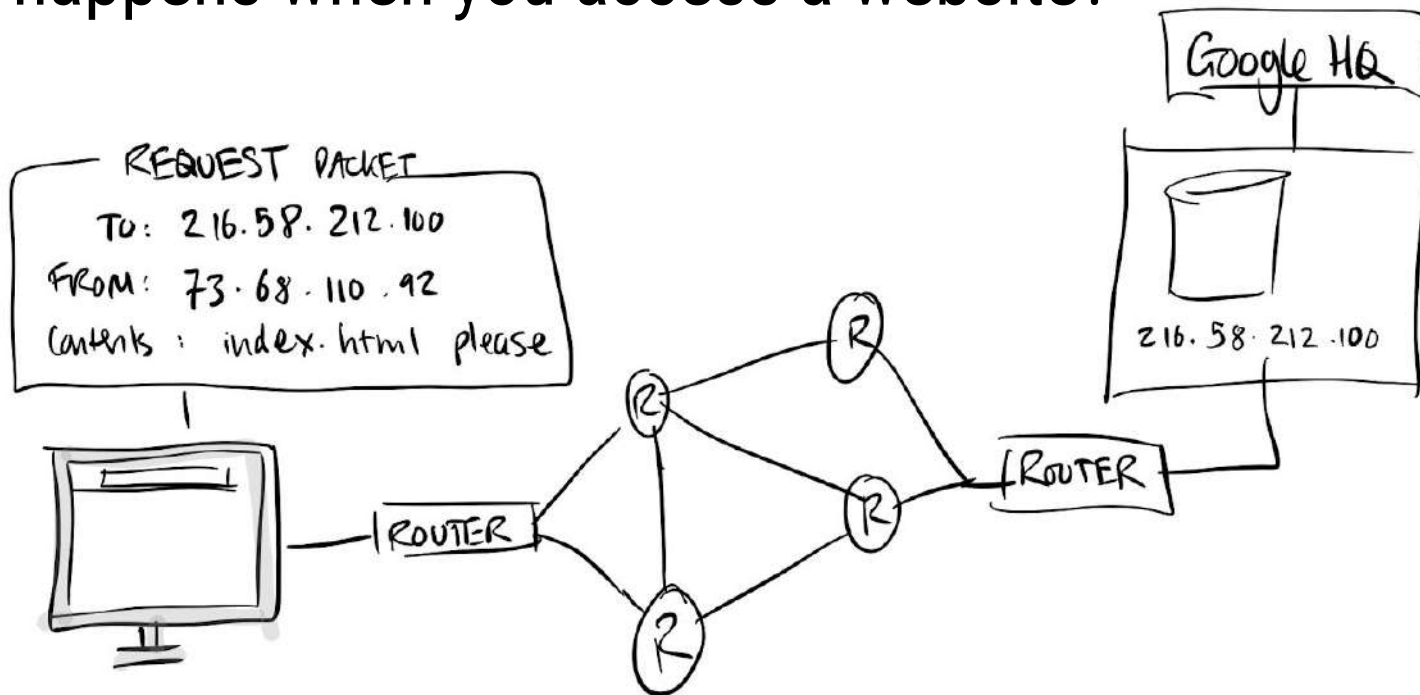
Step 1: The URL for a requested web page is entered into a web browser. The web browser generates an HTTP request to send to a web server **but** it doesn't yet know the IP address of the server!

What happens when you access a website?



Step 2: The OS on the client computer sends a request to its nearest DNS server to ask for the IP address that maps to the domain name (host name) specified in the requested URL.

What happens when you access a website?



Step 3: The client computer can now send its HTTP request over the Internet to the desired website based on its IP address.

Accessing a via DNS website - a summary

1. The **URL** of a web page is entered into a web browser on the **client**
2. The client asks its nearest **DNS server** for the **IP address** of the **server** that **hosts** the web page or website
3. The **IP address** of the web server is sent back to the **client**
4. The client sends an **HTTP request** to the IP address of the web server
5. The web server **responds** with the requested **page**, sending it directly to the client using the client's **IP address** specified in the HTTP request **packet**

Uses of the Internet

There are many uses of the Internet, but amongst the main ones are:

- Hosting websites on web servers
- Email
- Secure file transfers
- Voice-over-IP (VOIP) and video calls (Skype, Facetime, etc)
- On-demand video services
- The Cloud

Web hosting

- Web hosting is one of the principal uses of the Internet.
- Every website (a collection of webpages) is **hosted** on a **web server**. A web server is a **dedicated computer on the Internet** that **responds to HTTP and HTTPS requests** by returning webpages.
- It is entirely possible for anyone to set up their own web server, however most people use a **web hosting company** who charge a **monthly fee** to host a website in their data centres. Larger and more popular sites that generate more monthly bandwidth will be charged more each month than a smaller site that generates less bandwidth.
- Web hosting companies also offer **service guarantees**, ensuring that a website will always be available to visitors as well as **regularly backing it up** and **applying appropriate security measures** to protect it from cyberattacks.

Video: HTTP and HTML



Video by [Code.org](https://code.org)

The Cloud

- "The Cloud" is a term that refers to **storage, services** and **applications** that **exist on the Internet**, rather than on a **local** device such as your PC or phone
- These services exist in giant **data centres** all around the world, some of them underground or even underwater (but none are in the sky!)



Examples of Cloud services and applications

- **File storage and sharing** - Google Drive, Dropbox, iCloud Drive
- **Streaming** services - Netflix, BBC iPlayer, Spotify
- Online **backup** - Mozy
- **Applications** - Google Docs, Microsoft Office 365, Gmail
- **Server hosting** - Amazon Web Services, Windows Azure

Accessing these services requires an **internet connection** and either a **browser** or a mobile **app** and often involves paying a **monthly access charge**.

Advantages of using the Cloud

- + Files and applications can be accessed from any Internet-connected device, anywhere in the world
- + Users can easily buy additional storage when they run out of space
- + Cloud applications are always the latest, up-to-date version
- + Data is regularly backed up and kept secure by the hosting company
- + Software and servers are maintained without having to employ specialist IT managers

Disadvantages of using the Cloud

- A reliable Internet connection is required to access your files and services
- Users have little control over the security of their data
- Data stored on the Cloud could be targeted by hackers more easily than if it was stored locally
- It can sometimes be unclear who legally owns the data that is stored
- Cloud providers can change their prices and ongoing subscription fees may become expensive

Video: A tour of Google's Data Centres



Video by Google [G Suite](#)

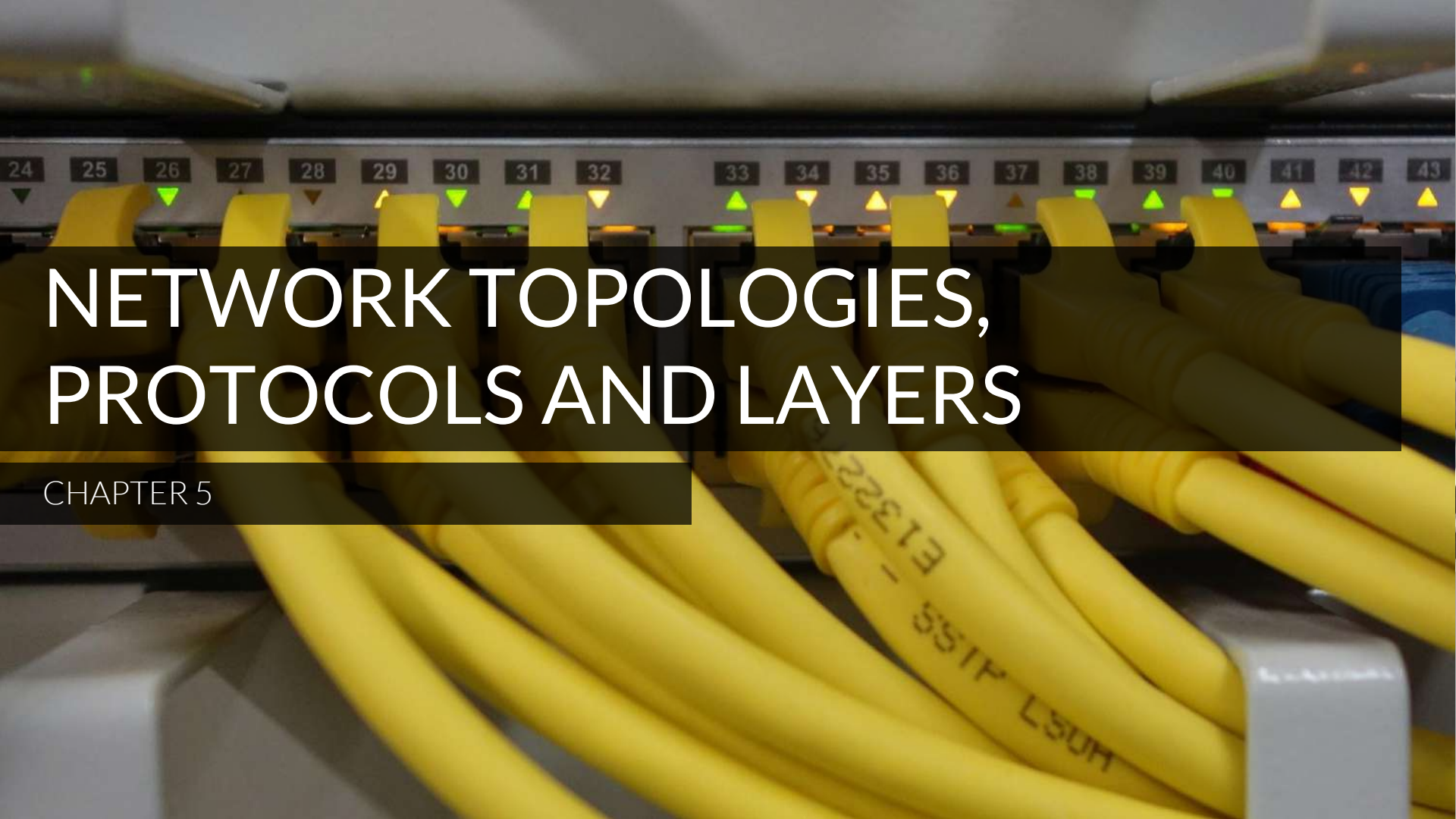
Virtual Networks

Virtual networks and Virtual Private Networks (VPN)

- A virtual network is a **software-based network** that can **share the same physical hardware** with other virtual networks.
- A common type of virtual network is a **Virtual Private Network (VPN)**.
- A VPN allows two or more devices on the Internet to communicate **as if they were on the same local area network (LAN)**.
- A VPN sets up a "**tunnel**" is setup between communicating devices. **Encryption is applied** to keep the communication between the devices private.
- Commercial organisations often use VPNs to allow employees to securely connect to their workplace network from home or other **remote location**. This is much cheaper than setting up a dedicated private WAN.

Virtual LANs (VLANs)

- A Virtual Local Area Networks (VLAN) is another form of virtual network.
- A VLAN allows an organisation to **divide one physical network** into distinct segments to operate **multiple, separate networks** across the same physical hardware and cabling.
- Advantages of doing this include:
 - **Simplifying the network design** by having different departments or types of device on their own network
 - **Increasing security** as devices on one VLAN cannot communicate with devices on another VLAN, even though they are physically connected together. This reduces the risk of an employee accessing systems that they shouldn't have access to (insider attacks) and the threat of malware spreading across all devices on the network.

A close-up photograph of a network switch or patch panel. Numerous yellow Ethernet cables are plugged into the ports, which are numbered 24 through 43. Small green and yellow indicator lights are visible above the ports. A semi-transparent dark banner is overlaid across the middle of the image, containing the title and chapter information.

NETWORK TOPOLOGIES, PROTOCOLS AND LAYERS

CHAPTER 5

Star and Mesh Topologies

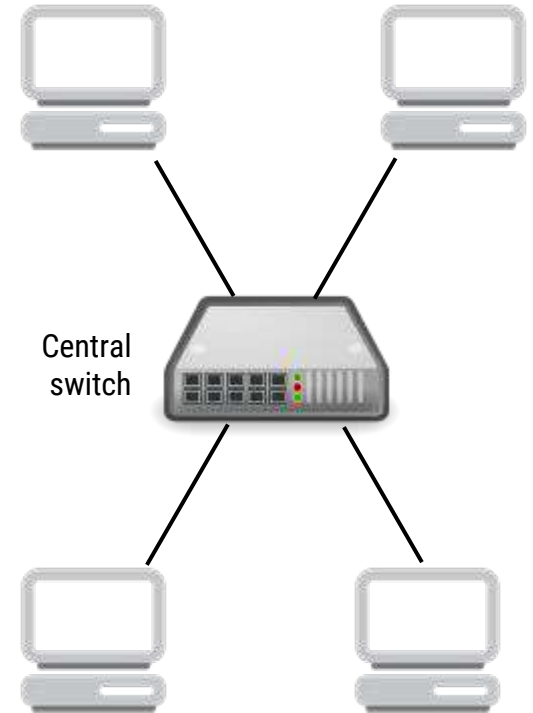
What are network topologies?

- A network's topology is the organisation of the connections between the devices.
- In other words, when we talk about network topologies, we are referring to how the network is 'laid out' and how the devices should connect to one another.
- There are two main topologies in use today that you need to know about: the **star** topology and the **mesh** topology.

The star topology

- A network that adopts the star topology has devices that are **indirectly connected** via a **central hub or switch**.
- The lines (representing connections) that come out from the central switch to the devices are said to form a shape that looks like a star (hence its name).
- Caution! The star topology should not be confused with client-server networks, even though they look similar in diagrams. Devices arranged in a star topology connect via a central switch, but the applications that they run could still adopt a peer-to-peer approach or a client-server model.

Devices (could be clients or servers if a client-server network or peers if a peer-to-peer network)



Advantages and disadvantages of the star topology

Advantages

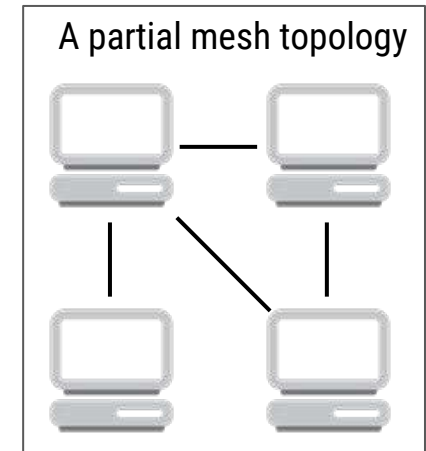
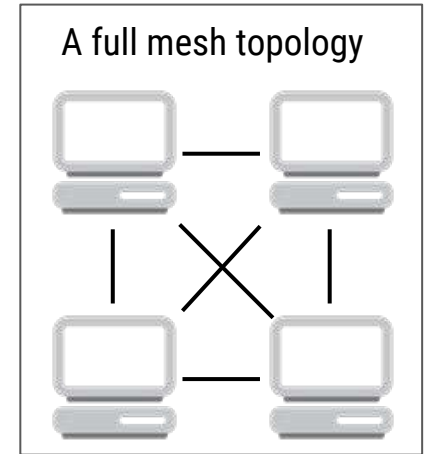
- ✓ The rest of the network is unaffected if the connection for a particular device fails
- ✓ The switch enables packets to be broadcast between devices without colliding
- ✓ The switch can screen packets and remove corrupted packets, increasing the security of the network
- ✓ It is very easy to add new devices to the network

Disadvantages

- x The whole network fails if the central switch fails (there is a single point of failure)
- x It can be expensive to lay cable all the way from the switch to each individual device

The mesh network topology

- In a network based on the mesh topology, all devices are connected either directly or indirectly to each other **without a central switch**.
- There are two types of mesh network: **full mesh** and **partial mesh**. In a full mesh network, every device is directly connected to every other device. In a partial mesh, direct connections only exist between some devices (though all devices can communicate with each other indirectly).
- Caution! Again, diagrams of the mesh topology can often look very much like those for peer-to-peer networks but they are not to be confused with each other. A mesh network topology can support a client-server network where one of the devices is a server and the others are clients that connect to it either directly or via each other.



Advantages and disadvantages of mesh networks

Advantages

- ✓ By removing the central switch there is **no single point of failure**, making the network **very robust**. If any device ("node") fails, data can be sent down a **different route** to its destination.
- ✓ The mesh topology is **well-suited to wireless networks** that can use each wireless device to route data through the network, thus **extending the range** of the network.

Disadvantages

- x **Wired** mesh networks are very **expensive** as every device needs to be wired to every other device. This is also very impractical in terms of the number of network interfaces required on every device.
- x As there is no central point that all data travels through it is **not possible to screen all data** for security purposes.

WiFi

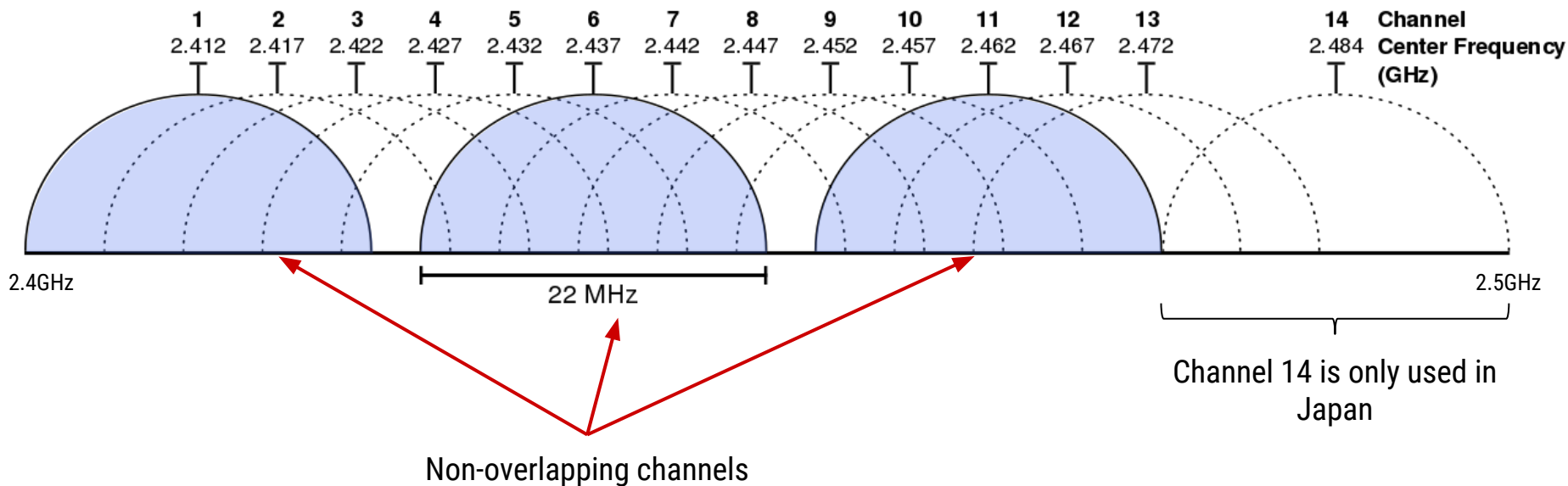
What is WiFi?

- WiFi is a set of standards (or **protocols**) that **define how network devices can communicate wirelessly** over radio waves.
- The official name of the **WiFi** protocol is IEEE 802.11x and it defines several "rules" such as:
 - What frequency should the radio waves be?
 - How powerful should the transmitter be?
 - How can data be secured?
 - What should happen if data is lost during transmission?
- There are different **WiFi standards** that operate at different speeds, for example 802.11g (54Mb/s), 802.11n (65 to 600Mb/s) and 802.11ac (78Mb/s to 3.2Gb/s).
- By creating devices that **adopt the same WiFi standards**, wireless devices from different manufacturers are **guaranteed to be compatible**.

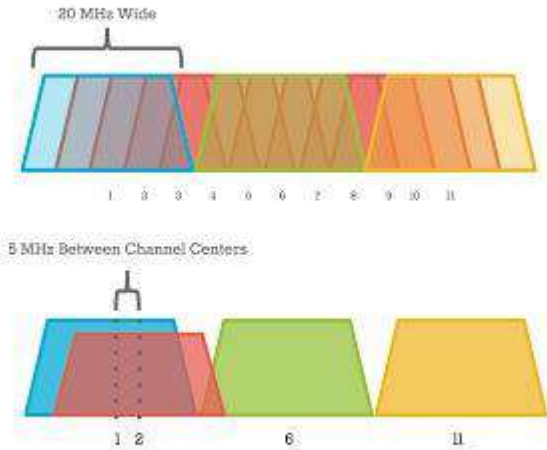
WiFi frequencies and channels

- Most WiFi standards transmit data using radio waves in one of two **frequency bands** - either 2.4GHz or 5GHz.
- In order to maximise how much data can be sent, each band is **subdivided into smaller channels**.
- There are 13 channels in the 2.4GHz frequency band and each spans a range of 22MHz, however **most of these channels overlap causing significant interference between wireless networks**. There are only 3 channels (1, 6 and 11) within the 2.4GHz band that don't overlap, reducing its overall effectiveness for supporting many networks.
- There are 24 **non-overlapping channels** within the 5GHz frequency band, allowing for many more wireless networks to exist side-by-side without interfering with one another.

WiFi channels within the 2.4GHz frequency band



Video: WiFi channels explained



Video by [MetaGeek](#)

WiFi encryption

- It's easy to **intercept** WiFi signals - they're just **radio waves** that anyone can pick up
- Different security measures can be used by Wireless Access Points to protect WiFi networks, including **WEP** (very basic), **WPA** (OK) and **WPA2** (best at the moment)
- WEP simply requires a **password** to join a WiFi network
 - The password can be easily **cracked** using a brute-force attack
 - None of the data is encrypted so it can still be **intercepted**
- WPA and WPA2 use **encryption** to **scramble** the data being sent on the network so that only the devices with the correct **key** (the sending device and the Wireless Access Point) can **decode** the data

Network protocols

Network protocols

- A network protocol is a **set of rules** that defines how **data is transmitted** between **connected devices**
- There are many different protocols that define rules for different aspects of networking, for example:
 - Ethernet - a protocol that defines how data should be physically transmitted between network hardware
 - HTTP - a protocol that defines how data should be transmitted between a web browser and a web server
 - WiFi (802.11x) - a set of protocols defining how data should be transmitted using radio waves

Rules that protocols often include

Remember that **protocol** is just a fancy word for "**rules**". These rules might include:

- How communication between devices should start and end (a process called **handshaking**)
- How **data packets** should be **structured** (for example, including sender and destination addresses specified along with the data)
- Which **network port** the communication should take place over (for example, HTTP operates on port 80)
- How to **check** if data has been **correctly received**
- **What to do** if something goes **wrong** such as a lost connection or data becoming corrupted.

Common networking protocols

Network application protocols

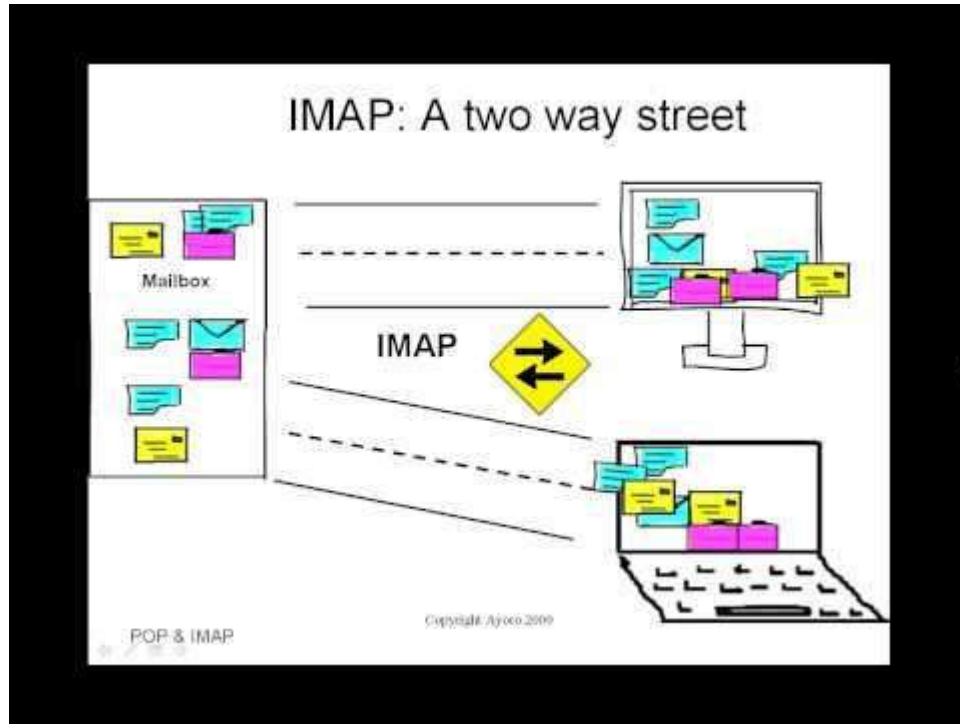
- **HTTP** - the Hypertext Transfer Protocol defines how data should be exchanged between web browsers (clients) and web servers as requests and responses.
- **HTTPS** - a secure version of HTTP that adds Secure Socket Layer (SSL) encryption.
- **FTP** - the File Transfer Protocol defines how binary and text data can be requested from a server by a client and how that data should then be transmitted to the client.

Common networking protocols

Network application protocols cont.

- **POP** - the Post Office Protocol is a common e-mail protocol that defines **how emails can be retrieved from a mail server** for a particular user. Mail clients that use POP to receive email do not report back any user activity with the email such as whether a message has been read or deleted from the client meaning that users inboxes are not kept in sync across devices.
- **IMAP** - the Internet Message Access Protocol is another e-mail protocol for **retrieving emails** however **mailbox activity is synchronised** between the client and server so that inboxes remain unified across multiple devices.
- **SMTP** - the Simple Mail Transfer Protocol defines how email messages are **sent** from an email client (such as Mail on your phone) to a mail server.

Video: IMAP vs POP3 email protocols



Video by [Ayoro France](#)

Common networking protocols

TCP/IP - two protocols that almost always work together

- **TCP** - the Transport Control Protocol **splits data** from applications into smaller **data packets** that can be sent across a network using packet switching. Each packet is made up of a **header and payload**. The header contains the **sequence number** of the packet and a **checksum** to allow recipients to determine whether it has been sent correctly. The payload contains the data from the application that needs to be sent.
- **IP** - the Internet Protocol defines how data packets should be sent between networks. An IP header is added to each packet that contains the **source and destination IP addresses for that packet**. Routers use this information to determine whether the packet's destination is on the local network or whether the packet should be passed onto another network.

Common networking protocols

Network interface protocols

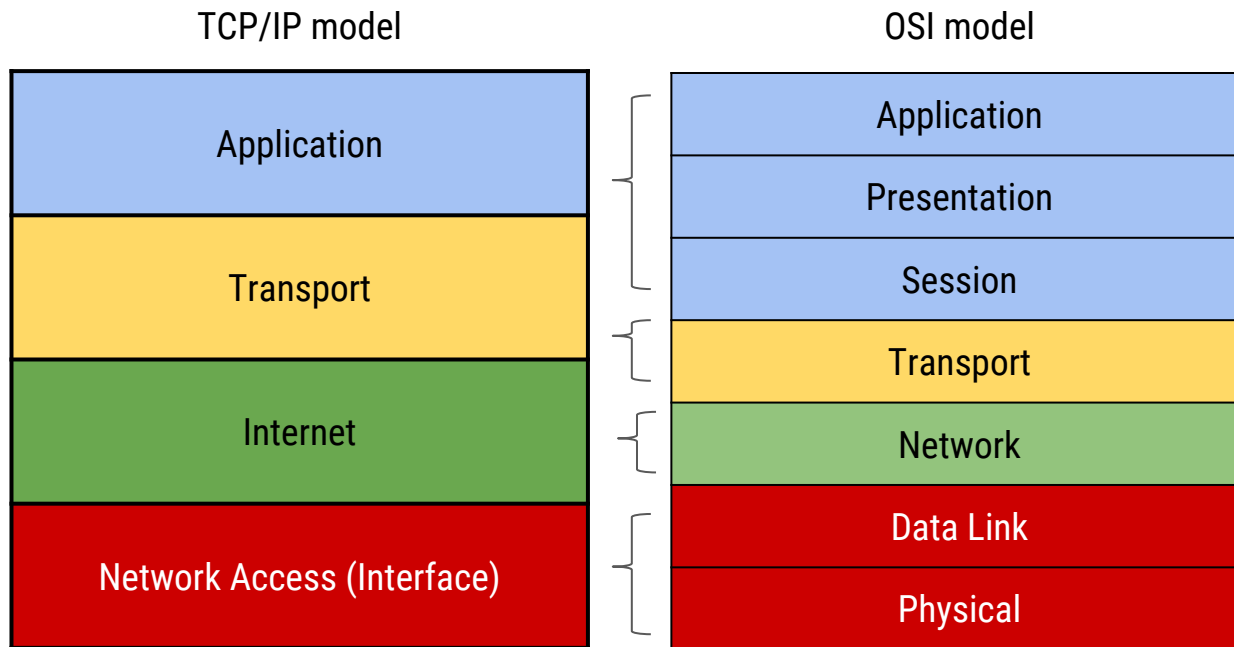
- **Ethernet** - defines how data should be physically transmitted between different pieces of network hardware via various transmission media. Ethernet uses MAC addresses to determine which device the stream of bits that make up a packet should be sent to. Ethernet also defines what to do if collisions occur on the network, where multiple packets are transmitted at the same time.
- **WiFi** - defines how data should be transmitted wirelessly via radio waves. The various WiFi standards determine the frequency bands and channels that should be used, the data transmission rates and how devices should be authenticated when they attempt to join the network.

Network protocol layers

Protocol layers

- To help conceptualise the many different network protocols, **protocol** (or network) **layer models** are often used.
- The most common protocol layer models are the 4-layer TCP/IP model and the 7-layer OSI (Open Systems Interconnection) model.
- In a protocol layer model, a protocol layer is simply a **group or collection of network protocols that work at a similar "level" within the networking process**.
- For example, one layer might group together all of the protocols that are used by network applications to send and receive data, this is called the Application Layer. Another layer might contain protocols that define how data should be sent between networks, this is called the Internet Layer in the TCP/IP model and the Network layer in the OSI model.

The TCP/IP and OSI network layer models



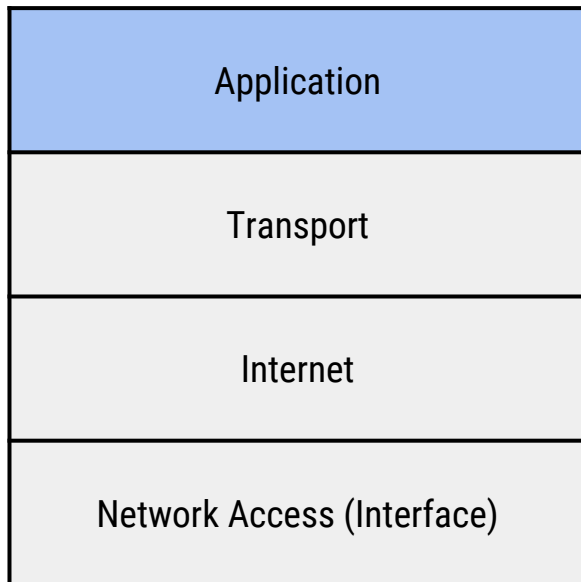
Note: You will not be expected to know the detail of the 7-layer OSI model, but it is helpful to see how it relates to the TCP/IP model as some textbooks and many websites use layer names from the OSI model instead of the TCP/IP model.

Typical protocols used within each layer

TCP/IP model

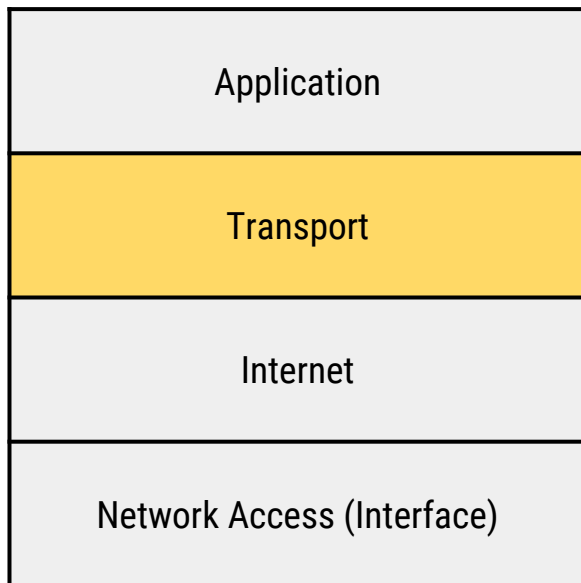
Protocol layer	Protocols operating in this layer
Application	HTTP/HTTPS, POP3, IMAP, SMTP, FTP
Transport	TCP, UDP
Internet	IP
Network Access (Interface)	Ethernet, WiFi

How data progresses through the TCP/IP network model



Applications **create** and **interpret** data, for example web browsers send request for pages and display the results.

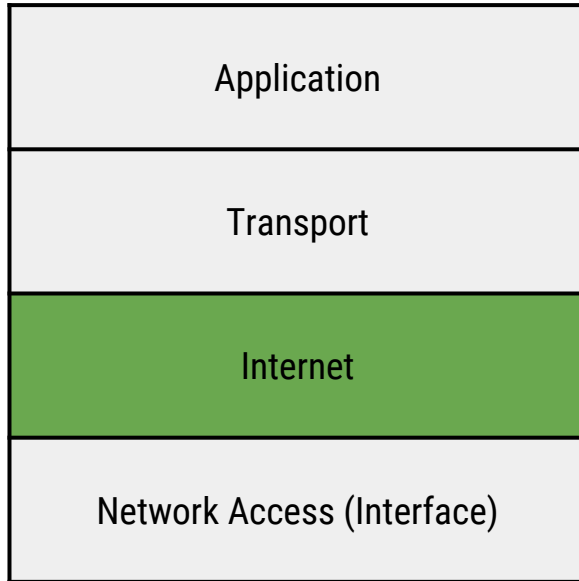
How data progresses through the TCP/IP network model



Data gets **split up** by **TCP** and **sequence numbers** are attached for sending.

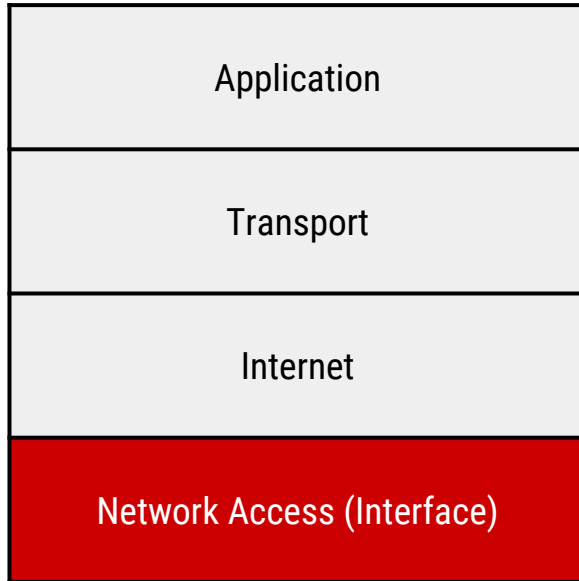
Upon receipt, TCP **checks** that packets have been received correctly and **reassembles** packets in the right order.

How data progresses through the TCP/IP network model



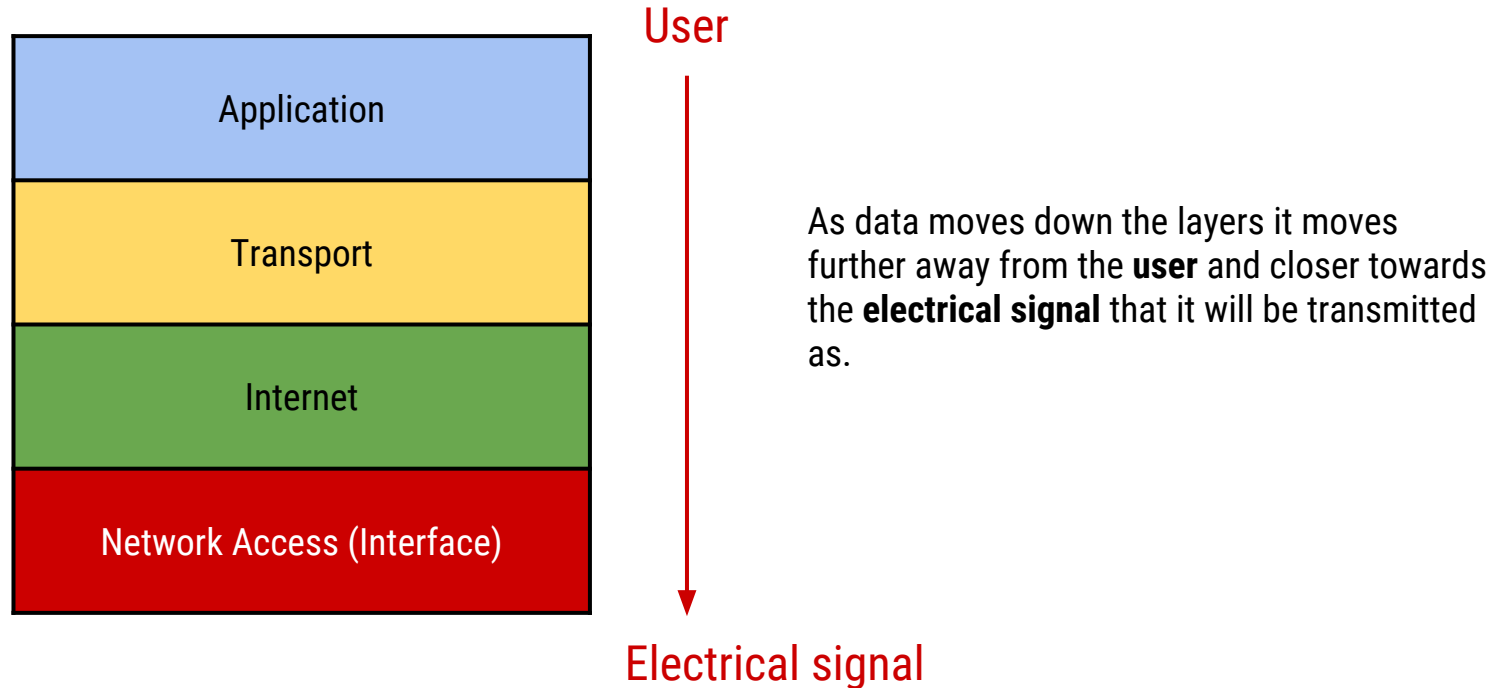
IP adds the **addresses** of the sender and destination to the packet so that **routers** know where to pass them to.

How data progresses through the TCP/IP network model



Network interface cards turn the packets into **electrical signals** that are sent to other devices via **switches**.

How data progresses through the TCP/IP network model

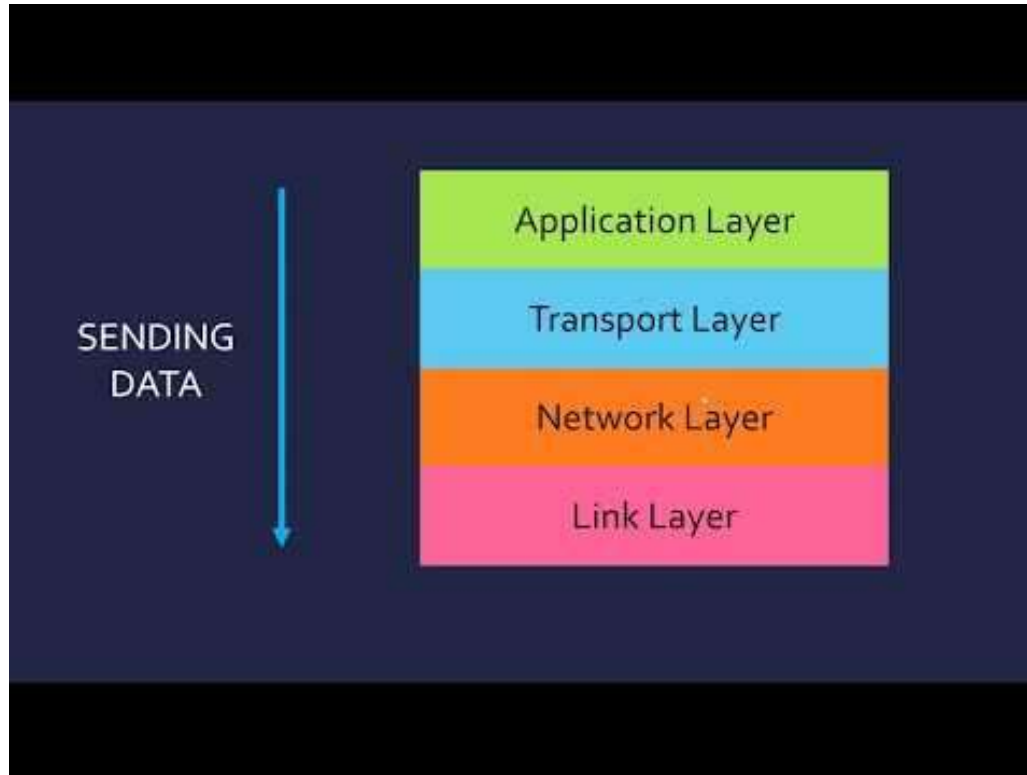


Advantages of using protocol layers

Each layer is **self-contained** but is able to send and receive data to/from the layers above and below it. This has several advantages, including:

- Software developers and hardware engineers can **specialise in understanding one layer of the network** without having to know how every layer works.
- Network layers can be **altered without affecting the other layers** (for example, the Internet didn't need to be redesigned when WiFi was invented)
- Network engineers can **diagnose and fix issues within individual layers** when problems occur.
- **Application developers can write networked applications without knowing the detail of how a network physically works.** They simply need to know how to send data from their application to the next layer down (transport), at which point the Operating System will take over the sending and receiving of data.

Video: The 4-layer network model explained



Video by Computer Science Tutor

Encapsulation

- As data is passed from one layer to another, additional information is applied to the data by protocols that operate at each layer.
- This process of "wrapping" data with more information as it passes through the layers is called **encapsulation**.
- For example, a web browser might generate a request for a web page at the Application layer. This will pass to the transport layer where it might be split up into smaller packets with a sequence number applied to each one. These packets will then pass to the Internet layer where the Internet Protocol will attach sender and destination IP addresses to each packet. Finally these packets will be sent to the Network Interface layer where they will be converted into Ethernet Frames and have the MAC address of the next piece of network hardware that should receive them added.

Network addresses

Types of network address - IP and MAC addresses

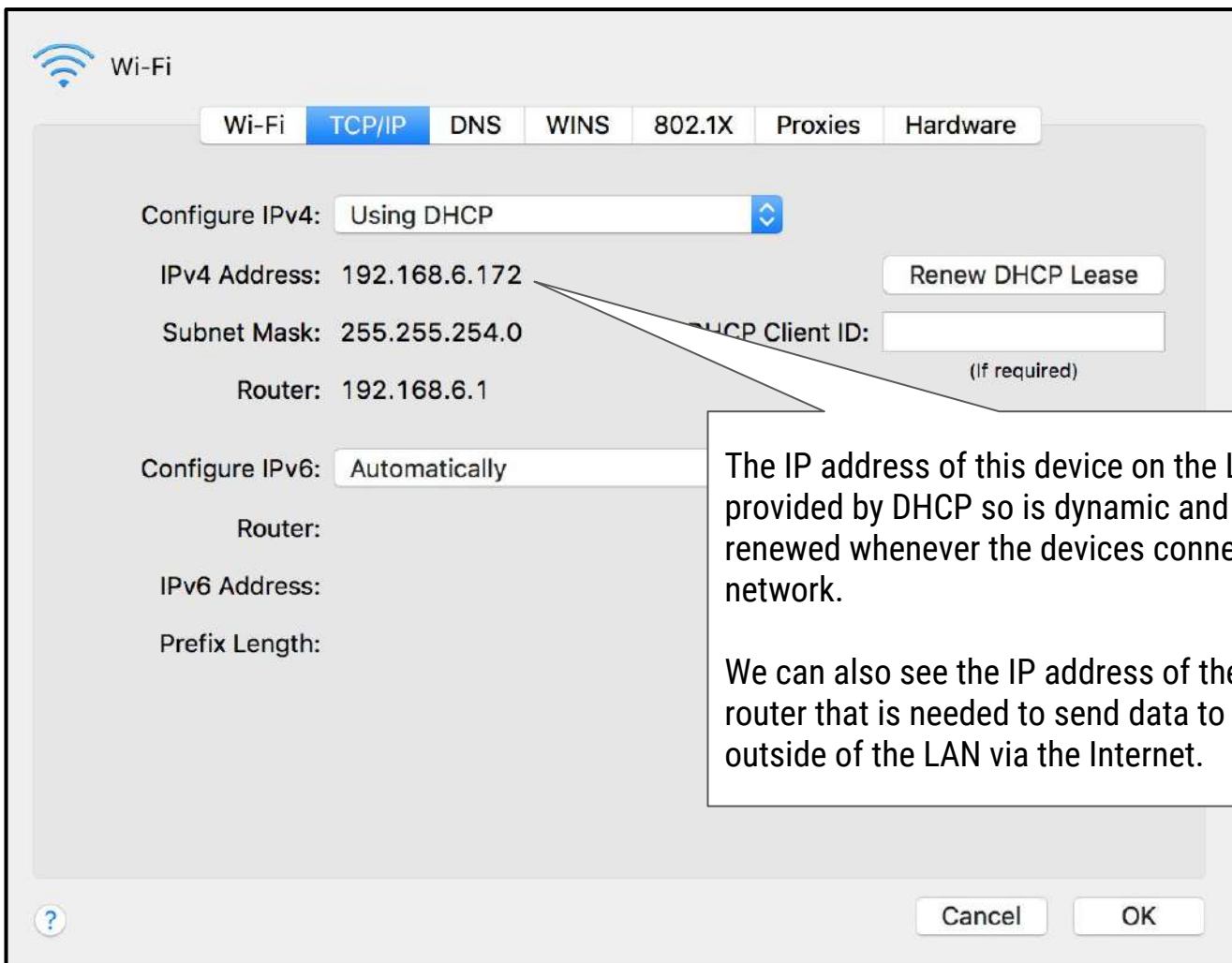
Just as an address is essential for sending a letter in the post, network addresses are used to identify where data packets need to be sent to and where they have come from.

There are two types of network address, each of which is used at different layers within the network:

- **IP addresses** - used by routers operating within the Internet layer to determine whether a packet's destination is on the local network or not
- **MAC addresses** - used by network switches operating within the Network Interface layer to identify which Network Interface (a component of a networked device) the data should be transmitted to.

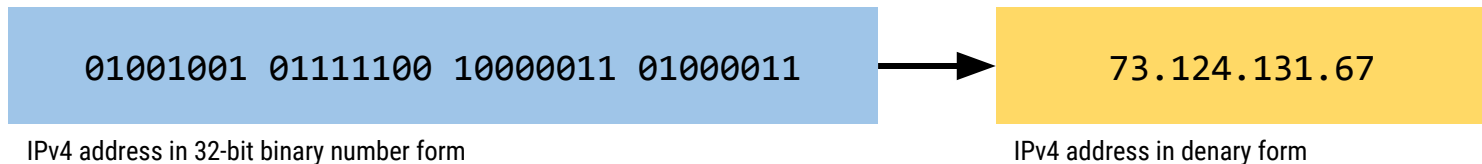
IP (Internet protocol) addresses

- IP addresses are used to communicate with devices **across different networks**, including across a wide area network (WAN).
- IP addresses must be **unique within the same LAN**, but are not universally unique.
- IP addresses can be either **static** (fixed, permanent) or **dynamic**. A device with a static IP address will have the same IP address every time it connects to the network. Devices with dynamic IP addresses will receive a different IP address each time they connect to a network. Dynamic IP addresses are provided by a DHCP (Dynamic Host Configuration Protocol) server.
- IP addresses are **used by network applications** to specify which networked device they wish to communicate with and **routers** to determine which network data should be sent to.



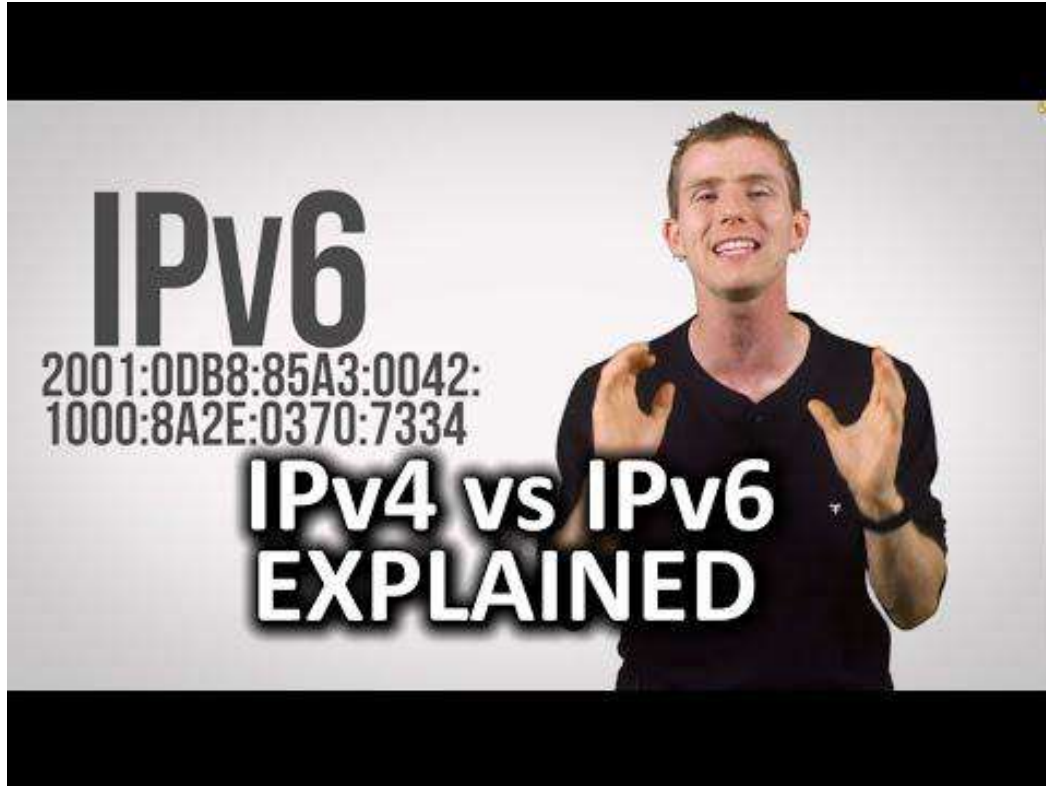
IP address structure and format

- There are two types of IP addresses - IPv4 (version 4) and IPv6 (version 6).
- IPv4 is the most common format and is made up of a 32-bit number that is most often displayed as four denary numbers separated by dots:



- With the advent of so many internet connected devices, we have started to run out unique IPv4 addresses from the 4.2 billion addresses that are available ($2^{32} = 4.2$ billion). IPv6 has been introduced to solve this problem by providing many more unique addresses ($2^{128} = 3.4 \times 10^{38}$).
- IPv6 addresses are 128-bit numbers, presented as eight groups of four hex digits (e.g. 2001:0db8:85a3:0000:0000:8a2e:0370:7334)

Video: Comparing IPv4 and IPv6

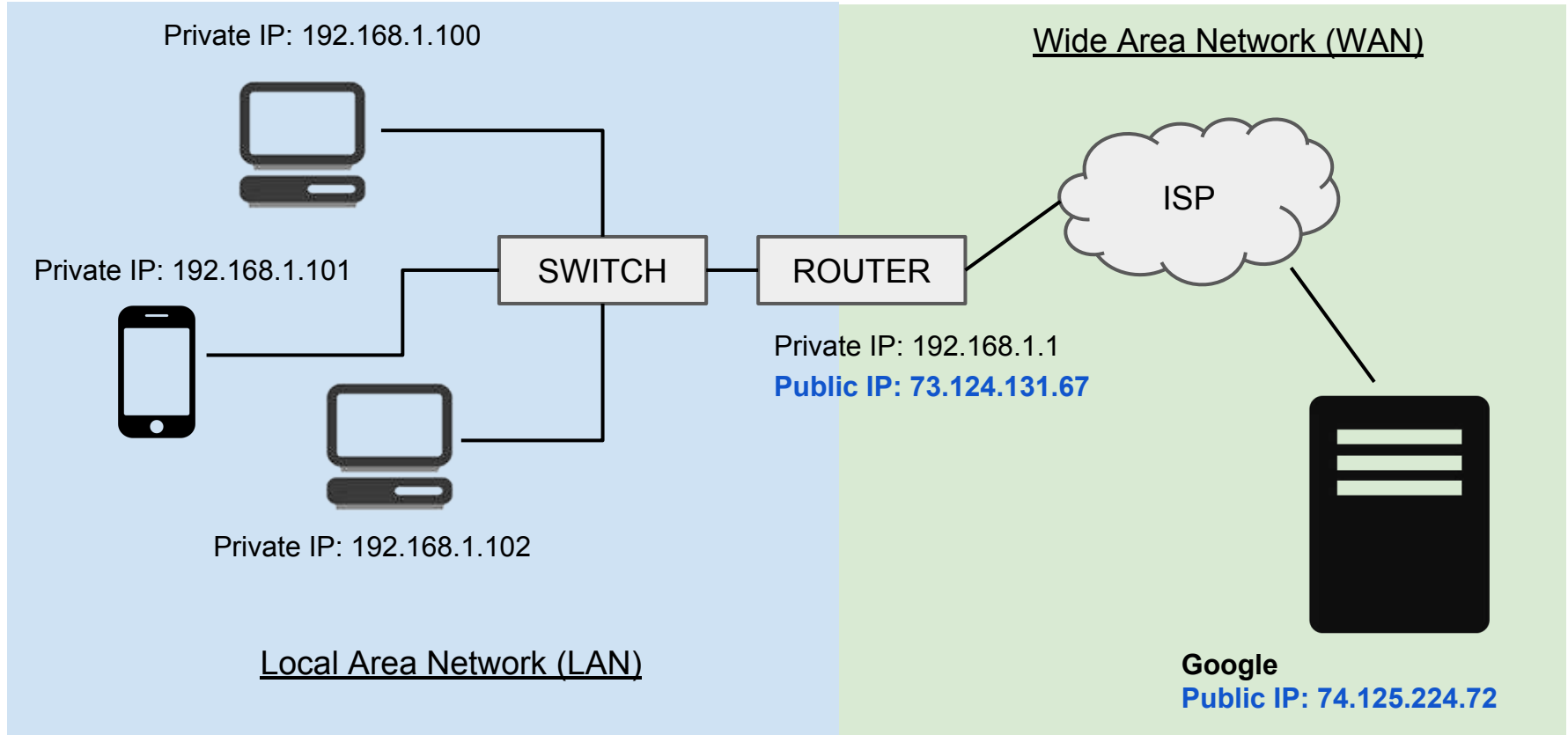


Video by [Techquickie](#)

Private and Public IP addresses

- Devices within a LAN have **private IP addresses** - these can be used to exchange data between devices on the LAN.
- The router that connects a LAN to a WAN (including the Internet) has a **public IP address** - this is used to exchange data with devices outside the LAN.
- When data (e.g. a requested web page) is sent back to the client device, it is sent to the router's public IP and forwarded within the LAN to the client's private IP address.
- Big companies such as Apple and Google **static public IP addresses** so that their services can always be reached, even if there was a failure of DNS, however these are **extremely expensive** due to the limited number of IP addresses that are available.

Private and Public IP addresses



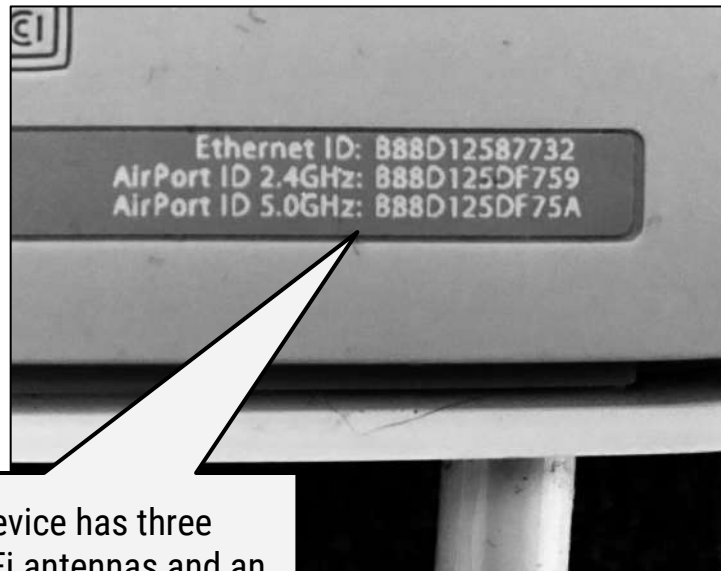
MAC addresses - purpose and format

- Every **network interface** on a device has a **unique MAC (Media Access Control) address**.
- They are **assigned** when devices are **manufactured** and **cannot be changed** once they are assigned.
- They are used by **network switches** and **network interface cards** to send data to their intended recipient.
- MAC addresses can be used to exchange data with devices on the **same local area network** (LAN).
- A device with **multiple interfaces** (e.g. wired ethernet port, wireless and bluetooth) will have **multiple MAC addresses**.

A MAC address for each network interface

MAC addresses uniquely identify **network interfaces**.

A device with multiple interfaces (e.g. wired ethernet port, wireless and bluetooth) will have multiple MAC addresses.



This network device has three interfaces (two WiFi antennas and an Ethernet port), so it can be sent data via three MAC addresses.

MAC address structure and format

- MAC addresses are traditionally **48-bit numbers** although there are 64-bit versions as well.
- MAC addresses are **represented as pairs of hexadecimal digits**, for example:

B8:8D:12:58:77:32

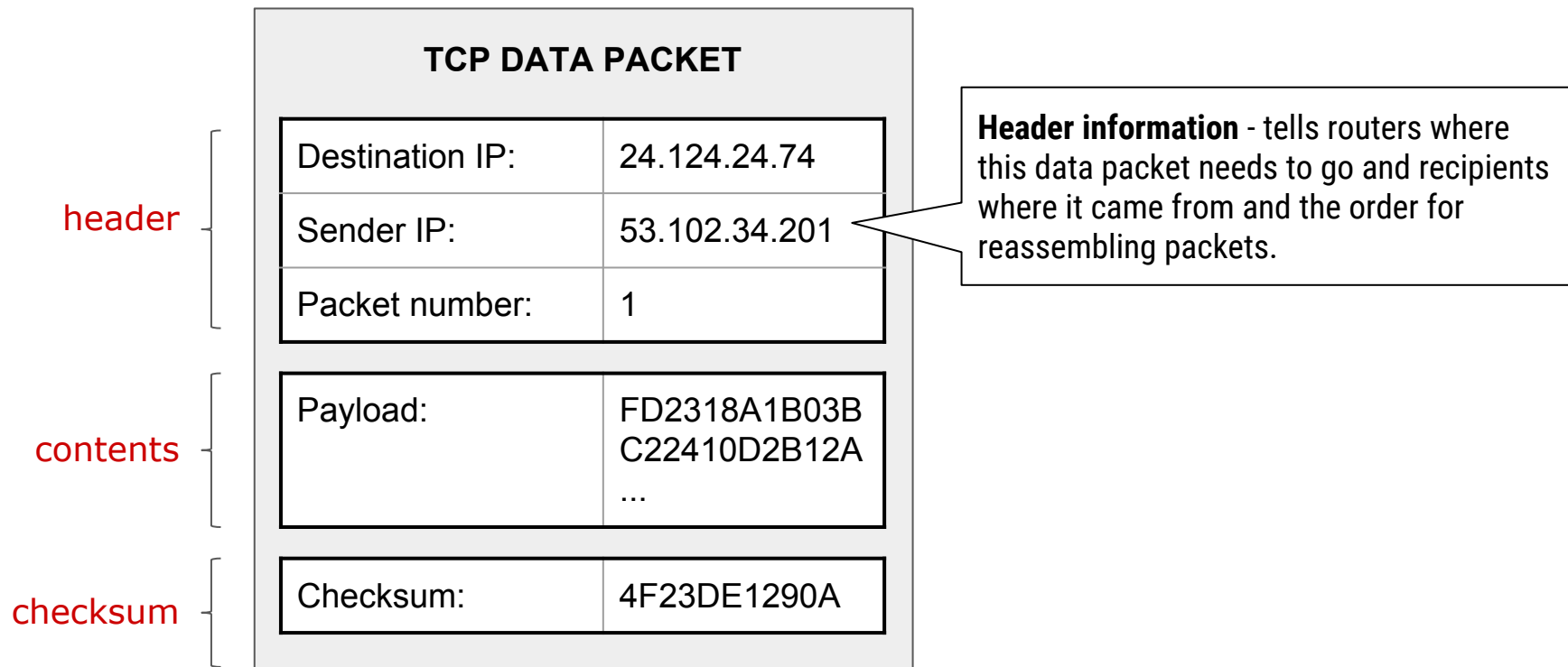
- The first 6 hex digits (24 bits) are unique to the manufacturer of the device, so it is possible to identify who manufactured a device from its MAC address.
- The remaining 6 hex digits uniquely identify the device or network interface from all those created by the same manufacturer.

Packet switching

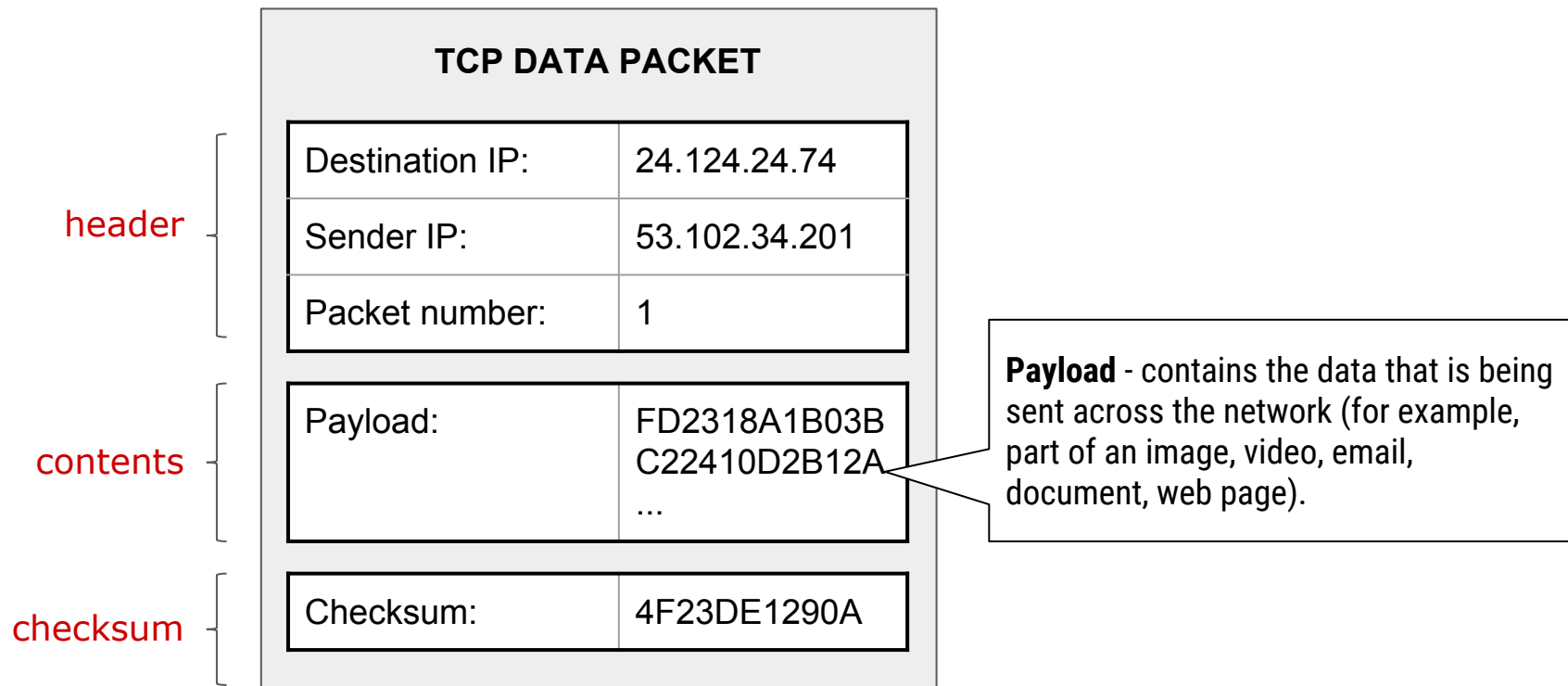
Packets

- When data is sent between networks (across a WAN, such as the Internet), it is broken down by TCP (Transmission Control Protocol) into smaller pieces called **packets** by sent by **routers**.
- A packet is made up of a **header** and a **payload**. The header contains important information necessary for sending the data to the correct place. The payload contains the actual data that is to be sent.
- Each packet is given a **sequence number** (stored in the header) so that multiple packets representing the same piece of data can be reassembled into the correct order by the receiving device, even if they arrive out of order.
- Routers use the **destination IP address** specified in the packet's header to know where to send it on to.

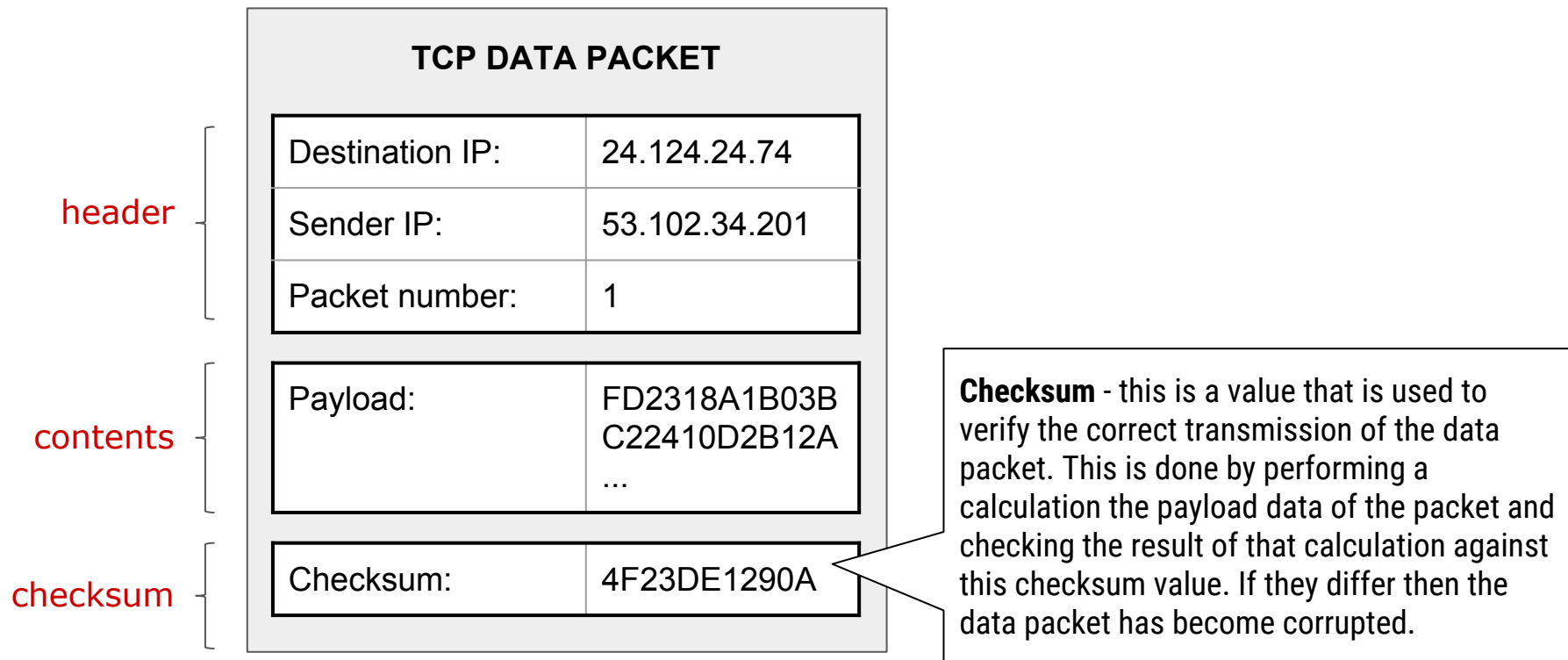
The structure of a TCP/IP data packet



The structure of a TCP/IP data packet



The structure of a TCP/IP data packet



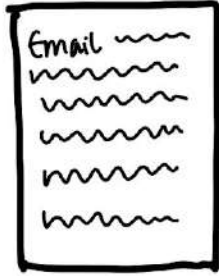
How are packets sent across the Internet?

- Data packets are sent from one device to another over the Internet via a process called **Packet Switching**.
- Packet Switching involves **breaking data into small packets** that are then **sent across a network of connected routers**, each taking a **different path** to reach its destination based on which path offers the **least congested route** at the time.
- If any path becomes unavailable, packets can simply take a different route making the system **very robust**.
- Because data is split into many smaller packets, each of which takes its own path to move quickly through the network, packet switching is a **very efficient** system for transmitting a lot of data.

The Packet Switching process

The following slides show the stages that data go through as they are broken down into packets and sent along the Internet to another computer via packet switching.

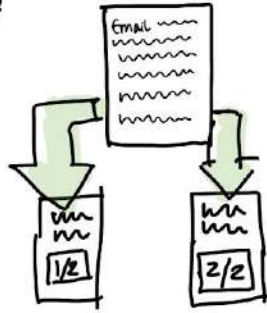
①



Data is generated by an application (for example, a email written in Microsoft Outlook).

This happens at the **application** layer in the TCP/IP layer model.

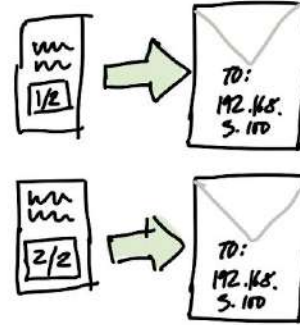
②



Data is split into packets by TCP and each one is given a sequence number.

This happens at the **transport** layer.

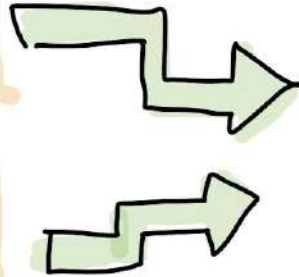
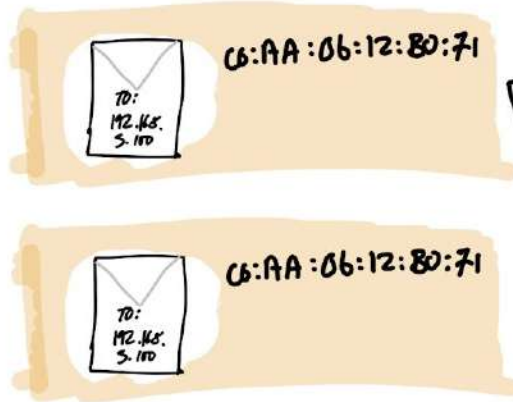
③



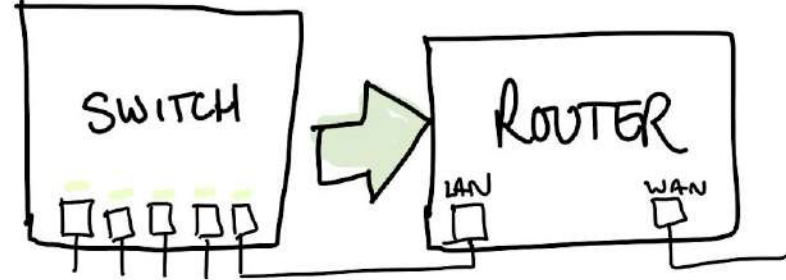
IP adds destination and sender **IP addresses** to each packet.

This happens at the **internet** layer.

④



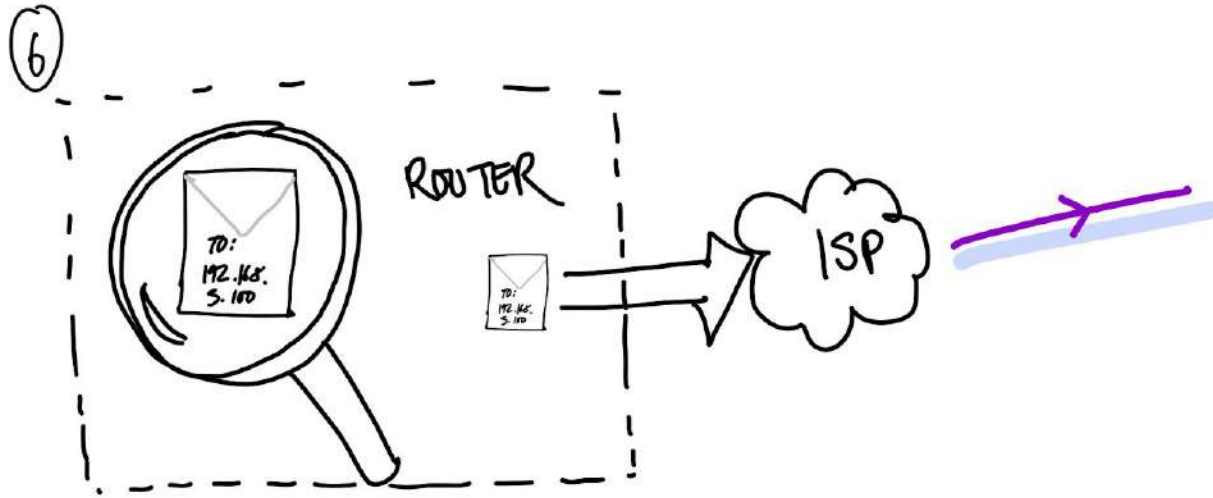
⑤



The IP addressed packets are **encapsulated** into an Ethernet **frame** that includes the **MAC address** of the router...

...each frame is then passed to the LAN's router by the switch.

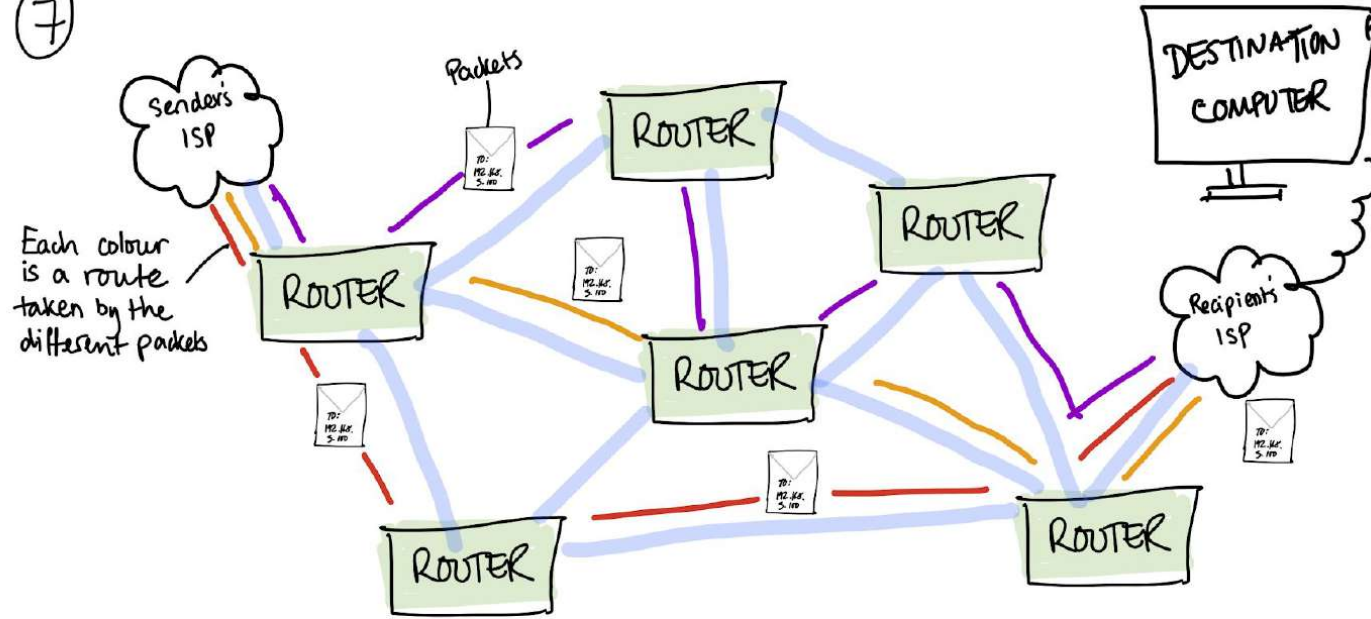
All this happens at the **network** layer.



The router **inspects** each packet to see if the destination IP address is on the LAN. If not then the packet is sent onto the Internet (WAN) via the sender's ISP.

This happens back at the **internet** layer.

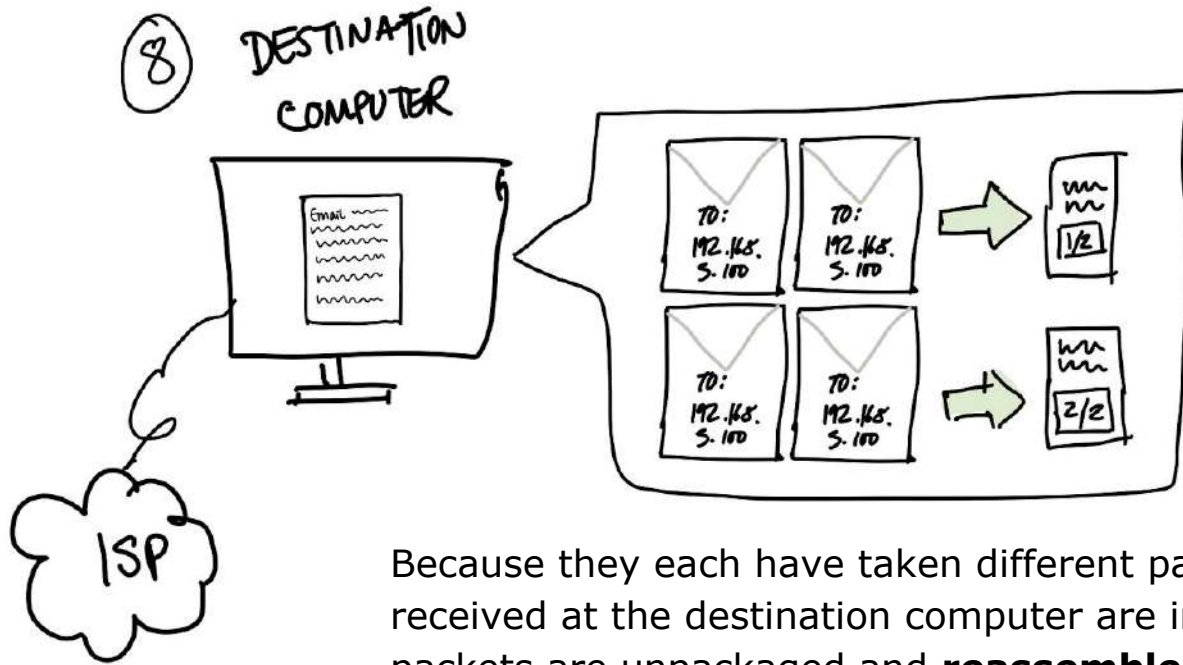
⑦



Each data packet is sent from the sender's ISP to the destination ISP via **connected routers**.

The route (or path) that each packet takes can be different, maximising efficient use of the network by avoiding bottlenecks or offline routers and utilising spare capacity.

This process is called **packet switching**.



Because they each have taken different paths, the packets that are received at the destination computer are in the wrong order. These packets are unpackaged and **reassembled** into the original binary data by TCP based on the **sequence number** of each packet.

If any packets are missing a message is sent back to the sender asking for the packets to be sent again. This makes packet switching a very **reliable** system.

Checking that data has been correctly received

- As data is being transmitted, **periodic checks** are made to see if all packets have been received by the recipient
- The recipient knows **how many** packets it should be expecting as each one states how many packets it belongs to (e.g. 3 / 16)
- If a packet hasn't been received within a certain time, a **timeout signal** is sent to the sender and the packet is **re-sent**
- When all packets have been received, their **checksum** is validated to ensure that they haven't been **corrupted** along their journey
- If all packets have been correctly received, a **confirmation message** (receipt) is sent to the sender

Video: Packets, routing and reliability



Video by [Code.org](https://code.org)

Video: There and back - a Packet's tale



Video by [World Science Festival](#)

A person wearing a dark hoodie is shown from the chest up, holding a red flag with both hands. The flag has some white binary code (0s and 1s) on it. The background is a blurred green field with vertical columns of blue binary code (0s and 1s) falling like rain, reminiscent of the 'Matrix' effect. A semi-transparent dark horizontal bar is overlaid across the middle of the image, containing the title and chapter text.

SYSTEM SECURITY

CHAPTER 6

Forms of attack

Forms of attack

Attacks on networks come in different forms:

- **Insider attack** - When someone inside an organisation gives away access details to others or uses their access to steal sensitive information.
- **Passive attack** - when a hacker performs 'eavesdropping' on a network by '**sniffing**' the data packets that are being sent.
- **Social Engineering** - When a person is exploited (or tricked) into giving away critical information that gives others access to the network or accounts.
- **Active attack** - when someone uses **malware or other technical methods** to compromise a network's security and take control over its devices.

Active attacks

Active attacks involve **actively doing something** to **compromise** or **disable** a network or its security.

Ways of actively attacking a network include:

- Intercepting data with a **man-in-the-middle** attack
- Releasing **malware** onto the network
- Launching a **Denial of Service (DoS)** attack
- Cracking a password via a **brute-force** attack

Video: An overview of different cyber attack methods



Video by [CrashCourse](https://www.youtube.com/c/crashcourse)

Threats posed to networks

Exam technique tip

Must be able to link a threat with the methods for identifying or protecting against it

Don't just learn each in isolation, consider how multiple preventative methods could be used together to protect against a particular type of attack.

Malware

- Malware is any kind of **malicious software** that is installed without your **knowledge** or **intent**
- There are many different forms of malware, including:
 - Viruses
 - Worms
 - Trojan horses
 - Ransomware
 - Spyware and keyloggers
 - Rootkits
 - Back doors

Video: Malware - Viruses, worms and trojan horses



Video by [TheCuriousEngineer](#)

Forms of malware

Viruses

- Small pieces of program code injected into other executable programs ('host' programs).
- Virus code is only run when the host program is executed by the computer.
- Viruses spread when users copy infected files or send them via email.
- Often cause damage to host systems such as deleting files.

Forms of malware

Worms

- Unlike viruses, worms are malicious computer programs that can be executed without needing to be inserted into a host program.
- Worms are self-replicating pieces of software that spread across networks without the need to be passed on by users.
- Worms can cause similar damage to viruses, including deleting a computer's files.

Forms of malware

Trojan horses

- Any form of malware that users are **tricked** into installing because they think it will be useful. For example a "printer driver" or "browser plugin" that, when installed, actually performs malicious actions on the host device.

Ransomware

- **Encrypts** the files on an infected computer and only decrypts them once payment has been made to the owner of the ransomware.
- Sometimes such malware doesn't actually encrypt anything but it still scares users into thinking it has and handing over payment.

Forms of malware

Spyware

- Gathers information about a user or organisation by doing things such as **keylogging** to capture login and password details, screen capture, switching on cameras and microphones.
- Often comes in the form of a trojan horse.

Rootkits

- Malware that modifies the computer's **Operating System** in order to avoid detection by antivirus software that sits 'on top' of the OS.
- Can make changes to low-level settings of a computer to expose it to further attacks.

Forms of malware

Backdoors

- Malware that opens up an **access channel** to a computer that other malware can use to take over the machine.
- Think of it as one burglar getting into a house via a window and then going down to open the back door for the other one to get in
- DNS hijacking and other MITM attacks often makes use of this.

How does malware get onto computers?

- Often malware is **willingly installed by users** who are **tricked** into thinking they are installing some important security update or software driver.
- Opening **attachments** in emails such as Word and Excel documents that include '**macros**' - small programs that you give permission to run on your computers or **executable programs** (ending in .exe).
- Once on a network, worms can **self-replicate** and spread to other computers.
- Once again, **people** are the weak link!

Example of a malware lure within Skype

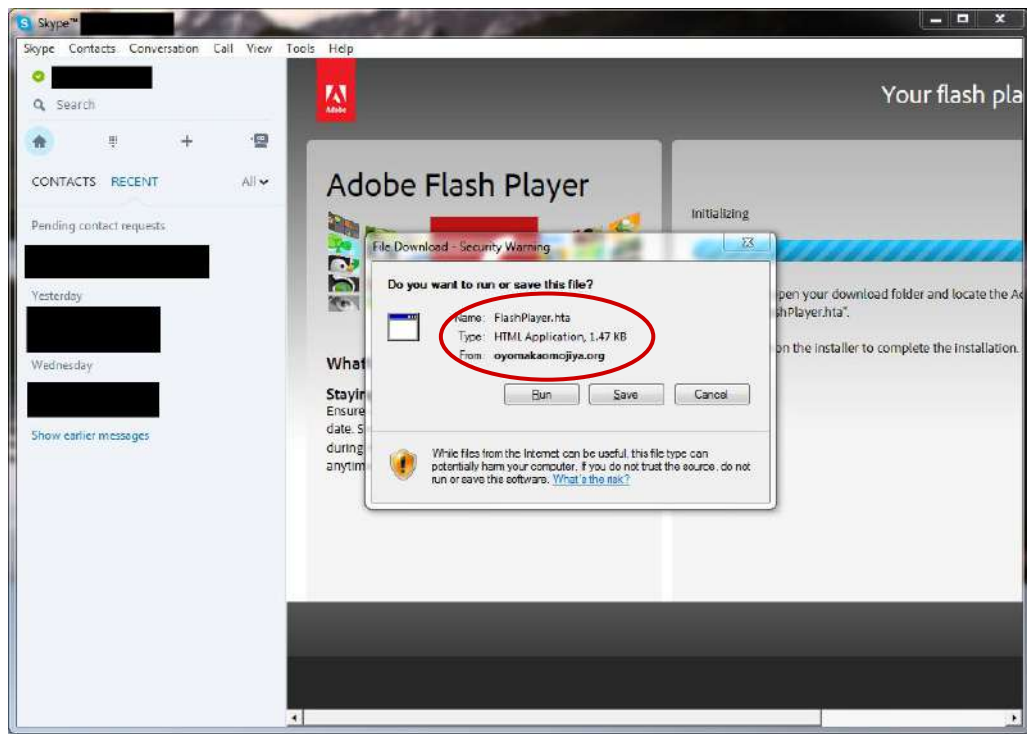


Image source <http://i.imgur.com/Q3F6yMW.png> (author unknown)

Protecting against malware

- Keeping web browsers and operating systems **up-to-date**.
- Installing **anti-malware software** such as antivirus.
- Implementing **user access levels** that prevent standard network users from being able to install software so they can't accidentally install malware.
- Using a **firewall** to restrict incoming and outgoing network traffic - this will prevent malware from being able to send sensitive data back to its source.
- **Educating users** about the risk of opening attached files in emails from unknown persons or clicking links on websites that are designed to scare them into action.

Video: Malware explained



Video by [Computerphile](#)

Social Engineering

- Often, no matter how much is spent on sophisticated technical protection such as firewalls, it is **people** who are the **weak point** in network security.
- Social engineering is a form of attack that involves **tricking people** into giving away critical information or access details.
- Social engineering could involve **cold-calling** someone, pretending to be from a bank or utility company and asking them to "confirm" their details.
- **Fear** is often used to put people off-guard and make them more likely to comply with instructions.

Video: Social Engineering in action



Warning: some use of bad language!

Video source: [oracle mind](https://www.oracle.com/mind)

Video: Staying safe on the phone



Video by [Action Fraud](#)

Protecting against social engineering

- The most effective means of protection against social engineering is **education and training** - people need to be made aware of the tactics of fraudsters in order to be on-guard against them.
- Company **security policies** should include instructions such as never proceeding to help a customer without them being able to confirm key security information such as their full address, date of birth or a PIN number.
- **Public awareness campaigns** and marketing by banks have been used in recent years to educate members of the public about the risks of social engineering.

Phishing

- Phishing is a common form of social engineering that utilises fake emails and websites to trick people into giving away their sensitive data.
- **Emails** are sent to thousands of people, claiming to be from a **known service** such as PayPal, Amazon or a bank.
- They always include an **action**, such as "click here to update your account" or a **link** to follow to the "real" website but which actually takes them to a **fake version** where their **login credentials** can be captured.
- Phishing emails often contain **clues** to show that they aren't legitimate - **educating users** about phishing is one of the main methods of preventing this form of attack.

Video: What is Phishing?



Video by [Safety in Canada](#)

Protection against Phishing

- **Look for clues** that an email might not come from a legitimate source such as **grammatical and spelling mistakes** or using a **generic greeting** such as "Dear customer" instead of the recipients full name.
- Check that the sender's **email address** and **domain** are legitimate.
- **Never** click a link in an email that asks you to update or enter your details.
- Ensure that links point to the legitimate website by hovering over them and looking for **https://** and the **proper domain name**.
- **Manually** go to a web browser to visit the website of the company that is apparently trying to contact you - if the request to update your details is real then they will have a prominent link to do so on their website.

Brute-force attacks

- A brute-force attack is where a hacker attempts to **crack a password** by trying thousands upon thousands of **different combinations** of letters and numbers until the right one is found.
- Special software is used to do this making it possible to make **millions of attempts in a second**.
- To speed things up, a brute-force attack often starts with a dictionary attack - a special type of brute-force attack that tries every known word in a language before attempting every possible combination of characters.



Image by
[OpenClipart-Vectors / 27440 images](#)

How long does it take to crack a password?

Password	Time to crack
password	< 1 second
qwertyuiop	< 1 second
Pa55word123	3 minutes
fish-and-chips	200 years
bad-badger/eats.dog	10,000,000+ years

*Times based on a brute-force tool being used on a standard PC
generated by <https://password.kaspersky.com/>*

Protecting against brute-force attacks

Networks can be protected against brute-force attacks by:

- Writing a **network policy** that requires **strong passwords** over a certain length and contain special characters. This will protect against dictionary attacks but won't ultimately prevent a brute-force attack from eventually succeeding.
- Using **two-factor authentication** so that even if a password is cracked, a second security "token" is required before access is granted.
- Restricting the number of **failed password attempts** before a user's account is disabled or "locked out" - either permanently or for a fixed period of time.

Denial-of-service (DoS) attacks

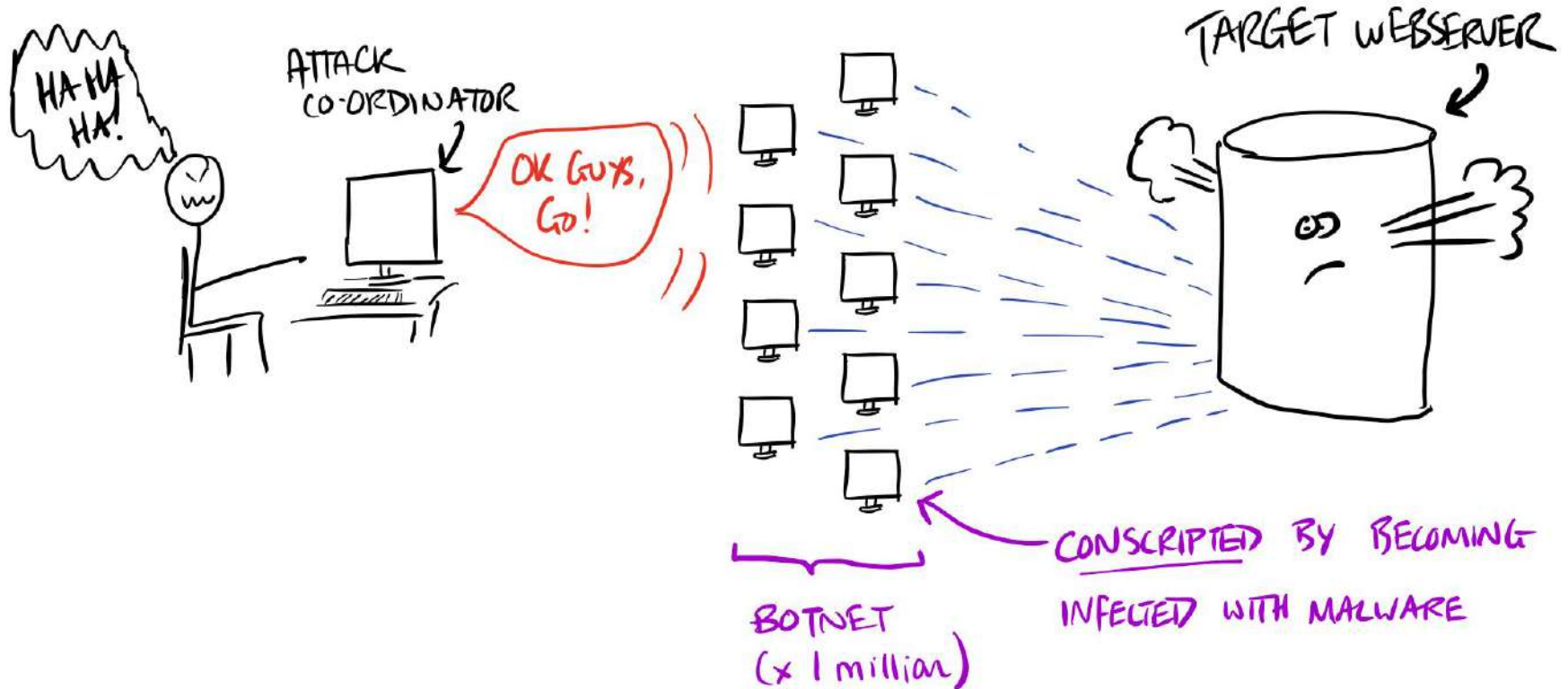
- DoS attacks seek to bring down servers by **flooding** them with many superfluous **requests** in an effort to **overload** them and prevent them from **responding** to legitimate requests.
- A **Distributed Denial of Service (DDoS)** attack uses an 'army' of compromised machines (a **botnet**) to launch a huge number of simultaneous requests upon the target machine.
- The people who own the conscripted botnet computers probably don't even realise they are taking part in this activity.

Video: DDoS Attacks Explained



Video by [Radware](#)

DISTRIBUTED DENIAL OF SERVICE



Protecting against Denial of Service (DoS) attacks

It is difficult to protect against a Denial of Service attack, however a range of methods can be used to minimise disruption:

- A **firewall** can be used to **inspect incoming** traffic and **reject packets** that originate from the same source or have identical contents
- A **firewall** could also be used to **restrict how many packets** can be accepted by a server with a particular timeframe to avoid being flooded with requests
- **Network monitoring and forensics** can be used to either identify a Denial of Service attack in real time or retrospectively analyse the attack to identify its source
- Whilst not a help to an organisation under attack, **anti-malware software** should be installed on computers to **avoid them becoming enlisted in a botnet** and being involved in future DDoS attacks.

Data interception and theft

- Data is a very valuable commodity. Personal data can be used to impersonate individuals to gain access to their assets or to act on their behalf. A company's financial data or trade secrets can be exploited by outsiders to gain a competitive advantage.
- Data theft is a serious issue for any organisation, not least because the loss of customers' personal information can result in **significant fines**.
- Data interception and theft can occur through a number of different methods, some passive and others active. These methods include: **insider attacks**, **Man-in-the-Middle (MITM) attacks** and **packet sniffing**.
- Data interception and theft is best prevented by **employing a range of preventative measures** including: **encrypting data**, implementing **user access levels**, requiring **strong passwords**, **preventing physical access** to computer systems and **educating users** about the risks of giving data away.

Insider threats and attacks

- An insider threat (or insider attack) is when someone with **privileged access** to information or systems uses that access to **steal information** from inside a company and give it to others **outside** the company.
- Their motivation may be **ethical** (as in the case of Edward Snowden) or it might be **financial** (stealing secrets to sell or blackmailing their employer for monetary gain).
- The risk of being attacked by insider users can be reduced by implementing different **user access levels** on the network the number of users with access to the most sensitive data is kept to an absolute minimum. A **firewall** can be used to screen outgoing data and **USB Flash Drive access can be disabled** on an organisation's computers to prevent data being copied in bulk.

Video: Insider Threats

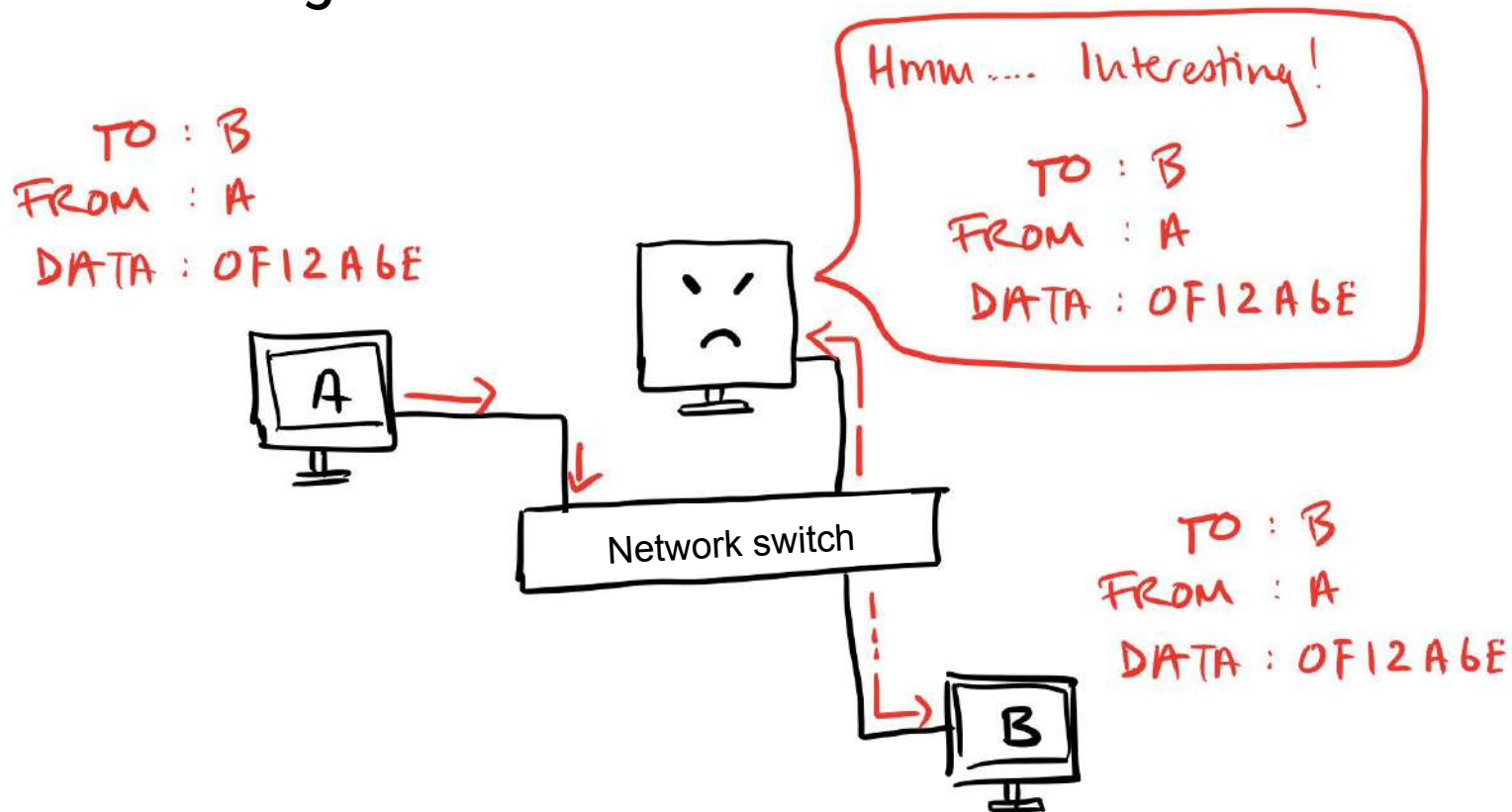


Video by [QinetiQ](#)

Packet sniffing

- Packet sniffing is a form of passive attack that **intercepts data** ("eavesdropping") as it is being transmitted across a network.
- The sniffing software reads and displays the contents of all the **data packets** being sent across the network.
- For this to work, **packet sniffing software** has to manipulate the information stored in the **network switch** so that it sends packets to the sniffing device rather than its intended destination. Once intercepted and read, the packets are then sent to their original destination by the sniffing device so that no one is aware that any interception has taken place.
- Packet sniffing software can also be run on **wireless devices** that can read packets sent across all wireless networks that are in range.
- Packet sniffing is a **passive form of attack** as nothing is actually being done to change or hinder any part of the network.

Packet sniffing



Protecting against packet sniffing

A range of methods can be employed to protect networks against packet sniffing

- Use enterprise-grade **managed switches** that include built-in protection against MAC address cloning and ARP table poisoning
- Use **MAC address authentication** so that only known devices can connect to the physical network
- Use **encryption** so that intercepted packets cannot be decoded - this is especially important for WiFi networks

Video: Packet sniffing on free public WiFi

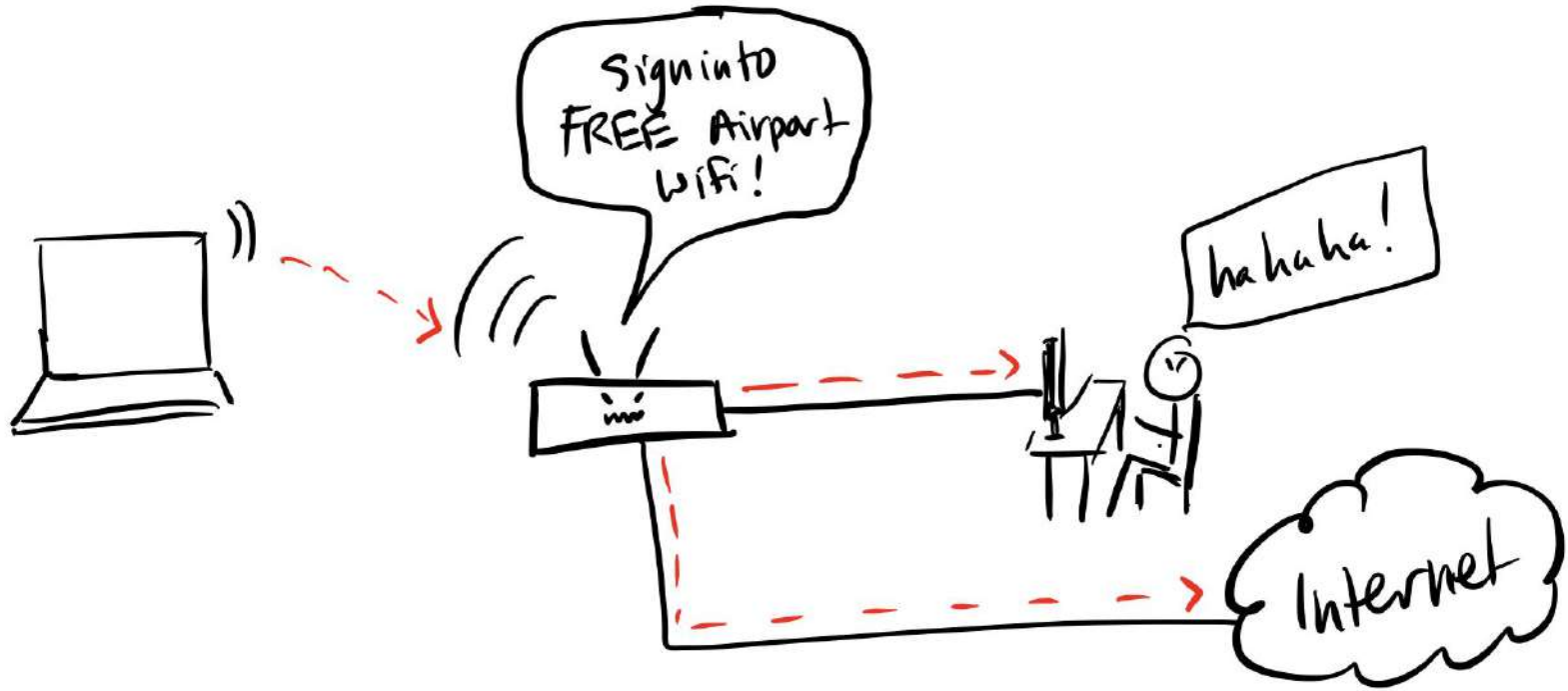


Video by [Android Authority](#)

Man-in-the-middle (MITM) attacks

- A man-in-the-middle attack involves **intercepting** a device's connection to the Internet.
- This is often achieved by luring users into signing into a **fake WiFi** hotspot.
- The operator of the fake WiFi network can then **sniff** all of the **packets** being sent by that user, giving them such information as:
 - What **websites** they are looking at
 - **Personal information** that they are transmitting, such as credit card details
 - Their **phone number**, which can be used to perform further attacks.

Man-in-the-middle (MITM) attacks



Video: Staging a man-in-the-middle attack

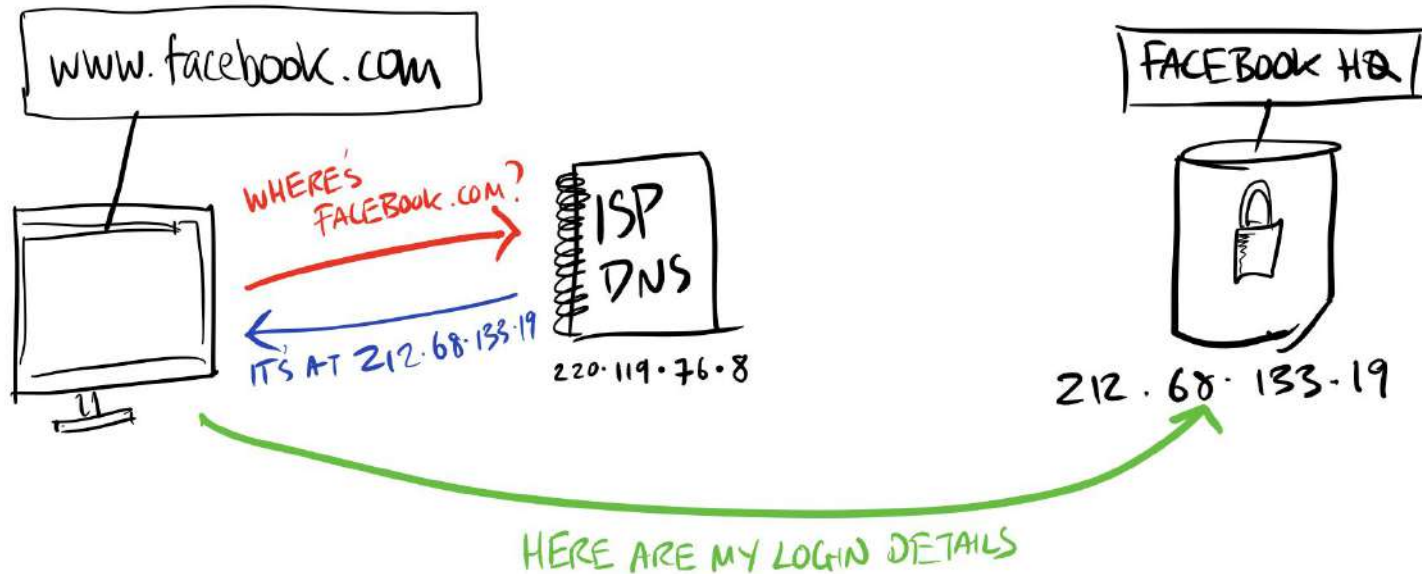


Video by [CBS This Morning](#)

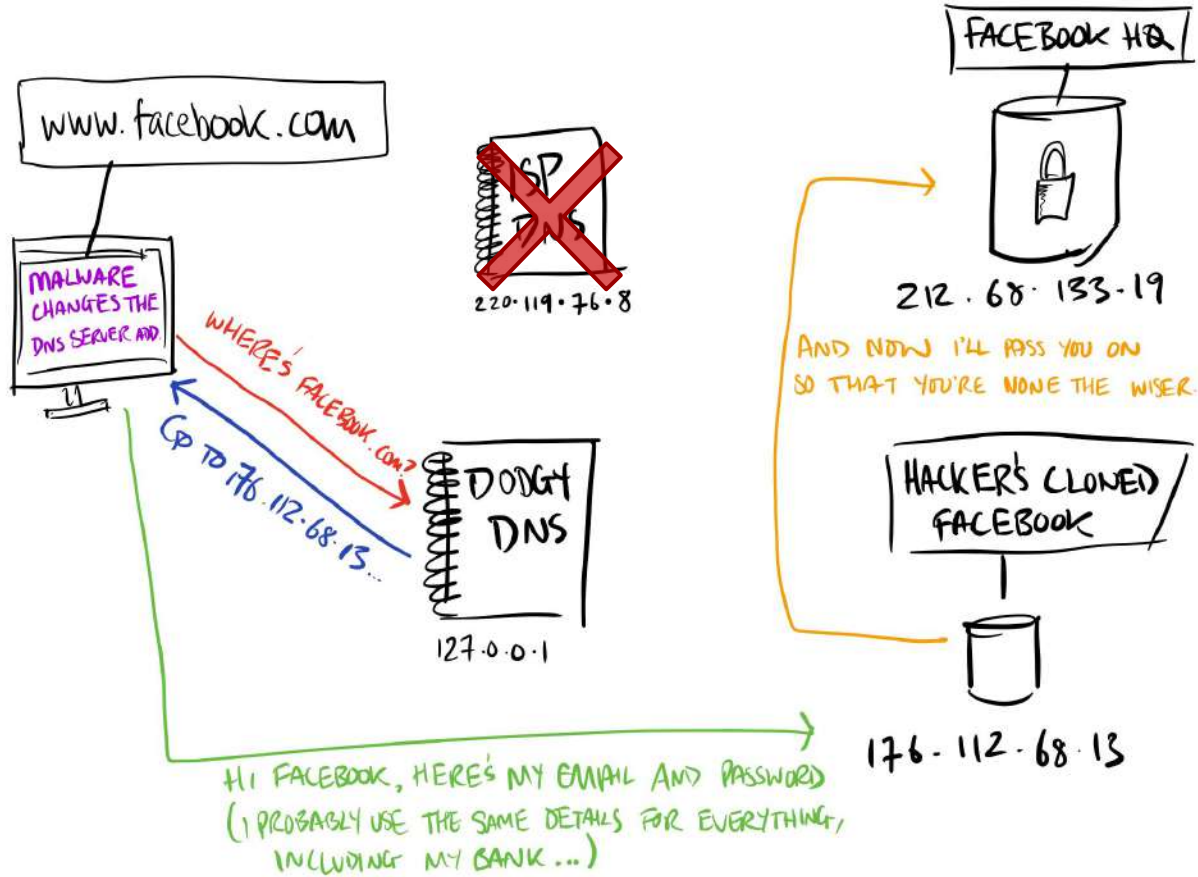
DNS spoofing (hijacking) - a MITM attack

- DNS spoofing is an example of a Man In The Middle (MITM) attack where a **user's device is compromised** so that it sends **DNS lookup requests** to a **rogue DNS server** rather than their ISP's genuine DNS server.
- When users navigate to a website, the rogue DNS server responds with the IP address for a server hosting a **fake copy** of the website.
- The user then enters their **login details** at the fake website, which are **captured** and used by hackers (possibly resold on the dark web).
- Often, the fake website then **redirects** the user onto the real version of the website and signs them in using the provided details so that users **aren't aware** that they have been attacked.

How it SHOULD work



WHEN IT GOES WRONG...



Protecting against MITM attacks

- Never join unencrypted, free public WiFi networks.
- Avoid sending sensitive data when connected to untrusted networks (including public WiFi and VPNs run by unknown organisations).
- Install anti-malware software on your computer (MITM attacks often involve using malware to change a computer's settings to send data to the wrong destination).
- Always ensure that websites are using secure (HTTPS) connections so that, even if data is intercepted, it is encrypted and cannot be read by unauthorised individuals.
- Do not proceed to visit a website if the computer or browser says that the connection has become compromised or is untrusted.

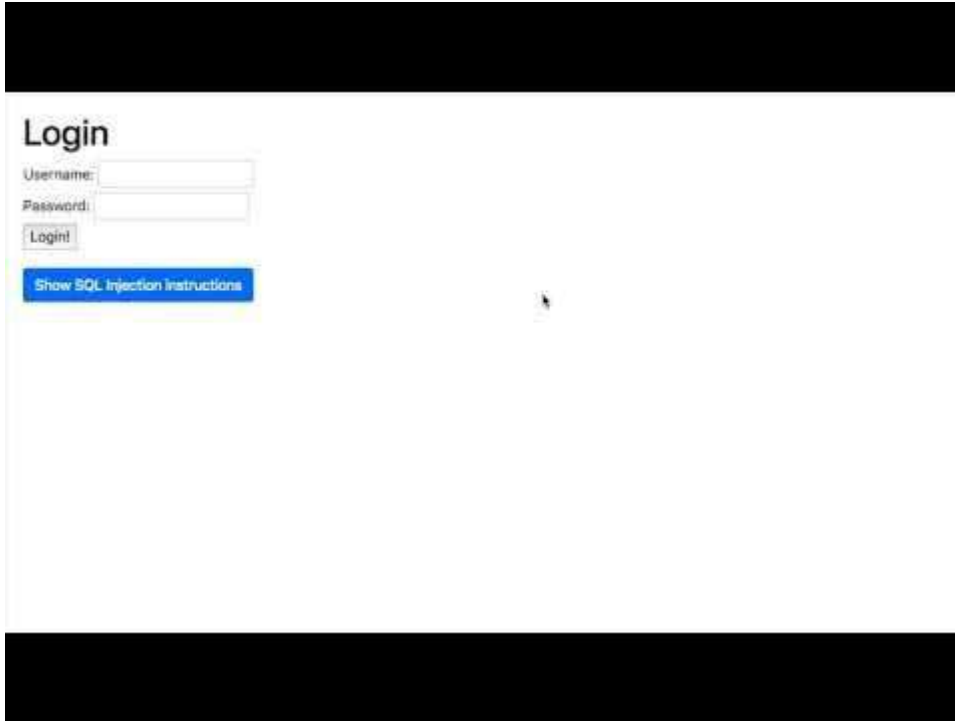
SQL injection

- SQL injection is a form of **active attack** that **exploits how SQL is processed** by databases on web servers in order to **circumvent the need to provide legitimate login credentials**.
- Instead of provide a username such as *admin*, in an SQL injection attack a hacker may type string that will result in a syntactically valid SQL expression that always yield results, such as:

```
' or 1=1 --
```

- These SQL commands are added to the SQL query that is **executed by the database engine** when it attempts to process the login details that it assumes it has been provided with.
- The malicious statement will always cause the database to return some results, making the web server "think" a user has been authenticated.

Video: SQL injection demo



The screenshot shows a web application interface. At the top, there is a large black rectangular redaction bar. Below it, the 'Login' section is visible, featuring a title 'Login', a 'Username:' label with an input field, a 'Password:' label with an input field, and a 'Login!' button. A blue button labeled 'Show SQL Injection Instructions' is positioned below the login fields. A yellow rectangular box highlights this button and the area immediately surrounding it. The rest of the page content is obscured by a large black rectangular redaction bar at the bottom.

Video by [Mr A W Dimmick](#)

SQL injections in more detail (1 of 2)

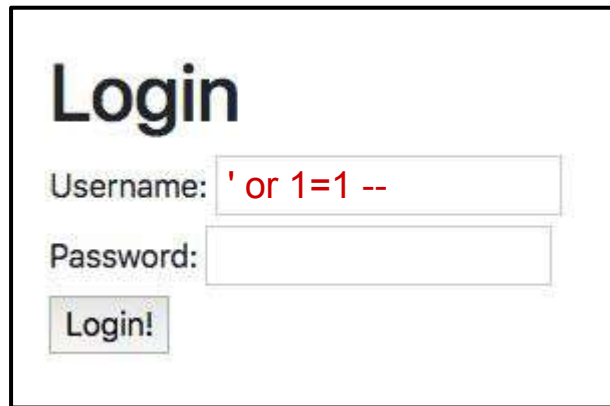
A standard user login SQL query searches for a record where the username and passwords fields match the values provided in the login form.

This query might look like:

```
SELECT * FROM user WHERE username =  
'{username}' AND password = '{password}'
```

If the string 'or 1=1 --' is provided as the username, this will be **inserted** into the SQL statement as follows:

```
SELECT * FROM user WHERE username=' ' or 1=1 -- AND  
password={password}
```



The image shows a login form with the title "Login". It contains two input fields: "Username:" and "Password:". The "Username:" field is filled with the text "' or 1=1 --" in red. Below the "Password:" field is a "Login!" button.

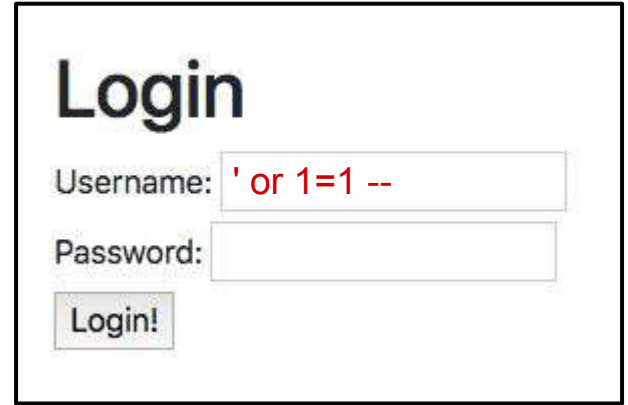
SQL injections in more detail (2 of 2)

```
SELECT * FROM user WHERE username='' or 1=1  
-- AND password={password}
```

The hijacked SQL query will **always** return a result because it now looks for a record where the username field is blank **or** 1=1 and 1=1 is always True, so the database always returns some data. In fact, it returns the details of **all** the users in the database.

The login algorithm simply looks at the first record it gets back and logs in with that user's details.

The `--` at the end of the "username" is how comments are prefixed in SQL, meaning that the rest of the query is simply ignored, so no attempt is made to match a password.



The image shows a login form with the title "Login". It contains two input fields: "Username:" and "Password:". The "Username:" field is filled with the text "' or 1=1 --". Below the "Password:" field is a "Login!" button.

Protecting against SQL injection attacks

These are a few methods that web developers can use to protect themselves from SQL injection:

1. **Input validation** - set password and username rules that don't permit characters such as ' and -- that can be used in an SQL injection attack
2. **Input sanitation** - inspect all user's input and remove special characters and SQL command words from their input before processing it
3. **User access levels** - set the database server to only allow certain users or systems (e.g. not the web server) to perform operations that can alter its contents

Identifying and preventing vulnerabilities

Penetration testing (pen testing)

- Penetration testing identifies **vulnerabilities** in a network's security by attempting a controlled (and permitted) **attack** on the network.
- This usually involves carrying out **several different types of attack** to see which are most successful.
- Good penetration testing will also test an organisation's **users** (staff / employees) to see how likely they are to fall for **social engineering** tricks.
- Penetration testing might also include testing the organisation's ability to **recover** compromised or lost data after an attack.
- By testing and improving network security, penetration testing can help organisations avoid large fines and public-relations damage that would result if their customers' personal data being stolen.

Video: What's a pen test?



Video by [SSL247](https://www.ssl247.co.uk)

Video: Penetration testing in a bit more detail

See an overview of what pen testing is, why you might want to do it and the overall processes involved.

A good video to watch if you are aiming for a Level 7/8/9 grade!



Video by [Rapid7](#)

Network policies

A network policy is a document written by a company that sets out the details about how their network is to be setup and maintained.

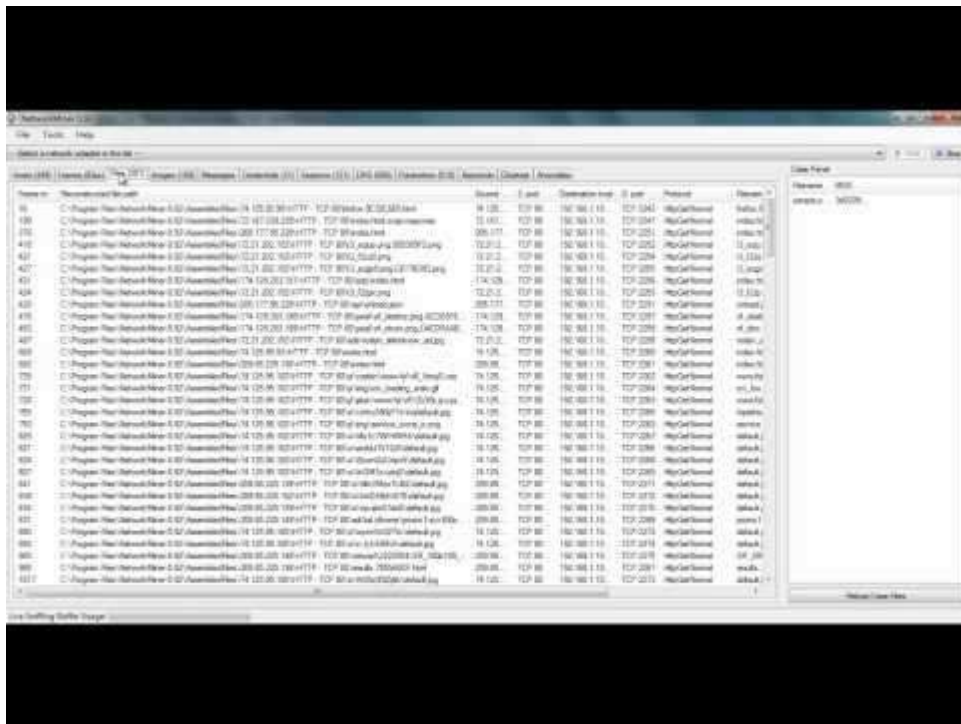
A good network policy will include details such as:

- How secure user passwords must be and how regularly they must be changed
- Which resources and services will be available to which groups of users by implementing user access levels and virtual networks
- What anti-malware software should be installed on the servers and client devices
- How often backups should be made and where they should be stored
- How often penetration testing will be performed to test the security of the system
- Which internal persons and external organisations should be contacted in case of a network attack
- What firewall system is to be used and what services/ports should be permitted
- Network diagrams that show where each of the switches is located within a building

Network forensics

- Network forensics involves **capturing**, **storing** and **analysing** network data to discover the **source** of security attacks.
- Software is used to **monitor** traffic and look for **unusual** activity on the network.
- Network traffic is usually transmitted and **lost**, making network forensics software essential for looking **back** over an event to identify what happened and where the attack originated.
- As well as identifying the source of an attack, analysing how the attack occurred can help an organisation make the **necessary changes** to prevent a similar attack in future.

Video: Conducting Network Forensics with Network Miner



This video is a little technical, but it does show you how network traffic that has already been captured can be retrospectively analysed and searched for potentially malicious traffic.

Video by [Anton Schieffer](#)

Antivirus and anti-malware software

Different forms of malware differ in how they operate. Therefore, antivirus and anti-malware utilities operate differently also.

- Antivirus software performs real-time '**on-access**' scans of files to detect whether they have been **infected** with virus.
- Anti-malware software performs **periodic** scans of your system, looking for **malicious applications** that it knows can run and harm your computer.
- Anti-malware software may also look for **signs** that malware has been at work, for example **backdoors** being left open.
- Both antivirus and anti-malware utilities identify targets and **quarantine** them. This prevents them from running and allows users to decide whether they would like to attempt to **clean** them or **destroy** them.

Antivirus and anti-malware software

- Most antivirus and anti-malware utilities rely upon up-to-date lists (**definitions**) of the latest viruses and malware to be looking out for and how to identify them by their code.
- More advanced anti-malware utilities can identify malware on the basis that it doesn't **behave** like ordinary software, even if it has no prior knowledge of it.

Video: Antivirus vs Anti-malware

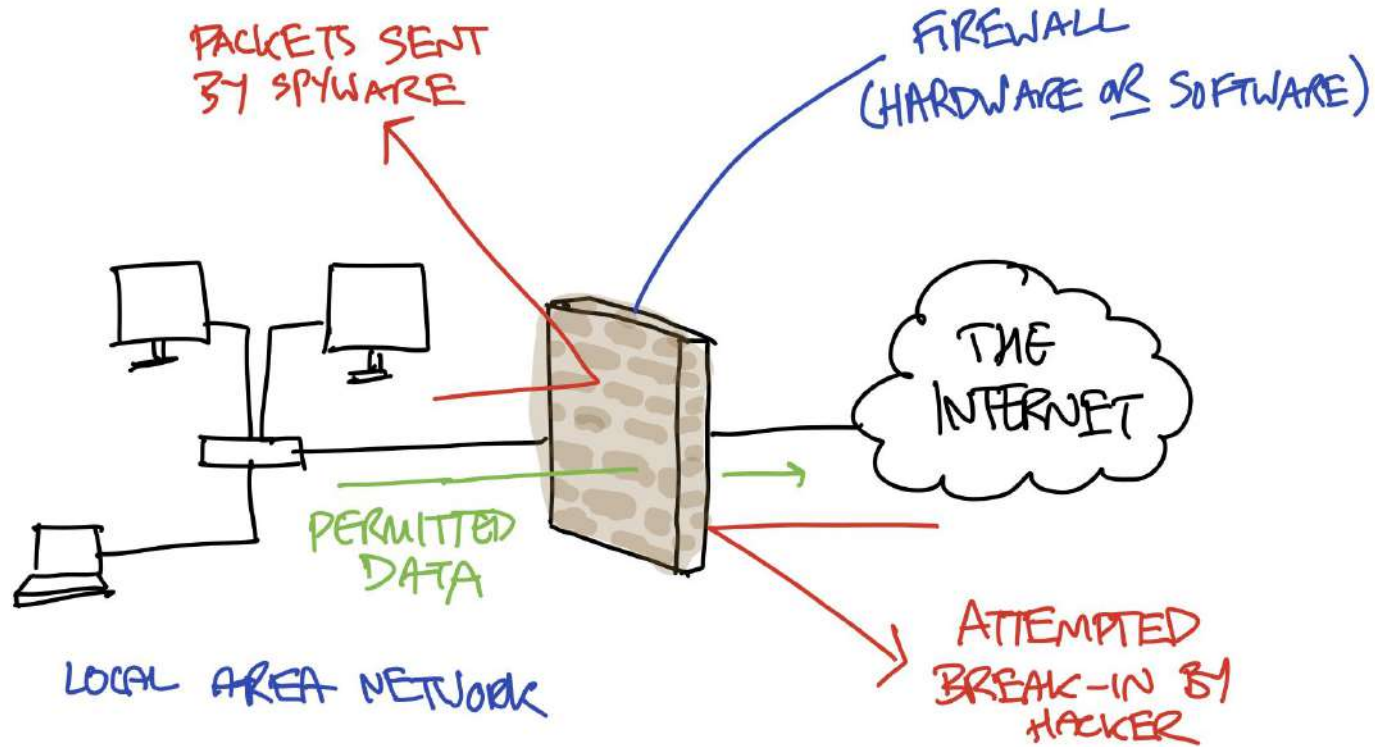


Video by [Techquicke](#)

Firewalls

- A firewall is either a **hardware device** or a **piece of software** that sits **between** a computer and the Internet.
- Firewalls **inspect** and **filter** incoming and outgoing data **packets**.
- If a packet fails to meet the filter **criteria** then it isn't allowed through the firewall.
- Firewalls protect computers and networks from **hackers'** attempts to break **into** a network from the outside.
- They also protect against attempts by **malware** to send data packets **out** of the network from infected machines.

Firewalls



Uses of Firewalls

- Firewalls are the main line of defence against Denial of Service (DoS) attacks and spyware (a form of malware).
- Different firewalls employ different techniques, including:
 - **Packet-filtering** firewalls - inspect each packet, dropping those that do not meet certain criteria
 - **IP-address filtering** - only permit traffic from certain (known) sources
 - **Port-blocking** - only permit traffic on certain network 'ports'.
Web-servers typically send data through 'Port 80'. Remote-access via SSH is typically on Port 22, so closing this on the firewall will prevent a lot of take-over attempts.

Video: Firewalls, as fast as possible



Video by [Techquickie](#)

User Access Levels

- Network users are typically arranged into **user groups**.
- Each group then has different **access rights** (or permissions) on the network, for example, in a school
 - Staff might be able to access **network folders** containing medical information about pupils but pupils cannot
 - Students might not be able to use **USB flash drives**
 - Students might be able to **write** to a folder to save/submit their work but not **read** from the folder, so that they can't see someone else's
 - Staff and Students might be prevented from **installing new software** applications.

Advantages of user access levels

Given that people are often the **weak point** in network security, implementing user access levels has clear advantages:

- Normal (non-administrator) users cannot give permission for **malware** to install, even if they are lured into clicking the 'install now' link
- Access to confidential information can be limited to those who need it - this helps protect against **insider attacks**.



However, there is a **risk** that an organisation might entrust all access rights to an **untrustworthy** network **administrator** who abuses their authority and blackmails the organisation!

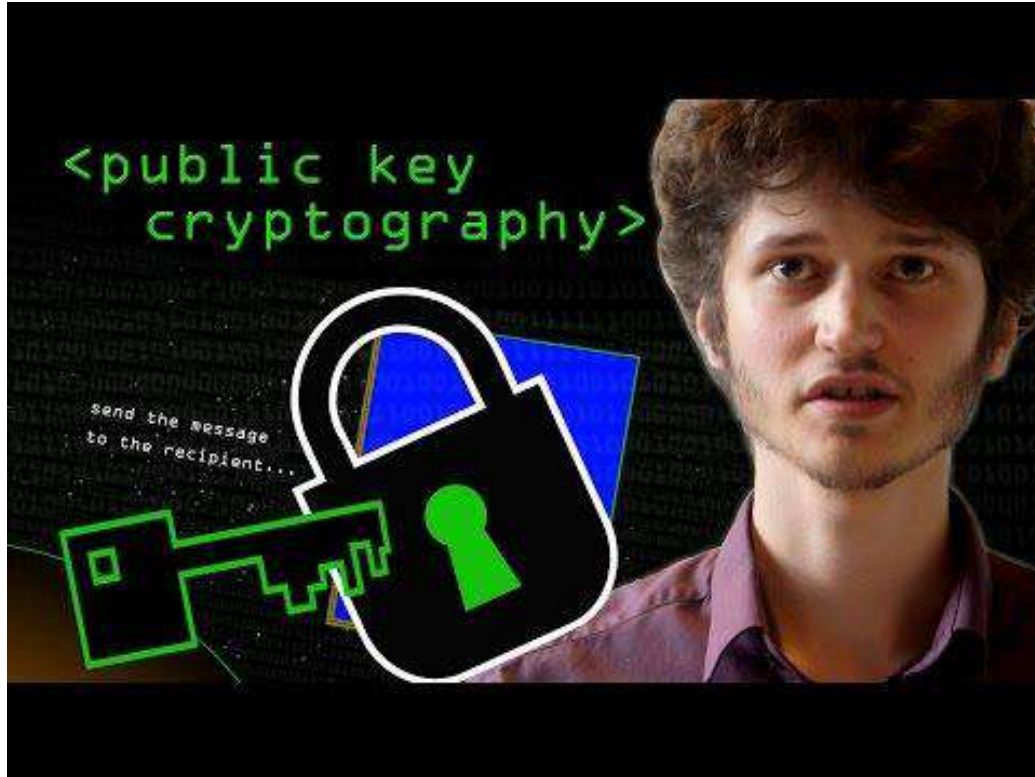
Passwords

- Passwords help prevent **unauthorised** users from accessing a network.
- However, passwords are only effective if (a) it remains **secret** and (b) it isn't easy to **crack** by brute-force attacks.
- To help keep them secure, passwords should be **long**, use a combination of **letters**, **numbers** and **symbols**, be **changed** regularly and **never written down**.
- Unfortunately, passwords are not a very good means of securing a system. They are hard to remember and easy to crack. Because of this, passwords are gradually being replaced with biometric authentication methods such as fingerprint readers and facial recognition.

Encryption

- Encrypting data means that it cannot be **read** by unauthorised persons, even if they manage to gain **access** to the data.
- The encrypted data can only be decrypted using the correct **key**.
- In practise, keys are made up of a pair of very large prime numbers, either 256- or 1024-bits long which would require an **impractical amount of time** for a modern-day computer to crack.
- As well as encrypting specific files, it is possible to encrypt all network **packets**, for example **WPA2** in WiFi networks and **VPN tunneling**.

Video: Public key cryptography (a form of encryption)



Video by [Computerphile](#)

The background is a dark, pixelated image of a computer screen. A yellow and black pixelated cursor is pointing at the word 'Settings' which is written in a light blue, pixelated font. Other visible elements include a wrench icon in the top left and the word 'Tools' in a similar pixelated font at the top center.

SYSTEM SOFTWARE

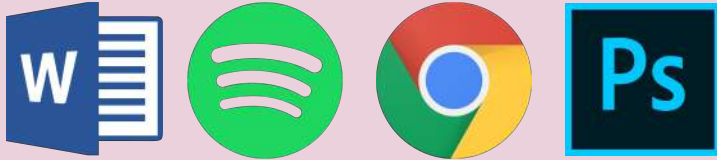
CHAPTER 7

What is system software?

Software can be split into two broad categories - application software and system software:

Application software

Programs that do specific tasks, such as write a letter (word processor) or edit a video.



System software

Controls the hardware in the computer and provides an environment (or platform) for applications to run.



Types of system software

Types of System Software include:

- **Operating Systems** (OS) manage the hardware in a computer and provide an environment for applications to run.
- **Device drivers** are small programs that tell an OS how to use a piece of hardware.
- **Utility software** are programs that perform maintenance tasks like recovering lost space on a secondary storage device.

Operating systems

What is an Operating System (OS)?

- A computer Operating System is an essential piece of software that is loaded when a computer **boots up**.
- Operating Systems are made up of many components that can **control the hardware** in the computer (e.g. graphics cards, memory, USB devices, network interfaces).
- Operating Systems also provide an **Application Programming Interface** (API). This is a set of code libraries that software developers can use to write applications that run on that OS. This is why Mac apps can't run on Windows, and so on.

Functions of Operating Systems

Operating Systems are responsible for many aspects of the running of a computer, each of which is controlled by a dedicated 'manager'.

Operating System (OS)

User Interface Manager

Provides the user interface that allows users to control the computer.

Device Manager

Allocates resources to external hardware devices and allows them to be used by applications.

Memory Manager

Controls the allocation of memory between applications.

User Manager

Authenticates and separates users of the computer.

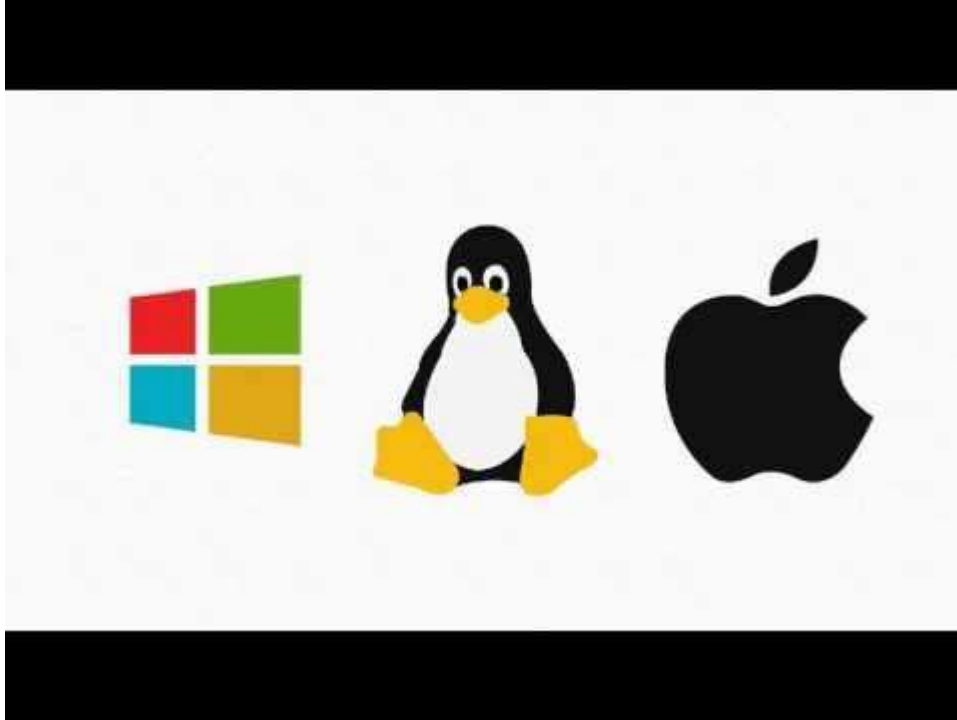
Process Manager

Controls the allocation of CPU cycles to multiple running applications.

File Manager

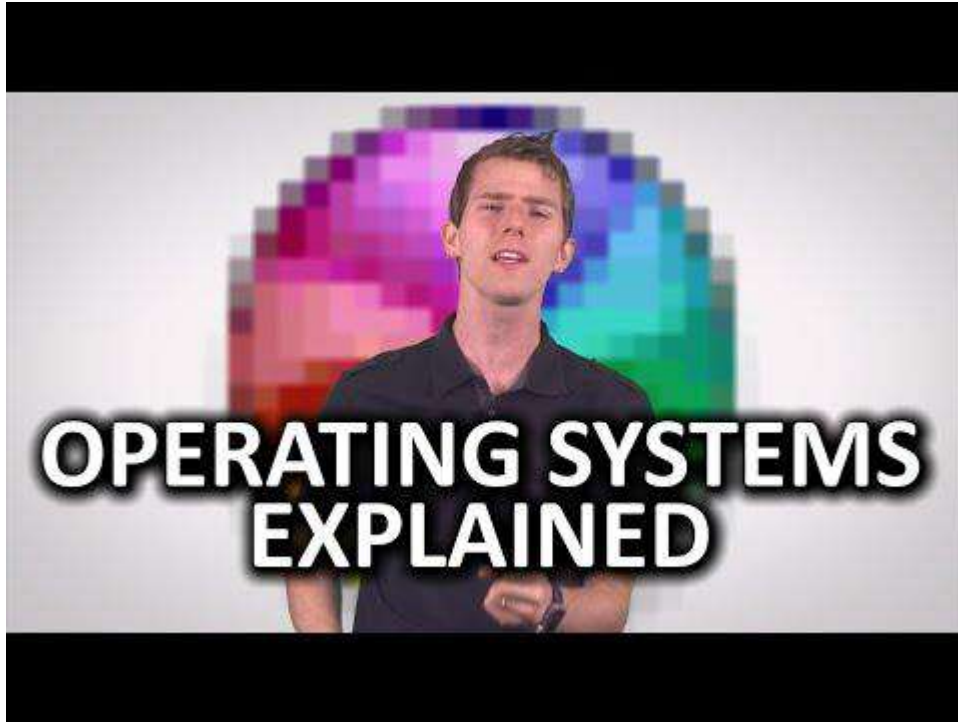
Controls the opening, reading and writing of files to storage and determines whether files are documents or executable programs.

Video: The main features of operating systems



Video by [OpenCanvas](#)

Video: Operating Systems as fast as possible



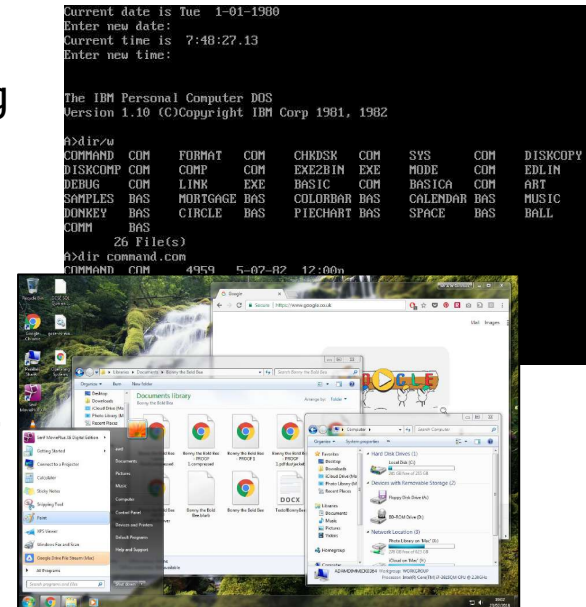
Video by [Techquickie](#)

User Interfaces

The User Interface allows users to **interact with the computer** by running programs, managing files and changing settings.

Early operating systems used **command-line interfaces** where a computer was controlled by typing **text commands** via a keyboard, a bit like using the Python IDLE but for *everything*!

Modern operating systems provide a **Graphical User Interface (GUI)** either based on the **Windows-Icon-Menu-Pointer (WIMP)** paradigm or a **multi-touch** interface.



DOS: An OS with a Command Line Interface

```
Current date is Tue 1-01-1980
Enter new date:
Current time is 7:48:27.13
Enter new time:

The IBM Personal Computer DOS
Version 1.10 (C)Copyright IBM Corp 1981, 1982

A>dir/w
COMMAND  COM      FORMAT  COM      CHKDSK  COM      SYS       COM      DISKCOPY COM
DISKCOMP COM      COMP      COM      EXE2BIN EXE      MODE      COM      EDLIN    COM
DEBUG    COM      LINK      EXE      BASIC   COM      BASICA    COM      ART       BAS
SAMPLES  BAS      MORTGAGE BAS      COLORBAR BAS      CALENDAR  BAS      MUSIC     BAS
DONKEY   BAS      CIRCLE   BAS      PIECHART BAS      SPACE     BAS      BALL      BAS
COMM     BAS

      26 File(s)
A>dir command.com
COMMAND  COM      4959    5-07-82  12:00p
      1 File(s)
A>
```



In DOS, everything was done by typing in commands!

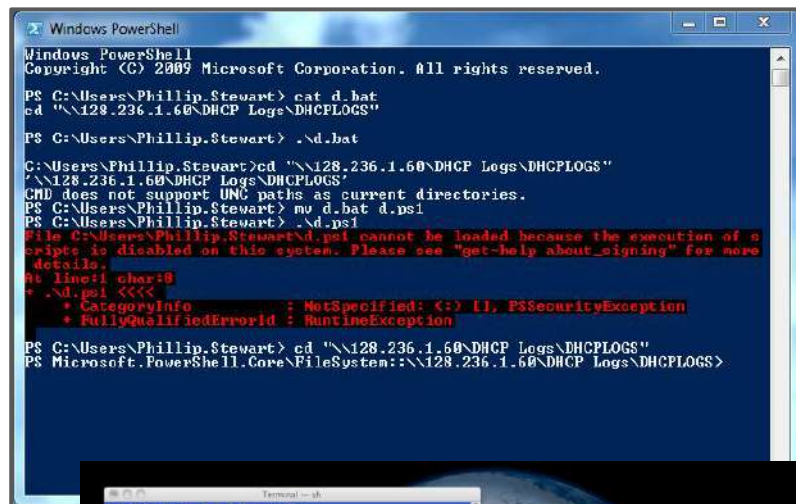
In order to use a computer, you had to remember every command to load programs, move files, delete files, print and so on.

You can still use a DOS-like CLI in Windows by running the **Command Prompt**. On Mac and Linux systems you can open the **Terminal** to control the computer via a CLI.

CLIs today

Command Line Interfaces are still available on modern operating systems for "**power users**" who want to do things with their computers that aren't available by the standard **graphical user interface (GUI)**.

CLIs **require less memory** and processing power than GUIs so are sometimes **used on low-power and embedded computers**.



Graphical User Interfaces (GUIs)

Almost all modern OSs provide a Graphical User Interface (GUI).

GUIs provide **visual representations** (abstractions) of the files and devices on a computer and allow you use these resources by clicking, dragging and now touching on their graphical representation (usually an icon).

Applications are shown in **windows** with features that are available via **menus**. Programs and documents are represented as **icons** and stored in **folders** or on a **desktop**. Interaction is typically by moving a **pointer** (cursor) that is controlled by a mouse.

GUIs have since evolved to allow control via touchscreen devices as well as using a mouse.

The Windows-Icon-Menu-Pointer (WIMP) GUI

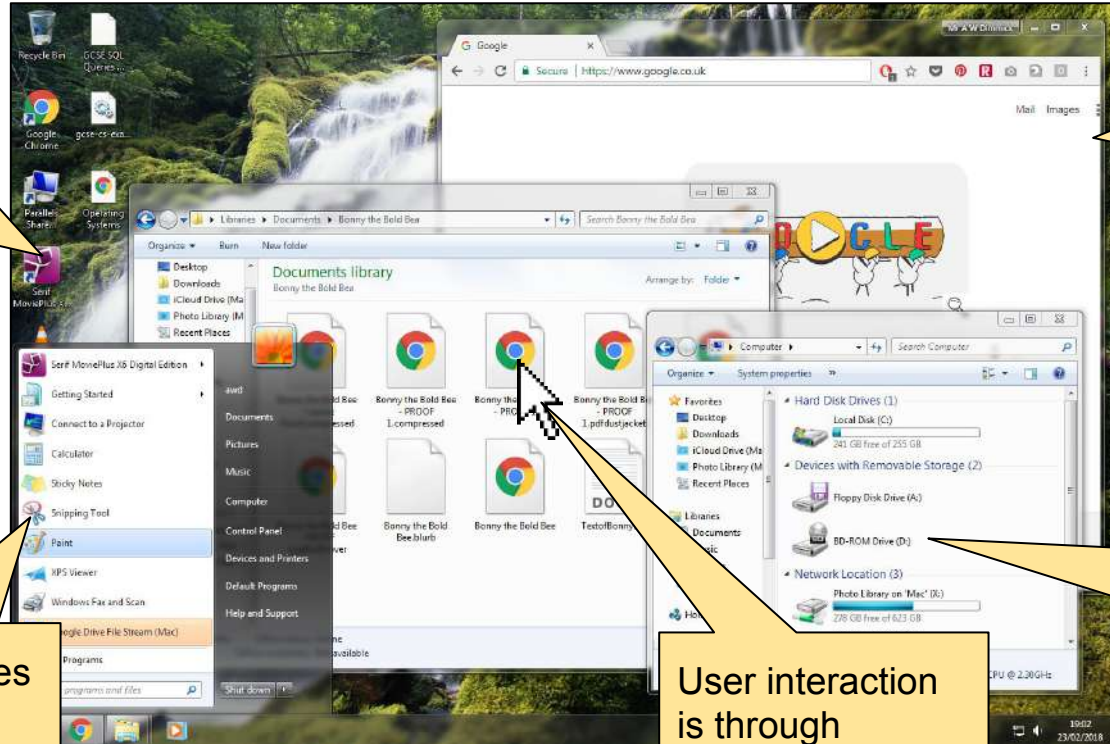
Programs and files (documents) are represented by clickable **icons**

Programs run in moveable, resizable **windows**

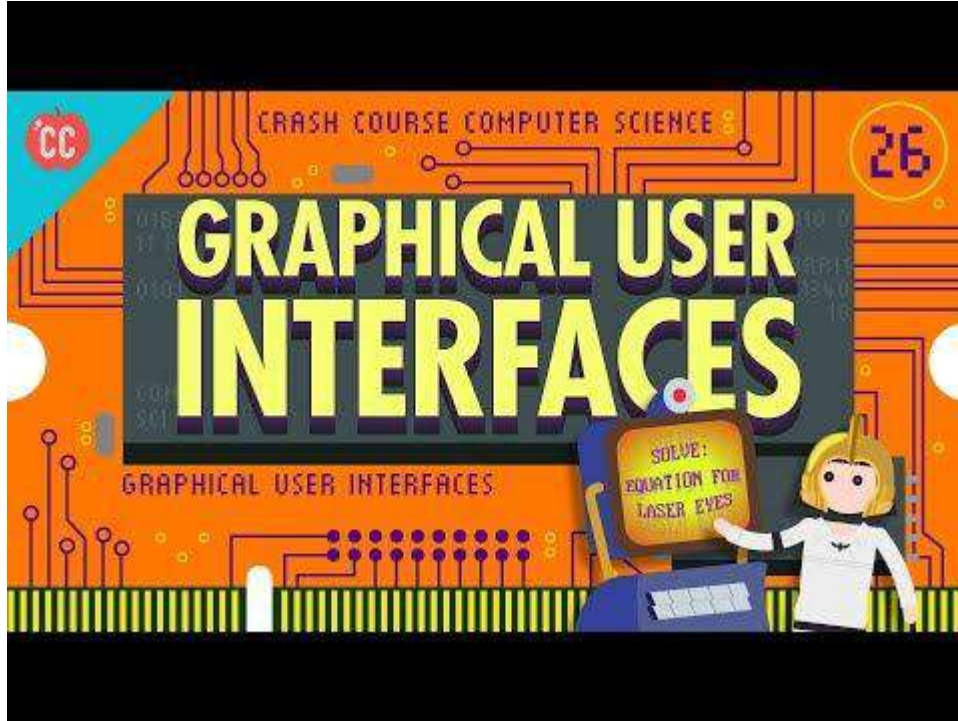
Processes, features and options are available via **menus**

Disk Drives (and other physical hardware) represented as **icons**

User interaction is through moving a **pointer**



Video: Crash Course CS: User Interfaces

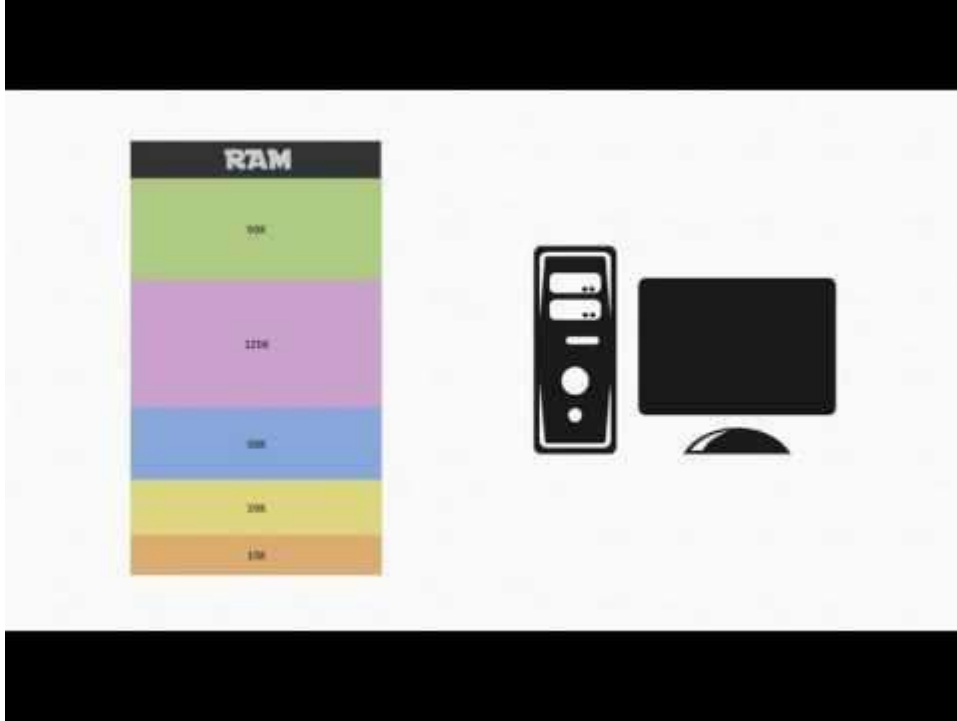


Video by [CrashCourse](https://www.youtube.com/c/crashcourse)

Memory Manager

- The Memory Manager component of the OS controls the allocation of the computer's **primary memory** (RAM) to each **process** that is running.
- The memory manager failing to manage memory well can result in a computer **slowing down** (when memory isn't freed up) or **crashing** (when processes access memory locations that contains data belonging to another process).

Operating Systems: The Memory Manager



Video by [OpenCanvas](#)

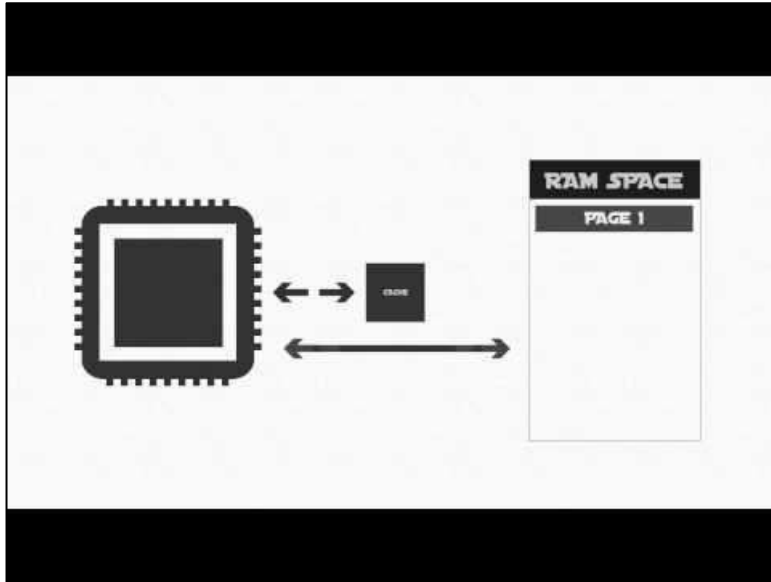
Multitasking

- Modern operating systems allow multiple applications to be run at once. This is called **multitasking**.
- In order for this to work, the OS must manage how memory (RAM) and CPU processing is allocated to each running process as, in reality, the CPU can only process one program's instructions and data at a time.
- The **memory manager** and **process manager** within the OS are responsible for this.

RAM (main memory)



Operating Systems: The Process Manager (Parts 1 and 2)



Video by [OpenCanvas](#)



Video by [OpenCanvas](#)

Peripheral (device) management

- Peripherals are **external devices** that are connected to a computer.
- A keyboard, mouse, printer, webcam are all examples of peripherals.
- Most peripherals are connected by **USB**.
- The Peripheral (Device) Manager keeps track of what devices are connected to which ports and allows applications to read data from and send data to each peripheral.

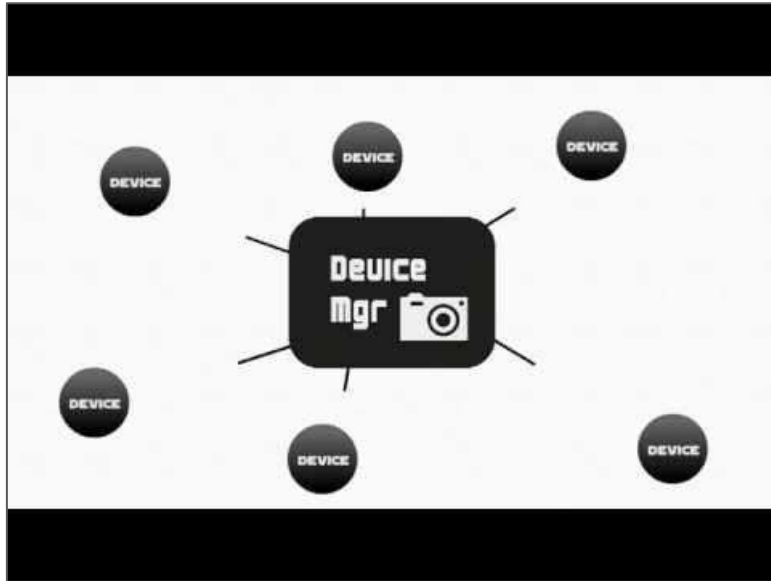


Device drivers

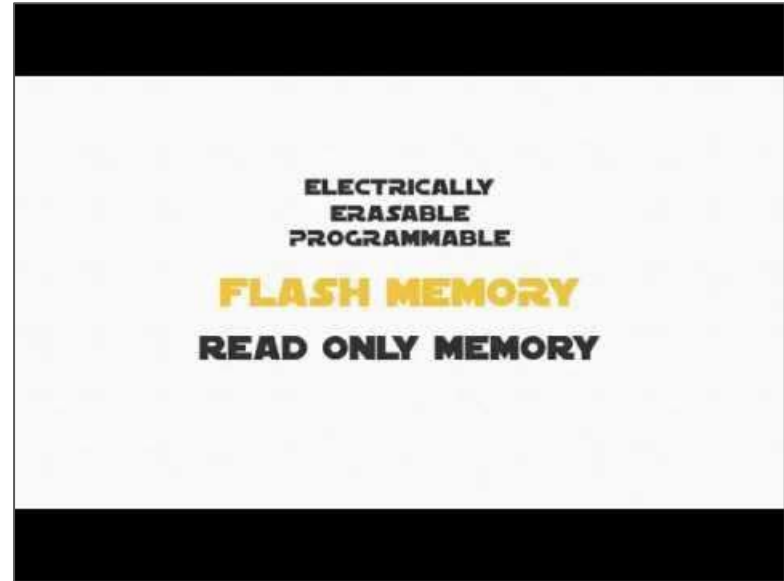
- Device drivers are small pieces of software that are loaded into the operating system to tell it how to communicate with (and make use) a particular peripheral device.
- Whenever a new printer is created, the manufacturer has to provide a device driver so that Windows or macOS can use it.



Operating Systems: The Device Manager (Parts 1 and 2)



Video by [OpenCanvas](#)



Video by [OpenCanvas](#)

Single and multi-user Operating Systems

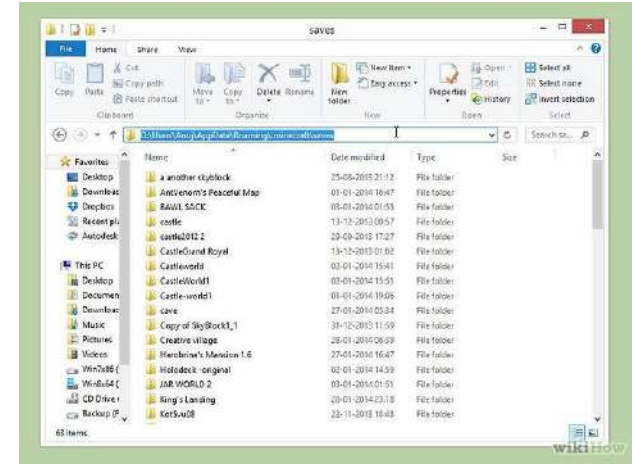
- Operating Systems differ in the number of simultaneous users that they support.
- An operating system that supports one user at a time (even if multiple users could potentially login) are called **single-user**.
- Operating Systems that allow several users to be connected (perhaps remotely) at once are called **multi-user**.
- MacOS and Windows are single-user OSs - even though multiple users can login, only one uses the system at a time. Mainframe and server OSs such as Unix and Linux are multi-user, allowing **multiple users to connect and request resources** at the same time.

User Manager

- The User Manager keeps track of **which users can access the system** and which are **currently logged in**. Access to the system is typically granted based on a **username** and **password** that matches those for known users stored in the OS.
- The User Manager is also responsible to providing **access** to different documents, applications and features (such as settings) to users based on their access rights.
- For example, a standard user should not be able to install applications or change a master password. This **security measure** is called **User Access Control** and is a responsibility of the User Manager.
- When a user logs in they expect to see their desktop, their documents and their apps - again, this is all managed by the User Manager.

File Manager

- Files are documents and executable programs that are stored on a computer's secondary storage.
- The File Manager is responsible for keeping track of **where specific files and folders are physically located in secondary storage**.
- The File Manager also determines the **type** of a file and either **executes** it (if it is a binary program file) or **sends it to an application** to be opened (if it is a document, such as a Photoshop file or a Word document).



Every file or folder is really just a collection of 0s and 1s represented on a secondary storage device. The File Manager allows users and applications to interact with these 0s and 1s as discrete files and folders.

File Manager

The File Manager also sets and manages the **user permissions** for files. Usually the permissions that a user or group of users can have for a file are:

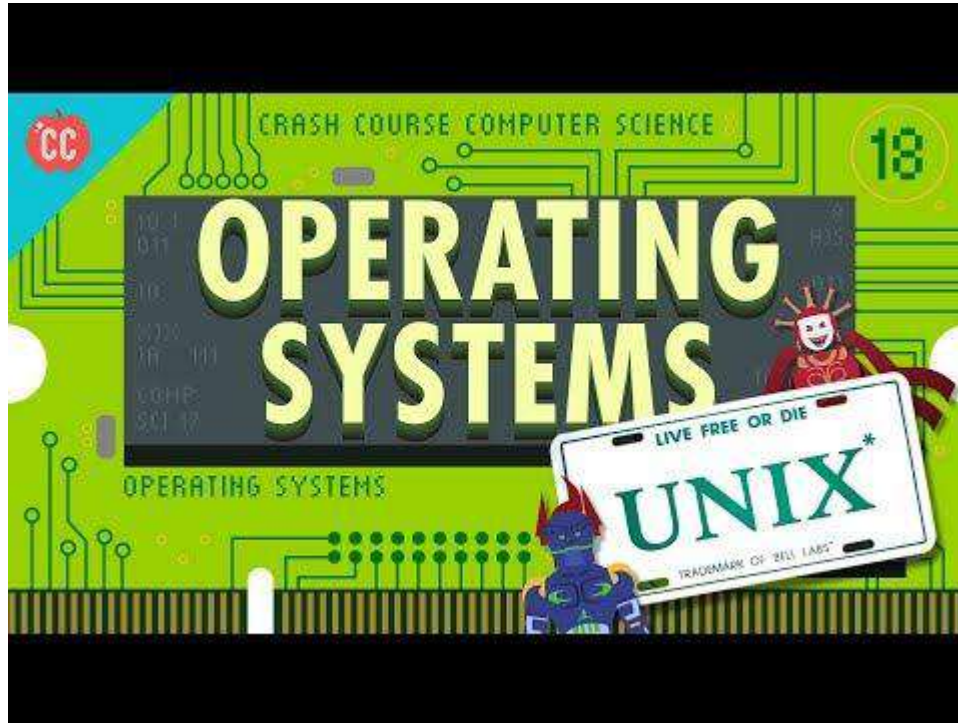
- **read** - whether a file can be read (opened, but not edited)
- **write** (or modify) - whether a file can be written (edited)
- **execute** - whether a binary program file can be run or not

Operating Systems: The File Manager



Video by [OpenCanvas](#)

Video: The roles and functions of Operating Systems



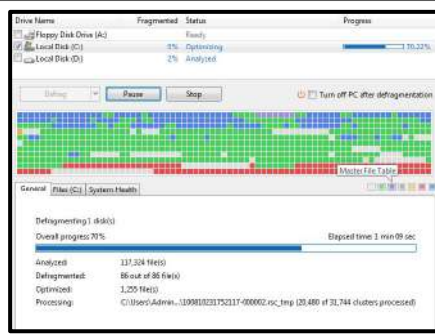
Video by [CrashCourse](https://www.youtube.com/c/crashcourse)

Utility system software

Utility software

- Utility software is a subset of system software (along with operating systems and device drivers).
- Utility software programs perform **specific tasks** to help **maintain** the system.

Types of utility software



Defragmentation

Improve hard disk performance by applying an algorithm to reorganise data



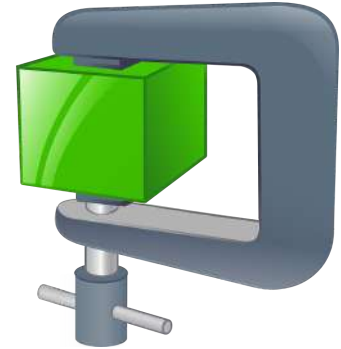
Encryption

Prevent unauthorised access to files by applying an algorithm to "scramble" data



Backup

Make copies of data that can be restored in the event of data loss



Compression

Apply an algorithm to reduce the space required to represent a file or its contents

Defragmentation utilities

Defragmentation utilities

- When something is broken up we can say it has been **fragmented** and each little piece is a **fragment**.
- When blocks of **data** that belong to the **same file** are split up and **stored in different places** on a hard disk we say that the data has become **fragmented**.
- A defragmentation (or "defrag") utility program **increases the performance of a computer** by **reorganising** the **data** stored on a **hard disk drive** so that data that belongs to the **same files** are stored in **adjacent blocks**.



This broken glass has become fragmented. Each piece of glass is a fragment.

Video: Defragmentation explained



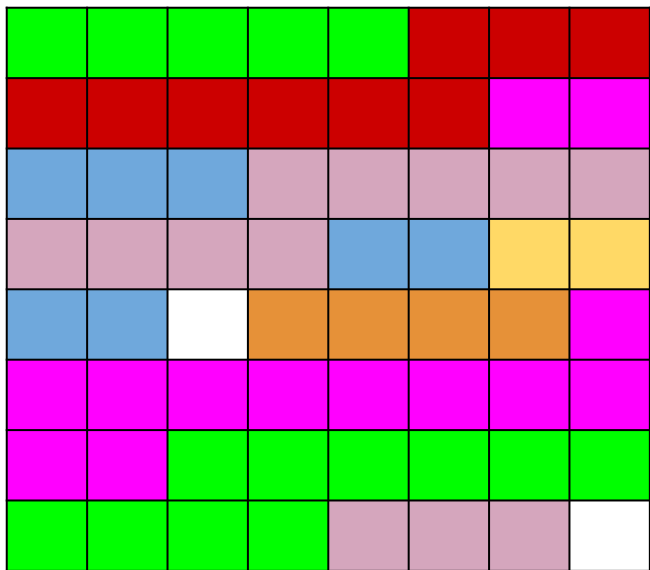
WATCH THIS! IT'LL SAVE YOU TIME!

This is a great overview that covers pretty much everything you need to know about data fragmentation and defragmentation in only a few minutes!

Video by [Techquickie](#)

What's the problem with fragmentation?

A model of fragmented data on a hard disk

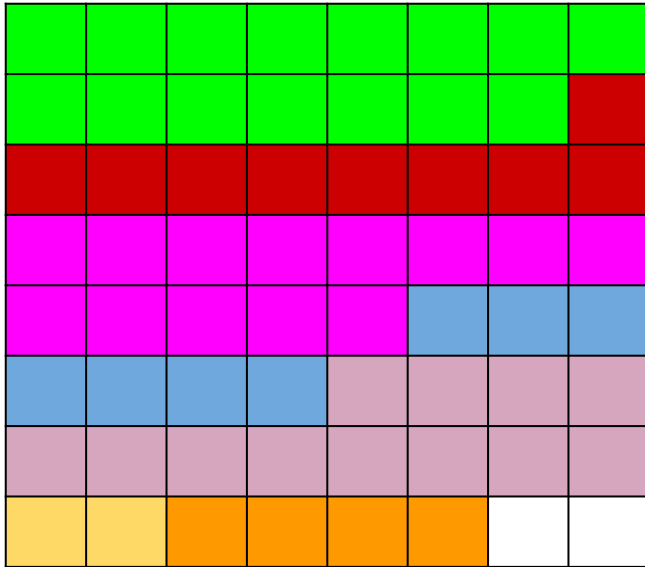


Each square represents a **block** on the hard disk. Coloured squares represent the data used for individual files.

- Recall that hard disk drives are **mechanical devices**
- The read/write head has to move to the correct **physical location** on a disk to read its 0s and 1s
- If a file's data is split across several locations on the hard disk, the read/write times for that file will **decrease**
- This can **dramatically affect the performance** of the computer

The solution: defragmentation utilities

A model of defragmented data on a hard disk

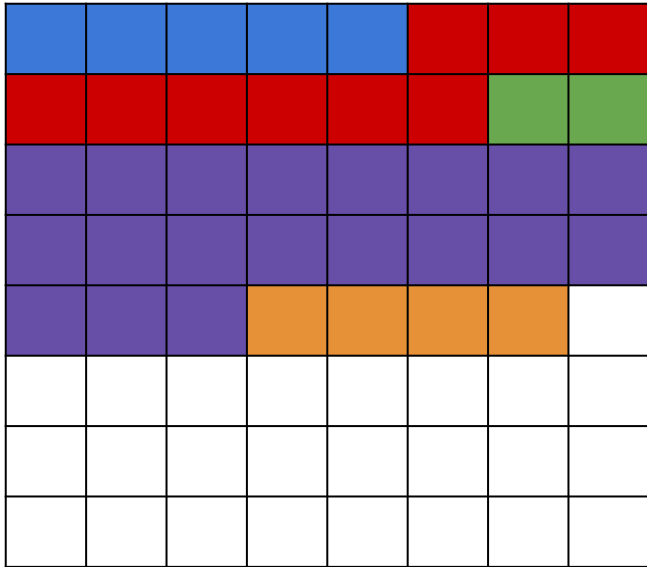


Each square represents a **block** on the hard disk. Coloured squares represent the data used for individual files.

- Defragmentation utilities **reorganise** the data blocks on a hard disk
- Data blocks that belong to the same file are **grouped together** in adjacent blocks
- **Free space is grouped together** at the end of the disk, which **reduces further fragmentation** when new files are written

How data becomes defragmented - 1

A model of the data on a hard disk

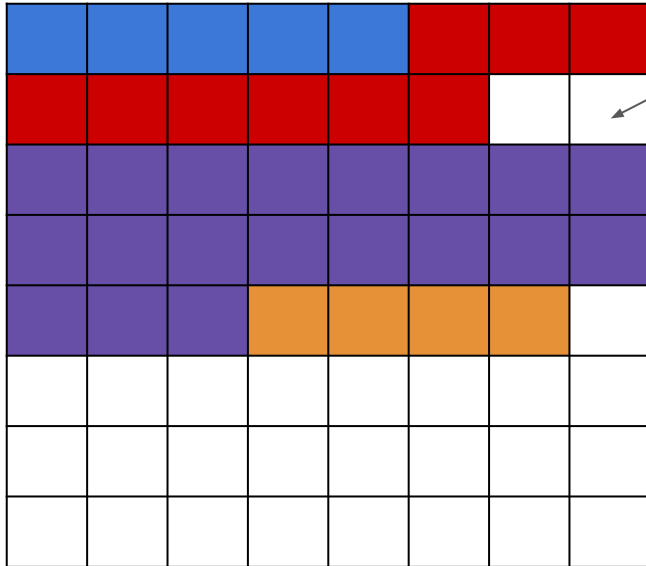


Each square represents a **block** on the hard disk. Coloured squares represent the data used for individual files.

- The diagram on the right represents an optimal organisation of data on a hard disk
- Each colour represents the data for a single file
- As you can see, each block for each file is organised so that they are together (adjacent) on the disk's surface

How data becomes defragmented - 2

A model of the data on a hard disk

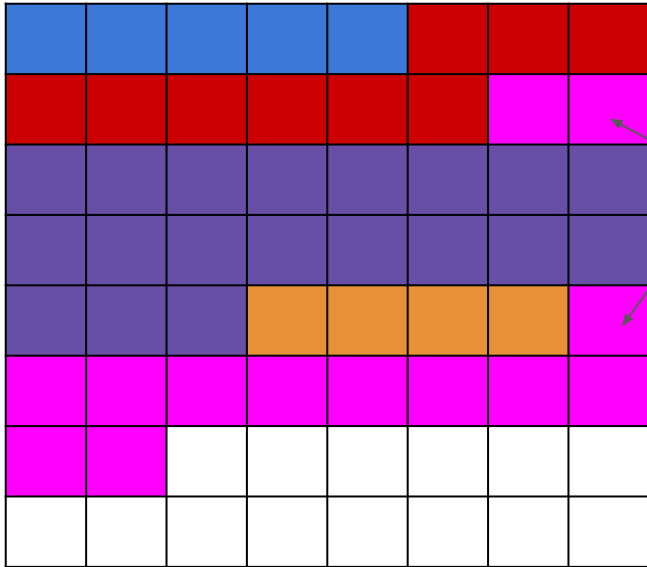


- Imagine the green file has been erased - these blocks have now been freed up

Each square represents a **block** on the hard disk. Coloured squares represent the data used for individual files.

How data becomes defragmented - 3

A model of the data on a hard disk



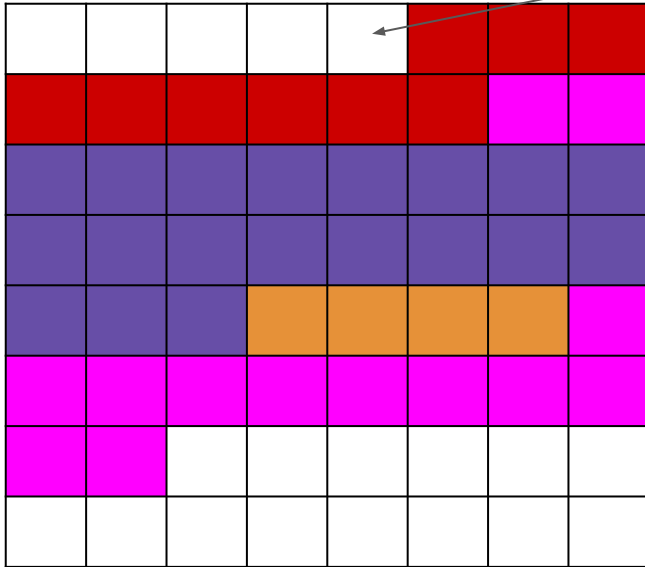
Each square represents a **block** on the hard disk. Coloured squares represent the data used for individual files.

- Imagine that a new, large file (maybe a photo) is stored on the disk.
- The OS begins by writing the photo's data to the first available blocks, but there are too few to store the whole file
- So the additional blocks are written elsewhere on the disk

How data becomes defragmented - 4

A model of the data on a hard disk

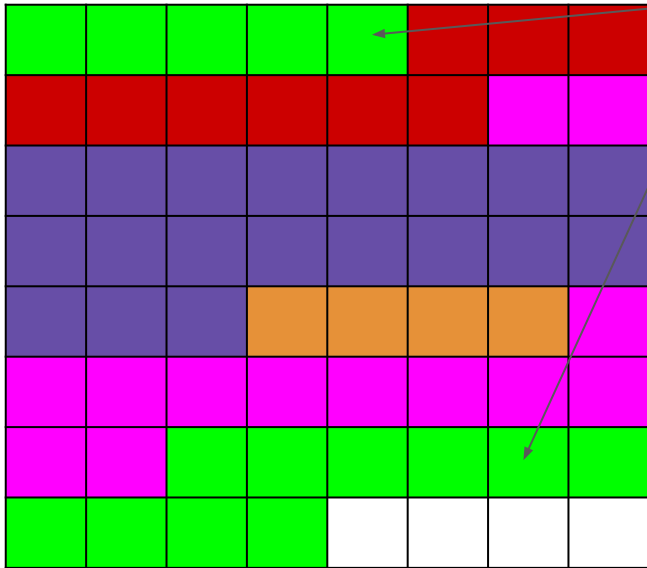
- Another file (the blue one) is erased



Each square represents a **block** on the hard disk. Coloured squares represent the data used for individual files.

How data becomes defragmented - 5

A model of the data on a hard disk

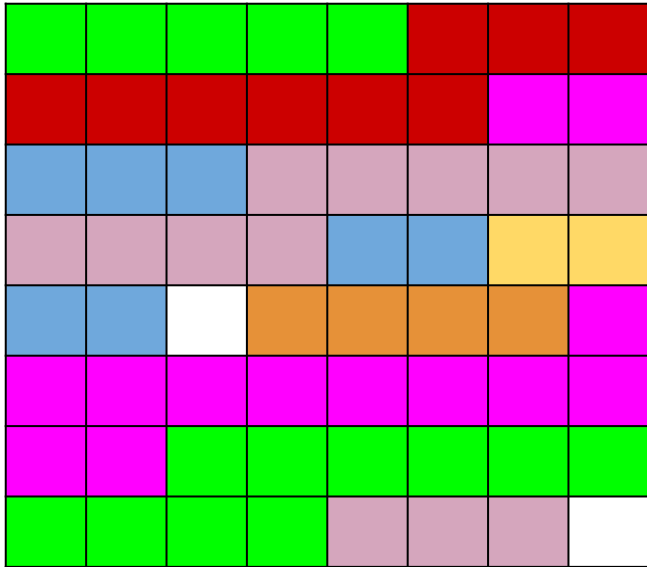


- And another large file is written to the disk
- This new file also becomes fragmented across the disk's available blocks

Each square represents a **block** on the hard disk. Coloured squares represent the data used for individual files.

How data becomes defragmented - 6

A model of the data on a hard disk



Each square represents a **block** on the hard disk. Coloured squares represent the data used for individual files.

- Eventually the data on the hard disk can become **very defragmented** with files split all over the disk's surface
- This **reduces read/write times** as the mechanical head within the hard disk has to do a lot of movement to read all of a file's data
- This also increases **wear and tear** on the disk, reducing its lifespan

SSDs and data fragmentation

- SSDs have **no moving parts**, therefore having data split around the flash memory locations does not slow down read/write times
- As a result, there is **no need to perform defragmentation** upon SSDs
- In fact, because SSDs have a **limited number of read/write cycles**, performing defragmentation can **reduce the life** of an SSD
- This is because defragmentation involves **lots** of reading and writing of data to the disk - billions of blocks of data need to be moved.



Don't defrag your SSD!
You'll kill it! (eventually..)

Image by [Shakib Saifi](#)

Backup utilities

The need for backup

- We store so much vital data on computers these days that losing this data could have **catastrophic consequences** for businesses, medical organisations, governments and consumers (no more Netflix?!!)
- A backup is a **copy of the data** on a computer or device (including servers) that is **stored externally** to the computer.
- In the event of data loss, data can be recovered (**restored**) from the backup.
- Data loss could occur for **many reasons** including fire, natural disaster, system failure, malware, hacking or even human failure.

Types of backup

There are two main types of backup that you need to know:

- **Full backup** - a copy is made of everything on the system onto backup media
- **Incremental backup** - only the files that have changed since the last full backup are copied

There is another type called *differential backup*, but this isn't on the GCSE specification, so don't worry about it for now.

Features of backup utility software

Backup utility software (or simply "Backup software") allows users to:

- **Select** which files and folders should be backed up
- **Schedule backups** (i.e. state when they should happen)
- Recover ("**restore**") lost data from backups
- **Manage** backup media
- Choose which **type** of backup should be performed, either **full** or **incremental** backup

Full backup

- A full backup involves copy **all of the data** on a system.
- Full backups take a **loooooong** time to complete (as a lot of data is being copied).
- Full backups take up **a lot of space**, so high-capacity storage media (ideally cheap per GB also) are used - for example, magnetic tape.
- Due to the amount of time involved, full backups are often made **overnight** when a network/server is not being used.
- Full backups are often taken once a week or once a month and are "topped up" by **incremental backups**...

Incremental backup

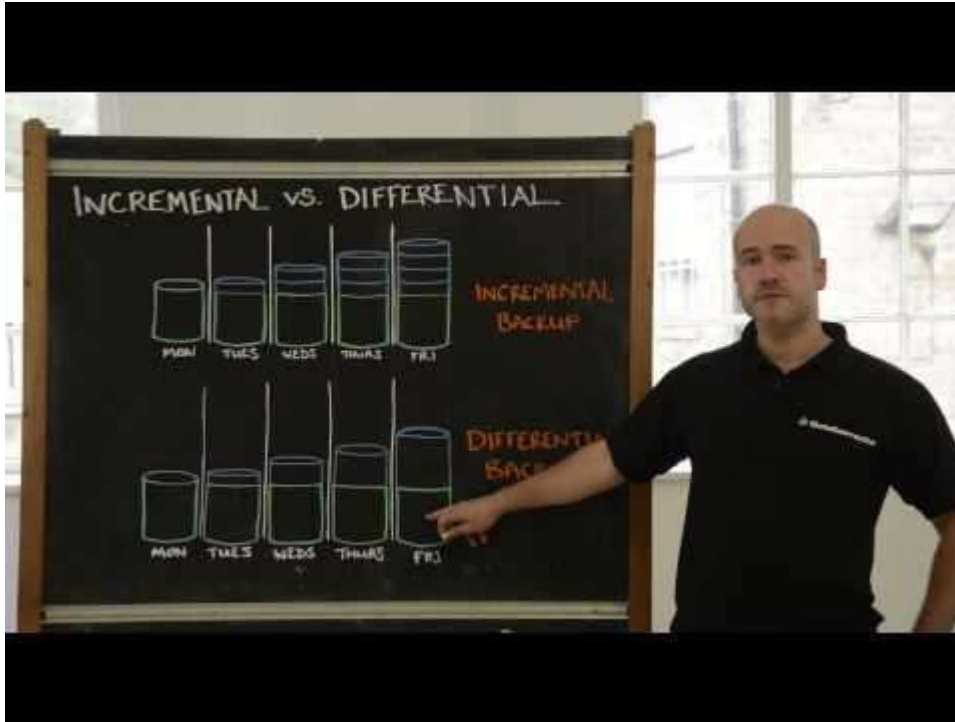
- An incremental backup only makes copies of the files that have been **created** or **changed** since the last backup (be it full or incremental).
- Incremental backups are a lot **faster** to complete as **much less data** is being copied.
- Incremental backups also use **much less space** and are therefore often done on hard disk drives.
- Incremental backups can be made whilst a network is in use, sometimes multiple times a day.
- Recovering a full system is **slow** as the last full backup must be restored followed by every subsequence incremental backup.

Exam tip!

You should be able to describe and compare both forms of backup, referring to:

- The nature / process of each type of backup (i.e. what happens in each type of backup)
- The time taken to complete each type of backup
- The relative amount of storage space required for each backup
- The impact on the network and its users whilst backup is taking place

Video: Incremental vs differential backup



This video will give you a basic explanation of how an incremental backup is made following a full backup.

Video by [Databarracks](#)

Backup storage media

- Magnetic tape, external hard disk and optical discs can be used to store system backups.
- Magnetic tape is most suited for **enterprises** with **large amounts of data** to backup from their servers. Magnetic tape is very **high-capacity** whilst being **small and portable**, allowing the backup to be kept **physically apart** from the original data. This is important for **reducing the risk of the backup being lost along with the original data** as a result of natural disaster, fire, flooding or theft.
- External Hard Disk drives and optical media are often used for **home or personal** backups.
- The Cloud can also be used to backup documents in personal and enterprise contexts.

Professional / enterprise backup



Home / personal backup



Cloud storage is really just loads of hard disks in a data centre run by some company, so technically it is enterprise backup!

Encryption utilities

Encryption utilities

- Encryption utilities "**scramble**" data by applying an **encryption algorithm** to the **binary data** that represents the file
- Encrypted data can only be decrypted by applying a secret "**key**" (in reality, a long/large **number**) to the decryption algorithm.
- Requiring a key to decrypt the data **prevents unauthorised users** from being able to access it.
- Good encryption utilities work closely with the Operating System to encrypt and decrypt your files "on-the-fly" as you need to access them.



Examples of encryption software

- Both Microsoft's BitLocker and Apple's FileVault are encryption utilities that are built-in within their operating systems.
- Both utilities encrypt either the contents of a user's "home" folder or the **entire hard disk** without the user having to do anything.
- Decryption of the file system occurs when an authorised user enters a **correct username and password combination** to log into the computer.
- There are many third-party encryption utilities that allow users to **encrypt specific files** with a **chosen key** before transferring the file to someone else to decrypt (they will likely need the same utility software to do this).

Compression utilities

Compression utilities

- Compression utilities reduce the amount of disk space required to store a file by applying a **lossless** compression algorithm to the file's binary data.
- Compression utilities can also **decompress** (or "extract") compressed files and folders so that they can be read.
- Compression utilities can compress **many** files (and folders) into a single compressed archive file. This, combined with reducing their size, makes transferring data across the Internet much **quicker** and convenient.
- Some compression utilities will allow you to **lock** the file with a **password**, adding security to files that will be sent online.
- Common file formats for compressed files include **.zip** and **.rar** files.

Video: Data compression explained



Video by [Techquickie](#)



ETHICAL, LEGAL, CULTURAL AND ENVIRONMENTAL CONCERNS

CHAPTER 8

WARNING!

The information in the slides is not meant to be an exhaustive reference for everything you could be asked in an exam.

It is **impossible** to revise every ethical, social and cultural issue that you could be asked about in an exam question.

The purpose of these slides is to develop your awareness of these **issue areas** and your ability to consider issues from different perspectives.

How to investigate and discuss Computer Science technologies

The impact of computing

Computing (and its related technologies) has had a monumental impact on almost all areas of life.

You need to be able to discuss the impact (both positive and negative) of computing on the following:

- **Ethics** - how have computers supported or encouraged us to disobey agreed, correct behaviour?
- **Society** - how has computing changed the way society is organised and the way we relate to one another?
- **Culture** - how has computing impacted the way that societies uniquely express themselves through thought, art and behaviour?

Video: Computer Science is changing everything



Video by [Code.org](https://code.org)

Investigating and discussing the impact of Computer Science

You need to be able to evaluate the legal, ethical, cultural and environmental issues related to the use of Computer Science. This includes an understanding of privacy.

To do this, you will need to consider these issues from the perspective of:

- The **stakeholders** involved (the various different people)
- The **law** (which laws apply to the situation)
- The **ethical** considerations surrounding the issue
- **Technologies** (what technology is involved)
- **Solutions** (what, if any, solutions can you propose for this issue?)

Level-of-response questions

Your understanding of this topic will be assessed in essay-style "Level of Response" questions.

You will need to write clearly and fluently and show a development of thought as you link your ideas together.

You will need to produce some form of plan for your response to the question.

Planning responses

Start by drawing out a grid and bullet-pointing everything you can think about for each element of your answer.

Stakeholders - Who is affected by or involved in the issue? - Try to think of less obvious groups of people!	Technologies - What technologies are involved? Artificial Intelligence? The Cloud? Networking? Databases? Surveillance?
Ethical, legal, cultural and environmental concerns - Which law does the issue apply to? - What will be the impact be on the environment or on society? - Is there a "right and wrong" angle that you can write about?	Solutions - What solutions might there be to the issue? - Can technology, the law, education, cultural change help?

LOR questions are not just ethical and legal...

You can be asked similar "Level of Response" questions on Computational Thinking topics, such as "How might Computational Thinking principles be applied to map design?".

To answer this, you would look at slightly different aspects:

- **Stakeholders** - who would design and use the maps?
- **Computational Thinking** - what computational thinking principles will be applied? Abstraction? Algorithms? Decomposition?
- **Impact** - what impact will the map have on society?
- **Solution** - what will the outcome be of applying the Computational Methods?

Approach to LOR questions

- Plan, don't panic - use a grid!
- Go beyond simply recalling facts - state **opinions** and attempt to justify them.
- Flesh out your plan into a properly written response.
- Use paragraphs, one for each square in your grid would be a good way to structure your response.
- Simply describing things without evaluating them, or expressing opinions on them, will limit you to the lowest marks.

How can you prepare for these types of questions?

- Get into the habit of **reading technology news articles** - <http://www.bbc.co.uk/news/technology> is an excellent source
 - Look for "for and against" arguments in the articles, particularly where ethical, cultural and environmental issues are raised
- **Learn** the relevant legislation
- Watch videos on these issues - [loads here!](#)
- Start **thinking** about these aspects when you consider any new technology, product, game or service
 - Practise thinking "how would I write about the impact on stakeholders of subscribing to Netflix" - who even are the stakeholders? Try to think of less obvious answers!

Stakeholders affected by technologies

Who are stakeholders?

Stakeholders are the **people** (or organisations) involved with an issue.

For example, when considering the impact on online movie streaming businesses such as Netflix, relevant stakeholders might include:

- Netflix's shareholders
- Customers (you and me)
- Traditional broadcasters (BBC, etc)
- Film writers, directors, actors of the content shown
- Movie studios
- Advertisers
- Technologists - software developers, web hosting companies
- Internet service providers

Ethical issues

Ethical considerations of technology

Computing raises the possibility for many ethical dilemmas:

- Is it OK to stream a movie from a site such as 123movies.com if you were never going to pay to see it anyway?
- Is it right to continue developing artificial intelligence if we know that it might result in people losing jobs due to greater automation of the workforce?
- Should internet businesses such as Netflix be able to pay ISPs to prioritise traffic to their services?

Writing about ethics in your exams

There is no possible way to revise all of the potential ethical issues that could be raised by an exam question.

Whatever issues are raised, try to:

- present a **balanced argument** that looks at both sides of the issues
- write about the **impact on the various stakeholders** - who are the "winners" and "losers"?
- wherever possible, use **evidence** to back up your position in the form of **news stories** and events that you have heard or read about

Ethical issues that you should be aware of

Having said that you cannot revise **all** ethical issues, you should be aware of the following:

- Not everyone has access to computers and the Internet which might put them at a disadvantage in terms of education, access to welfare and business opportunities - this is called the **Digital Divide**.
- For a long time, a key principle of the Internet was that all data packets are treated equally - this is called **Net Neutrality** and it is being threatened, particularly in the US, by companies paying ISPs to prioritise their traffic.
- Electronic devices are typically assembled in factories that employ many people, but sometimes for **low wages and long hours**. Consumers have little or no choice in the matter - is this right?

Ethical issues that you should be aware of

Having said that you cannot revise **all** ethical issues, you should be aware of the following:

- The governments of some countries limit what their citizens can access online. This is called **censorship** and it occurs in China, Russia and North Korea amongst other places.
- There can be positive sides to censorship, however. For example, in the UK access to pornography and gambling websites might be restricted by ISPs to protect children.

Video: Inside Apple's iPhone Factory



Watch this video and consider the following:

- What ethical issues are being raised by the report?
- What impact would working in this factory have on the workers and their families?
- What are the working conditions like? How do they compare to the conditions that other people from the same part of the world might experience in other jobs?
- How different are the lives of those who make and use the devices featured in the report?

Video by [ABC News](#)

Video: Ethical issues raised by the Internet of Things



Video by [Computerphile](#)

Privacy issues

Privacy and the Internet

It is extremely easy to track online activity such as:

- Websites that you have visited
- Search terms that you have entered into Google
- Videos that you have watched
- What you have purchased

With the advent of mobile devices and the Internet of Things (IoT), it is also easy to record:

- Where you have been (location data, GPS)
- What you look like (Facial recognition)

Video: Tracking the trackers



Video by [TED](#)

Privacy and the government

Many governments' Intelligence Agencies have permission to track their citizens' online behaviour in certain circumstances.

This is often done in the "National Interest" or when there are concerns about "National Security" but it is difficult to ascertain when this is the case.

Government agencies and Police forces may ask ISPs to release information about their customers such as what websites they have visited.

A recent case in the US involved the FBI asking Apple to help break into a terror suspects' iPhone - something that Apple were unwilling to do on the basis of it requiring them to build a "backdoor" into their system that could impact on the privacy of every iPhone user.

Video: Your social media likes expose more than you think



Video by [TED](#)

Environmental impact of Computer Science

Video: China's e-waste problem



Watch this video and consider:

- What was the main issue raised by the video?
- What environmental impacts were discussed?
- What solutions or opportunities were presented?

Video by [Al Jazeera English](#)

The environmental impact of Computing

Computing devices have many negative impacts on the environment:

- They contain **precious metals** and other **scarce raw materials** that must be mined from the Earth
- Their production produces a **significant amount of carbon dioxide**
- Their ubiquitous use requires a **huge amount of energy** on a global scale
- They contain **hazardous and toxic materials** that are difficult to recycle

This is e-waste



The growing problem of e-Waste

- e-Waste refers to discarded electronic devices
- Tens of millions of tonnes of waste is generated **every year**
- There are a number of **causes** of this problem:
 - **Consumers** replacing working devices with newer models each year
 - **Manufacturers** designing products that are **difficult to repair**, making it cheaper to replace with new units
 - **Manufacturers** designing devices with **planned obsolescence**
 - **Manufacturers** using **lower quality components** so that devices break after a year or two

Where does all the e-Waste go?

In the UK and EU nations, the **Waste Electric and Electronic Equipment** (WEEE) directive imposes rules for how e-waste should be dealt with safely and promotes its reuse and recycling.

This is typically **more expensive** than simply throwing things away.

To reduce financial impact, electronics manufacturers and other organisations often send e-waste to countries where there are less strict rules, for example China and India. Here, **toxic chemicals** can get into the food and water supply and **children and adults can be poisoned or hurt** whilst trying to find precious **raw materials** that they can sell.

Video: The impact of e-waste



Video by [Rappler](https://www.rappler.com)

Video: What happens to discarded electronics?



Video by [Techquickie](#)

Natural resources and raw materials

Computers, smartphones and other electronic devices use a range of **natural resources** and **raw materials**.

- Cases are often made from **plastics** (derived from **crude oil**)
- Batteries might contain chemicals such as **lead** or **lithium**
- Chips might contain gold, copper, platinum and other minerals that are **very rare**

Mineral extraction is **expensive**, **energy intensive** and **pollutive**. It is also often dangerous and can result in injuries or death if the mines are not well managed. Because many of these materials occur naturally in the developing world, people who work in mines may have limited rights and safety.

Video: What's a smartphone made of?



Video by [TED-Ed](#)

Video: Smartphone sustainability



Video by [edeos- digital education GmbH](#)

How much energy does it take to run the Internet?

The vast majority of the Internet exists on servers in Data Centers.

Data centers are filled with servers that produce **heat**, so they need to be **cooled**. This often involves **air conditioning** systems, which produce **pollutants** and require a **lot of energy** (electricity) to run.

Server virtualization is used to run several virtual servers on each piece of physical hardware, meaning that no servers are sitting around not being used.

The US Department of Energy has calculated that the data centers in the US alone require 70 billion kWh each year - nearly 2% of the entire power consumption of the US ([source](#)).

Video: How much energy does the Internet use?



Video by [SciShow](#)

What are companies doing about the environment?

Tech companies are under **enormous pressure** to reduce their environmental impact.

Greenpeace publishes an [annual report on the impact of major electronics companies](#), looking at factors such as the **energy required to produce devices**, the **repairability** or shelf life of products, the amount of **energy the company uses**, the amount of **precious raw materials** used, the amount of **toxic or non-recyclable materials**.

Over the years, this report has had a significant impact on companies such as Apple, who have responded by improving their environmental footprint considerably.

Video: Apple's environmental efforts



Video by [VICE News](#)

Video: EcoDataCenter - The world's greenest data center



Remember that you should try to write about how technology can provide **solutions** to problems as well as the issues it raises.

This video presents a great example of how technology is being used to solve the problem of energy consumption by data centres that power the web.

Video by [EcoDataCenter](#)

Positive impacts of technology on the environment

It's not all bad! The advancement of computing has had many positive impacts on the environment, including:

- Computer **modelling** allows scientists to monitor and predict the effects of **climate change**
- Computer models are used to test and improve the **efficiency of renewable energy sources** such as wind turbines
- Computer modelling enables car designers to create more **efficient engines** that produce less pollution
- **Online communication** reduces the need for travel for meetings, reducing the impact of pollution from cars and aeroplanes

Social and cultural implications of Computer Science

Writing about social impact in exams

Again, it is **impossible to revise for every possible social situation** that might arise in an exam question.

Instead, you need to (again) be prepared to offer a **balanced view** that takes into account common-sense and less-obvious ideas.

Write about the persons involved (**stakeholders** - not just the obvious ones) and **whether technology can offer any solutions** to the issues as well as problems.

Technology's impact on Society

Technology has impacted several aspects of society, for example:

Health

- ✓ Technology can improve health through computerised imaging, modelling diseases, AI can diagnose conditions faster and more accurately than doctors.
- ✗ Too much use of technology can be bad for health - eye strain, RSI, neck and back problems, lack of exercise are all common health problems that are associated with the use of computers at work.

Technology's impact on Society

Technology has impacted several aspects of society, for example:

Social interaction

- ✓ Social networks allow people to connect with long lost friends or family anywhere in the world and keep up to date with what is happening in their lives.
- ✗ Users of social networks can receive an idealised view of others' lives, causing them to feel less satisfied with their own life. Users can become obsessed over reactions to their content and this can have a significant impact on their mental wellbeing. Cyberbullying and trolling can be distressing and difficult to police.

Technology's impact on Society

Technology has impacted several aspects of society, for example:

Education

- ✓ Online learning platforms provide access to courses at some of the world's best universities for free to anyone with an Internet connection.
- ✗ Technology is expensive and poorer schools (perhaps not in the UK) might not be able to afford computers and devices, putting those children at a disadvantage. This is called the **Digital Divide**.

Technology's impact on Society

Technology has impacted several aspects of society, for example:

Welfare

- ✓ Applying for benefits and other government services saves time and is much quicker than manually filling out and processing forms. This saves the government money that can be spent on improving services.
- ✗ Those who need state support the most might be the least likely to have access to or the capability to use computers, for example the elderly, disabled or poorer persons. By forcing welfare applications online, these citizens might miss out (**Digital Divide** again).

Technology's impact on Society

Technology has impacted several aspects of society, for example:

Work-life-balance and wellbeing

- ✓ Mobile devices allow employees the freedom to work from anywhere and can help them compete more effectively by being available at all times.
- ✗ Having "work in your pocket" can result in an intrusion on family time, resulting in poorer relationships and less rest. This has a significant impact on the wellbeing of employees, as well as family members (e.g. children feeling ignored by parents).

A funny look at the unintended consequences of tech



Video by [TED](#)

Examples of cultural impacts of technology

Examples of aspects of culture that have been affected by technology include:

- **Individualism** - easily posting updates, tweets and photos about or of oneself might lead to a more selfish society
- **"Netiquette"** - people often behave differently online from in the real world, leading to more instances of people feeling free to express offensive or inconsiderate views
- **Fake news** - the main source of news for young adults is social media (Facebook, Twitter, Snapchat, etc). The platforms are often used by publishers of Fake News and people rarely question the validity of their news sources.

Examples of cultural impacts of technology

Examples of aspects of culture that have been affected by technology include:

- **Richer cultural experiences** - online media allows for people to discover music and film and art that individuals have created and published, offering a far broader range than mainstream publishers
- **24/7, "always-on" culture** - technology has meant that we never switch off - from work, or entertainment. Many people are addicted to their devices and find they lack the rest and peace that they need to survive, leading to a huge increase in the popularity of wellbeing activities such as yoga and mindfulness.

Globalisation

The Internet has had a massive impact on culture, accelerating **globalisation**.

- Organisations such as Google, Facebook, Twitter and Apple operate in **all countries** around the world
- E-Commerce sites such as eBay and Amazon can **source and deliver products anywhere** in the world
- Local farmers in **remote parts of the world** can **sell products directly** online
- Streaming services such as Spotify and Netflix **export Western media** throughout the world.

Technology's impact on traditional news media

The media has always played a huge role in influencing a society's culture. Traditionally this has been newspapers, radio and TV.

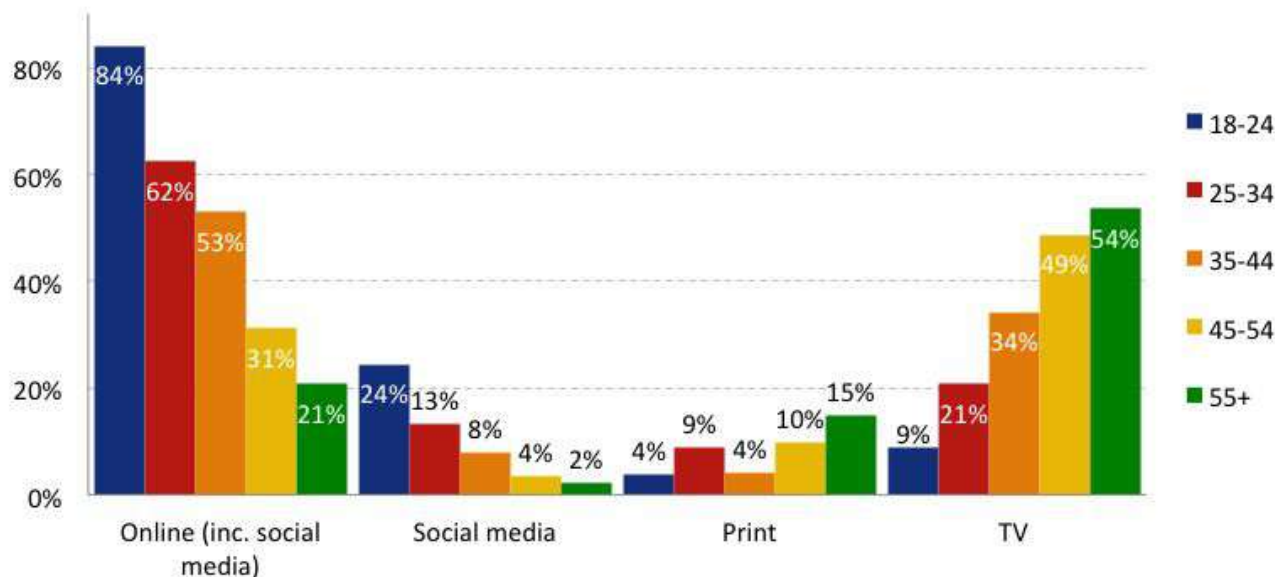
Since the advent of **Web 2.0**, traditional media has had much less influence in the face of platforms that host **user-generated content** such as YouTube, Twitter, Facebook and Instagram.

The **problem** is that, unlike traditional media, these companies are not **held to account** by governments and, as their content is provided by anyone, **can it be trusted?**





MAIN SOURCE OF NEWS (by age group)



Q4. You say you've used these sources of news in the last week, which would you say is your MAIN source of news?

Base: Base: All 18-24s/25-34s/35-44s/45-54s/55+ who have used a news source in the last week: UK= 220/271/353/392/714



Reuters Institute
for the Study of Journalism

How a handful of tech companies control billions of minds every day



This video is 17 minutes long - make a cup of tea, relax and enjoy!

Video by [TED](https://www.ted.com)

Open source vs. proprietary software

Open Source and Proprietary Software

The terms **Open Source Software** (OSS) and **Proprietary Software** relate to the **permitted use** of software and its **source code** as defined by **licenses**.

The essential differences are:

- Open Source Software - the source code is freely and legally available for anyone to view, modify and distribute
- Proprietary Software - the source code is not available for viewing, modification or distribution

Video: What is Open Source?



Video by [Danielle Thé](#)

Open Source licenses

Not all open source software can be used in exactly the same ways. Different **licenses** determine what can be done with the software's source code.

Code that is "remixed" or modified must be **distributed under the same license terms** as the original software.

This can get very **complicated** when a project utilises many different open-source components, each with different licenses.

It might also **limit your ability to sell software** if your project utilises open-source components whose license prohibits its sale.

Benefits of Open Source software

There are many advantages to Open Source software, including:

- Open source software is usually **free** for anyone to use
- Popular software is **maintained** by a **large community of contributors**, ensuring that it is **continually being testing and improved**, leading to **greater reliability and security** of the software product
- Open Source projects **encourage collaboration** and the sharing of ideas which can contribute to benefits in other software projects
- Changes and improvements can often be made more **rapidly** to open source software than proprietary software

Disadvantages of Open Source Software

Whilst OSS is often great, there are some things to be aware of:

- No warranty or guarantee, increasing **risk** to business that depends on OSS
- Smaller projects **may not receive regular updates** or contain **bugs** as they have not been subjected to mass testing by a large community of users and contributors
- Using open source components within your software requires that **your software is also distributed under an open source license**. Depending on the original licenses, you might still be able to charge for your software so your competitors can see your source code.
- Official **customer support might not be available**, however this isn't always true (it's how Open Source project make money)

Proprietary Software

- Proprietary software is software where the source code is not "open" to others to **view and modify**.
- Typically, you do not own the software you have paid for but rather you buy a **license** to **use** the software with a number of restrictions.
- Modifying, copying and redistributing proprietary software is **prohibited** under the Copyright, Designs and Patents Act as well as the Digital Millennium Copyright Act (US).
- Products such as Microsoft Windows and Adobe Photoshop are examples of proprietary software.

Benefits of Proprietary Software

In exchange for your hard-earned cash, proprietary software vendors typically provide a number of benefits:

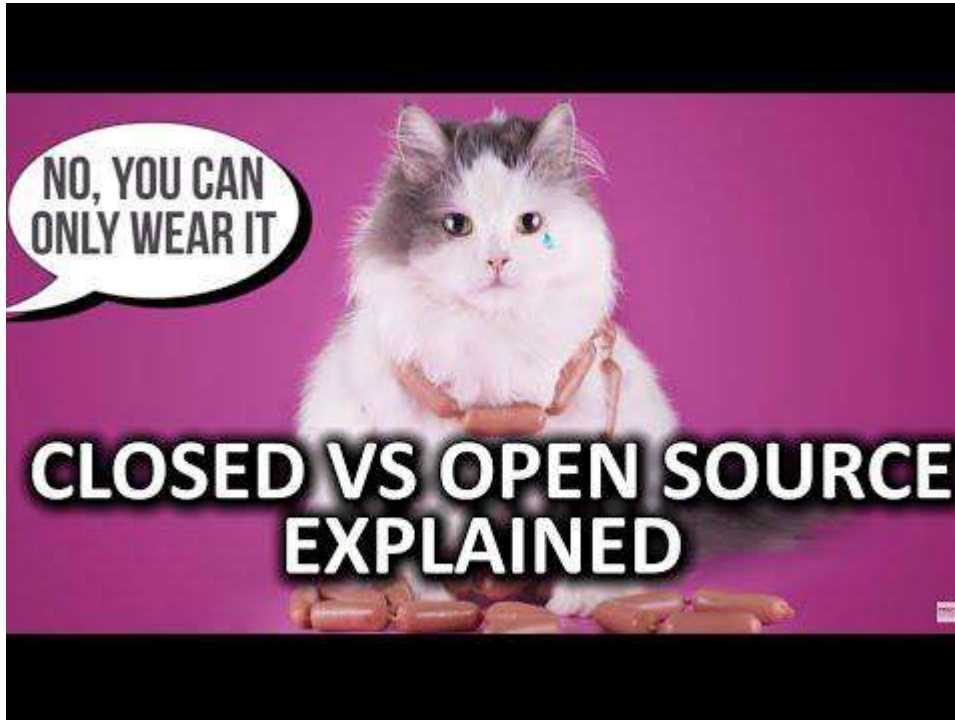
- The software comes with a **warranty or guarantee**
- **Customer support** is available (often for free, at least initially)
- The software should be **thoroughly tested and reliable** (though nothing can be perfectly tested before release). If any issues are found, **patches and updates** are quickly released.
- Proprietary software is sometimes **easier to install and use** for average users than downloading and installing open source software (which sometimes requires compiling from the source code).

Disadvantages of Proprietary Software

Of course, being proprietary means that this software comes with restrictions:

- The **source code is not available** so you cannot fix or modify it (if you have the skill and inclination)
- The product **might not do exactly what you want it to do** (more of an issue for businesses than home users)
- Can be **expensive**, either due to a high up-front cost or an ongoing subscription fee (Adobe Photoshop is a good example)
- Older versions **may not be maintained** as vendors want you to pay to upgrade to the latest version

Video: Open vs Closed (proprietary) software



Make the most of watching this video!

Whilst watching, write a list of advantages and disadvantages of open source software.

Video by [Techquickie](#)

Computing and the Law

Computing and the Law

There are a number of areas where the use of computers can introduce legal problems, such as:

- People agreeing to Terms and Conditions without reading them
- Using social media to spread slander or act abusively
- Downloading or streaming copyright material including music, films and TV shows
- "Hijacking" someone's social media account to impersonate them, post as them or, in any other way, act as if you were them

These examples demonstrate that, even though a law might exist to protect individuals or organisations, it can sometimes be difficult to enforce these laws online.

Terms and conditions examples

- Thousands of people sign up to 10,000 hours of community service by accepting (and not reading) Terms and Conditions for a free public WiFi service ([source](#))
- Over 90% of people in US admit to agreeing to legally-binding terms and conditions without reading them ([source](#))
- Failing to adhere to Terms and Conditions is unlikely to have serious consequences ([source](#), [another source](#))

The Data Protection Act 1998

The Data Protection Act (1998)



The Data Protection Act determines **what organisations can do** with **personal data** that they have collected and **defines the rights of individuals** over the data that is stored about them.

There are **eight principles** of the DPA that set out what can and cannot be done with individuals' personal data.

Under the DPA, individuals have the right to see what information is stored about them, claim compensation for damages caused by a breach of the Act and have inaccurate information corrected or removed.

Organisations that store personal data must register with the government and state what they will use the information for.

Video: Explaining the Data Protection Act



Video by [Skill Boosters](http://www.skillboostersvideo.com)

The 8 principles of the Data Protection Act

1. **Data must be used and processed in a fair and lawful way** - i.e. you cannot use that data for any illegal purposes.
2. **Data must only be used for the stated purpose** - if you've asked for an address to deliver a purchased item, you cannot use that address to send marketing material (without permission).
3. **Data should be adequate, relevant and not excessive for the specified use** - if someone has ordered some shoes from your online shop you don't need to know their age and gender, so don't ask for it.
4. **Data must be accurate and kept up-to-date** - if someone joined your mailing list 10 years ago and you haven't heard from them since, you should probably remove them

The 8 principles of the Data Protection Act

5. **Data should not be kept longer than necessary** - if you had a contract with EE but left them for Vodafone, EE shouldn't hold on to your data
6. **Data should only be used according to the rights of the data subject** - e.g. if someone asks for their personal information not to be shared, it shouldn't be shared
7. **Data should be kept safe and secure** - it is the organisation's responsibility to keep your data safe from hackers or loss through other means
8. **Data must not be transferred to organisations within other countries that do not offer a similar level of protection** - if you run an online shop but it's hosted on a server in Turkey then you might be breaking this law. Generally, data collected in the EU should stay in the EU.

Video: The Data Protection Act's principles explained



Video by [Teaching and Learning Resources for Me](#)

GDPR - Replacing the Data Protection Act

The Data Protection Act was introduced in 1998 and has since become a little outdated.

New EU legislation - the [General Data Protection Regulation \(GDPR\)](#) - is replacing it. The GDPR requires companies to ask individuals to confirm that they are happy for organisations to keep using their data.

Fortunately, the OCR specification doesn't mention the GDPR so they cannot ask you any direct questions about it, however you might gain yourself some points if you make reference to it in your Level-of-Response answers!

Computer Misuse Act 1990

The Computer Misuse Act (1990)

The Computer Misuse Act was introduced to protect against hacking and cybercrime.

It introduced three offences:

1. Attempting to access a computer or its contents without authorisation
2. Attempting to access a network or device with the intent of committing further criminal activity
3. Intending to modify or destroy a computer system, software or data without authorisation, for example creating, obtaining or deploying malware



Copyright Designs and Patents Act 1988

Copyright

- Copyright is a **legal framework** to **protect the use** of **intellectual property** that you (or someone else) have created.
- Written, drawn and recorded content are all subject to copyright. That includes, books, films, TV programmes, software, music, Youtube videos, Instagram pics, podcasts and even live performances.
- Copyright automatically applies to **any original creative work** that is physically expressed - you do not need to register your work for it to be protected by copyright legislation.
- **Different countries have different copyright laws** - the law that applies to a particular work will be that of the country where the work was produced.
- UK copyright expires 70 years after the holder's death, at which point they enter the **public domain** and can be used by anyone.

The Copyright, Designs and Patents Act (1988)

The Copyright, Designs and Patents Acts (1998) is the UK law that states how intellectual property may be used.

Intellectual Property is anything that someone has created - it could be a book they have written, a photo they have taken, a song they have recorded or an app that they have made.

Under the Act, it is **illegal** to:

- **make copies** of copyrighted material without the copyright holder's permission
- **share** copyrighted files (including downloaded movies and music)
- **use** unlicensed software
- **plagiarise** someone's creative work

Using copyrighted material

- Copyright applies automatically when intellectual property is created, however it is possible to **request permission** of the copyright holder to use their material.
- You can usually request permission by **writing to the creator or publisher, stating what you intend** to use their work for.
- The copyright holder does not have to grant permission but if they do, they can choose to do so for free or for a **fee**.
- Often, copyright holders will grant permission to students who wish to use their content for "educational purposes" without redistributing it.

Patents - protecting ideas and inventions

- Patents are legal protections over **inventions**. Unlike copyright, patents cover **ideas and concepts** rather than physically expressed intellectual property.
- In computing, patents can apply to both **hardware** (such as multi-touch screens in iPhones) and to **software** (such as the MP3 compression algorithm) inventions.
- Patent holders can **grant licenses** for other companies to use their patented ideas. Companies such as Apple and Samsung have spent billions of dollars protecting their own patents and suing each other over the unlicensed use of patents.
- Patents have **expiration dates** by which they must be renewed, otherwise the invention is freely available for others to capitalise upon.

Essential-standards patents

- If a company owns a patents to a "standard-essential" concept, the patent holder is **legally required to grant licenses** to that patent under "**fair, reasonable and non-discriminatory terms**" to all companies, including **competitors**.
- A "standard-essential" concept is an invention that is **essential for a standardised technology to work**. For example, the patent that covers how data is converted into radio waves to be sent over WiFi and the received and decoded.
- This legal provision protects against one company owning a **monopoly** on an essential technology.
- Sometimes companies **work together** to create corporate entities to own and maintain standards-essential technologies, such as the Bluetooth Special Interest Group that includes Microsoft, Apple, IBM and Intel.

Patent trolls

Patent trolls are companies that are set up to do nothing other than buy other companies in order to acquire their patents.

This allows them to then collect royalty fees on the use of their patents. They might also seek to sell those patents onto biggest technology companies.

The average Patent Troll defence costs \$2million and can take up to two years to resolve! This can have a massive impact on stifling innovation, particularly in smaller organisations and startups.

Video: Beating a Patent Troll



Watch this TED talk describing the story of defeating a patent claim from a patent troll.

Video by [TED](#)

Issues related to the Copyright, Designs and Patents Act

There are a number of issues around Copyright that have been raised by advances computing:

- The Internet makes **file sharing very easy**, making it difficult to **enforce** copyright.
- Digital representation of text, images, audio and videos allows for **perfect copies** to be made.
- Digital assets can be easily remixed and included in new projects, but there is a lack of clarity over how much can be used under "fair use" rights.
- Distributed peer-to-peer services such as BitTorrent allow files to be shared without a single, central server being responsible for hosting copyright files.
- There is uncertainty over whether website hosts are responsible for the illegal content uploaded by their users.

Issues related to the Copyright, Designs and Patents Act



The use of copyright material has recently been relaxed to allow for "fair use" such as using small amounts of copyright material within mash-ups, parodies and remixes.

However, it isn't clear exactly what constitutes "fair use" with the final say being down to a Judge's opinion.

Would you want to take that risk?

Video by BBC/[BhamUrbanNewsUK](#)

Creative Commons licensing

Creative Commons - a licensing framework

Creative Commons is a **licensing framework** that allows content producers to publish their intellectual property with **licenses that give permission** for others to use their work.

There are four basic types of license that you need to know about:

1. CC **Attribution** - Work can be shared, copied or modified as long as the content creator is credited
2. CC **Share-alike** - Any work that is used must be redistributed under the same license terms as the original
3. CC **Non-commercial** - Works can be used, but only if no profit is made from them.
4. CC **No-derivatives** - Work can be shared or redistributed, but not modified.

Licenses can be combined, for example: **CC-Non-Commercial-Attribution**

Finding content that isn't subject to copyright



Video by [CharliMarieTV](#)

Finding Creative Commons content

There are many places that you can find Creative Commons content online, including:

- Youtube
- Soundcloud
- Flickr
- Pixabay
- Google Images

CreativeCommons.org has an excellent search tool to help you find media from a variety of sources that is licensed under Creative Commons terms.

Freedom of Information Act 2000

Freedom of Information Act (2000)

The Freedom of Information Act states that **public bodies** must **regularly publish certain information** and provides members of the general public the right to **request information that is held by public bodies**.

Public bodies include:

- hospitals
- schools
- police forces
- armed forces
- local councils
- national government

Freedom of Information Requests

Any member of the public can make a **Freedom of Information Request**, there are no limits on age, gender, occupation, etc.

However, the public body **does not have to release requested information** if, for example, doing so would endanger national security or cause harm to an individual. In this case, the requester can appeal to the Information Commissioner (an independent body that can overrule the decision).

Information that can be requested includes **data stored on computers** such as letters, emails, CCTV footage, statistics, activity logs and financial information.

The background of the entire slide is a dense, repeating pattern of small, circular, multi-colored motifs. These motifs are arranged in a grid-like fashion, with each circle containing a different combination of colors and patterns, creating a vibrant, pixelated effect. The colors include various shades of red, orange, yellow, green, blue, purple, and pink, all set against a dark, almost black, background.

ALGORITHMS

CHAPTER 9

Computational thinking

What is Computational Thinking?

Computational thinking is a way of **understanding complex problems** in order to **form a solution** that can be implemented by either a computer, a human or a combination of both.

There are three key elements to computational thinking:

- Decomposition - breaking a problem down into smaller pieces
- Abstraction - removing details that are not relevant to solving the problem
- Algorithmic thinking - identifying logical steps from the problem to a solution to produce a reusable algorithm.

Abstraction

Abstraction is a computational thinking technique that **simplifies a problem** by **removing unnecessary detail** so that you can **focus on the important parts** that are relevant to the problem.

For example, a school might want a computer system to store details about pupils and staff. Each pupil or staff member would be represented as a row in a database with fields for the most important information like their name, address, date of birth and so on (rather than what they had for breakfast or who their favourite actor is). Each entry in the database is an **abstraction** of the real-life person. It is a simplified form that represents the relevant details about them for this particular "problem".

Examples of abstraction

The London Underground map is a classic example of abstraction.

It focuses on the important details such as the station names and the routes from one to another along the various train lines.

The lines are **not** realistically representative of the actual paths taken by each train.

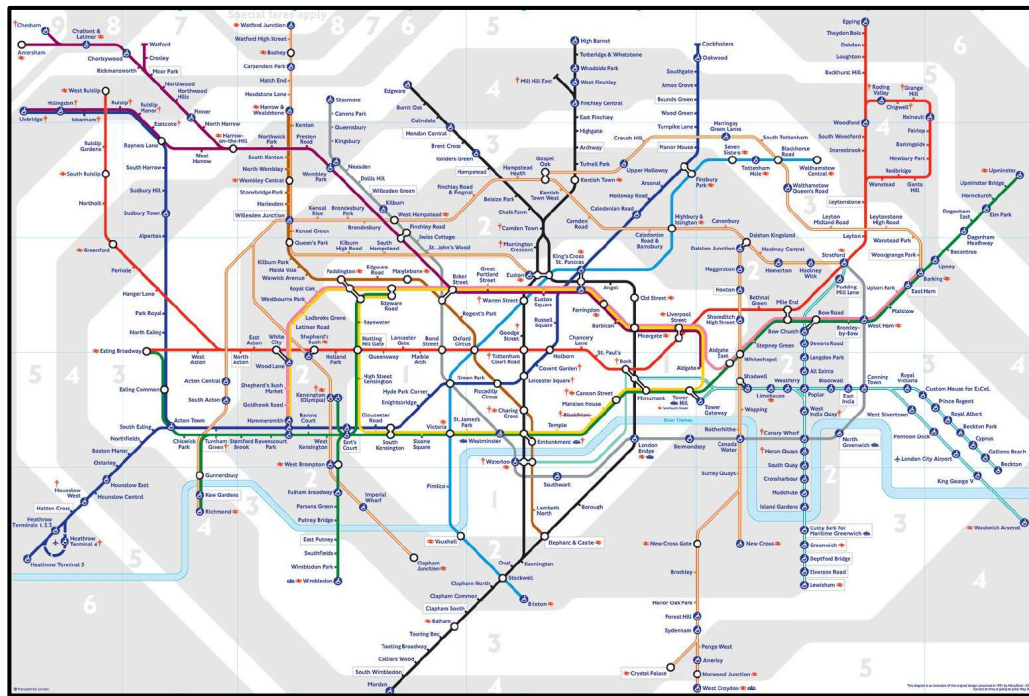


Image by Transport for London

Tube map abstraction vs real paths and distances

Tube map abstraction

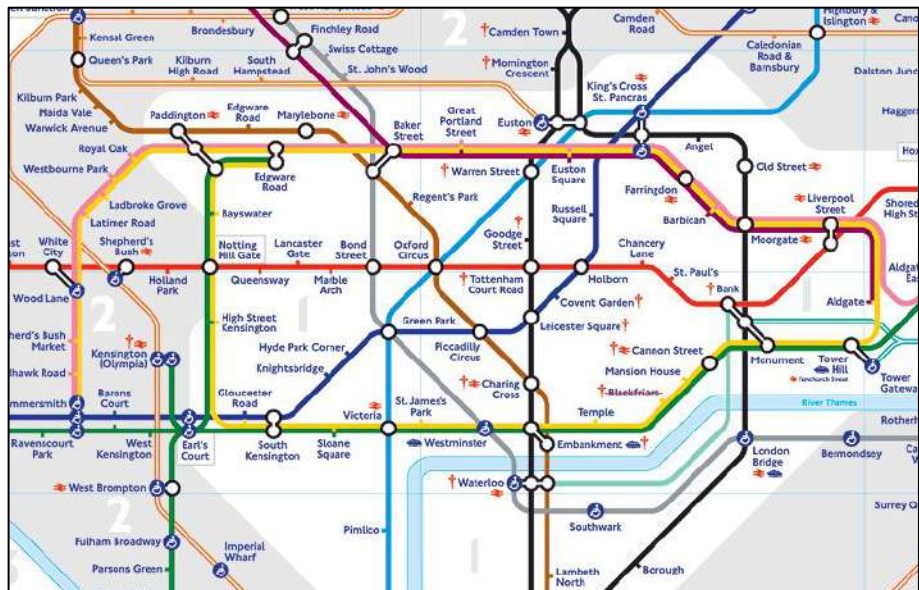


Image by Transport for London

Realistic distances and locations

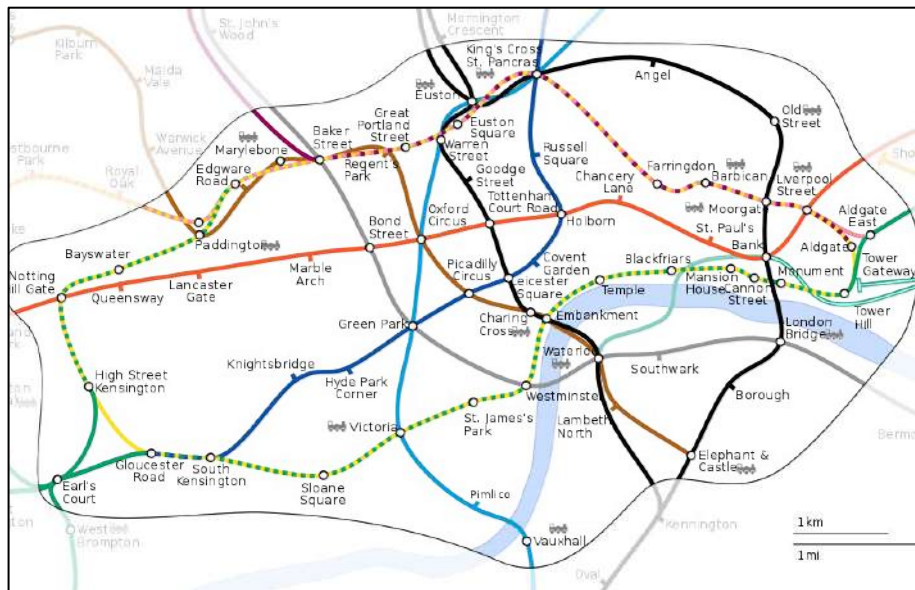


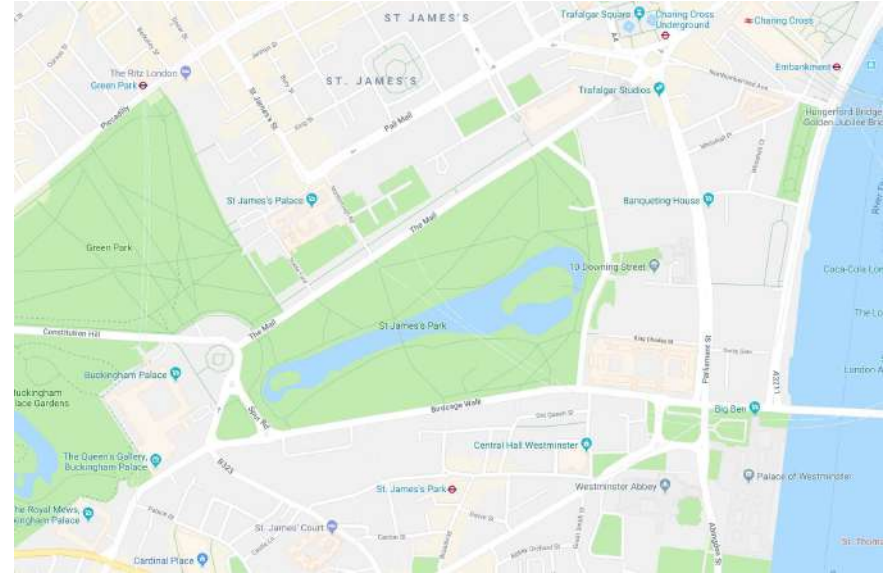
Image by [Jc86035](#)

All maps are examples of abstraction



The satellite image shows every tree and building as well as roads (although the roads aren't always very clear).

Images from Google Maps



The map, whose purpose is to help people drive or walk around an area, leaves out details about each and every building and tree in order to focus on the most important information, namely roads and major buildings and landmarks.

All maps are examples of abstraction

- A map **represents something from the real world**
- It **removes unnecessary details** (plants, trees, colour of houses)
- It **represents the relevant information** that you need with symbols or shapes (such as buildings and roads)
- The purpose of the map will determine what information is left out. For example a map for climbers and hill walkers might include information about the relative height of the landscape, whereas a driving map may focus on showing roads more clearly.
- When planning a route from one location to another, a mapping app needs to know which roads connect those locations but it doesn't need to know every detail of the landscape between those locations.

Icons in a WIMP GUI are abstractions

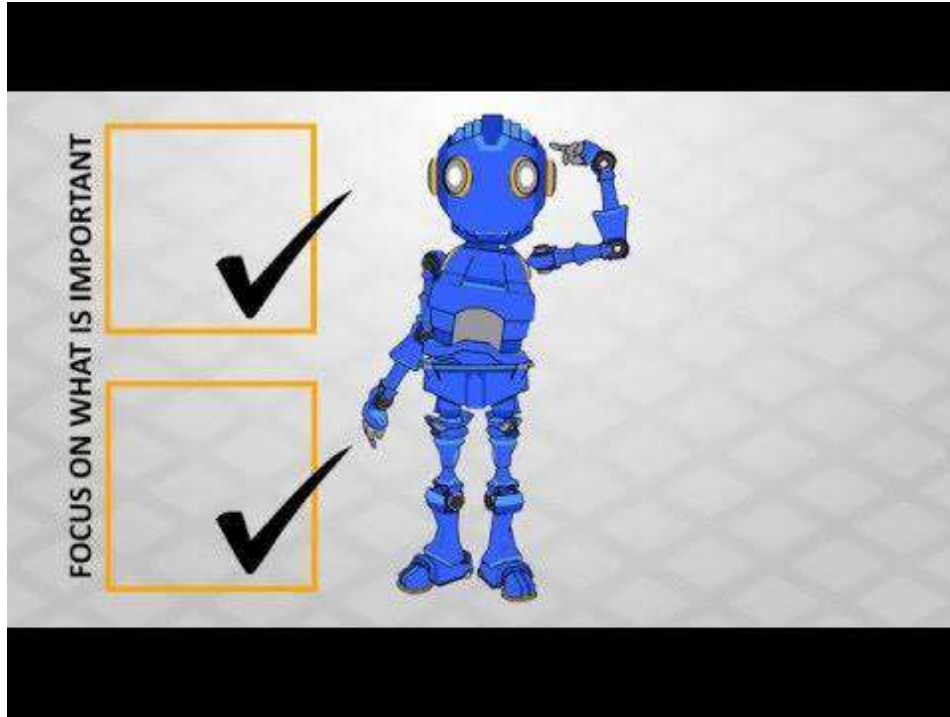
- Icons **represent** programs and files within the UI
- The actual programs and files are binary data stored as electrical or magnetic charges on secondary storage
- When we double-click an icon to open it we give a command to the OS to load that binary data into memory
- When we drag-and-drop an icon to move it from one folder to another the OS manages the physical movement of data on the disk



225975-specification-
accredited...e-j276.pdf

Not actually a file, just
a representation of
one!

Video: Abstraction explained



Video by [Robotics Academy](#)

Other examples of abstraction

In programming

- **Variables** in programs are used to represent real entities such as the name of a player or their score.
- A piece in a board game might be represented in a computerised version of that game as a **grid coordinate** whilst the board itself might be represented as a **2D array**.

In computer systems

- **Tables** in a database represent real-life entities (people, products, etc)

In the real world

- **Money** has no real value (it's just paper, or maybe not even that!), but it represents the value of goods and services

Decomposition

- Decomposition means **breaking down** a **complex problem** into **smaller, more manageable parts**.
- Each smaller part can then be **solved individually**.
- Decomposition allows large teams to each take a part of a problem and work on it.

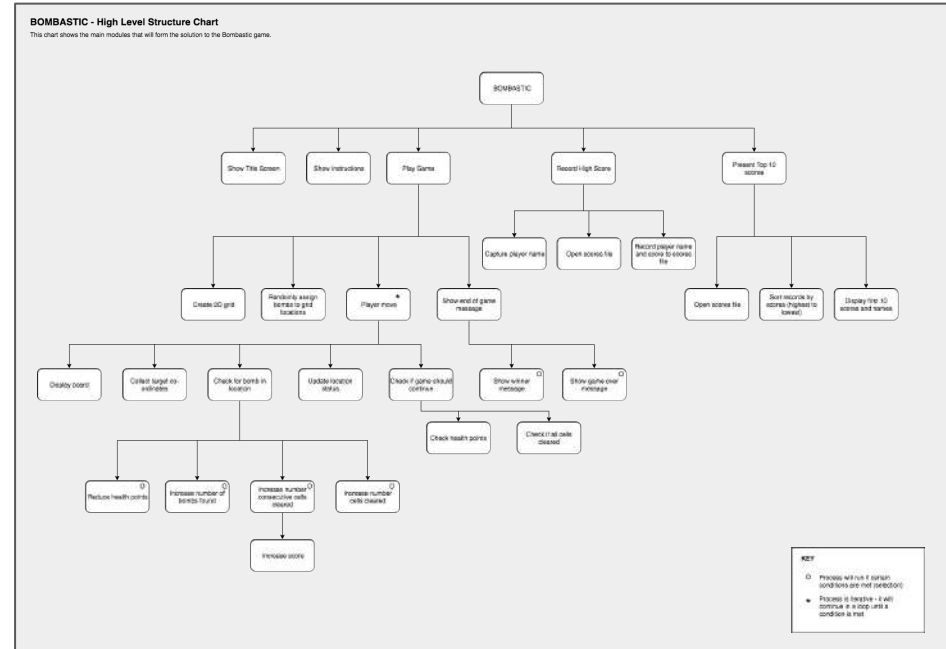


Just like in real life, "decomposition" in computational thinking means to break something down into smaller parts.

Image by [Hbreton19](#)

Structure charts

- Structure charts are used to break a large program down into the smaller parts that make it up.
- Each box represents a smaller problem to be solved. The lines show which bigger problem each box is a part of.
- Structure charts are a very useful tool when designing a solution to a big problem, such as a creating a computer game.



An example of a structure chart showing the parts that make up a computer game.

Algorithmic thinking

- Algorithmic thinking is a way of solving a problem by producing algorithms.
- An algorithm is a **reusable set of instructions** (or a series of steps or logical processes) **to solve a problem.**
- Algorithms can be written as a set of numbered steps to follow such as a cooking recipe or assembly instructions, however in computing they are presented as pseudocode or flow diagrams.

Algorithm building blocks

There are three main building blocks of all algorithms:

- **Sequence** - a series of steps that are to be completed one-by-one.
- **Selection** - steps that only occur if a certain condition is met.
- **Iteration** - steps that repeat, either for a fixed number of times or until a condition is met.

We use each of these algorithm structures when programming.

Video: What is an algorithm?



Video by [TED-Ed](#)

Applying Computational Thinking to solve problems

Computational Thinking is all about applying techniques from the world of Computer Science to solve problems, either in computing or in real life.

There are four main stages to solving any problem and different computational thinking techniques can be applied at each one:

- **Analyse** the problem to fully understand it through decomposition and abstraction of the problem.
- **Design** a solution to the problem using algorithmic thinking and abstraction.
- **Implement** the solution either by writing a program based on the algorithms or, if a real-world situation, follow the steps in the algorithm.
- **Test and evaluate the solution** to tests each of the outcomes from the initial computation thinking. For example, has each part identified through decomposition worked? Are the abstractions accurate enough? Do the algorithms work effectively and efficiently?

Producing algorithms using pseudocode and
flow diagrams

Video: Representing algorithms in flow diagrams and pseudocode



Video by [Cambridge GCSE Computing Online](https://www.cambridgegcsecomputingonline.com/)

What is pseudocode?

```
INPUT num1
INPUT num2
IF num1 > num2 THEN
    OUTPUT "Yes"
ELSE
    OUTPUT "No"
```

This algorithm is written in **pseudocode**:

- Pseudocode is a way to write out algorithms using **code-like** statements
- It is **not** an actual programming language
- There is **no "correct" way** to write it
- It is used to **plan algorithms**, focussing on the **logic** and **steps** rather than language-specific **syntax**

OCR's guide to pseudocode

Whilst it is true that there is no "correct" way to write pseudocode, OCR will always present pseudocode in a consistent way in exam papers.

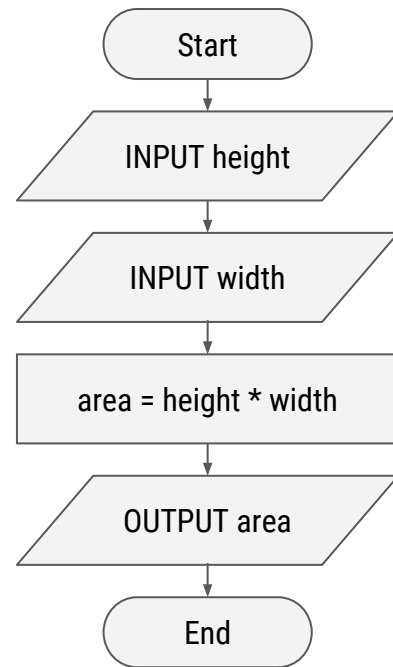
You should read through the [OCR guide to pseudocode](#) to ensure that you are familiar with how they will represent each of the Programming Techniques covered on the course using pseudocode.



Image by OCR

Using flow diagrams to represent algorithms

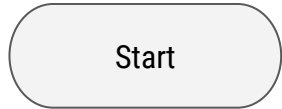
- Flow diagrams **visually represent** the steps that make an algorithm.
- A **standard set of shapes** are used to represent different types of step, such as running a sub-process or taking one path or another depending on a condition being met.
- The arrows in a flow diagram represent the **flow of control** through the algorithm.



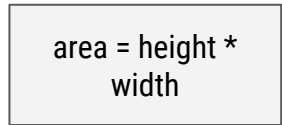
A flow diagram representing an algorithm for calculating the area of a rectangle.

Standard flow diagram shapes

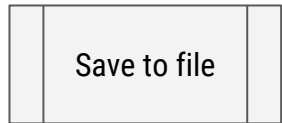
You should be familiar with the following standard shapes used in flow diagrams.



Terminator - is used at the start and end of the flow diagram. You should include "Start", "End" or if the process is a function that returns a value then the shape should include "Return" followed by the value (or variable) being returned.



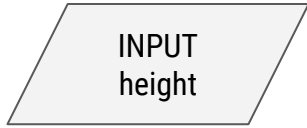
Process - this is used to represent a process. A short but clear description of the process should be written in the box. If you are describing several steps then you should use several process rectangles instead of one.



Sub-process - a rectangle with extra vertical lines on each side represents a whole other process that is to be run at this stage. This is very useful for a complex algorithms that are made up of several smaller processes, each of which is described in its own flow diagram. Include the name of the sub-process within the rectangle.

Standard flow diagram shapes

You should be familiar with the following standard shapes used in flow diagrams.



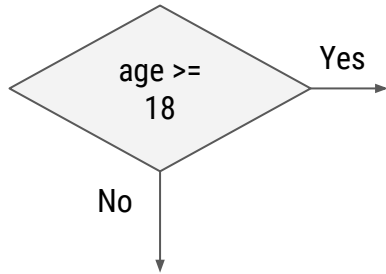
Input/Output - a parallelogram is used to represent a stage in the algorithm where data is either input by a user or output to a display. You should write either "INPUT" or "OUTPUT" within the shape to make clear which process is being represented.

If data is being input, you should also state the name of a variable that will be used to store the value.

If data is being output you should include the name of the variable that contains the output value or the message that is to be displayed.

Standard flow diagram shapes

You should be familiar with the following standard shapes used in flow diagrams.

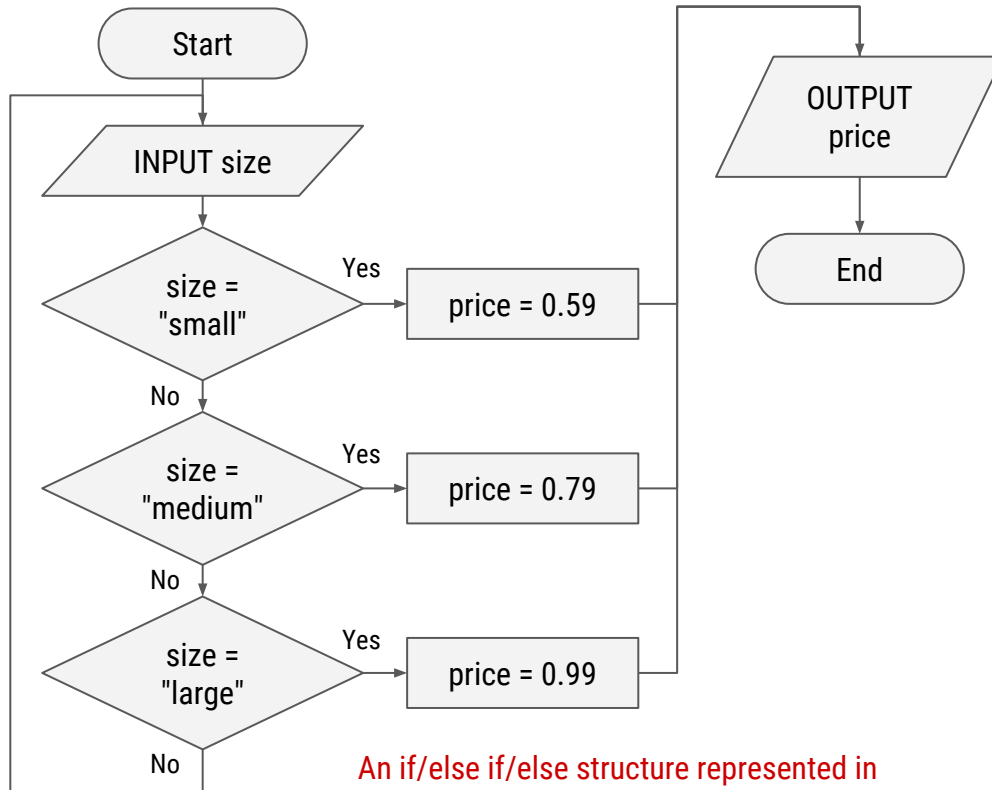


Decision - a diamond shape is used to represent a point of decision (or selection). You should include the condition that the decision depends on within the shape. Again, try to keep this short but specific.

A decision diamond must have **two labelled lines** coming out of it to show what sequence of steps should follow whether the condition has been met or not. These lines should be labelled with either "Yes" and "No" or "True" and "False".

It doesn't matter whether the "yes/true" or "no/false" labels go on the line coming down or to the side of the diamond.

Representing if/else if/else in a flow diagram

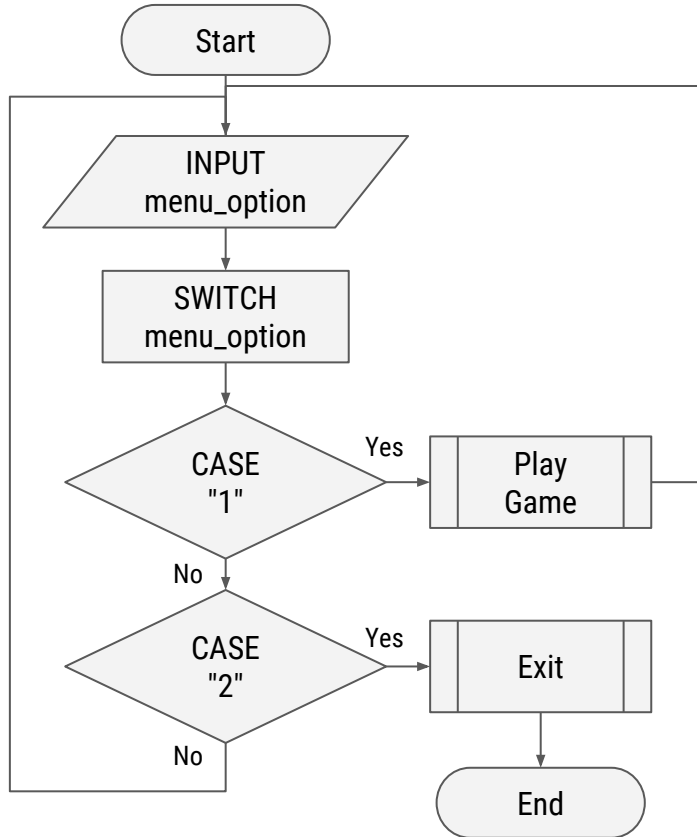


An if/else if/else structure represented in a flow diagram

An if/else if/else selection structure is used when multiple conditions are to be tested resulting in a different thing happening each time.

if/else if/else structures can be represented in flow diagrams using multiple decision diamonds flowing into each other from their "No" branches.

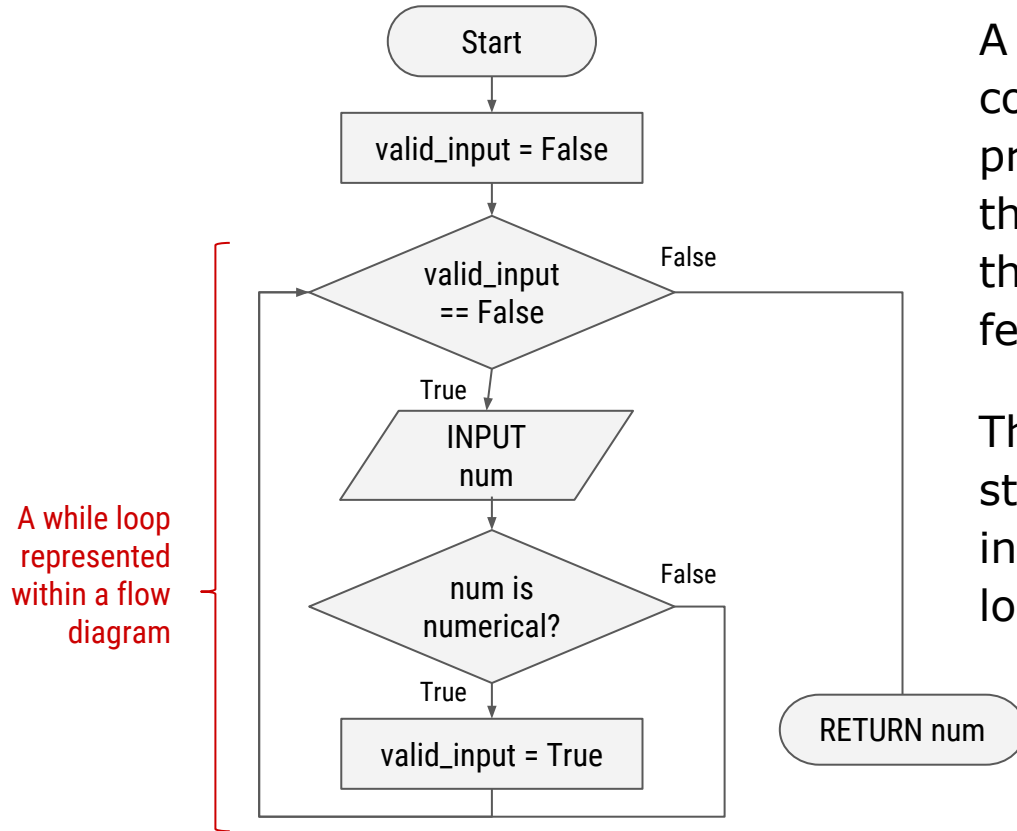
Representing switch-case structures in a flow diagram



A **switch-case structure** can be used to check whether **a single variable** has been assigned **one of many possible values**.

Switch-case structures are neater than using if/else if when testing for a particular value of a single variable but they cannot test multiple conditions that involve different variables.

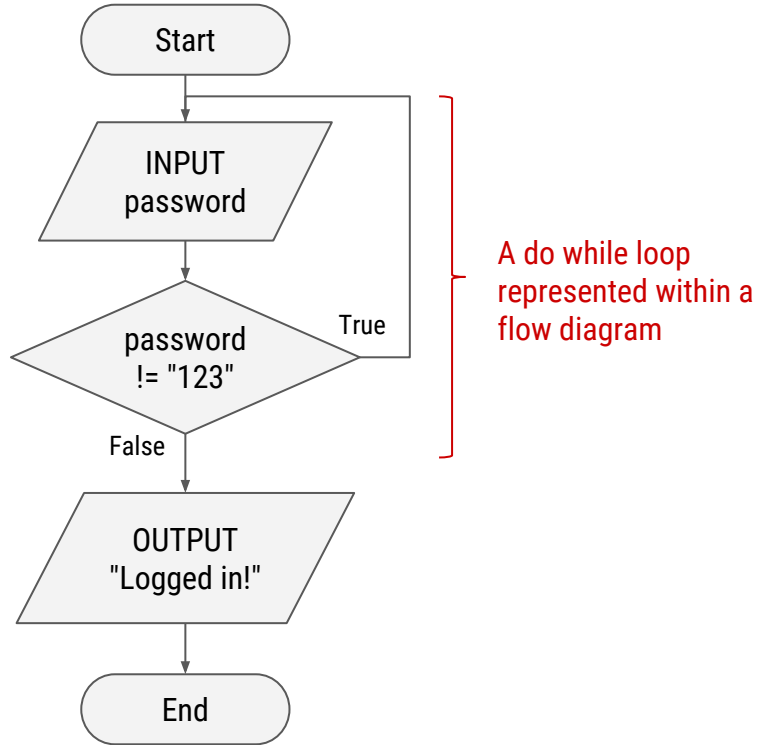
Representing while loops in a flow diagram



A **while loop** (a form of condition-controlled iteration) is presented by a decision diamond that has the "Yes" branch leading to the steps to be repeated and then feeding back into itself.

The "No" branch should skip past the steps to be repeated and straight into the steps to take place after the loop has completed.

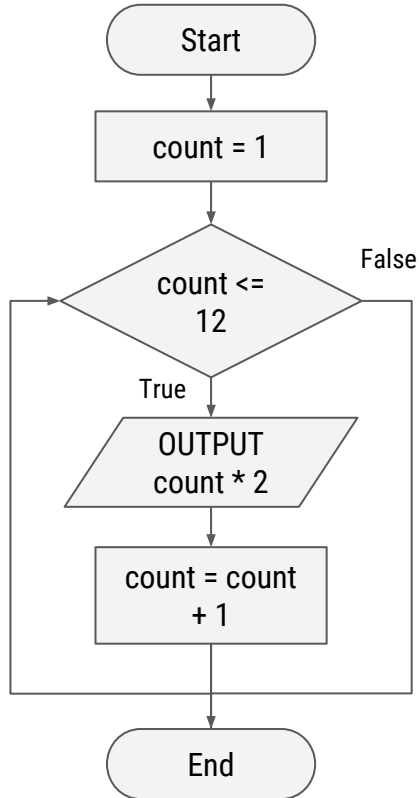
Representing do...while loops in a flow diagram



A **do...while loop** (a form of conditional iteration) is presented by the steps that will occur at least once, followed by a decision diamond that has the "Yes" branch leading back to the start of the steps to be repeated.

The "No" branch proceeds to the steps that take place after the loop has completed.

Representing for loops in a flow diagram

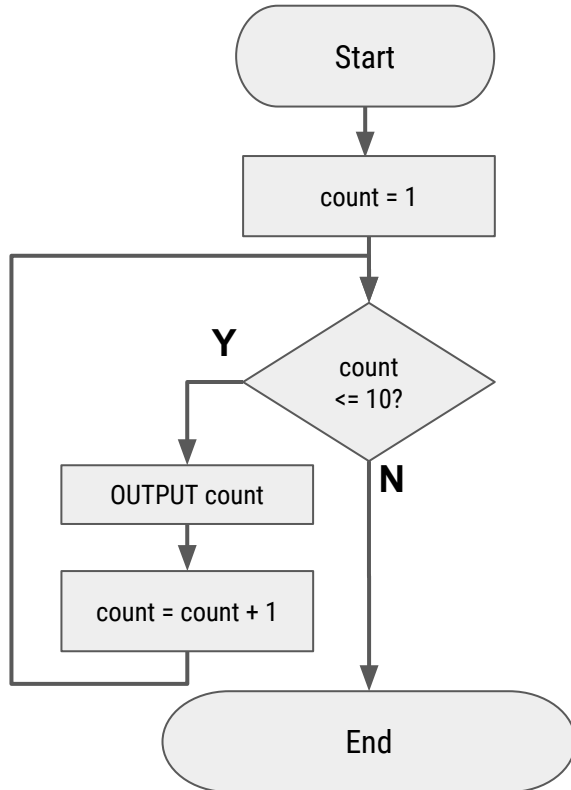


A for loop represented within a flow diagram. The for loop will run 12 times.

A **for loop** (count-controlled) iteration structure is really just a specialised form of while loop where the condition checks whether a counter variable has reached a limit.

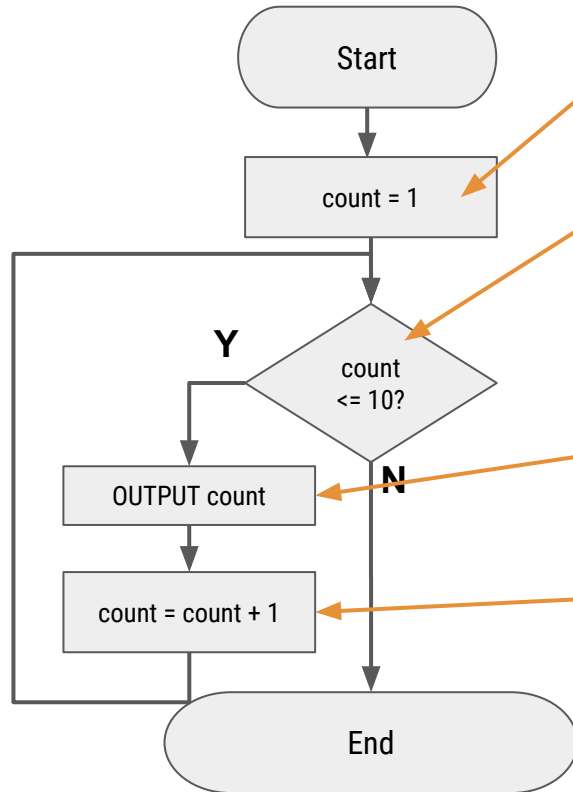
This counter variable must be given an initial value before the loop begins and incremented within the while loop (otherwise it will continue to run forever).

Representing for loops in flowcharts



- Set the incrementing variable (count in this case) to 1 at the start of the iteration
- Use selection (diamond) to check whether the incrementing variable is **less than or equal to** the maximum number of iterations required
- Show the processes that need repeating off the "Y" branch from the selection shape
- Don't forget to **increment** the counting / incrementing variable!

Representing for loops in flowcharts



- Set the incrementing variable (count in this case) to 1 at the start of the iteration
- Use selection (diamond) to check whether the incrementing variable is **less than or equal to** the maximum number of iterations required
- Show the processes that need repeating off the "Y" branch from the selection shape
- Don't forget to **increment** the counting / incrementing variable!

Worked example: Writing an algorithm to
solve a problem

Worked example: Writing an algorithm to solve a problem

Write an algorithm using pseudocode for a vending machine that runs each time a coin is entered to check whether enough money has been provided to dispense a can of drink that costs 65p.

- The algorithm should allow a user to keep entering coins until the required amount have been provided
- If not enough money has been provided the algorithm should output the amount that still needs to be entered.
- If the right amount of money has been entered the drink should be dispensed with no change given.
- If too much money has been provided, the drink should be dispensed and change should be given.

Worked example: Writing an algorithm to solve a problem

Start by analysing the problem, applying decomposition and abstraction.

- Decomposition - break down the problem
 - The system needs to allow users to enter money
 - The system needs to check whether enough has been entered
 - The system needs to be able to dispense a drink
 - The system needs to be able to calculate and dispense change
- Abstraction - represent only the relevant details
 - The solution doesn't need to know which drink has been selected; there is no need ask users to "make a selection"
 - The solution does need to represent the required amount for a drink (65p) and how much has been entered so far

Worked example: Writing an algorithm to solve a problem

Now we understand the problem, we need to design the solution.

- Use algorithmic thinking to produce a solution
 - Amounts of money entered and the total required can be represented as variables
 - The basic sequence is: enter money, check if money is enough, if so, dispense drink and any change
 - Indefinite (condition controlled) iteration will be required to allow the user to keep entering money until the required amount is provided
 - Selection will be used to determine whether change needs to be given.

Worked example: Writing an algorithm to solve a problem

Implement the solution as an algorithm using pseudocode.

```
total_entered = 0
cost_of_drink = 65
while total_entered < cost_of_drink
    INPUT money
    total_entered = total_entered + money
    remaining_amount = cost_of_drink - total_entered
    OUTPUT "Amount remaining: " + remaining_amount
dispense_drink()
if total_entered > cost_of_drink then
    change_amount = cost_of_drink - total_entered
    give_change(change_amount)
```

This algorithm is written in **pseudocode** - "code-like" English statements that represent each step of the process.

Worked example: Writing an algorithm to solve a problem

Test the solution using a series of input values that represent money entered.

```
total_entered = 0
cost_of_drink = 65
while total_entered < cost_of_drink
    INPUT money
    total_entered = total_entered + money
    remaining_amount = cost_of_drink - total_entered
    OUTPUT "Amount remaining: " + remaining_amount
dispense_drink()
if total_entered > cost_of_drink then
    change_amount = cost_of_drink - total_entered
    give_change(change_amount)
```

money	total_entered	output
-	0	Amount remaining: 65
10	10	Amount remaining: 55
50	60	Amount remaining: 5
10	70	Drink dispensed Change given: 5p

This is a trace table. It is used to "dry-run" an algorithm to see how it performs as it progresses.

Interpreting, correcting and completing
algorithms

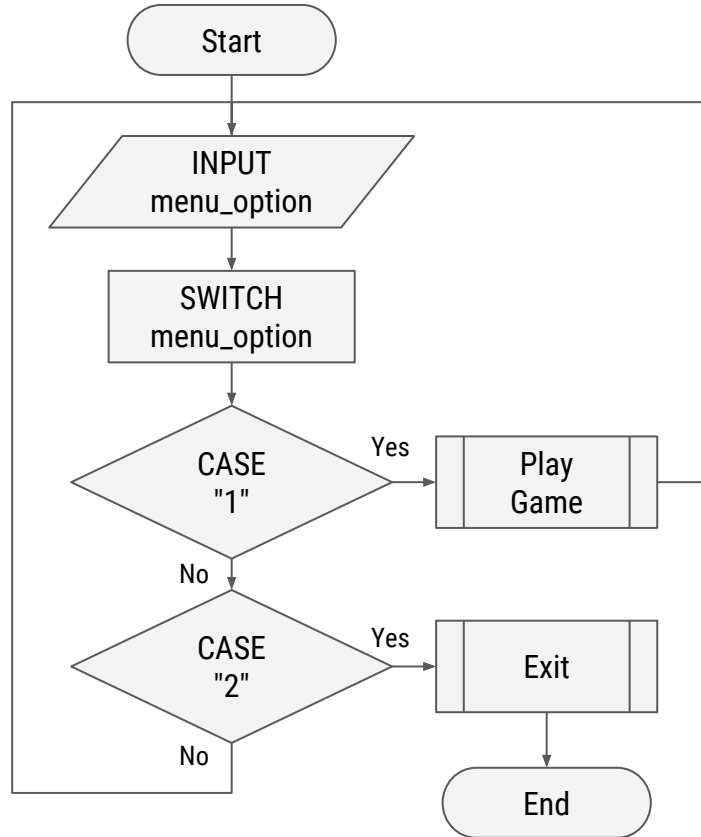
Interpreting algorithms

You need to be able to look at an algorithm and explain what it does.

The best way to approach this type of exam question is to look for clues such as the names of variables, the data being input and any output messages being given.

To make sure that you cover everything in the algorithm, start at the top and write out the effect of each and every stage shown in the algorithm (presented as either a flow diagram or pseudocode).

Interpreting algorithms - an example



The variable identifiers in this algorithm suggest that it is for a main menu in a game.

An explanation of the algorithm might be as follows.

The algorithm starts by asking the user to enter an option from the menu, this is assigned to a variable called 'menu_option'. A switch structure is then used to select which action should occur next depending on the value of 'menu_option'.

If the value assigned to 'menu_option' is "1" the "Play Game" sub-routine is executed after which the process begins again at the main menu. If the value assigned to 'menu_option' is "2" the "Exit" subroutine is executed, after which the program ends. If any other value is assigned to 'menu_option' the user is asked to input their option again.

Correcting algorithms

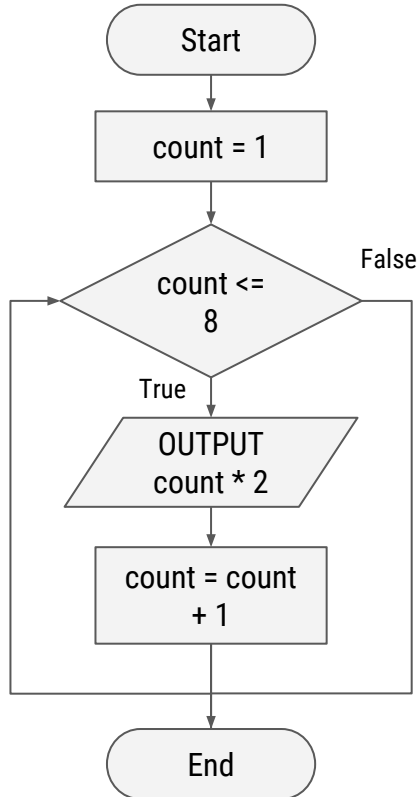
You should be able to look at an algorithm and identify and fix errors such as:

- incorrect **variable identifiers** being used
- incorrect **relational operators** such as checking whether a value is less than X, instead of less than *or equal to* X)
- missing processes, such as not incrementing a counter variable in a count-controlled iteration structure which can cause the algorithm to enter an **infinite loop**.

Completing algorithms

You should be able to follow the steps of a simple algorithm in an exam and state what its final outcome will be. A good way to do this is to use a trace table that shows the state of variables at each step of the algorithm.

Completing algorithms - an example



count	output
1	2
2	4
3	6
4	8
5	10
6	12
7	14
8	16
9	-

To complete the trace table for the algorithm on the left, start by writing the first value of 'count' into the 'count' column and then follow the algorithm through to find the output and write this into the table.

Continue to follow the algorithm, which will increment the value of count, so write this into the next row in the trace table.

Keep doing this until all the algorithm reaches its end.

Video: Tracing an algorithm with multiple variables

```
INPUT x
INPUT y

FOR count = 1 TO y
  result ← x * count
  OUTPUT result
NEXT
```

x	y	count	result	output
5	7	1	5	5
		2	10	10
		3	15	
		4		
		5		

Video by [Jonathan Molson](#)

Searching algorithms

Search algorithms

A search algorithm is a set of instructions for **finding a specific item** of data (a "target") **within a data set**.

Computer systems can store and process **billions** of pieces of data so it is vital that computers can find the information they are looking for **efficiently**.

Two different types of search

There are two different search algorithms that you need to know about:

- Linear search
- Binary search

Each algorithm is **effective** - whichever you choose they will either find the target or determine that the target isn't present

However, each algorithm **differs** in terms of its **complexity** and **efficiency**.

Linear search - learn these steps!

The linear search algorithm is very simple:

1. Start with the **first item** in the list
2. **Compare** the item to the **target** - if it matches then stop the search
3. Move onto the **next item** in the list
4. **Repeat steps 2-3** until the item is found or every item has been checked
5. If every item has been checked and the target has not been found then it is not in the list

Linear search - example

Use linear search to find the item "cabbage" in the following list:

tomato	potato	carrot	cabbage	onion



1. Compare first item in the list ("tomato") to "cabbage" (the target) - they don't match, move to next item

Linear search - example

Use linear search to find the item "cabbage" in the following list:

tomato	potato	carrot	cabbage	onion



1. Compare first item in the list ("tomato") to "cabbage" (the target) - they don't match, move to next item
2. Compare "potato" to "cabbage" - they don't match, move to next item

Linear search - example

Use linear search to find the item "cabbage" in the following list:

tomato	potato	carrot	cabbage	onion



1. Compare first item in the list ("tomato") to "cabbage" (the target) - they don't match, move to next item
2. Compare "potato" to "cabbage" - they don't match, move to next item
3. Compare "carrot" to "cabbage" - they don't match, move to next item

Linear search - example

Use linear search to find the item "cabbage" in the following list:

tomato	potato	carrot	cabbage	onion



1. Compare first item in the list ("tomato") to "cabbage" (the target) - they don't match, move to next item
2. Compare "potato" to "cabbage" - they don't match, move to next item
3. Compare "carrot" to "cabbage" - they don't match, move to next item
4. Compare "cabbage" to "cabbage" - they match, end search

Linear search - super easy but slow

Linear search is very **easy to implement** and can be used on **unordered data** but it can be quite **slow** to find a particular item of data.

In the worst case, linear search has to check **every item** in a list to determine whether the target is present.

- If a single check took 0.001 seconds it would take 1,000 seconds to search a dataset containing 1 million items.

Binary search - super efficient but data must be sorted

Binary search is a very **efficient** search algorithm but **only works** on **sorted** data.

Binary search works by comparing the middle item in the dataset to the target and "cutting off" half the dataset each time.



People would instinctively use a binary search method to find a name in an old-fashioned printed phone book.

Binary search - learn these steps!

1. Find the middle item in the **sorted** dataset using $(n+1) / 2$ (round up if necessary)
2. Compare the middle item to the target:
 - a. if it matches then the item is found
 - b. If the target is **smaller/comes before** the middle item then "cut off" the second half of the dataset including the middle item
 - c. If the target is **larger/comes after** the middle item then "cut off" the first half of the dataset including the middle item
3. Repeat steps 1 and 2 on the new, smaller dataset until the target has been found **or** the dataset cannot be split any further

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98



1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:

$$\text{middle item} = (13 + 1) / 2 = \underline{7\text{th item}}$$

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98



1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:

$$\text{middle item} = (13 + 1) / 2 = \underline{7\text{th item}}$$

2. Compare middle item to target: 76 (target) comes **after** the middle value so cut off first half of the dataset

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98

1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:
middle item = $(13 + 1) / 2 = \underline{7\text{th item}}$
2. Compare middle item to target: 76 (target) comes **after** the middle value so cut off first half of the dataset
3. Repeat steps 1 and 2 on the new dataset

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98



1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:

middle item = $(6 + 1) / 2 = 3.5$ so round up to the 4th item

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98



1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:
middle item = $(6 + 1) / 2 = 3.5$ so round up to the 4th item
2. Compare middle item to target: 76 (target) comes **before** the middle value (90) so cut off **second half** of the dataset

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98

1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:
middle item = $(6 + 1) / 2 = 3.5$ so round up to the 4th item
2. Compare middle item to target: 76 (target) comes **before** the middle value (90) so cut off **second half** of the dataset
3. Repeat steps 1 and 2 on the new dataset

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98



1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:

$$\text{middle item} = (3 + 1) / 2 = 2$$

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98



1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:
middle item = $(3 + 1) / 2 = 2$
2. Compare middle item to target: 76 (target) comes **before** the middle value (78) so cut off **second half** of the dataset

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98

1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:
middle item = $(3 + 1) / 2 = 2$
2. Compare middle item to target: 76 (target) comes **before** the middle value (78) so cut off **second half** of the dataset
3. Repeat steps 1 and 2 on the new dataset

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98



1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:

$$\text{middle item} = (1 + 1) / 2 = 1\text{st item}$$

Binary search - example

Use binary search to find the number **76** in the following dataset. Show the steps involved.

1	13	24	32	45	56	62	76	78	81	90	96	98



1. Find the middle item - use $(n + 1) / 2$ where n is the number of items in the dataset:
middle item = $(1 + 1) / 2 = 1\text{st item}$
2. Compare middle item to target: 76 (target) equals the middle value (76) so end the search - the target has been found!

How efficient is binary search?

It might feel like there are a lot of steps to binary search, but we were able to search a dataset of 13 items with only **four** searches.

That's a lot better than linear search, which would have taken 8 searches on this dataset.

In fact, binary search is **so efficient** that it can search **1 million items** in only **20 searches!**

It could also search 8 billion items (everyone on this planet) in only 33 searches!

Comparing linear and binary search

Characteristic	Linear search	Binary search
Algorithm complexity	Very simple	More complex
Efficiency	Inefficient	Very efficient
Suitable for	Small lists or unordered lists	Large, ordered lists
Data ordering	Dataset does not need to be ordered	Can only search ordered lists (having to order the data first can make it less efficient overall)

Sorting algorithms

Efficient searching needs sorted data

Binary search is brilliant but has a fatal flaw - it only works with sorted data.

So, if we are going to have efficient search we also need an **efficient sorting algorithm!**

A sorting algorithm is a **set of instructions** to **arrange a list of data** into a **particular order** (usually low to high).

Three sorting algorithms

You need to know the details of **three** sorting algorithms:

1. Bubble sort
2. Insertion sort
3. Merge sort

Bubble sort

The Bubble Sort algorithm is **simple** but **slow**. Here are the steps:

1. Start with the first pair of items (first two items) - if the second item is smaller than the first, swap them
2. Move onto the next pair of item two items (i.e items 2 and 3) - if the second item is smaller than the first, swap them
3. Repeat steps 2 until the last pair has been sorted - this is the end of the first **pass**
4. Keep repeating steps 1-3 until **no swaps are made**, the list is now sorted.

Video: Bubble sort visualisation



Video by [AllTrueFalse](#)

Bubble sort example

State the steps involved in a bubble sort on the following items:

	23	10		87		64		12

1. Compare first pair: $23 > 10$ so swap them

Bubble sort example

State the steps involved in a bubble sort on the following items:

	10	23		87		64		12

1. Compare first pair: $23 > 10$ so swap them

Bubble sort example

State the steps involved in a bubble sort on the following items:

10		23	87		64	12

1. Compare first pair: $23 > 10$ so swap them
2. Compare second pair: $23 < 87$ so no swap needed

Bubble sort example

State the steps involved in a bubble sort on the following items:

10	23		87	64		12

1. Compare first pair: $23 > 10$ so swap them
2. Compare second pair: $23 < 87$ so no swap needed
3. Compare third pair: $87 > 64$ so swap them

Bubble sort example

State the steps involved in a bubble sort on the following items:

10	23		64	87		12

1. Compare first pair: $23 > 10$ so swap them
2. Compare second pair: $23 < 87$ so no swap needed
3. Compare third pair: $87 > 64$ so swap them

Bubble sort example

State the steps involved in a bubble sort on the following items:

10	23	64		87	12	

1. Compare first pair: $23 > 10$ so swap them
2. Compare second pair: $23 < 87$ so no swap needed
3. Compare third pair: $87 > 64$ so swap them
4. Compare fourth pair: $87 > 12$ so swap them

Bubble sort example

State the steps involved in a bubble sort on the following items:

10	23	64		12	87	

1. Compare first pair: $23 > 10$ so swap them
2. Compare second pair: $23 < 87$ so no swap needed
3. Compare third pair: $87 > 64$ so swap them
4. Compare fourth pair: $87 > 12$ so swap them

Bubble sort example

State the steps involved in a bubble sort on the following items:

10	23	64	12	87

1. Compare first pair: $23 > 10$ so swap them
2. Compare second pair: $23 < 87$ so no swap needed
3. Compare third pair: $87 > 64$ so swap them
4. Compare fourth pair: $87 > 12$ so swap them
5. First pass is completed
6. Repeat the process until a pass is made with no swaps

Only the **first pass** has been completed at this stage.

The whole process needs to now be repeated until no swaps are made.

Evaluating Bubble Sort

Pros

- ✓ The Bubble Sort algorithm is simple to understand and implement.
- ✓ It is an efficient way to find out if a list is already sorted
 - If the first pass is completed with no swaps then the list must be sorted. This can be determined with only $n-1$ comparisons where n is the size of the list.
- ✓ Little memory is used as only a single value is stored at a time (the value being swapped).

Cons

- X Sorting a list is very inefficient and therefore takes a long time to complete on long lists.

Insertion sort

The Insertion Sort algorithm is a very simple algorithm:

1. Start with the second item in the list
2. Compare the current item to the item to its left - if they are out of order, swap them
3. Continue comparing the current item with the item on its left until no swap is made
4. Move onto the next item (move the 'current item' one along the list)
5. Repeat steps 2 to 4 until the last item has been sorted

Struggling? Try this explanation of insertion sort, it's pretty good:

<http://courses.cs.vt.edu/~csonline/Algorithms/Lessons/InsertionSort/index.html>

Video: Performing an insertion sort



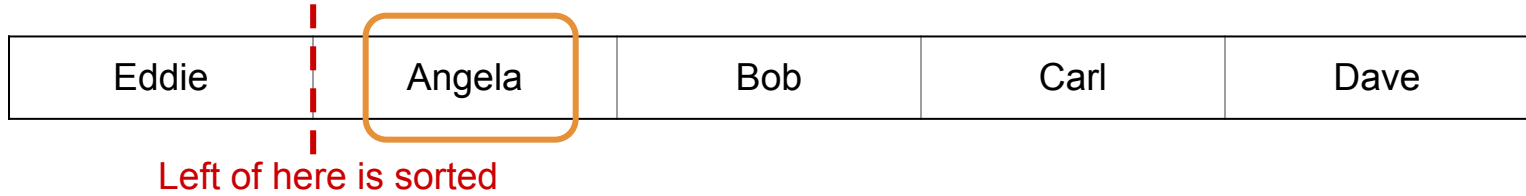
Insertion Sort **in 2**



Video by [Michael Sambol](#)

Insertion sort example

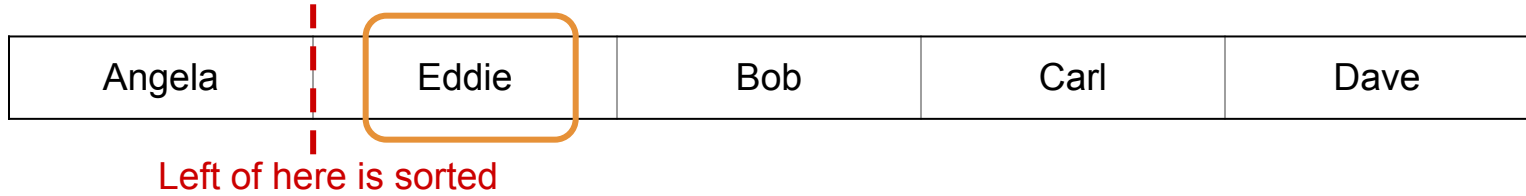
State the steps involved in an insertion sort on the following items:



1. Start with the second item (Angela) and compare it to the item to its left (Eddie)

Insertion sort example

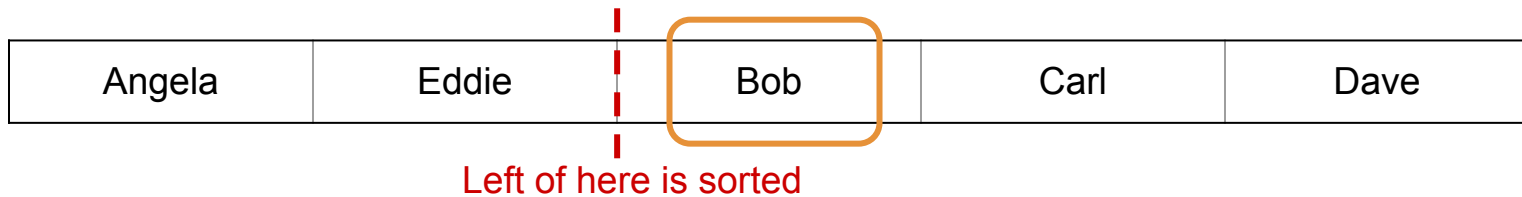
State the steps involved in an insertion sort on the following items:



1. Start with the second item (Angela) and compare it to the item to its left (Eddie)
2. Items are out of order, so swap them

Insertion sort example

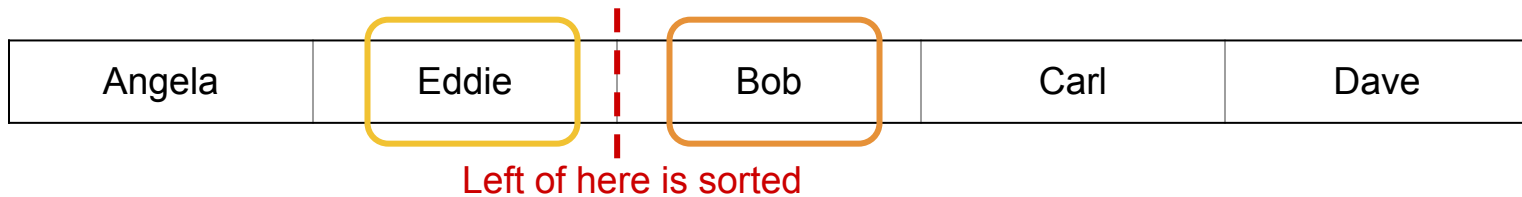
State the steps involved in an insertion sort on the following items:



1. Start with the second item (Angela) and compare it to the item to its left (Eddie)
2. Items are out of order, so swap them
3. Move onto next item

Insertion sort example

State the steps involved in an insertion sort on the following items:



1. Start with the second item (Angela) and compare it to the item to its left (Eddie)
2. Items are out of order, so swap them
3. Move onto next item
4. Compare current item (Bob) to the item to its left (Eddie)

Insertion sort example

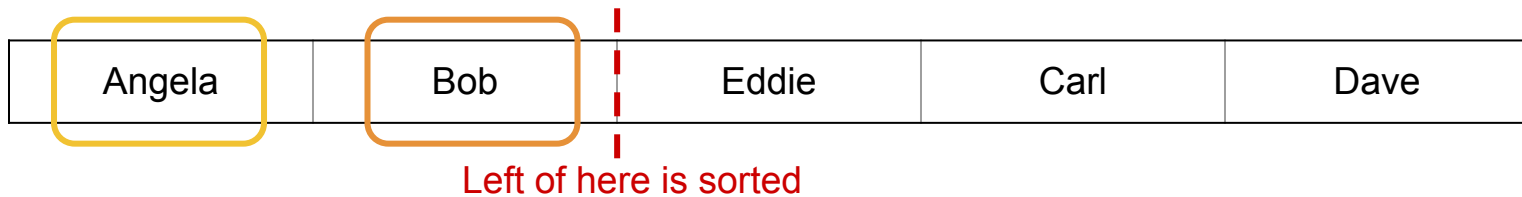
State the steps involved in an insertion sort on the following items:



1. Start with the second item (Angela) and compare it to the item to its left (Eddie)
2. Items are out of order, so swap them
3. Move onto next item
4. Compare current item (Bob) to the item to its left (Eddie)
5. They are out of order, so swap them

Insertion sort example

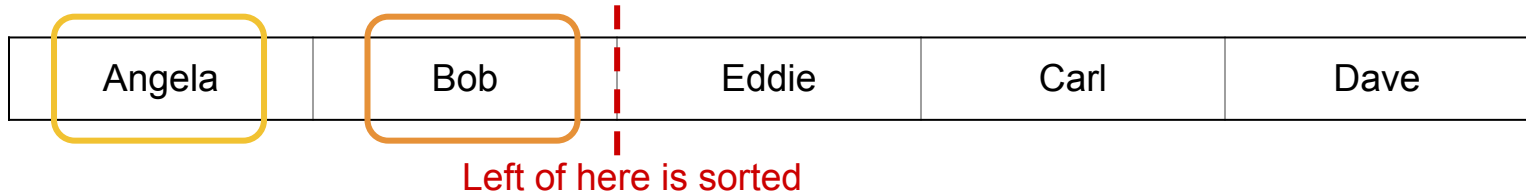
State the steps involved in an insertion sort on the following items:



1. Start with the second item (Angela) and compare it to the item to its left (Eddie)
2. Items are out of order, so swap them
3. Move onto next item
4. Compare current item (Bob) to the item to its left (Eddie)
5. They are out of order, so swap them
6. Compare current item (Bob) to its left (Angela)

Insertion sort example

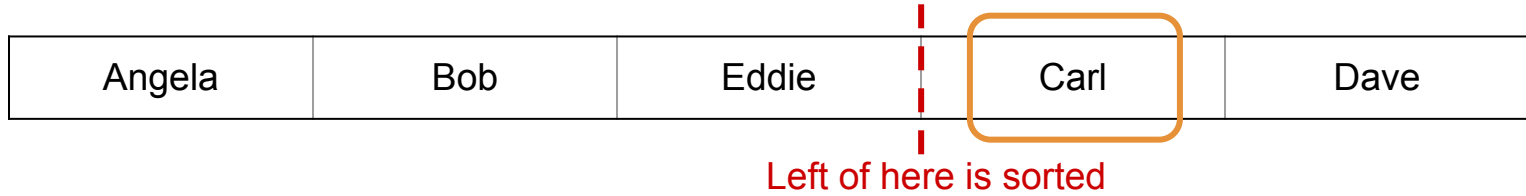
State the steps involved in an insertion sort on the following items:



1. Start with the second item (Angela) and compare it to the item to its left (Eddie)
2. Items are out of order, so swap them
3. Move onto next item
4. Compare current item (Bob) to the item to its left (Eddie)
5. They are out of order, so swap them
6. Compare current item (Bob) to its left (Angela)
7. They are in order, so need to swap further

Insertion sort example

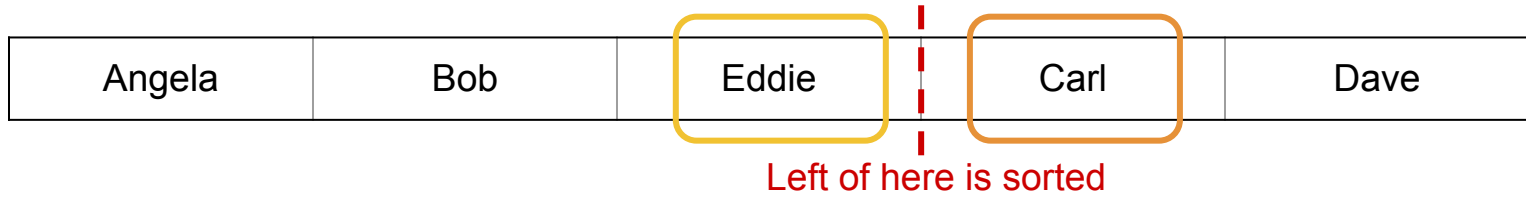
State the steps involved in an insertion sort on the following items:



8. Move onto next item (Carl)

Insertion sort example

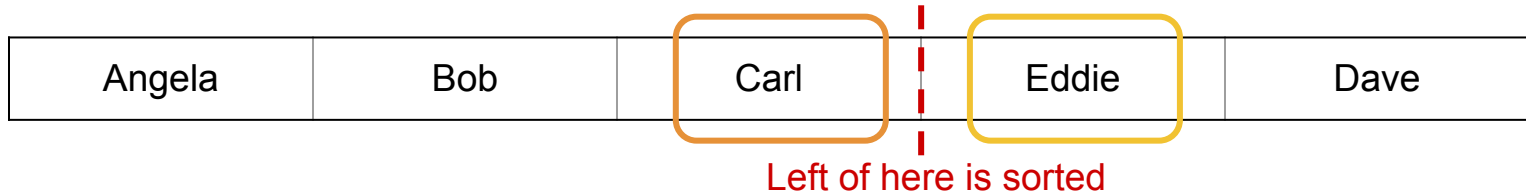
State the steps involved in an insertion sort on the following items:



8. Move onto next item (Carl)
9. Compare current item to item on the left (Eddie)

Insertion sort example

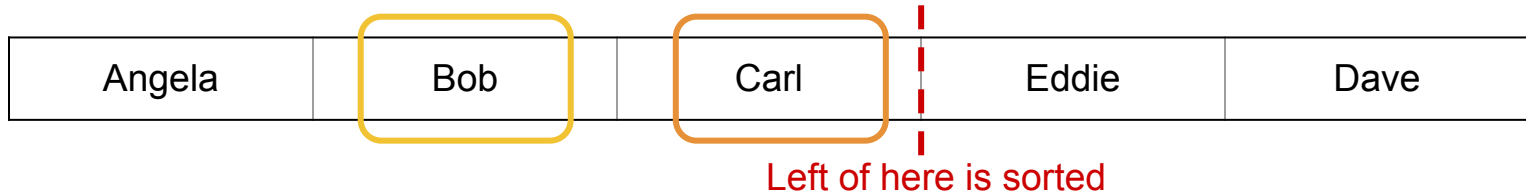
State the steps involved in an insertion sort on the following items:



8. Move onto next item (Carl)
9. Compare current item to item on the left (Eddie)
10. They are out of order, so swap them

Insertion sort example

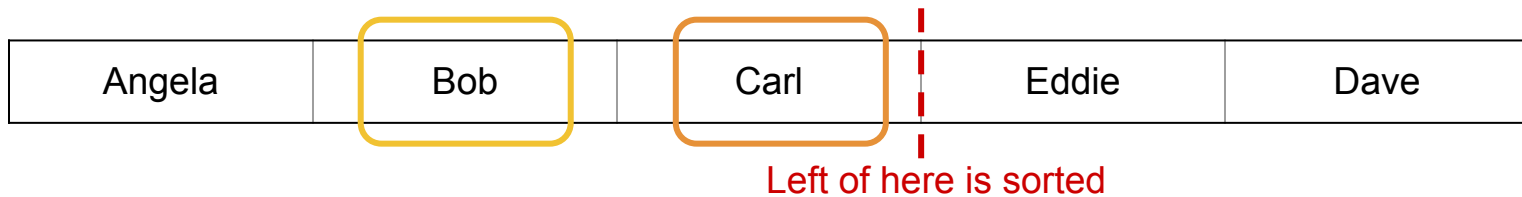
State the steps involved in an insertion sort on the following items:



8. Move onto next item (Carl)
9. Compare current item to item on the left (Eddie)
10. They are out of order, so swap them
11. Compare to next item on the left (Bob)

Insertion sort example

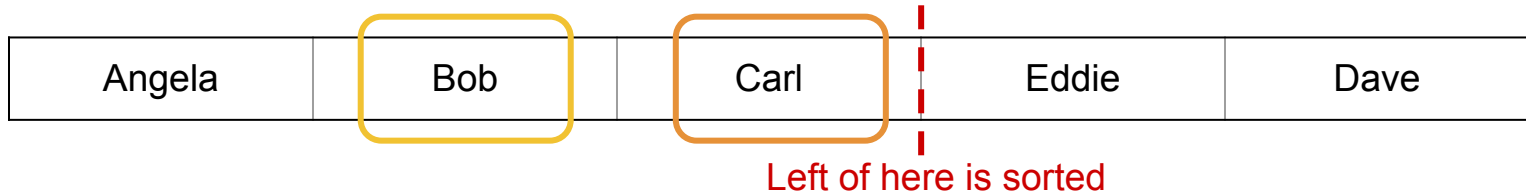
State the steps involved in an insertion sort on the following items:



8. Move onto next item (Carl)
9. Compare current item to item on the left (Eddie)
10. They are out of order, so swap them
11. Compare to next item on the left (Bob)
12. They are in order, so no swap

Insertion sort example

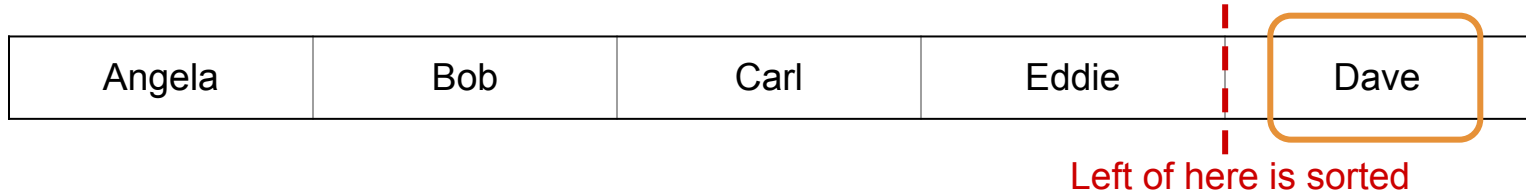
State the steps involved in an insertion sort on the following items:



8. Move onto next item (Carl)
9. Compare current item to item on the left (Eddie)
10. They are out of order, so swap them
11. Compare to next item on the left (Bob)
12. They are in order, so no swap
13. Since no swap was made, no further checks are needed for this item - it must be inserted into the correct place as we know the left side is sorted up to this point

Insertion sort example

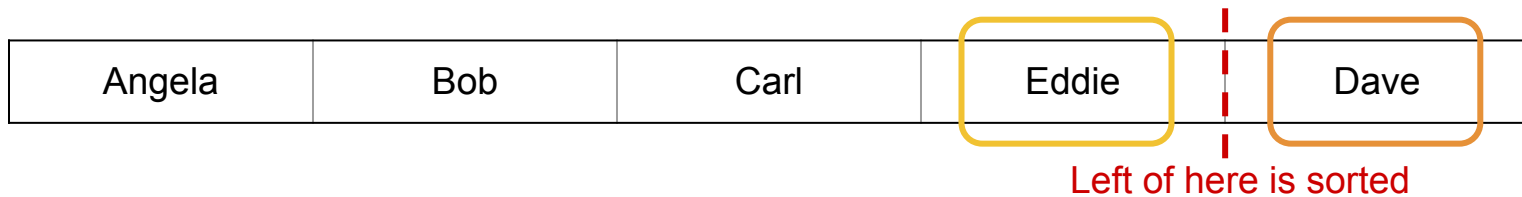
State the steps involved in an insertion sort on the following items:



14. Move onto next item (Dave)

Insertion sort example

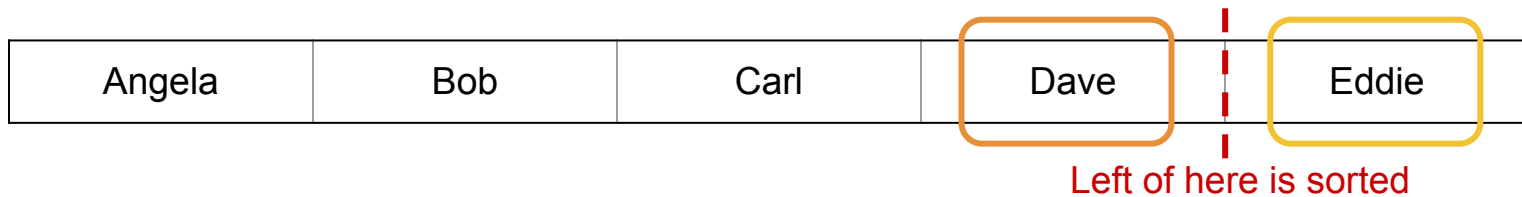
State the steps involved in an insertion sort on the following items:



14. Move onto next item (Dave)
15. Compare current item (Dave) to the left (Eddie)

Insertion sort example

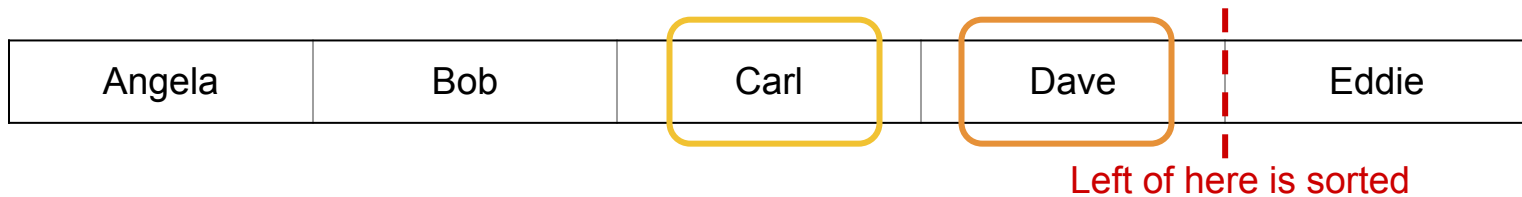
State the steps involved in an insertion sort on the following items:



14. Move onto next item (Dave)
15. Compare current item (Dave) to the left (Eddie)
16. They are out of order, so swap

Insertion sort example

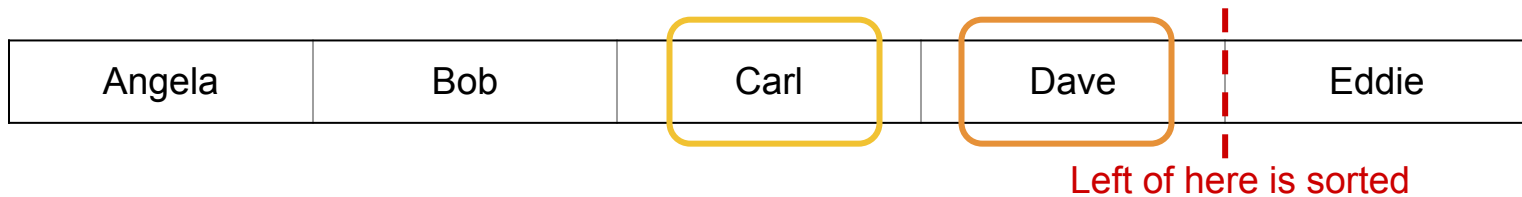
State the steps involved in an insertion sort on the following items:



14. Move onto next item (Dave)
15. Compare current item (Dave) to the left (Eddie)
16. They are out of order, so swap
17. Compare to next item on left (Carl)

Insertion sort example

State the steps involved in an insertion sort on the following items:



14. Move onto next item (Dave)
15. Compare current item (Dave) to the left (Eddie)
16. They are out of order, so swap
17. Compare to next item on left (Carl)
18. They are in order, so no swap is needed.

Insertion sort example

State the steps involved in an insertion sort on the following items:

Angela	Bob	Carl	Dave	Eddie



14. Move onto next item (Dave)
15. Compare current item (Dave) to the left (Eddie)
16. They are out of order, so swap
17. Compare to next item on left (Carl)
18. They are in order, so no swap is needed.
19. No further checks are needed on the left hand side and Dave was the last item in the list to sort so we are finished!

Evaluating Insertion sort

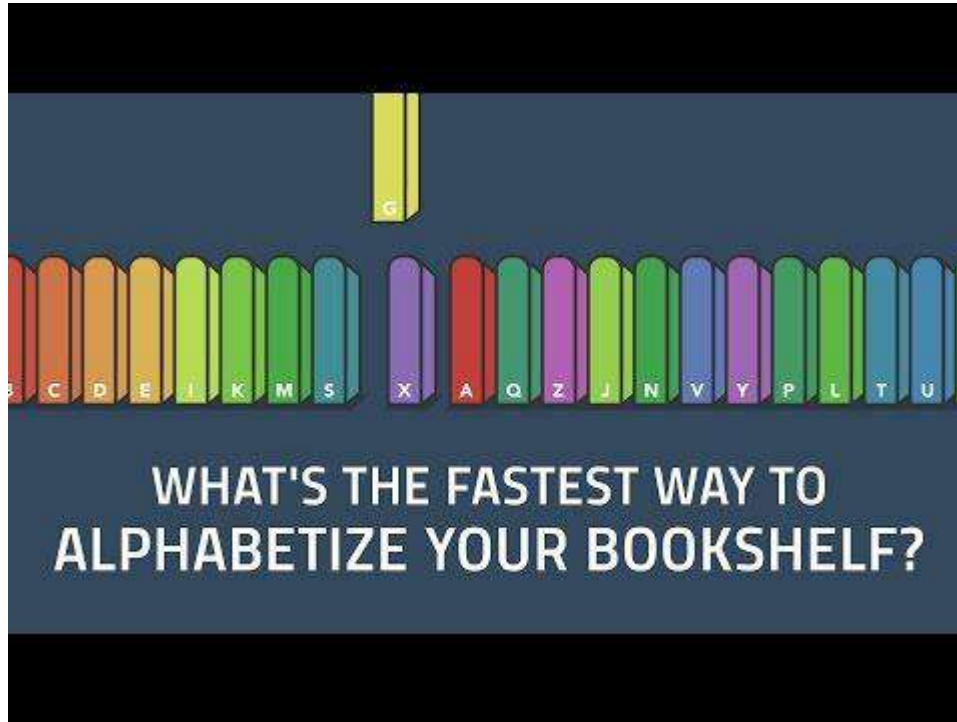
Pros:

- Insertion sort is a simple algorithm that is easy to implement
- Little memory is used as the sorting is done on the original list
- It can quickly check whether a list is already sorted

Cons:

- Insertion sort does not cope well with very large lists - in fact it's no better than Bubble Sort, requiring $n(n-1) / 2$ comparisons in the worst case

Video: comparing bubble and insertion sorts



Video by [TED-Ed](#)

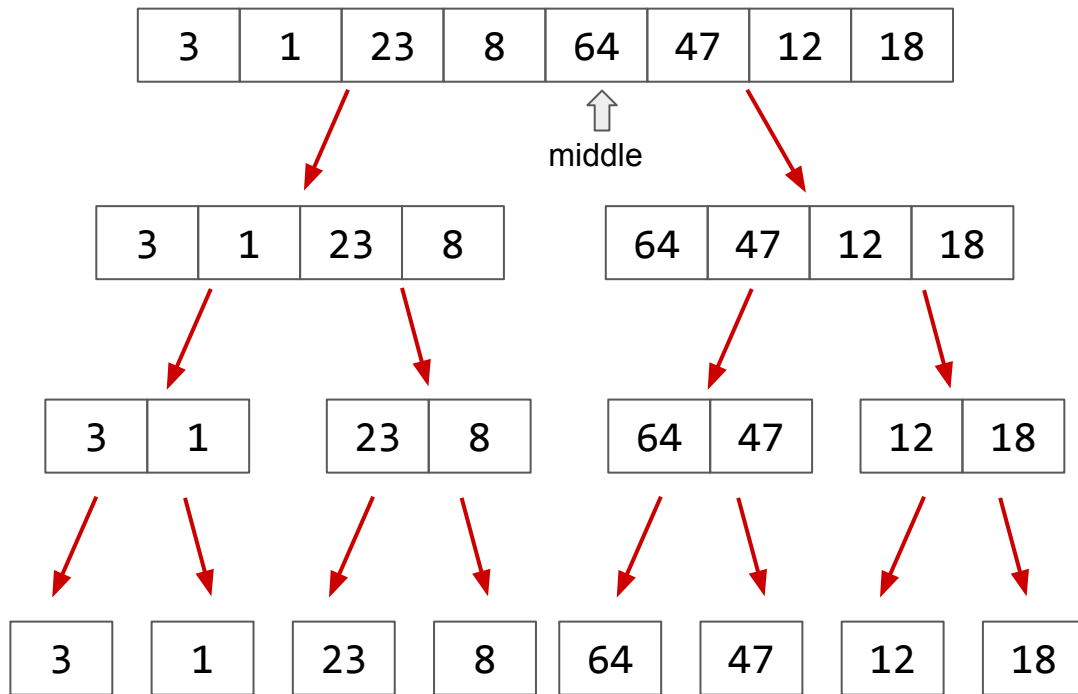
Merge sort

- Merge sort is a **very efficient** sorting algorithm
- It is an example of a "**divide and conquer**" algorithm where the problem is broken down into smaller steps (divided), which are conquered in turn
- It's efficiency is down to the fact that:
 - Small lists can be sorted quicker than long lists
 - Merging ordered lists is easier than unordered lists

Performing a merge sort

Part 1 - Divide the original list

1. Identify the midpoint of the list ($\text{mid} = (n+1) / 2$; round up if necessary)
2. Split the list of items into two sublists - the second list should start at the middle item
3. Repeat steps 1 and 2 on each sublist until all the sublists contain only 1 item

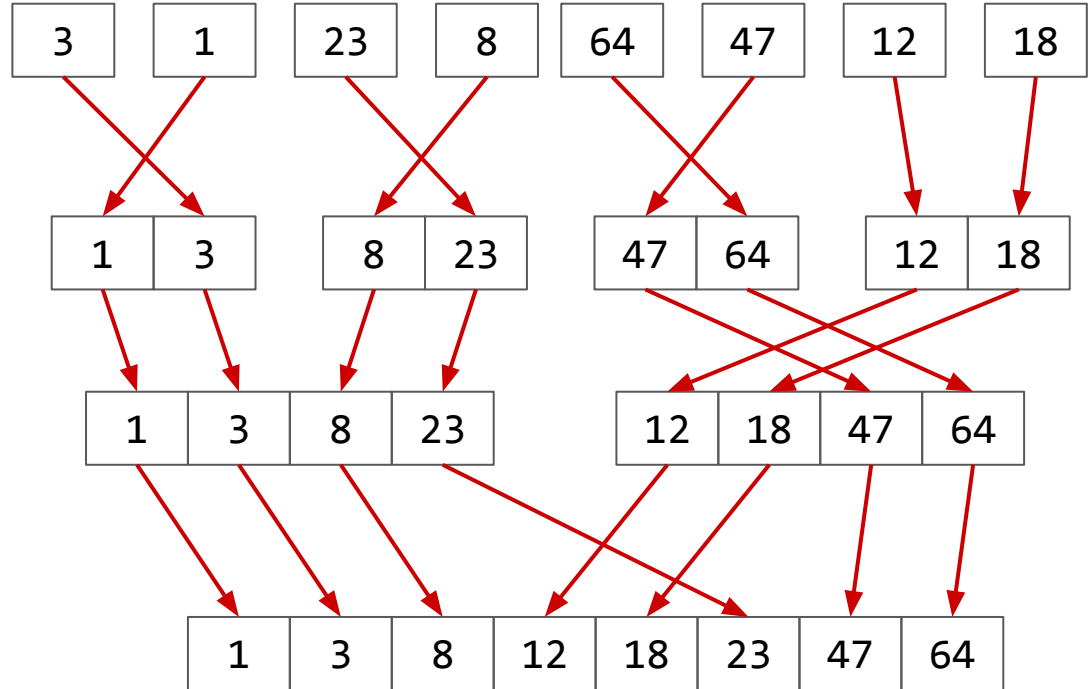


The "cheat" method is to realise that dividing the original list will always result in every item in the list being put into its own single-item list.

Performing a merge sort

Part 2 - Merge the sublists in order

1. Merge each pair of sublists to form a new list by comparing the first value in each list and putting the smallest into the new list first
2. As an item is added to the new merged list, remove it from the front of the original list.
3. Repeat step 1-2 to merge all of the new sublists
4. Continue merging sublists until there is one final, sorted list



Video: performing a merge sort



Merge Sort in 3



Video by [Michael Sambol](#)

Evaluating merge sort

Pros

- Merge sort is very efficient and much quicker than bubble sort and insertion sort for **large lists**
- The number of steps **doesn't increase** very quickly as the dataset grows

Cons

- Merge sort is **slower** than insertion sort for **small lists**
- **Additional memory** is used to store each of the temporary sublists
- The whole algorithm is run even if the list is already sorted

Another example of completing the 'merge'
steps of a merge sort

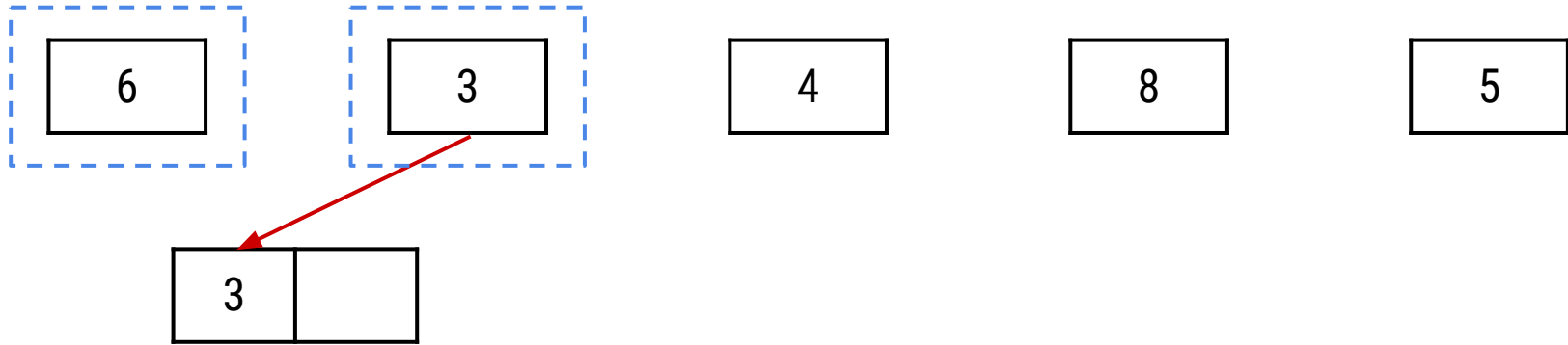
6

3

4

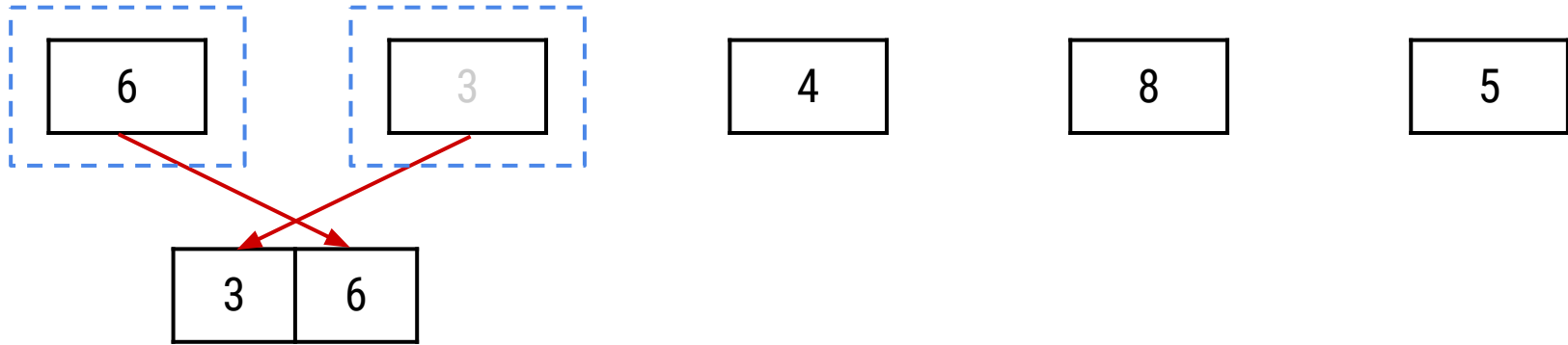
8

5



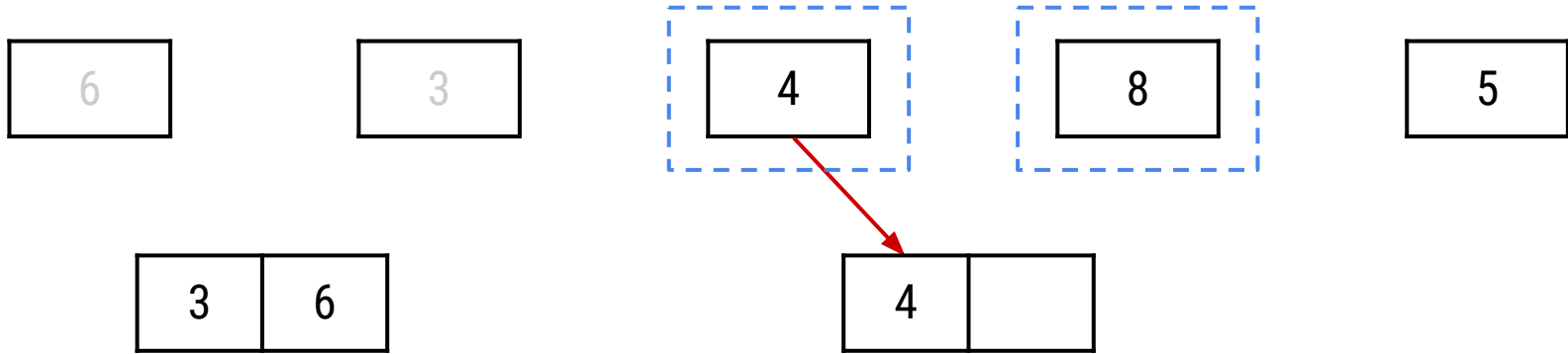
Merging algorithm

- Compare the first item in each of the two lists to merge
- Add the smaller of those items to a new (temporary) list
- Remove that item from its parent list
- If both lists still contain items, repeat the process of comparing the first item in each list
- Once one list is empty, copy over each of the remaining items in the other list.



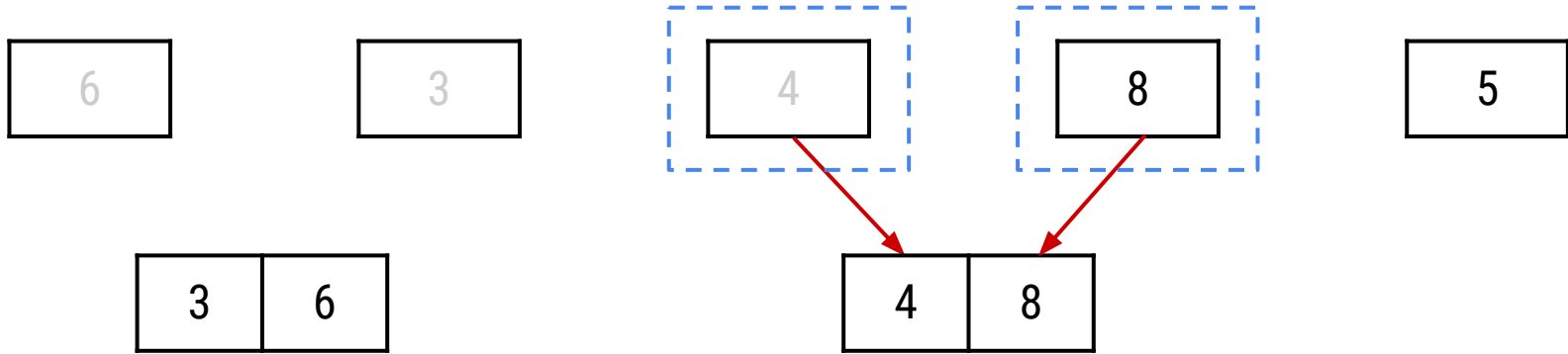
Merging algorithm

- Compare the first item in each of the two lists to merge
- Add the smaller of those items to a new (temporary) list
- Remove that item from its parent list
- If both lists still contain items, repeat the process of comparing the first item in each list
- Once one list is empty, copy over each of the remaining items in the other list.



Merging algorithm

- Compare the first item in each of the two lists to merge
- Add the smaller of those items to a new (temporary) list
- Remove that item from its parent list
- If both lists still contain items, repeat the process of comparing the first item in each list
- Once one list is empty, copy over each of the remaining items in the other list.



Merging algorithm

- Compare the first item in each of the two lists to merge
- Add the smaller of those items to a new (temporary) list
- Remove that item from its parent list
- If both lists still contain items, repeat the process of comparing the first item in each list
- Once one list is empty, copy over each of the remaining items in the other list.

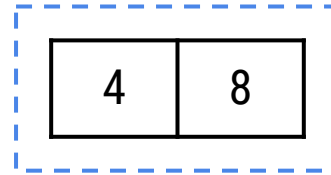
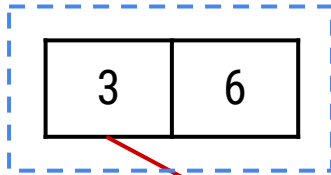
6

3

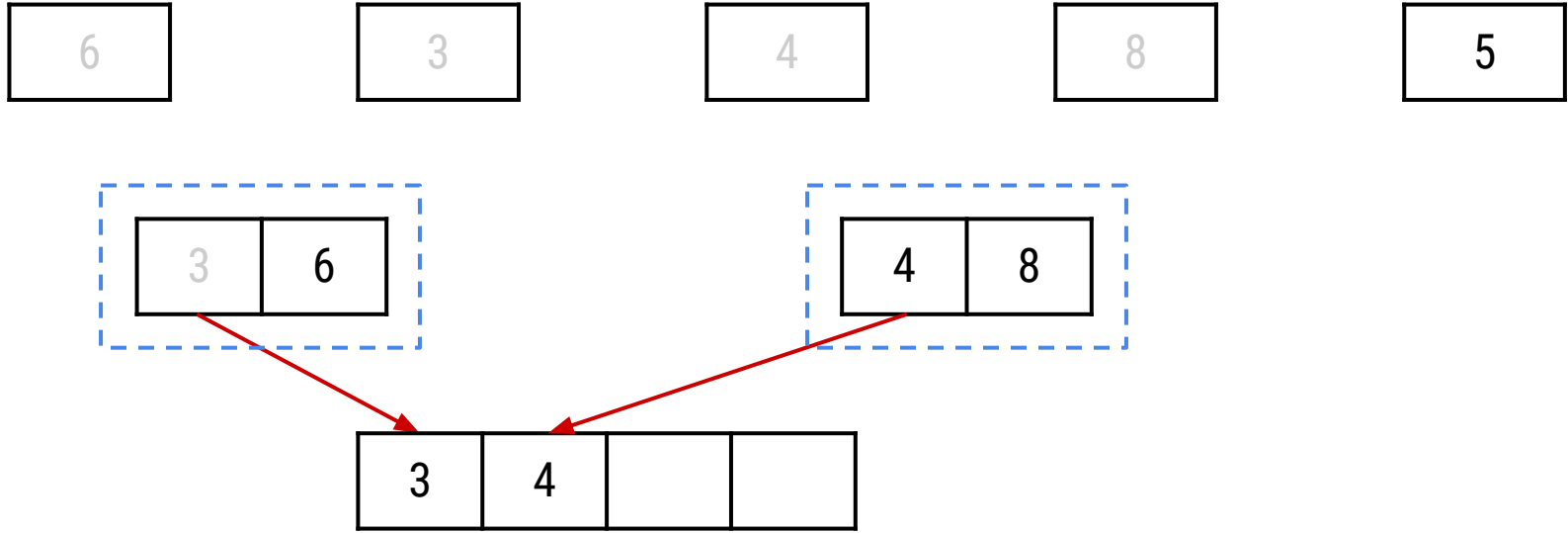
4

8

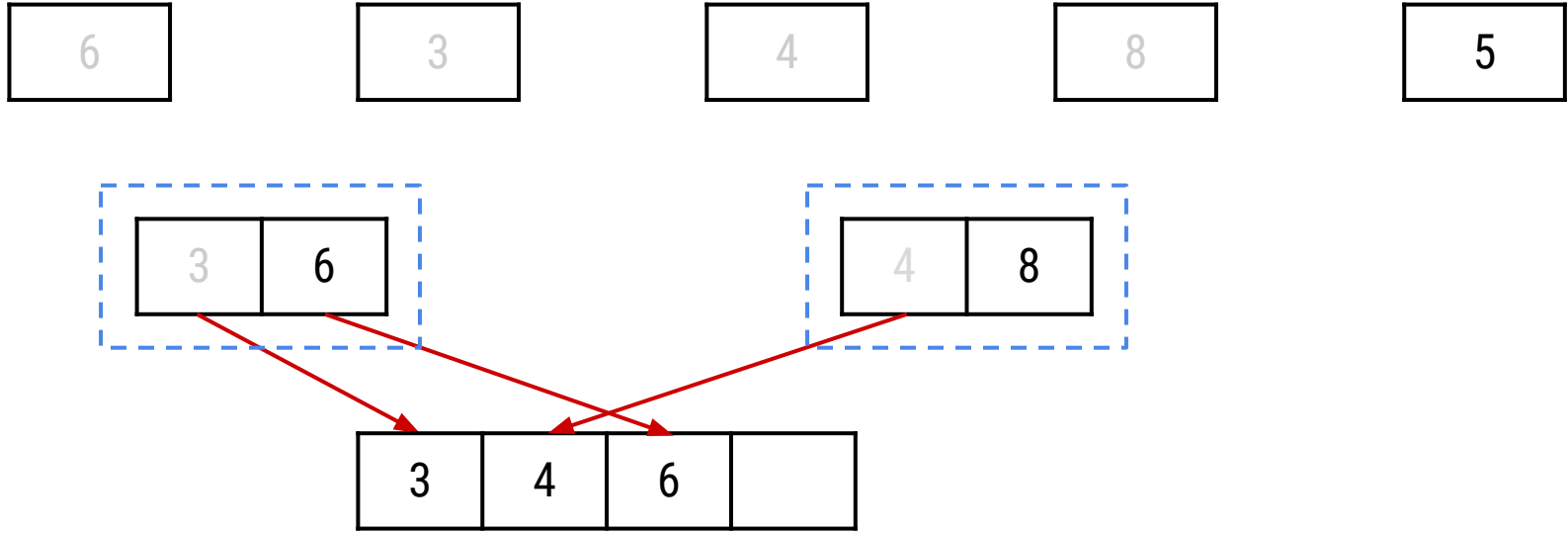
5



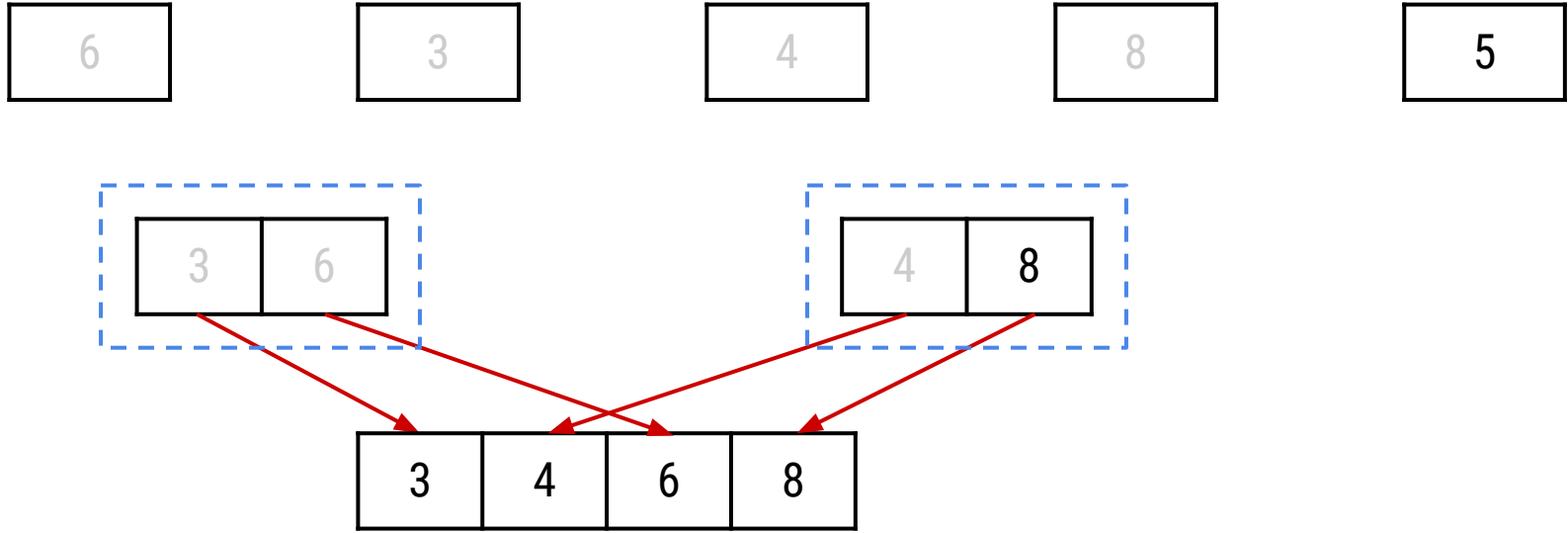
Continue merging, always comparing the **front** items of each list and **removing** the smallest item from the sublist when it added to the merge list.



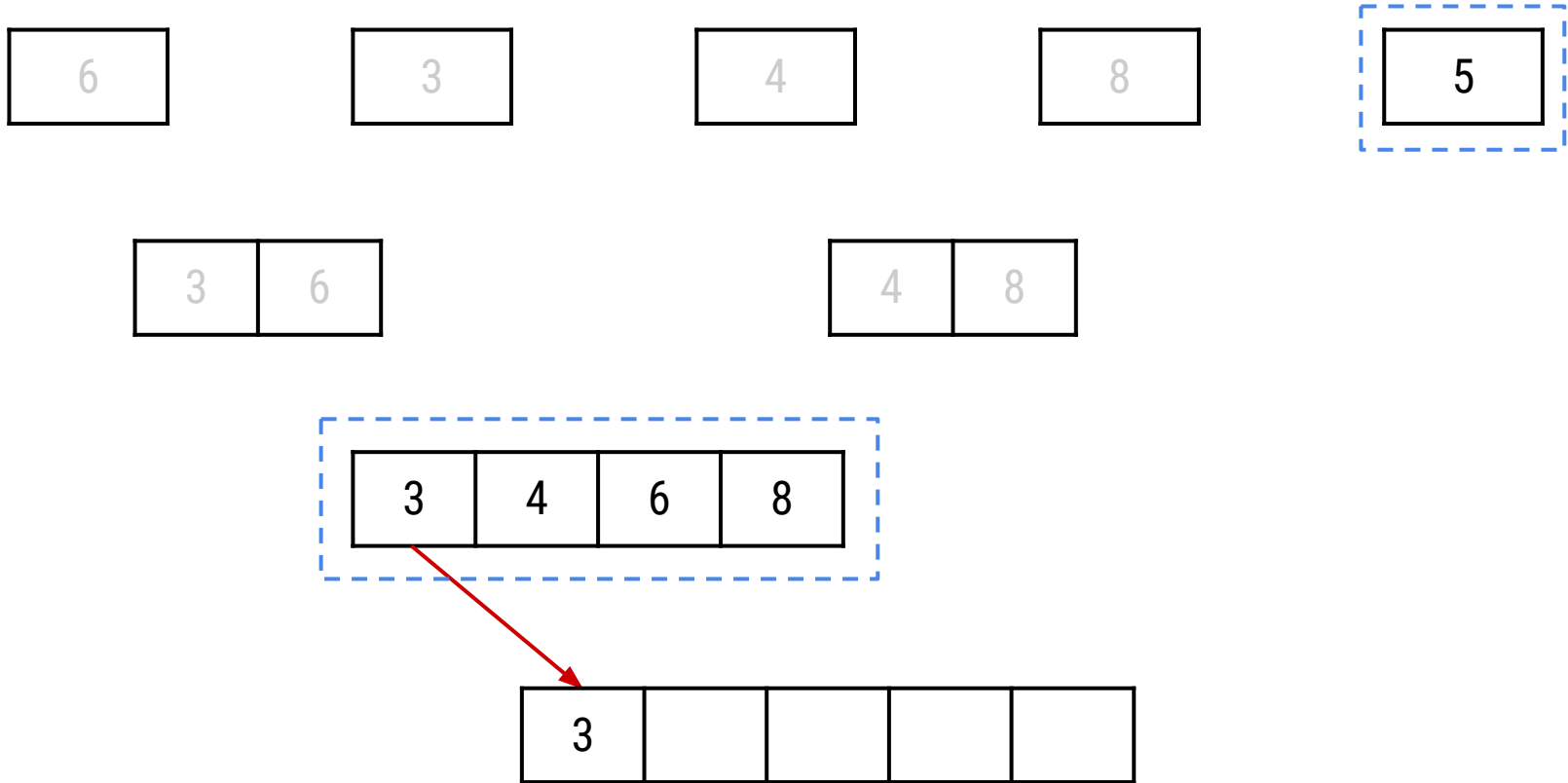
Continue merging, always comparing the **front** items of each list and **removing** the smallest item from the sublist when it added to the merge list.



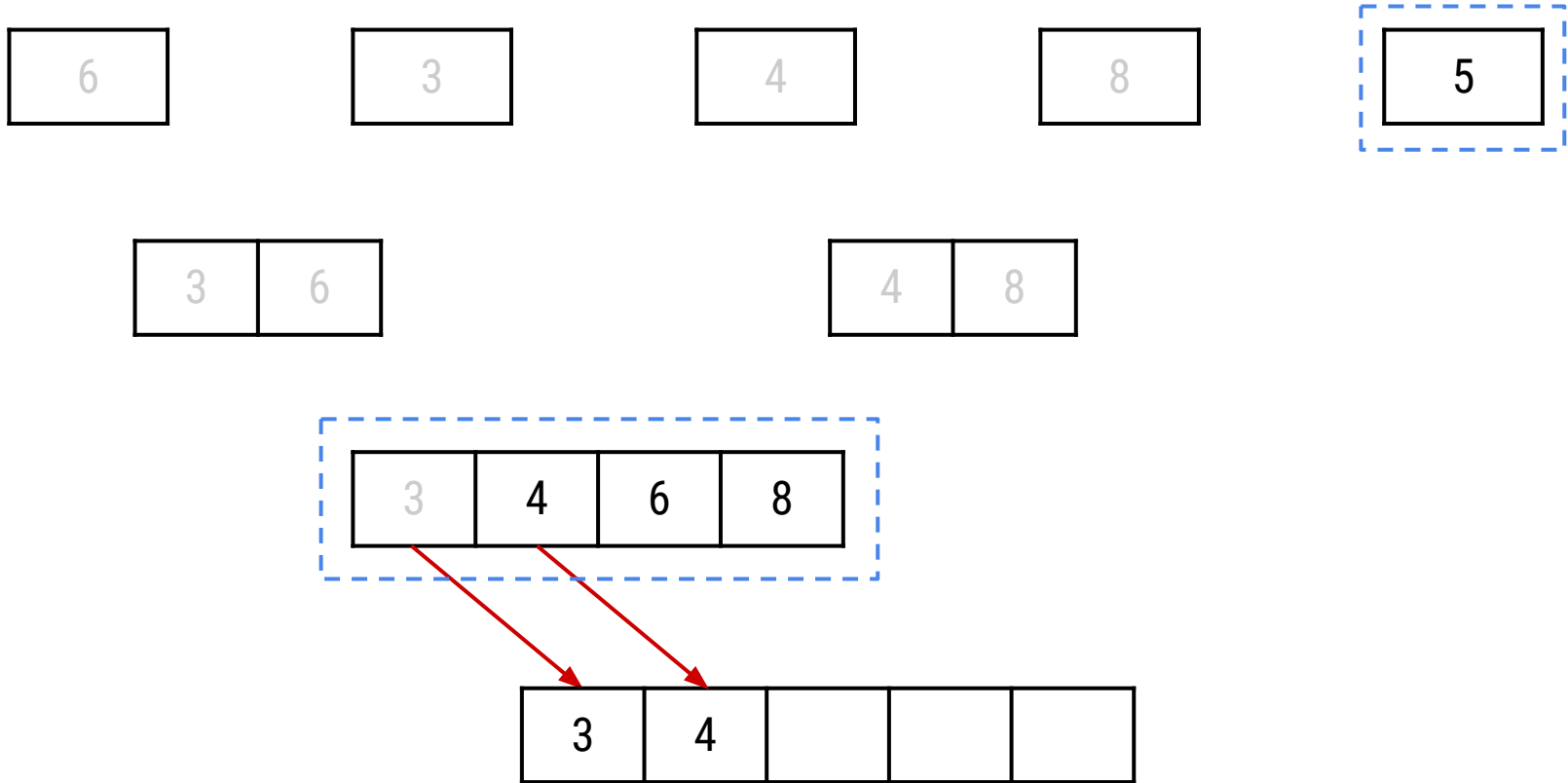
Continue merging, always comparing the **front** items of each list and **removing** the smallest item from the sublist when it added to the merge list.



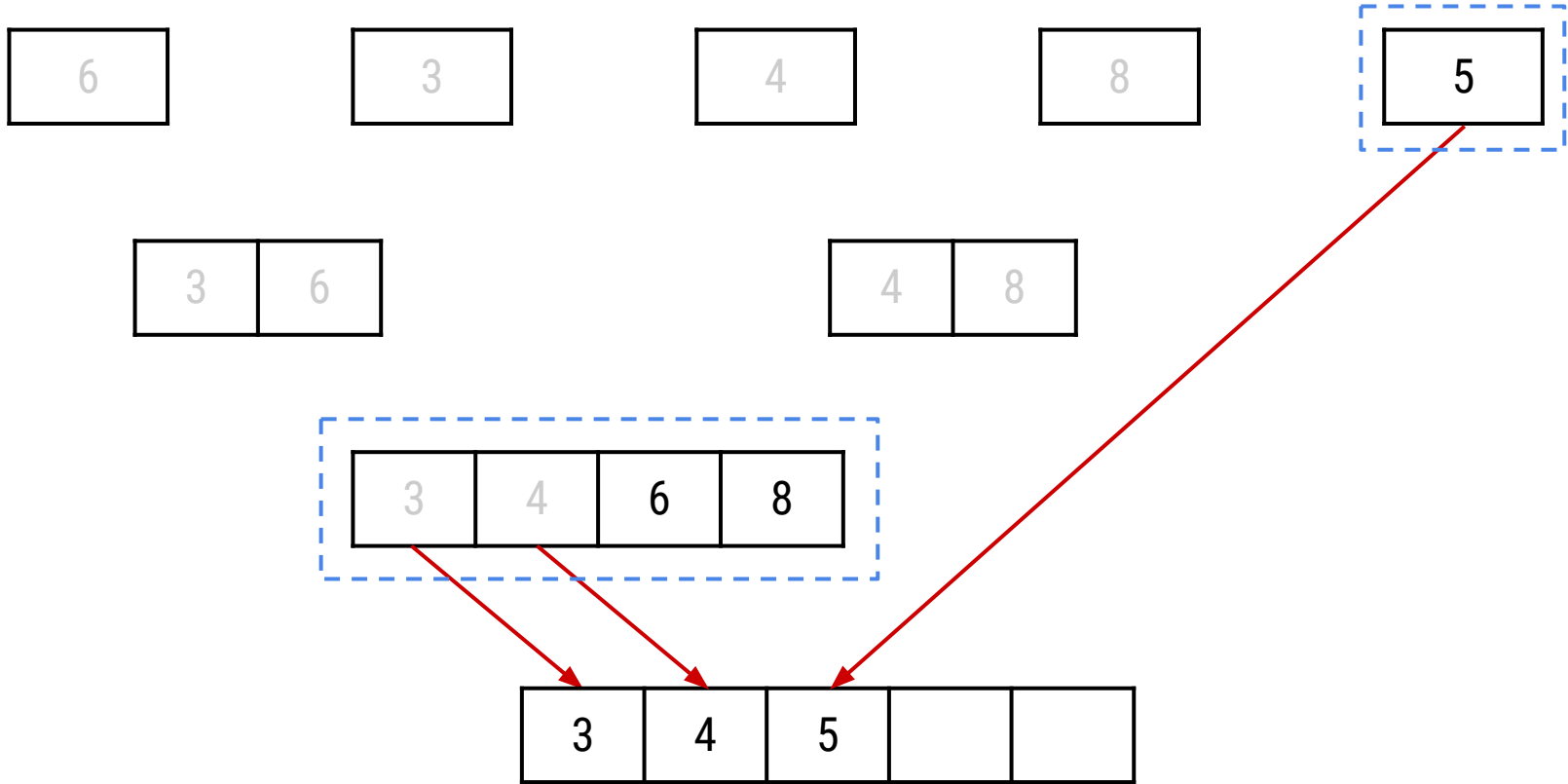
Continue merging, always comparing the **front** items of each list and **removing** the smallest item from the sublist when it added to the merge list.



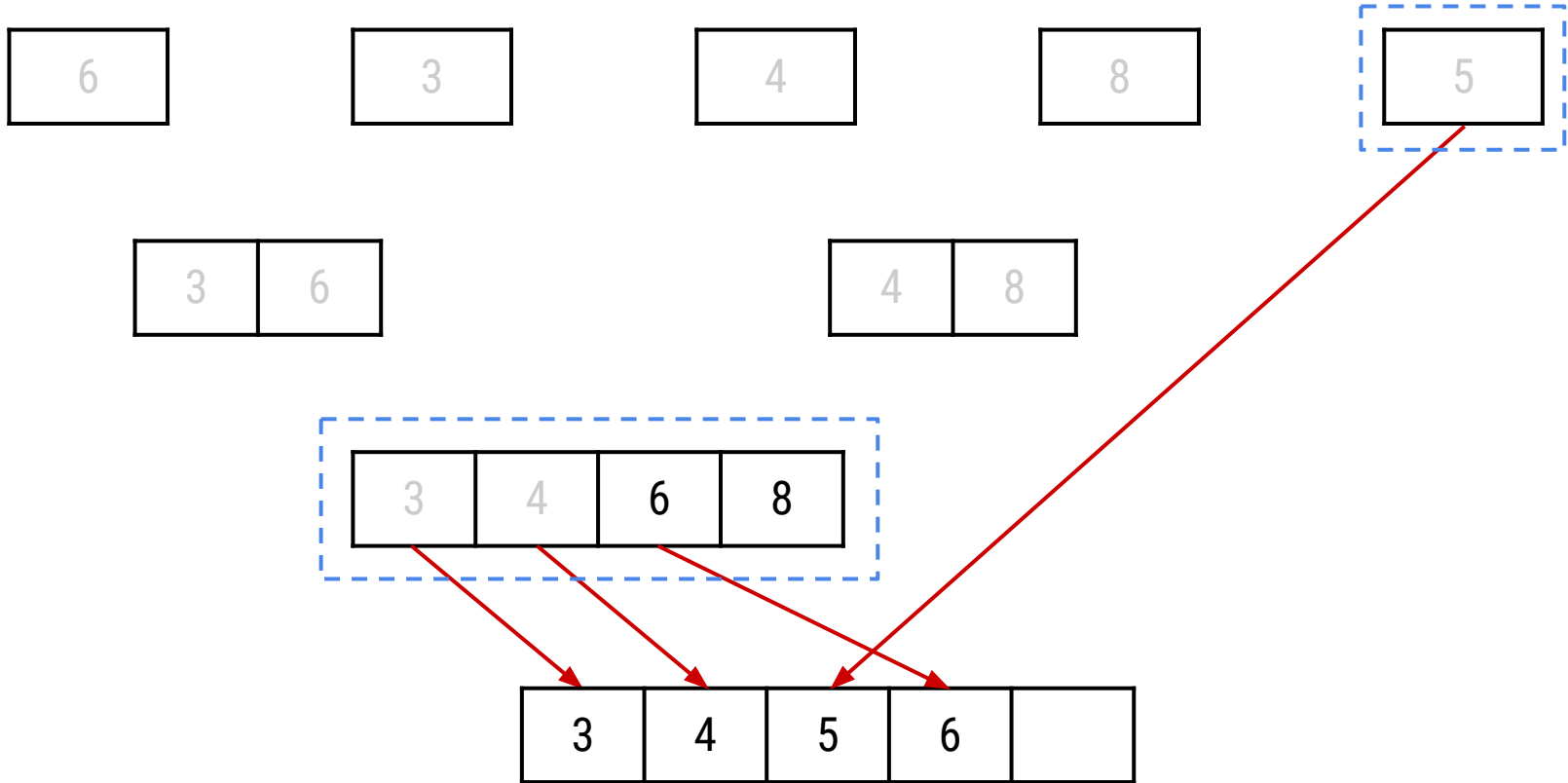
Continue merging, always comparing the **front** items of each list and **removing** the smallest item from the sublist when it added to the merge list.



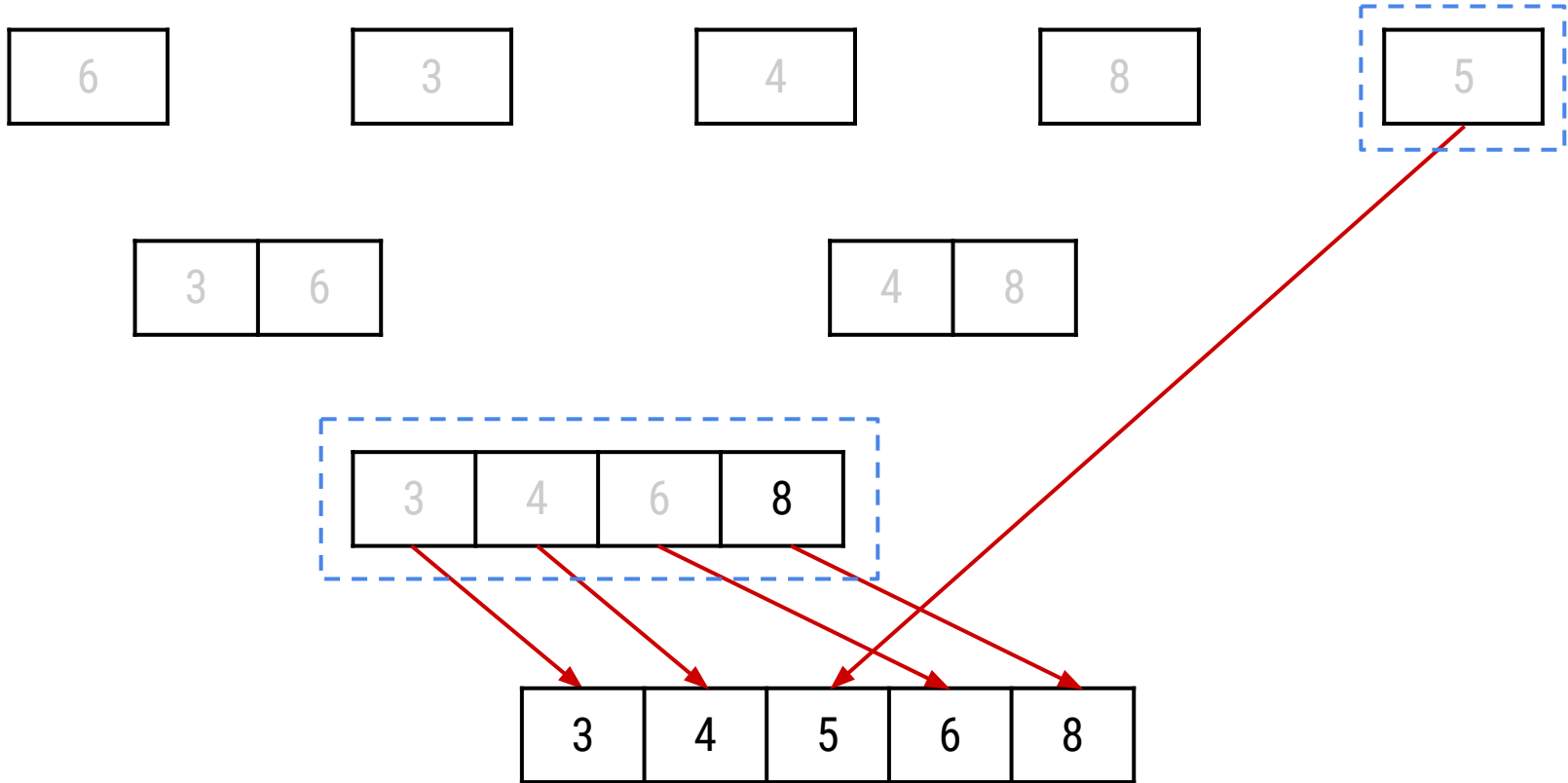
Continue merging, always comparing the **front** items of each list and **removing** the smallest item from the sublist when it added to the merge list.



Continue merging, always comparing the **front** items of each list and **removing** the smallest item from the sublist when it added to the merge list.



Continue merging, always comparing the **front** items of each list and **removing** the smallest item from the sublist when it added to the merge list.



Continue merging, always comparing the **front** items of each list and **removing** the smallest item from the sublist when it added to the merge list.

Comparing the efficiency of each algorithm

Click the link to see a visual comparison of several different sorting algorithms (yes, there are more than just bubble, insertion and merge sort!)

<https://www.toptal.com/developers/sorting-algorithms>

PROGRAMMING TECHNIQUES

CHAPTER 10

Programming fundamentals

Variables and constants

- Variables are **named locations in memory** that are **used to store temporary values that can change** as the program is run
- Constants are **named locations in memory** that are used to store **values that do not change** throughout the program's execution
- The value stored by variable or constant can be set or recalled by using the **identifier** (name) of the variable
- The process of setting a value to a variable or constant is called **assignment**

Variables and constants

Python 3

Variable or
constant?

Variable or
constant?

```
PI = 3.14159
```

```
radius = input("Enter the  
radius of a circle: ")
```

```
area_of_circle = PI * radius *  
radius
```

```
print("The area of the circle  
is:", area_of_circle)
```

Variables and constants

Python 3

Constant!

Variable!

```
PI = 3.14159
```

```
radius = input("Enter the  
radius of a circle: ")
```

```
area_of_circle = PI * radius *  
radius
```

```
print("The area of the circle  
is:", area_of_circle)
```

Variable declaration

- In some programming languages, variables need to be **declared** before they can be used to store data.
- Declaring a variable means to **state its name** (identifier) and the **type of data** that it will contain, for example

```
var player_score Int
```

- Python doesn't require that variables are explicitly declared before they are used. Instead, they are **automatically declared upon first use**. Python also doesn't require that the type of data being stored is stated, it simply **infers the data's type** from the data that is assigned to (stored in) the variable.

Choosing a suitable identifier

- It is important that variable identifiers (names) are **clearly descriptive of the data** that they hold.
- For example, `player_name` is clearly going to be used to store the name of the player in a game. It would be perfectly possible to write `x = "Mary"`, but it wouldn't be at all clear what the purpose of variable `x` is.
- Using sensible, clear and descriptive variable identifiers helps to ensure that code is **easily understood** by yourself and other developers and will **reduce the need for documentation** and the **likelihood of making errors**. If errors do appear, the code will be easier to understand and therefore the source of errors will be easier to **identify and fix**.

Rules for choosing good identifiers

1. Identifiers **cannot include spaces**. Underscores or "CamelCase" can be used where an identifier is to be made from two or more words.
2. Identifiers can contain **letters, numbers and underscore symbols (_) only**. However, in Python, variable identifiers cannot start with a number but can contain a number anywhere else in the identifier.
3. Certain **reserved words cannot be used as identifiers** if they are existing built-in commands, statements or functions. For example, you cannot use 'if' as a variable identifier.
4. Identifiers are **case sensitive**. For example, myAge and myage are two different identifiers and cannot be used interchangeably. Forgetting capital letters in identifiers is a very common cause of syntax error!
5. Identifiers should be **short**, but not so short as to be meaningless or unclear.

Assignment

Assignment simply means giving a variable a value. Rather than "setting" a variable to a value or "putting a value in the variable", the technical term is to "assign a value to a variable."

Python 3

The following is an example of how to declare a variable and assign a value to it in Python:

```
player_name = "Bob"
```

Note the format - the variable's identifier (name) always goes on the left, followed by a single = (a single equals sign is called the assignment operator) and then the data to be stored.

Using a variable's value

To access a variable's data, simply use the identifier of the variable wherever its data is required.

Python 3

The following is an example of how to access the value of a variable via its identifier in Python:

```
print("Hello " + player_name)
```

Assigning a new value to a variable

Python 3

It is easy to assign new data to a variable. Simple state the variable's identifier, followed by the assignment operator (=) and the value to be assigned, just as when assigning the variable's initial value.

The following example shows a new value being assigned to the player_name variable and then accessing this new value from the variable.

```
player_name = "Mary"  
print("Hello " + player_name)
```

Constants

- Constants are very much like variables, except that their assigned values **do not change** during the execution (running) of a program.
- Constants are very useful for representing values that you don't want to have to re-type throughout the program, either because they are awkward or because you would have to replace many instances of a fixed value if you were to change this value during your design of the program.
- In some programming languages, constants are declared using a specific keyword such as `let` or `const`. In Python, there is no specific declaration for a constant, so CAPITAL IDENTIFIERS are used to help identify constants within the program.

Examples of using constants

Python 3

In the example Python program to the right, `PI` has been declared as a constant value at the start of the program.

`PI` can now be used in place of the actual value of pi wherever it is needed - namely within the two subroutines `circle_area()` and `circle_circumference()`.

```
PI = 3.14159265

def circle_area(radius):
    return PI * radius * radius

def circle_circumference(radius):
    return 2 * PI * radius
```

Examples of using constants

Python 3

In this example, a constant is used for defining the maximum number items that can be stored in a database.

This is a value that won't change during a program's execution but may need to be referred to in several places.

During the development of the program, the developer might want to alter this value so defining it as a constant means that they can do this in one place and have the change reflected throughout their program.

```
MAX_ITEMS = 5000

items_list = []

def add_item(item):

    if len(items_list) > MAX_ITEMS:
        print("Sorry! Maximum number of
              items exceeded!")

    else:
        items_list.append(item)
```

Inputs and outputs in Python

Python 3

- Data is output to the screen using the **print()** function
- Data is collected via the keyboard by the **input()** function
- The input() function **returns** the entered data which must be **assigned** to a variable if it is going to be used in the program
- In pseudocode, OUTPUT() is sometimes used in place of print().

```
print("Welcome user!")

name = input("What is your
name? ")

print("It's great to meet
you", name)
```


Inputs and outputs in Python

Python 3

The value returned by `input()` is assigned to the 'name' variable

The value assigned to 'name' can be accessed and included in this output message

```
print("Welcome user!")
```

```
name = input("What is your  
name? ")
```

```
print("It's great to meet  
you", name)
```

Outputting special characters (new lines and tabs)

Python 3

A **special character** is one that affects the output of String, such as adding a new line within the string, or tabbing (indenting).

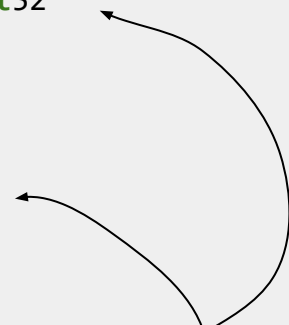
Special characters (also called 'escaped characters') start with a **backslash** (\), for example:

- \n - newline - puts what follows on a new line
- \t - tab - adds a 'tab' (spacing) within the string

```
print("Hello\nthere!")
>> Hello
    there!

results_message = "Results\n-----\nPerson
1:\t32\t33\nPerson 2:\t48\t52"

print(results_message)
>> Results
-----
Person 1:    32    33
Person 2:    48    52
```



The actual string might look messy and hard to read, but the printed output is much nicer!

Numerical inputs

The `input()` function always returns data as **string**, because a keyboard can only provide text data.

If we want to use the input as a number for a mathematical or relational operation, we first need to convert it from a string into a numerical data type, such as an integer, using a method called **casting**.

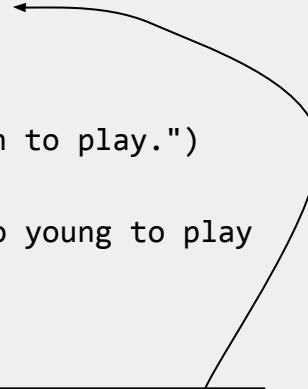
We will look at this in more detail when we look at data types, but for now here is a quick example.

Python 3

```
player_age = input("Please enter your age: ")

player_age = int(player_age)

if player_age >= 10:
    print("You are old enough to play.")
else:
    print("Sorry, you are too young to play
          this game.")
```



int() is a built-in function for converting string data into integer data.

Comments (# and """)

Python 3

Comments are added to a program to write a message for other people to see but that will be **ignored** by the Python interpreter.

This is very useful when developing a program with other people to **explain how a part of it works**.

It is also useful to **leave notes and reminders for yourself** as you work on a more complex program.

```
# Use a hash symbol to create a  
single-line comment
```

```
print("This is not a comment, it will  
be printed")
```

```
"""
```

```
You can use three quotation marks to  
write a multi-line comment.
```

```
You finish the comment with three more  
quotation marks.
```

```
"""
```

Operators

Operators

- Operators are **symbols and statements** that **perform specific functions** within a program. We have already come across one operator - the assignment operator (=) which is used to assign a value to a variable.
- The values used either side of an operator are called **operands**. So, in the simple mathematical expression $1 + 2$, + is the operator and 1 and 2 are the operands.
- There are different types of operator:
 - Arithmetic (mathematical) operators
 - Relational operators
 - Boolean (logical) operators

Arithmetic (mathematical) operators

Arithmetic operators are used to perform mathematical calculations.

The main arithmetic operators are:

Operator (pseudocode)	Operator (Python)	Purpose	Example
+	+	addition	$2 + 2 = 4$
-	-	subtraction	$6 - 4 = 2$
/	/	division	$7 / 2 = 3.5$
*	*	multiplication	$2 * 5 = 10$
^	**	exponentiation (raising to the power of)	$3 ^ 2 = 9$

MOD and DIV operators

There are two other mathematical operators that you need to know for the OCR GCSE course - MOD and DIV.

MOD

The MOD operator performs a "modulo" operation which finds the **remainder of a division of two numbers**.

For example, 5 divided by 2 is 2.5. If we were to think of this as fractions we could say:

$$\frac{5}{2} = 2 \frac{1}{2}$$

Therefore 5 MOD 2 is 1 - because the remainder when 5 is divided by 2 is 1 (2 goes into 5 twice, 2 * 2 is four, leaving a remainder of 1 unit).

There is no MOD keyword in Python, instead the % symbol is used instead:

Python 3

```
mod_calculation = 5 % 2
print(mod_calculation)
```

Uses of MOD - testing for odd and even numbers

Python 3

MOD can be useful for determining whether a number is odd or even.

Every even number will return no remainder when divided by 2 whereas every odd number leaves a remainder of 1.

```
def test_if_even(number):  
    if number % 2 == 0:  
        return True  
    else:  
        return False  
  
print(test_if_even(6))  
>> True  
  
print(test_if_even(7))  
>> False
```

Uses of MOD - pagination

MOD can be used to check whether a number is multiple of any other number.

This might be useful if we wanted to pause an output every 10 lines we could do the following (this is called 'pagination' - splitting up a long output into 'pages').

Python 3

```
line_number = 0

while line_number < 30:

    line_number = line_number + 1
    print("Hello!")

    if line_number % 10 == 0:
        input("\nPress Enter to continue...")
```

DIV

The DIV operator performs a special type of division called Integer division or Floor division.

The result of a DIV operation is the **whole part** (quotient) of the result of dividing one number by another. For example, $7 \text{ DIV } 2 = 3$ whereas $7/2 = 3.5$

Note that DIV does not round to the nearest whole number. If anything, it's a rounding down operation but it's easier (and more accurate) to think of DIV as a division where anything after the decimal point has been sliced off and lost.

As with MOD, Python doesn't have DIV operator, instead it uses two forward slashes - `//` - to perform this operation:

Python 3

```
div_example = 18 // 4 # This will return 4 as 18/4 is 4.5 - the whole number part of this is 4.
print(div_example)
>> 4
```

Relational operators

Relational operators are used to determine the relationship between two values, such as whether one is bigger or smaller than another or whether they are both equal.

Operator (pseudocode and Python)	Purpose	Examples
>	Greater than	$5 > 2 = \text{True}$, $5 > 5 = \text{False}$
>=	Greater than or equal to	$5 >= 5 = \text{True}$, $4 >= 5 = \text{False}$
<	Less than	$3 < 10 = \text{True}$, $40 < 18 = \text{False}$
<=	Less than or equal to	$6 <= 6 = \text{True}$, $7 <= 6 = \text{False}$
==	Equal to (note the double equals - that's because a single equals is used to assign values to variables)	$2 == 2 = \text{True}$, $2 == 3 = \text{False}$
!=	Not Equal to	$2 != 3 = \text{True}$, $2 != 2 = \text{False}$

Relational operators

Python 3

The result of all relational operators is either **True** or **False** (note the capital T and F!) - these are called "Boolean values", named after the famous mathematician and logician George Boole.

Because relational operators result in True or False values, they can be used in selection statements such as if/else.

```
num1 = 4
num2 = 9

if num1 < num2:
    print("num1 is smaller than num2")
else:
    print("num1 is bigger than num2")

num3 = 4
num4 = 4

if num3 == num4:
    print("num3 is the same as num4")
else:
    print("num3 and num4 are different")
```

Boolean (logic) operators

Boolean operators perform **logic operations** and always result in **True** and **False** values. The Boolean operators for GCSE are:

- **AND** - evaluates to True if both operands are True; False if either operand is False
- **OR** - evaluates to True if either of the two operands are True; False if both operands are False
- **NOT** - inverts / negates a logical value so True becomes False and False becomes True

In Python, the Boolean operators are written in lowercase letters - and, or, not.

Using Boolean operators with selection statements

Pseudocode

You can test multiple conditions within an IF/ELSE structure by using **Boolean operators**:

- AND
- OR
- NOT

Remember that Boolean operators always return a value that is either **True** or **False**.

```
INPUT username
INPUT password
IF username == "Bob" AND password == "123"
THEN
    logged_in = True
ELSE
    logged_in = False
END IF
```


Using Boolean operators with selection statements

Python 3

Boolean operators are used with selection statements to form complex "conditions" that must be evaluated before the program can proceed.

Let's use the example of boarding a flight where the traveller must have both their passport and boarding pass:

```
got_passport = True
got_boarding_pass = False

if got_passport == True and got_boarding_pass:
    print("You're all set to fly!")
else:
    print("Sorry, you need both your passport and
          boarding pass to fly.")
```

Forming more complex conditions with Boolean operators

Python 3

Boolean expressions can be combined by wrapping expressions in brackets - (). The whole expression in the brackets evaluates to either True or False.

Let's pretend you are allowed to go out but only if you've done your homework and tidied your room.

If you haven't got any homework then it doesn't matter whether you've done it or not.

```
got_homework = True
done_homework = False
tidied_room = True

if tidied_room == True and (got_homework == False or
                             (got_homework == True and done_homework == True)):
    print("You can go out!")
else:
    print("Too bad - you've got work to do!")
```

Generating random numbers

Python 3

Whilst not strictly a mathematical operation, it is very useful to be able to generate a random number within a program.

In pseudocode, a statement to get a random number between a minimum and maximum value could be represented as:

```
random_no = get_random(min, max)
```

For example,

```
dice_value = get_random(1,6)
```

Generating a random integer between 1 and 6 in Python

```
import random
```

```
dice_value = random.randint(1,6)
```

```
if dice_value == 6:  
    print("You rolled a 6! You win.")  
else:  
    print("You lose :(")
```

Controlling program flow

What is program control?

Program control determines what "path" or "flow" will be taken through an algorithm.

The simplest algorithms will be a series of steps that follow one after the other every time it is run (sequence).

More complex algorithms will have parts that repeat (iteration) or parts that only happen if certain conditions are met (selection).

Larger processes might be broken into smaller parts (subroutines) that each have their own algorithms to define what they do. These can be run (called) from the "master" process as and when they are needed.

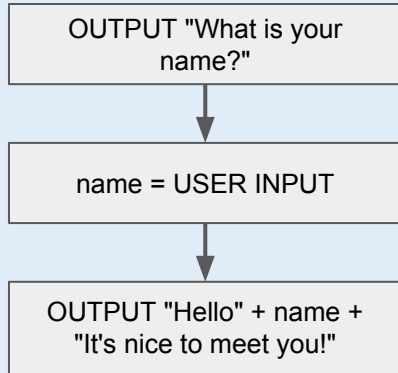
The three building blocks of algorithms

Recall that all algorithms are built upon some combination of these three "building blocks":

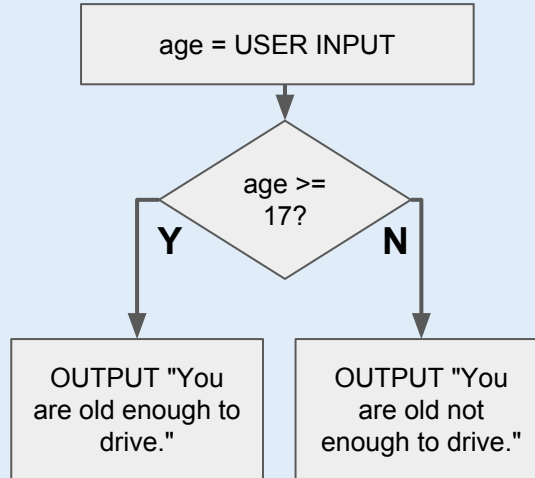
- **Sequence** - each step is executed in the order that it is written. All algorithms are made up of at least a basic sequence of instructions.
- **Selection** - controls the flow of an algorithm by checking whether a **condition** is met or not before determining which steps should follow.
- **Iteration** - controls the flow of an algorithm by repeating a set of instructions either for a fixed number of times or whilst a set **condition** is met.

The three building blocks of algorithms

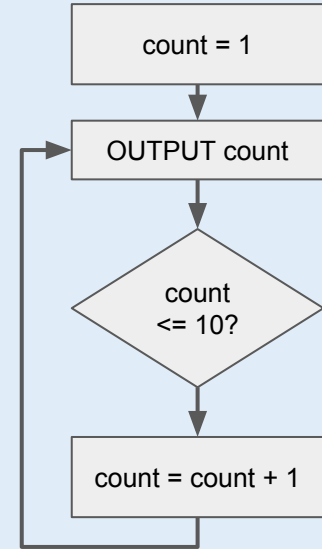
Sequence



Selection



Iteration



Selection structures: IF/ELSE and SWITCH/CASE

Selection structures allow for steps within an algorithm or program to only be followed if one or more condition has been met.

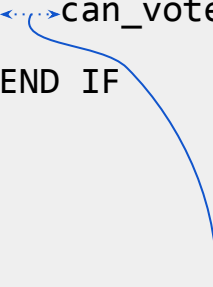
There are **two** selection structures that you need to be familiar with:

- IF/ELSE structures run different instructions depending on whether a condition is met or not
- SWITCH/CASE run different instructions depending on the value of a variable

IF/ELSE example

Pseudocode

```
INPUT age
IF age >= 18 THEN
    can_vote = True
ELSE
    can_vote = False
END IF
```



- IF/ELSE structures check if a condition is met (true)
- If the specified condition is met, the instructions within the "IF" part are run
- If the specified condition is not met the instructions in the "ELSE" part are run
- ELSE-IF (elif in Python) allow for multiple checks to be made

Notice the use of **indentation** - this is essential in Python for showing that a line belongs to a "branch" of the IF structure. It also helps to keep code really easy to read.

SWITCH/CASE example

Pseudocode

```
OUTPUT "Choose an option,  
       either 1, 2 or 3:"  
INPUT menu_option  
SWITCH menu_option:  
    CASE "1":  
        play_game()  
    CASE "2":  
        show_high_scores()  
    CASE "3":  
        quit_game()  
END SWITCH
```

- SWITCH/CASE statements check the value of a variable
- A different CASE statement is used for each possible value of the variable to determine what happens "in that case"

Note: There is no switch/case structure in Python, but you still need to know about it for your exams!

More complex selection with ELSE IF

Pseudocode

It is sometimes necessary to test for **multiple conditions** to determine what outcome should be presented.

For example, if a baby is crying it would be sensible to test whether they are hungry, tired or their nappy needs changing.

ELSE IF is an efficient way to test for multiple conditions as further checks will not be made against any other conditions once one has been found that meets a condition.

```
IF baby_hungry == True THEN
    cause_of_crying = "feed baby"
ELSE IF nappy_full = True THEN
    cause_of_crying = "nappy needs changing"
ELSE IF baby_tired = True THEN
    cause_of_crying = "baby needs a nap"
ELSE
    cause_of_crying = "who knows!"
OUTPUT "Baby crying! Action to take: " +
    cause_of_crying
```

Nested IF statements

Pseudocode

IF statements can be **nested** within another IF statement as another way to produce complex conditions.

You should be careful when using nested IF statements; in many cases it might be neater and clearer to use AND operators to test for multiple things being true.

```
IF owns_a_car = True THEN
  IF has_passed_driving_test == True THEN
    OUTPUT "You can drive by yourself!"
  ELSE
    OUTPUT "You need to drive with someone else who can drive."
  END IF
ELSE
  OUTPUT "You don't have a car, so why are we worried about this?"
```



This "inner" IF statement is **nested** within the outer IF statement. **It will only run** if the output IF statement condition is met.

Two types of iteration: definite and indefinite

There are **two** types of iteration structures that you need to be familiar with:

- **Indefinite iteration** - a block of code that will repeat whilst a specified **condition is true**. This is often called **condition-controlled** iteration and is done with WHILE and DO...WHILE loops.
- **Definite iteration** - a block of code that will repeat a **known number of times**. This is often called **count-controlled** iteration and is done with FOR loops.

Two indefinite (condition-controlled) iteration structures

While loops



Can do in
Python

- A condition is stated at the **start** of the loop
- The code within the loop will run repeatedly **whilst the loop condition is met** (true)
- The code **may not run at all** if the condition isn't met when the loop is first encountered

Do...until loops



Cannot do in
Python

- A condition is stated at the **end** of the loop
- The code within the loop **will always run once**
- The code within the loop will then **repeat** if the loop condition is met

Definite (condition-controlled) iteration with WHILE loops

Pseudocode

```
ask_again = True
WHILE ask_again == True
    OUTPUT "Would you like me
           to ask this pointless
           question again?"
    INPUT continue
    IF continue == "no" THEN
        ask_again = False
    END IF
END WHILE
```

- Very similar to an if statement that repeats
- State the **condition test** within the while statement
- Code that is to be repeated must be **indented** under the start of the loop
- All **relational** and **Boolean operators** can be used in the condition test

Definite iteration with DO...WHILE loops

Pseudocode

```
DO
    OUTPUT "Enter your password: "
    INPUT password
WHILE password != "letmein"
```

- A DO...WHILE loop will always execute its code at least once.
- The code in the loop will repeat if the condition at the end of the loop is True.
- Again, it's not possible to write DO...WHILE loops in Python, but other programming languages support them.

Infinite loops

Python 3

```
while True:  
    get_player_move()  
    redraw_player()
```

- It is possible to write a loop that will continue to run by simply stating **True** as the loop condition
- This is quite common in games and programs where a main menu is displayed after every part of the program has completed or where set of sprites need to be continually redrawn on the screen.

Breaking out of an infinite loop

Python 3

```
while True:
    cont = input("Would you
                  like me to ask
                  again?")
    if cont == "no":
        break
```

- An infinite loop can be used as an efficient way to keep asking for an input until valid data is provided
- However, it is essential that there is a way to end the loop!
- To do this, use the **break** keyword

Definite (count-controlled) iteration with FOR loops

Python 3

Example 1 - Pseudocode

```
for num = 1 to 10:  
    print(num)  
next num
```

Example 2 - Python

```
print("Counting down...")  
for count in range(0, 11):  
    print(10 - count)  
print("Blast off!")
```

- FOR loops repeat code a **known number of times**
- An **iterator variable** is created in the FOR statement (in this case, num)
- The value of this variable is **updated** each time the loop runs
- The value of the variable can be **accessed** by the code within the for loop structure

Using for loops in Python

For loops are really useful in Python. They can be used to **iterate** through any list of values:

- To iterate through a range of numbers, use the `range(min, max)` function.
- To iterate through an array or list, simply state the name of the list to be iterated over.
- To iterate through the characters in a string, simply state the string or variable that holds the string to be iterated over.

Example 1

```
for i in range (1,11):  
    print(i) # Will print the numbers 1 to 10
```

Example 2

```
list = ["apple", "orange", "lemon"]  
for item in list:  
    print(item) # Will print each item in the  
                list on a different line
```

Example 3

```
capitals_count = 0  
message = "How mAny cAPitals?"  
for letter in message:  
    if letter.isUpper() == True:  
        capitals_count = capitals_count + 1  
print("Number of capital letters is: " +  
      str(capitals_count))
```

Producing structured code with subroutines

Subroutines

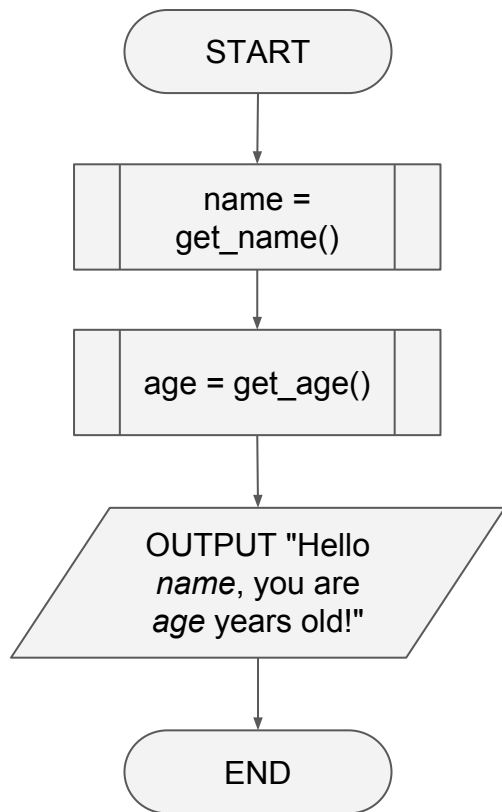
Subroutines are **named** "mini programs" inside your program

Using subroutines allows you to structure your program so that it is:

- **Easier to read:** There are fewer long blocks of code to understand
- **More efficient:** Blocks of code can be written once and reused many times
- **More reliable:** Each subroutine can be individually tested to ensure it works

Subroutine example

Python 3

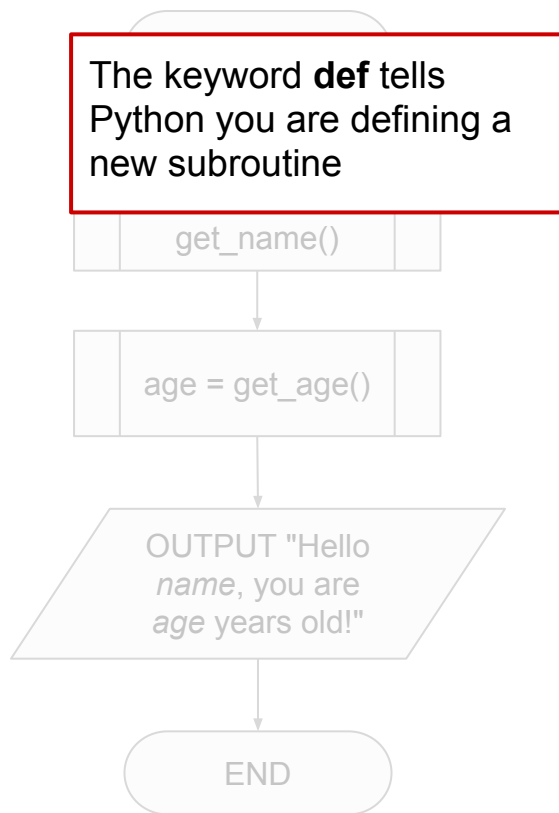


```
def get_name():  
    n = input("What is your name? ")  
    return n  
  
def get_age():  
    a = input("What is your age? ")  
    return a  
  
name = get_name()  
age = get_age()  
  
print("Hello {0}, you are {1} years  
old!".format(name, age))
```

Subroutine example

Python 3

The keyword **def** tells Python you are defining a new subroutine



```
def get_name():  
    n = input("What is your name? ")  
    return n
```

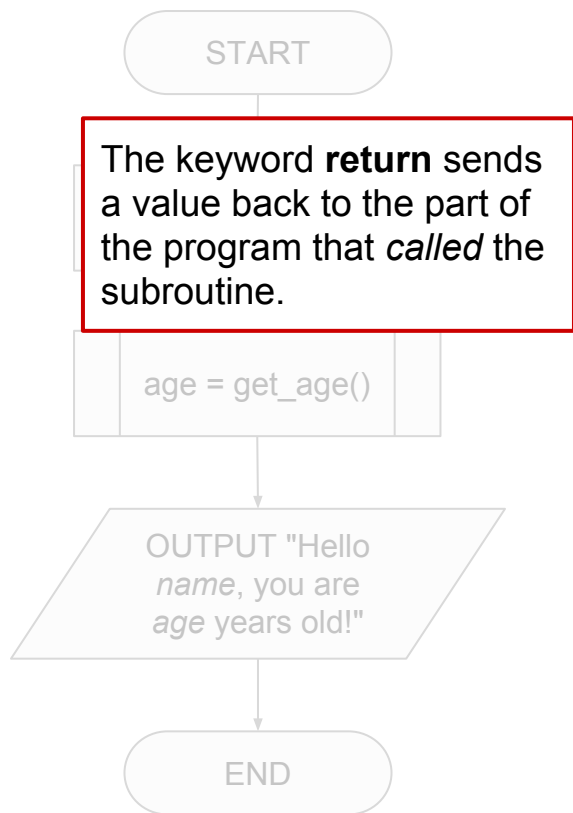
```
def get_age():  
    a = input("What is your age? ")  
    return a
```

```
name = get_name()  
age = get_age()
```

```
print("Hello {0}, you are {1} years  
old!".format(name, age))
```


Subroutine example

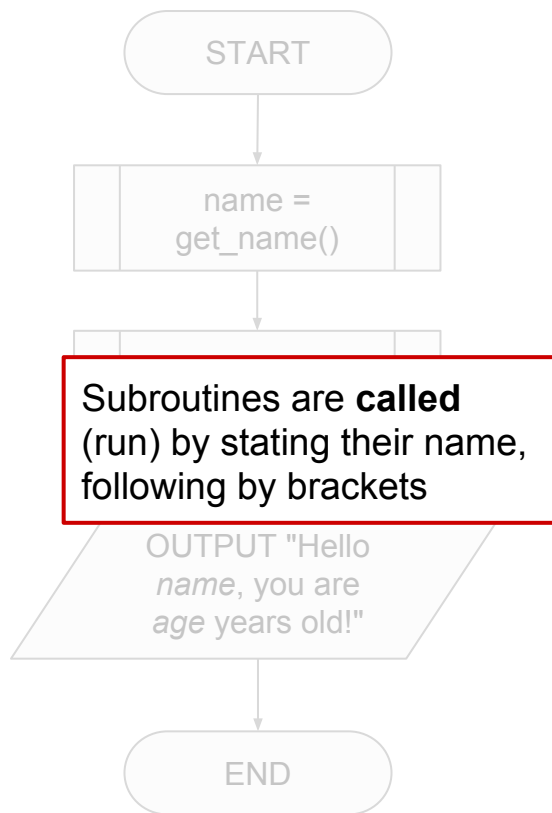
Python 3



```
def get_name():  
    n = input("What is your name? ")  
    return n  
  
def get_age():  
    a = input("What is your age? ")  
    return a  
  
name = get_name()  
age = get_age()  
  
print("Hello {0}, you are {1} years  
old!".format(name, age))
```

Subroutine example

Python 3



```
def get_name():  
    n = input("What is your name? ")  
    return n
```

```
def get_age():  
    a = input("What is your age? ")  
    return a
```

name → get_name()

age → get_age()

```
print("Hello {0}, you are {1} years  
old!".format(name, age))
```

Parameters and arguments

Python 3

Parameters are special **variables** used to pass values into a subroutine

Arguments are the **actual values** that are passed in

Parameters are declared inside the brackets that follow a subroutine's name.

```
def rect_area(height, width):  
    a = height * width  
    return a
```

```
area = rect_area(2, 4)
```

```
print("The area of a rectangle of  
height 2 and width 4 is  
{0}".format(area))
```

Parameters and arguments

Python 3

Parameters are special variables used to pass values into a function.

Arguments are the values that are passed in to the function.

Parameters are declared inside the brackets that follow a subroutine's name.

Parameters are declared inside the brackets after the subroutine name

```
def rect_area(height, width):  
    a = height * width  
    return a
```

```
area = rect_area(2, 4)
```

```
print("The area of a rectangle of  
height 2 and width 4 is  
{0}".format(area))
```

Parameters and arguments

Python 3

Parameters are special variables used to pass values into a function.

Multiple parameters are separated by a **comma**

Arguments are the actual **values** that are passed in

Parameters are declared inside the brackets that follow a subroutine's name.

```
def rect_area(height, width):  
    a = height * width  
    return a
```

```
area = rect_area(2, 4)
```

```
print("The area of a rectangle of  
height 2 and width 4 is  
{0}".format(area))
```

Parameters and arguments

Python 3

Parameters are special variables used to pass values **into** a subroutine

Arguments are values that are passed to a subroutine

Parameters are defined by brackets that follow a subroutine's name.

The **values** stored in the parameters can then be accessed within the subroutine by stating the parameter name

```
def rect_area(height, width):  
    a = height * width  
    return a
```

```
area = rect_area(2, 4)
```

```
print("The area of a rectangle of  
height 2 and width 4 is  
{0}".format(area))
```

Parameters and arguments

Python 3

Parameters are special variables used to pass values **into** a subroutine

Arguments are the actual **values** that are passed in

Parameters are declared inside the brackets that follow a subroutine's name.

Arguments (values) for the parameters are given when the subroutine is called

```
def rect_area(height, width):  
    a = height * width  
    return a
```

```
area = rect_area(2, 4)
```

```
print("The area of a rectangle of  
height 2 and width 4 is  
{0}".format(area))
```

Procedures and Functions

There are two different types of subroutine that you need to know about - **procedures** and **functions**.

Procedures and functions are very similar:

- they are both made up of sets of instructions
- they can both accept values via parameters
- they are both given names so that they can be called

However, there are a few key differences between them.

Procedures

Python 3

Procedures are the **simplest** form of subroutine.

Procedures are **blocks of instructions** that run but don't necessarily **return** a value back to the part of the program that called it.

They **may have** one or more parameters, but they don't have to.

```
# Define the procedure
def say_hello_ten_times():

    name = input("Hello, what is your
                  name? ")
    print ("Hello {}".format(name) *
          10)

# Call (run) the procedure
say_hello_ten_times()
```

Functions

Python 3

Function are subroutines with **at least one parameter** and **always return an output value**.

You can think of a function like a box that you put a value into, it does something to that value and then spits out the result.

Functions are defined in Python in just the same way as procedures (i.e. by using the **def** keyword), and they always include at least one **return** statement.

```
def get_area_of_rectangle(height,
                           width):
    area = height * width
    return area

# Call the function and assign its
return value to a variable
rect_area = get_area_of_rectangle(12,
                                   8)

# Use the variable elsewhere in our
program
print("The area of the rectangle is
      {0}".format(rect_area))
```

Functions

Python 3

Function are subroutines with **at least one parameter** and **always return**

This function has two parameters - **height** and **width**

You can think of a function like a box that you put a value into, it does something to that value and then spits out the result.

Functions are defined in Python in just the same way as procedures (i.e. by using the **def** keyword), and they always include at least one **return** statement.

```
def get_area_of_rectangle(height, width):  
    area = height * width  
    return area  
  
# Call the function and assign its  
return value to a variable  
rect_area = get_area_of_rectangle(12,  
                                   8)  
  
# Use the variable elsewhere in our  
program  
print("The area of the rectangle is  
      {0}".format(rect_area))
```

Functions

Python 3

Function are subroutines with **at least one parameter** and **always return**

The function **returns** the value of area once it has done its processing

You can think of a box that you put a value into, it does something to that value and then spits out the result.

Functions are defined in Python in just the same way as procedures (i.e. by using the **def** keyword), and they always include at least one **return** statement.

```
def get_area_of_rectangle(height,
                           width):
    area = height * width
    return area

# Call the function and assign its
# return value to a variable
rect_area = get_area_of_rectangle(12,
                                   8)

# Use the variable elsewhere in our
# program
print("The area of the rectangle is
      {0}".format(rect_area))
```

Functions

Python 3

Function are subroutines with **at least one parameter** and **always return an output value**.

You can think of a function like a box that you put a value into, it does something to that value and then spits out a value.

The return value is **assigned** to a variable when the function is **called**

Functions are defined in just the same way as procedures (i.e. by using the **def** keyword), and they always include at least one **return** statement.

```
def get_area_of_rectangle(height,
                           width):
    area = height * width
    return area

# Call the function and assign its
return value to a variable
rect_area = get_area_of_rectangle(12,
                                   8)

# Use the variable elsewhere in our
program
print("The area of the rectangle is
      {0}".format(rect_area))
```

Functions

Python 3

Function are subroutines with **at least one parameter** and **always return an output value**.

You can think of a function like a box that you put a value into, it does something to that value and then spits out

The value returned by the function can then be **used** later in the program

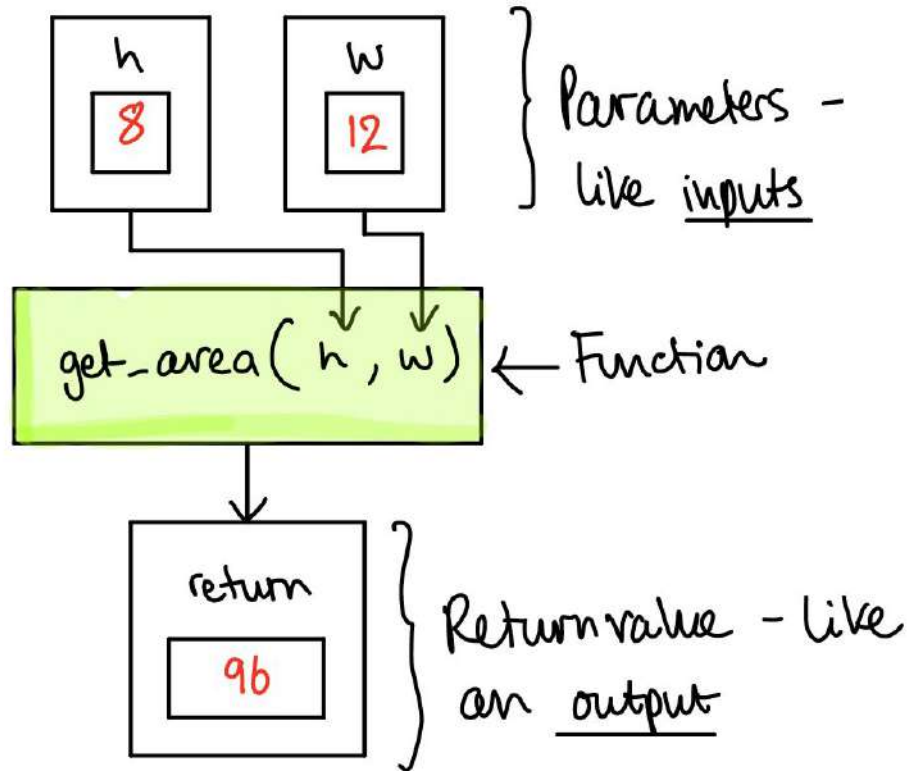
Function just the same way as procedures (i.e. by using the **def** keyword), and they always include at least one **return** statement.

```
def get_area_of_rectangle(height,
                           width):
    area = height * width
    return area

# Call the function and assign its
return value to a variable
rect_area = get_area_of_rectangle(12,
                                   8)

# Use the variable elsewhere in our
program
print("The area of the rectangle is
      {0}".format(rect_area))
```

Functions



Writing subroutines in Pseudocode

Pseudocode

When writing subroutines in Pseudocode a few things must be remembered:

- State the type of subroutine, either a Procedure or Function
- Parameters must be defined within brackets after the name (identifier) of the subroutine
- Indent the subroutine's code
- Functions must include at least one RETURN statement
- End the subroutine with an END X at the end

```
PROCEDURE say_hello()  
    OUTPUT "Hello!"  
END PROCEDURE  
  
FUNCTION get_area(height, width)  
    area = height * width  
    RETURN area  
END FUNCTION
```


Local and global scope

Python 3

By default, variables that are declared inside a subroutine can only be used within the subroutine.

These variables are said to have **local scope** - they can only be accessed locally.

The **global** keyword can be used to declare a **global variable**.

Global variables can be accessed by **any** part of the whole program, including inside any subroutine.

```
player_score = 0
player_lives = 5

def touch_enemy():
    global player_score, player_lives
    player_score = player_score + 100
    player_lives = player_lives - 1

def check_game_over():
    global player_lives
    if player_lives == 0:
        return True
    else:
        return False
```

Local and global scope

Python 3

By default, variables that are declared inside a subroutine can only be used within

These **scope** locally.

These variables are declared outside of any subroutines - they have **global** scope

The **global** keyword can be used to declare a **global variable**.

Global variables can be accessed by **any** part of the whole program, including inside any subroutine.

```
player_score = 0
player_lives = 5
```

```
def touch_enemy():
    global player_score, player_lives
    player_score = player_score + 100
    player_lives = player_lives - 1
```

```
def check_game_over():
    global player_lives
    if player_lives == 0:
        return True
    else:
        return False
```

Local and global scope

Python 3

By default, variables that are declared inside a subroutine can only be used within the subroutine.

These variables are said to have **local scope**.

The **global** keyword must be used inside subroutines that need to access the global variables

Global variables can be accessed by **any** part of the whole program, including inside any subroutine.

```
player_score = 0
player_lives = 5

def touch_enemy():
    global player_score, player_lives
    player_score = player_score + 100
    player_lives = player_lives - 1

def check_game_over():
    global player_lives
    if player_lives == 0:
        return True
    else:
        return False
```

Local and global scope

Python 3

By default, variables that are declared inside a subroutine can only be used within the subroutine.

These variables are said to have **local scope**.

Multiple variable identifiers can be stated after **global**, each one separated by a **comma**

The **global** keyword is used to declare a **global variable**.

Global variables can be accessed by **any** part of the whole program, including inside any subroutine.

```
player_score = 0
player_lives = 5

def touch_enemy():
    global player_score, player_lives
    player_score = player_score + 100
    player_lives = player_lives - 1

def check_game_over():
    global player_lives
    if player_lives == 0:
        return True
    else:
        return False
```

Local and global scope

Python 3

By default, variables that are declared inside a subroutine can only be used

with

The

scope

As parameters, **height** and **width** have **local scope** to `rect_area()`

local

defined locally.

The **global** keyword can be used to declare a **global variable**.

Global variables can be accessed by **any** part of the whole program, including inside any subroutine.

```
def rect_area(height, width):  
    a = height * width  
    return a
```

```
area = rect_area(2, 4)
```

```
print("The area of a rectangle of  
height 2 and width 4 is  
{0}".format(area))
```

Local and global scope

Python 3

By default, variables that are declared inside a subroutine can only be used within the subroutine.

These variables are said to have **local scope** - they can only be accessed locally.

The **global** keyword can be used to declare a variable as global.

Global variables can be accessed by **any** part of the program, including inside any subroutine.

Local variables cannot be accessed outside of their subroutine, so this line of code would **fail**

```
def rect_area(height, width):  
    a = height * width  
    return a  
  
area = rect_area(2, 4)  
  
print("The height of the rectangle is",  
      height)
```

Local and global scope

Python 3

By default, variables that are declared inside a subroutine can only be used within the subroutine.

These variables are said to have **local scope** - they can only be accessed locally.

The **global** keyword can be used to

Instead we have to use the **return value** from the subroutine to print the area

Global variables can be used by any part of the program, including inside any subroutine.

```
def rect_area(height, width):  
    a = height * width  
    return a
```

```
area = rect_area(2, 4)
```

```
print("The area of a rectangle of  
height 2 and width 4 is  
{0}".format(area))
```

Data types and casting

Data types

Every piece of data that is stored in a variable or constant has a **type**.

The type of a piece of data **determines how it is stored** and **what you can do with it** within a program.

Data type	Description	Examples
Integer	Whole numbers	12, 0, -142
Float/Real	Fractional numbers (number with decimal points)	0.42, 1.00, -588.432
String	Text data	"Hello"
Character	A single item of text data	'a'
Boolean	Logical values, either True or False	True, False

Why to data types matter?

Python 3

The type of an item of data determines how the data will be stored and what can be done with it.

Some **operators** are **ambiguous** - they can mean more than one thing.

For example, `+` is used in Python to mathematically add two numbers together *and* join two strings of text together.

Python works out which operation to perform based on the type of the data on either side of the operator.

```
10 + 5
```

```
>>> 15
```

```
"Hello my name is " + "Bob"
```

```
>>> Hello my name is Bob
```

```
"Hello" + 2
```

```
>>> Traceback (most recent call last):  
      File "<pyshell#6>", line 1, in <module>  
        "Hello" + 2  
TypeError: must be str, not int
```

Why to data types matter?

Python 3

The type of an item of data determines how the data will be stored and what can be done with it.

Some **operators** are **ambiguous** - they can mean more than one thing.

For example, `+` is used in Python to mathematically add two numbers together *and* join two strings of text together.

Python works out which operation to perform based on the type of the data on either side of the operator.

```
10 + 5
```

```
>>> 15
```

Integer + Integer : perform mathematical addition

```
"Hello my name is " + "Bob"
```

```
>>> Hello my name is Bob
```

```
"Hello" + 2
```

```
>>> Traceback (most recent call last):  
      File "<pyshell#6>", line 1, in <module>  
        "Hello" + 2  
TypeError: must be str, not int
```

Why to data types matter?

Python 3

The type of an item of data determines how the data will be stored and what can be done with it.

Some **operators** are **ambiguous** - they can mean more than one thing.

For example, `+` is used in Python to mathematically add two numbers together *and* join two strings of text together.

Python works out which operation to perform based on the type of the data on either side of the operator.

```
10 + 5
```

```
>>> 15
```

String + String : perform string concatenation

```
"Hello my name is " + "Bob"
```

```
>>> Hello my name is Bob
```

```
"Hello" + 2
```

```
>>> Traceback (most recent call last):  
      File "<pyshell#6>", line 1, in <module>  
        "Hello" + 2  
TypeError: must be str, not int
```

Why to data types matter?

The type of an item of data determines how the data will be stored and what can be done with it.

Some **operators** are **ambiguous** - they can mean more than one thing.

For example, + is used in Python to mathematically add two numbers together *and* join two strings of text together.

Python works out which operation to perform based on the type of the data on either side of the operator.

Python 3

10

>>>

"He

>>>

A **runtime error** is caused because the data types of the two **operands** for the + operation do not match.

Python doesn't know whether to attempt a **mathematical addition** or a **string concatenation** as the + operator can be used for either depending on the type of data it is used with.

"Hello" + 2

```
>>> Traceback (most recent call last):  
      File "<pyshell#6>", line 1, in <module>  
        "Hello" + 2  
TypeError: must be str, not int
```

Casting data types

Python 3

Casting is the process of **converting data from one type to another**.

The most common case of casting is converting numbers that users have typed in via `input()` statements into integers or floats.

This is because `input()` **always** returns a **string**, even if the only thing typed in is a numeric value.

You will **always** need to cast an input to an integer or float if you plan to do any mathematical operation with it, including testing whether it is greater or less than another number.

```
num = input("Please enter a number: ")

# num currently contains a string

>>> print(type(num))

<class 'str'>

num = int(num)

# num is now of Integer type

>>> print(type(num))

<class 'int'>
```

Casting functions

Pseudocode & Python 3

There are special built-in casting functions for each of String, Integer and Float data types:

- `str()` - converts values to String type.
- `int()` - converts Strings that represent numbers into their equivalent Integer values; also converts Float values to Integers by "dropping" the decimal part.
- `float()` - converts Strings that represent numbers and Integers into equivalent Float values.

str() example

```
num1 = 3
num2 = 4
result = num1 + num2
print("The result is: " + str(result))
```

float() and str() example

```
x = input("Enter a number to double: ")
double_x = float(x) * 2
print(x + " doubled is " + str(double_x))
```

int() example

```
menu_choice = input("Enter a choice, 1 2 or 3: ")
menu_choice = int(menu_choice)
if menu_choice < 0 or menu_choice > 3:
    print("Choice must be between 1 and 3!")
```

Manipulating Strings

Manipulating Strings

String data is a collection of individual characters "strung together". All text data is of the string type.

There are a number of useful techniques for manipulating strings that you might need to know for an exam or completing your NEA task, including:

- Joining strings together (**concatenation**)
- Finding the length of a string
- Obtaining a specific character or substring from a string
- Testing whether a string contains a character or substring
- Converting a string into upper or lower case
- Using a for loop to process each character in a string

For more details of the **string methods** available in Python:

https://www.w3schools.com/python/python_ref_string.asp

String concatenation

Pseudocode & Python 3

String concatenation simply means to join strings together.

This is done using the + operator with string data for each operand.

When joining non-String data such as Integers or Floats, these values need to be **cast** to Strings first.

In Python, any data type can be cast to a string using the build-int function `str()`.

```
string1 = "Hello "  
string2 = "World!"  
  
output_string = string1 + string2  
  
print(output_string)  
>> Hello World!
```

Substituting values with .format()

Python 3

Python offers a String method `.format()` that allows you replace **placeholders** - `{}` - within strings with data values when the program runs.

For each placeholder used, add an index value (starting at 0) and then simply list the values to be substituted as arguments to `.format()` in the order that they should appear.

One of the best things about using `.format()` to substitute values is that there is no need to cast numeric types to strings first!

```
# Replace {} placeholders with substituted values
```

```
x = 4
```

```
y = 3
```

```
result = x + y
```

```
print("The result of {0} + {1} is  
{2}".format(x, y, result))
```

```
# Format a float value to 3 d.p.
```

```
two_thirds = 2 / 3
```

```
print("Two-thirds to 3 d.p. is  
{0:.3f}".format(two_thirds))
```

Accessing a specific character within a string

Pseudocode & Python 3

To access a particular character within a string, state the **identifier** for the string followed by **square brackets** - `[]`.

Within the square brackets state the **index** of the character that is to be accessed.

String indexes start at 0 so to access the fourth character, the index 3 would be used.

```
message = "How are you doing today?"
```

```
print(message[4])
```

```
>> a
```

```
print(message[23])
```

```
>> ?
```

Finding the length of a string

Pseudocode & Python 3

Use `len()` or `.length` to find the length of a string.

Finding the length of a string can be very useful for **validating inputs**, for example testing whether a value has been entered at all (length will be 0 if not) or a password is longer than a specified number of characters.

```
# In Pseudocode
message = "Hello there!"
print(message.length)
>> 12
```

```
# In Python 3
message = "Hello there!"
print(len(message))
>> 12
```

Obtaining a substring from a string

Pseudocode

A substring is a string made up from part of a bigger string.

In pseudocode, use this function:

`stringname.substring(startPosition, numberOfCharacters)`

- *startPosition* is the *index* to start the substring (0 being the first character).
- *numberOfCharacters* is the number of characters that the substring should contain from the starting position.

```
message = "This is a long string."

part_of_string = message.substring(0,6)

print(part_of_string)
>> This i

end_of_string = message.substring(15,7)

print(end_of_string)
>> string.
```

Obtaining a substring from a string

Python 3

To obtain a substring in Python, string "slicing" is used.

State the identifier of the string, followed by square brackets - [].

Within the square brackets state the starting index, then a colon (:) and then the end index (leave blank if you want a substring from the starting index through to the end of the string).

The starting index can be negative to start X characters from the end of the string.

```
message = "This is a long string."

part_of_string = message[0:6]

print(part_of_string)
>> This i

end_of_string = message[15:]

print(end_of_string)
>> string.

last_three_characters = message[-3:]
print(last_three_characters)
>> ng.
```

Testing whether a string contains a substring

Python 3

Python offers a very easy way to test whether a substring can be found within another string - the `in` keyword.

`in` returns a `True` value if the substring is found, meaning it can be used with `if` statements.

This is a really useful technique for validating input such as a range of menu options - simply test whether the user's input is found in a string made up of all valid inputs.

```
test_string = "ad"
message = "I had a nice day"

if test_string in message:
    print("Substring found!")
else:
    print("Substring not present.")

# Validating input example
choice = input("Enter your choice (A, B or C): ")

if choice in "ABC":
    do_something()
else:
    print("Invalid choice!")
```


Replacing characters within a string

Pseudocode & Python 3

The `.replace()` method can be applied to a string return a new string that replaces a specified substring (possibly a single character) with another substring.

```
story = "Jack and Jill went up the hill."

adjusted_story = story.replace("Jack", "Tom")

print(adjusted_story)

>> Tom and Jill went up the hill
```

Converting strings to upper and lower case

Python 3

The `.upper()` and `.lower()` methods return a version of the string in all-capitals or all-lowercase format.

The original string isn't changed, rather the `.upper()` or `.lower()` methods returns a new string.

These methods are really useful for allowing users to enter either a capital or lowercase version of a letter at a menu prompt - simply convert their input to upper or lowercase before testing it.

```
normal_text = "Hello there!"  
shouting = normal_text.upper()  
print(shouting)  
>> HELLO THERE!
```

Case-insensitive input example

```
choice = input("Do you wish to continue  
(y/n)? ")  
if choice.lower() == 'y':  
    print("Ok, everything's erased!")  
else:  
    print("Good choice. Your data is safe.")
```

It doesn't matter whether the user enters "Y" or "y" - whatever they enter will be converted to lowercase and then tested to see if it is "y".

Iterating through each character in a String

Pseudocode & Python 3

For loops can be used to iterate through each character in a String.

This is really useful for going through a string character by character to check that it meets validation rules. An example of this might be checking a password contains at least one capital letter and symbol.

Examples are given to show how to do this in pseudocode and Python.

Iterating through a string in Pseudocode

```
message = "Count the vowels"
```

```
vowels_count = 0
```

```
FOR i = 0 to (message.length - 1)
```

```
    IF message[i].upper() in "AEIOU" THEN
```

```
        vowels_count = vowels_count + 1
```

```
    END IF
```

```
NEXT i
```

In Python

```
message = "Count the vowels"
```

```
vowels_count = 0
```

```
for letter in message:
```

```
    if letter.upper() in "AEIOU":
```

```
        vowels_count = vowels_count + 1
```

Example: Validating an email address

For an email address to be valid, it must contain one @ symbol and at least one . symbol. Other acceptable symbols are - and _ along with all letters and numbers.

Pseudocode

```
email = input("Enter an email address: ")
valid_email = True  # Assume valid email unless proven
count_at_sign = 0
count_dots = 0

FOR i = 0 to (email.length - 1):
    IF (i == 0 AND email[i] == '@') OR (i == 0 AND
        email[i] == '.') THEN
        valid_email = False
    ELSE IF email[i] == '@' THEN
        count_at_sign = count_at_sign + 1
    ELSE IF email[i] == '.' THEN
        count_dots = count_dots + 1
    ELSE IF email[i] in "!#$%^&*()+={}[ ]:;'
        \"\\|<>,?/~±$":
        valid_email = False
    END IF
NEXT i
```

```
# Having gone through every character, check there are
the right number of @ and . symbols

IF count_at_sign != 1 OR count_dots == 0 THEN
    valid_email = False
END IF

# Use the final state of valid_email to report whether
the email address is valid or not.

IF valid_email == True THEN
    print("Email is valid")
ELSE
    print("Invalid email")
END IF
```

Arrays

Arrays - used for storing multiple values

An array is:

- a **data structure**
- that stores **multiple values**
- under a **single identifier**

Arrays are really useful when you have lots of **related data** that you don't want to store in multiple variables.

Arrays are a little like pigeon holes...

Arrays are made up of multiple slots (**elements**), each of which can contains an item of data.

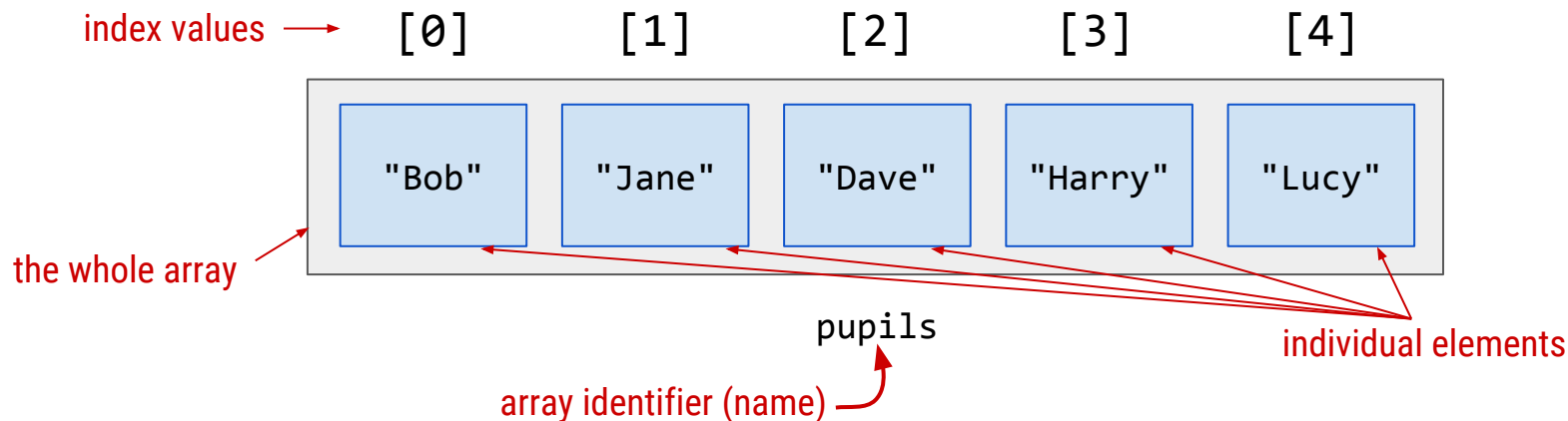
Each element has its own **index** - a number that can be used to access the data inside.



Image by [addentry](#)

Arrays and made of multiple elements

- Each array is made up of a number of **elements**
- Elements are like '**slots**' that can each be used to store **one thing**
- Each element has its own **index value** (index values start at 0 in most programming languages)



Representing arrays in Pseudocode

Pseudocode

Like all data, arrays are assigned to variables.

When declaring a new array you need to specify its size (i.e. how many items of data it should store).

To access an element ("slot") within the array, state its identifier and then the index of the element within square brackets.

```
# Declare a new array called todo_items that  
will be able to store 10 pieces of data
```

```
array todo_items[10]
```

```
# Assign some values to elements in the array
```

```
todo_items[0] = "Tidy room"
```

```
todo_items[1] = "Do homework"
```

```
todo_items[2] = "Call Nan"
```

```
# Access the data within a specific element
```

```
OUTPUT todo_items[0]
```

```
# Use a FOR loop to access each item in the  
array by its index
```

```
for i = 0 to 10
```

```
    OUTPUT todo_items[i]
```

```
next i
```

Iterating over an array with a FOR loop

Pseudocode

FOR loops offer a really efficient way to process the contents of an array.

In order to use a FOR loop to iterate through each item in an array you need to:

- **increment** the value of a counter variable
- use the counter to access each **index** within the array

```
items = ['Cat', 'Dog', 'Rabbit', 'Hamster']
```

```
FOR i = 0 TO 3  
    OUTPUT items[i]  
END FOR
```

```
>> Cat  
>> Dog  
>> Rabbit  
>> Hamster
```

Note that array index values in pseudocode might sometimes start at 1, not 0!

Arrays and Lists

The terms **Array** and **List** are often used interchangeably but there are actually some important differences between them.

Arrays

- Arrays are **fixed sized** data structures
- You must specify the size of the array when you declare it
- Sufficient memory is reserved for the array to store its data when it is declared

Lists

- Python doesn't use arrays, instead it uses Lists
- Lists work just like arrays but they are not fixed-size
- You can **append** addition items on the end of the list, so they never get full

Using lists in Python - declaring and assigning values

Python 3

- To declare a new, empty list, simply declare the identifier and assign it the value `[]` .
- To add (append) values to the list, use the `.append()` method.
- You can also specify initial values when you declare a list.

```
pupils = [] # declare a new, empty list  
under the identifier 'pupils'
```

```
pupils.append("Bob") # add 'Bob' to the  
end of the pupils list
```

```
pupils.append("Dave")
```

```
pupils.append("Harry")
```

```
pupils.append("Lucy")
```

```
pre-populated-list = ['Bob', 'Dave',  
'Harry', 'Lucy'] # declare a list called  
'pre-populated-list' that contains items
```

Using lists in Python - accessing and changing values

Python 3

- To access a value in a list, **state the identifier** of the list, followed by the **index number** of the value within **square brackets []**
- Remember that index values start at 0, so the first item in the list is accessed by `list_name[0]`
- Values can be **assigned** to elements within the list by using the same syntax, i.e. state the name of the list followed by the index of the element that is to store the value

```
pupils = ['Bob', 'Dave', 'Harry', 'Lucy']
```

```
print(pupils[1]) # prints the second item  
in the list, 'Dave'
```

```
pupils[1] = "Jane" # assigns the value  
'Jane' to the second element in the list
```

```
print(pupils[1]) # prints the second item  
in the list, which is now 'Jane'
```

Two ways to iterate over a list in Python

Method 1 - access values via their index

Python 3

```
beatles = ["John", "Paul", "George",  
"Ringo"]
```

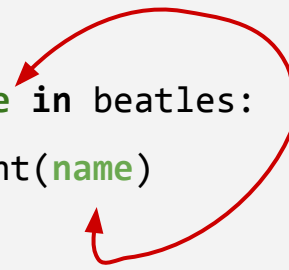
```
for i in range(0,4):  
    print(beatles[i])
```

Method 2 - let Python assign the each value in the list to the incrementing variable.

Python 3

```
beatles = ["John", "Paul", "George",  
"Ringo"]
```

```
for name in beatles:  
    print(name)
```



You get to pick the identifier for the incrementing variable used in the for loop. Remember to keep it sensible!

Two-dimensional arrays

- Arrays can store any type of data within their elements, including other arrays.
- An array that contains arrays is called a **two-dimensional array**.
- Two-dimensional arrays are well-suited to **storing tabular data** (i.e. tables of data) such as the results of several assignments for every pupil in a class.
- All of the data stored in a two-dimensional array is still accessed via a single identifier.
- Using two-dimensional arrays significantly simplifies programs that handle tabular data.

Two-dimensional array example

Pseudocode

The pseudocode on the right shows how the data in the following table could be stored in a two-dimensional array.

Pupil name	Test 1 (%)	Test 2 (%)	Test 3 (%)
Aled	80	82	76
Dan	50	55	63
Imran	76	84	86
Zoe	92	88	89

```
# Declare an new 4-by-3 array
```

```
array test_scores[4,3]
```

```
# Add data for each pupil
```

```
test_scores[0] = [80, 82, 76]
```

```
test_scores[1] = [50, 55, 63]
```

```
test_scores[2] = [76, 84, 86]
```

```
test_scores[3] = [92, 88, 89]
```

```
# Access specific results - double [][]!
```

```
print(test_scores[1][1]) # Dan's Test 2 score
```

```
print(test_scores[3][2]) # Zoe's Test 3 score
```

```
print(test_scores[0][0]) # Aled's Test 1 score
```

```
# Change a result
```

```
test_scores[2][2] = 88 # Updated Imran's Test 3  
                        score to 88
```


Two-dimensional array example

Pseudocode

The pseudocode on the right shows how the data in the following table could be stored in a two-dimensional array.

Pupil name	Test 1 (%)	Test 2 (%)	Test 3 (%)
Aled	80	82	76
Dan	50	55	63
Imran	76	84	86
Zoe	92	88	89

```
# Declare an new 4-by-3 array
```

```
array test_scores[4,3]
```

```
# Add data for each pupil
```

```
test_scores[0] = [80, 82, 76]
```

```
test_scores[1] = [50, 55, 63]
```

```
test_scores[2] = [76, 84, 86]
```

```
test_scores[3] = [92, 88, 89]
```

```
# Access specific results - double [][]!
```

```
print(test_scores[1][1]) # Dan's Test 2 score
```

```
print(test_scores[3][2]) # Zoe's Test 3 score
```

```
print(test_scores[0][0]) # Aled's Test 1 score
```

```
# Cha
```

```
test_scores[1][2] = 88 # Dan's Test 3 score to 88
```

Row index

Column index

Two-dimensional array example

Python 3

The pseudocode on the right shows how the data in the following table could be stored in a two-dimensional array.

Pupil name	Test 1 (%)	Test 2 (%)	Test 3 (%)
Aled	80	82	76
Dan	50	55	63
Imran	76	84	86
Zoe	92	88	89

```
# Declare an new empty list
```

```
test_scores = []
```

```
# Add data for each pupil
```

```
test_scores.append([80, 82, 76])
```

```
test_scores.append([50, 55, 63])
```

```
test_scores.append([76, 84, 86])
```

```
test_scores.append([92, 88, 89])
```

```
# Access specific results - double [[]]!
```

```
print(test_scores[1][1]) # Dan's Test 2 score
```

```
print(test_scores[3][2]) # Zoe's Test 3 score
```

```
print(test_scores[0][0]) # Aled's Test 1 score
```

```
# Change a result
```

```
test_scores[2][2] = 88 # Updated Imran's Test 3  
                        score to 88
```

Iterating over a two-dimensional array

Pseudocode

To iterate over a two-dimensional array a **nested FOR loop structure** must be used.

The **outer** FOR loop iterates through each **row** in the table, the **inner** FOR loop iterates through each **column** in that row.

Pupil name	Test 1 (%)	Test 2 (%)	Test 3 (%)
Aled	80	82	76
Dan	50	55	63
Imran	76	84	86
Zoe	92	88	89

```
# Declare an new 4-by-3 array  
array test_scores[4,3]
```

```
# Add data for each pupil  
test_scores[0] = [80, 82, 76]  
test_scores[1] = [50, 55, 63]  
test_scores[2] = [76, 84, 86]  
test_scores[3] = [92, 88, 89]
```

```
FOR row = 0 to 3  
    OUTPUT "Results for Pupil " + (row + 1)  
    FOR col = 0 to 2  
        OUTPUT "Test " + (col + 1) ":"  
        OUTPUT test_scores[row][col]  
    NEXT col  
NEXT row
```

Using two-dimensional arrays for board games

Pseudocode

Another common use of a two-dimensional array is to represent a 2D board game, for example of a chess board or Battleships grid.

The values stored in the array's elements can represent the position of pieces / counters within the game.

	0	1	2	3	4	5	6	7
0	D	D						
1								
2								
3			F					
4								
5								
6								
7						F		

```
array board[8,8] # Declare empty board
```

```
# Place destroyer
```

```
board[0][0] = "D"
```

```
board[0][1] = "D"
```

```
# Place frigats
```

```
board[3][2] = "F"
```

```
board[7][5] = "F"
```

```
# Attempt hit
```

```
OUTPUT "Enter the row and column to strike:"
```

```
INPUT row
```

```
INPUT col
```

```
IF board[row][col] != "" THEN
```

```
    board[row][col] = "X"
```

```
    OUTPUT "Hit!"
```

Using two-dimensional arrays for board games

Python 3

```
board = [] # Declare empty board list

# Fill board list with empty rows and columns
for row in range(0,7):
    row_cells = []
    for col in range (0,7):
        row_cells.append(" ")
    board.append(row_cells)

# Place destroyer
board[0][0] = "D"
board[0][1] = "D"

# Place frigats
board[3][2] = "F"
board[7][5] = "F"

...
```

```
...

# Attempt hit
row = input("Enter the row to strike: ")
row = int(row)
col = input("Enter the column to strike: ")
col = int(col)

if board[row][col] != " ":
    board[row][col] = "X"
    print("Hit!")
```

File handling

Files

Without files we can only store data in memory. This data is lost when our programs stop running.

Files allow us to store data so that it persists even when our programs are not running.

This allows program to 'carry on where they left off' when they are executed.

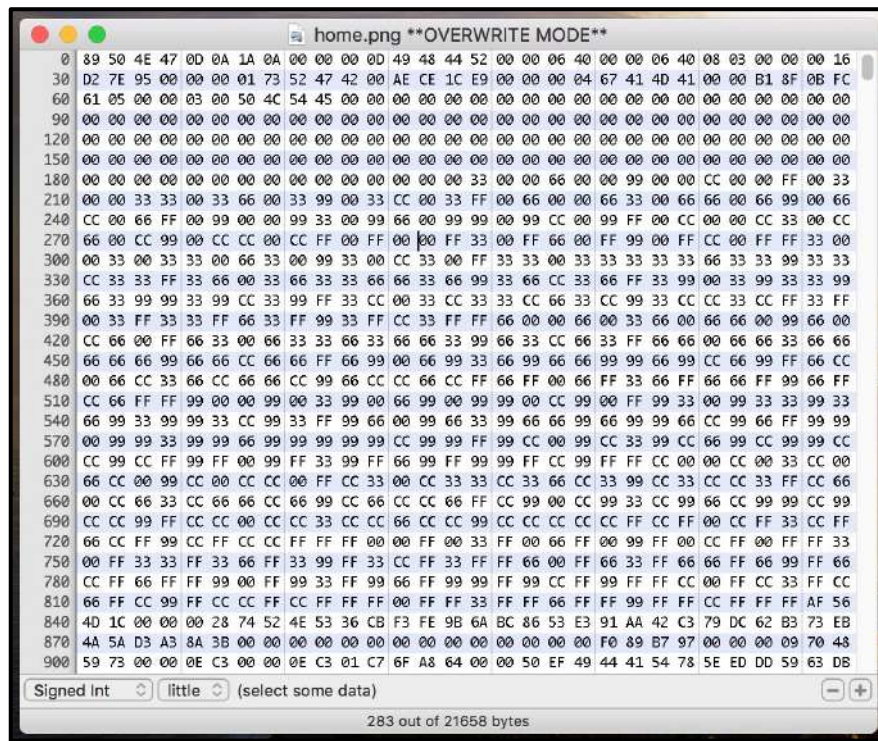
Binary and text files

Files can come in two different types:

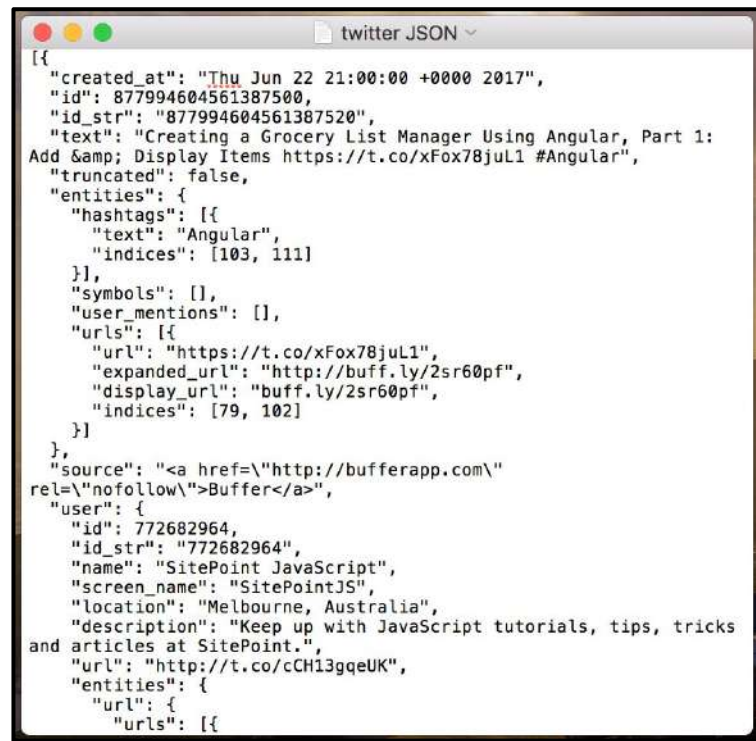
- **Binary** files - consist of binary data (e.g. 10111001 1101011) that is read and processed by specific applications
- **Text** files - consists of text (string) data (e.g. 'Hello!' or even '<h1>This is an HTML header</h1>') that can be read and processed by many applications

For GCSE, we will be focussing on reading and writing **text** files in Python and Pseudocode.

Contents of binary and text files



Contents of a binary file (a PNG image file)



Contents of a text file (a tweet encoded in JSON)

Pros and cons of text files

Text files have many advantages:

- Lots of programs can read their data, meaning they are ideal for storing data that will be exchanged between applications
- They are quite lightweight - it doesn't take a lot of data to store plain text
- They are simple to use in programming languages

Pros and cons of text files

However, there are disadvantages:

- Anyone can read their contents - this isn't great if you are storing passwords or source code for a key product
- Not all types of data can be efficiently represented as text (e.g. sound and video data)

File handling

Pseudocode

There are four main steps to file handling:

1. **Open** the file for use, even if it's a new file that is being written, it still has to be "opened" within the Operating System's file manager.
2. **Assign** the opened file to a variable so that it can be accessed within the program.
3. **Read** from or **write** to the file.
4. **Close** the file so that it can be used by another application or process.

Opening a file to read its contents

```
my_file = openRead("file.txt")  
OUTPUT my_file.readlines()  
my_file.close()
```

Opening a file to write (save) to

```
save_data = openWrite("save.txt")  
save_data.writeLine("Player1-3425")  
save_data.close()
```

Looping over every line in a file

```
my_long_file = openRead("story.txt")  
WHILE NOT my_long_file.endOfFile()  
    OUTPUT my_long_file.readLine()  
END WHILE  
my_long_file.close()
```

Reading text files in Python

Python 3

The basic process of reading a file in Python involves:

1. **Opening** the file in **read** mode
2. **Assigning** the opened file to a variable
3. Doing something with the file's **contents**
4. **Closing** the file once you are finished - this is a really important step!

```
# Steps 1 and 2
file = open('my_file.txt', 'r')

# Step 3
print(file.read())

# Step 4
file.close()
```

File opening modes

Python 3

Files can be opened in different **modes**:

- **'r'** - use this to **read** a file
- **'w'** - use this to **overwrite** a file
- **'a'** - use this to **append** to a file (add data to the end of the file)

Note that you use the same `open()` function no matter whether you plan to read or write to the file.

```
# open a file for reading
file = open('my_file.txt', 'r')

# open a file for overwriting
file = open('my_file.txt', 'w')

# open a file for appending
file = open('my_file.txt', 'a')
```

Doing things with a file

Python 3

Once you've opened a file for reading there are number of things that you can do with its contents:

- Read the whole file with `.read()`
- Read a certain number of bytes with `.read(bytes)`
- Read a single line of the file with `.readline()`
- Get a list that contains all of the lines in the file with `.readlines()`
- Loop through each line of the file with a for loop

```
whole_file_contents = file.read()

first_four_bytes = file.read(4)

first_line = file.readline()

file_lines = file.readlines()

for line in file:
    print(line)
    input("Press Enter to read the next
        line...")
```

Processing a file line-by-line

You will often want to process a file line-by-line, particularly if that file contains structured data such as a list of username and password combinations.

Python 3

users.txt

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.txt', 'r')  
  
username = input("Enter your username: ")  
password = input("Enter your password: ")  
  
for line in file:  
    line = line.rstrip("\n")  
    row_data = line.split(',')  
    if row_data[0] == username and row_data[1] == password:  
        logged_in = True  
  
file.close()
```


Processing a file line-by-line

You will often want to process a file line-by-line, particularly if that file contains structured data such as a list of username and password combinations.

users.txt

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.txt', 'r')
```

Open the 'users.txt' file in read mode

```
username = input("Enter your username: ")
```

```
password = input("Enter your password: ")
```

```
for line in file:
```

```
    line = line.rstrip("\n")
```

```
    row_data = line.split(',')
```

```
    if row_data[0] == username and row_data[1] == password:
```

```
        logged_in = True
```

```
file.close()
```

Processing a file line-by-line

You will often want to process a file line-by-line, particularly if that file contains structured data such as a list of username and password combinations.

users.txt

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.txt', 'r')
```

```
username = input("Enter your username: ")
```

```
password = input("Enter your password: ")
```

```
for line in file:
```

```
    line = line.rstrip("\n")
```

```
    row_data = line.split(',')
```

```
    if row_data[0] == username and row_data[1] == password:
```

```
        logged_in = True
```

```
file.close()
```

Get the user to enter their
username and password

Processing a file line-by-line

You will often want to process a file line-by-line, particularly if that file contains structured data such as a list of username and password combinations.

Python 3

users.txt

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.txt', 'r')
```

```
username = input()
```

```
password = input()
```

Work through each line in the file, line-by-line

```
for line in file:
```

```
    line = line.rstrip("\n")
```

```
    row_data = line.split(',')
```

```
    if row_data[0] == username and row_data[1] == password:
```

```
        logged_in = True
```

```
file.close()
```

Processing a file line-by-line

You will often want to process a file line-by-line, particularly if that file contains structured data such as a list of username and password combinations.

Python 3

users.txt

```
Bob,letmein
Dave23,password
Jenny99,iloveben
Mark.Brown,s3cur3
big_mike_xx,password123
```

```
file = open('users.txt', 'r')
```

```
username = input("Enter your  
password = input("Enter your
```

```
for line in file:
```

```
    line = line.rstrip("\n")
```

```
    row_data = line.split(',')
```

```
    if row_data[0] == username and row_data[1] == password:  
        logged_in = True
```

```
file.close()
```

Remove the hidden newline character (`\n`) from the end of each line and split each line wherever a comma (,) is found. Return the split parts in a list.

Processing a file line-by-line

You will often want to process a file line-by-line, particularly if that file contains structured data such as a list of username and password combinations.

Python 3

users.txt

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.txt', 'r')  
  
username = input("Enter your username: ")  
password = input("Enter your password: ")  
  
for line in file:  
    line = line.rstrip() # Strip trailing newline character  
    row_data = line.split(',')  
    if row_data[0] == username and row_data[1] == password:  
        logged_in = True  
  
file.close()
```

Test to see if the data in the first part of the line is the same as the username that was entered...

... **and** check if the second part of the line is the same as the password that was entered.

Processing a file line-by-line

You will often want to process a file line-by-line, particularly if that file contains structured data such as a list of username and password combinations.

Python 3

users.txt

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.txt', 'r')  
  
username = input("Enter your username: ")  
password = input("Enter your password: ")  
  
for line in file:  
    line = line.rstrip("\n")  
    row_data = line.split(",")  
    if row_data[0] == username and row_data[1] == password:  
        logged_in = True  
  
file.close()
```

If the username and password entered match a row in the file, assign the value True to the logged_in variable.

Processing a file line-by-line

You will often want to process a file line-by-line, particularly if that file contains structured data such as a list of username and password combinations.

Python 3

users.txt

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.txt', 'r')  
  
username = input("Enter your username: ")  
password = input("Enter your password: ")  
  
for line in file:  
    line = line.rstrip("\n")  
    row_data = line.split(',')  
    if row_data[0] == username and row_data[1] == password:  
        logged_in = True  
  
file.close()
```

Close the file once we are done with it.

The 'with' method

Python 3

The **with** keyword allows us to open the file, work with its contents and then have the final **automatically closed** when we are done with it.

This is the **best practise** way to open files in Python as it is the **safest**.

However, the pseudocode that you will see in an exam will not use the with method, instead it will use `file = open(...)` and `file.close()`.

```
with open('my_file.txt', 'r') as file:  
    print(file.read())
```


The 'with' method

Python 3

When using with you still use the **open()** function, passing the filename and file mode parameter arguments.

You also still need to specify a variable that to assign the opened file to - this is done after the **as** keyword.

```
with open('my_file.txt', 'r') as file:  
    print(file.read())
```

The 'with' method

Python 3

When using with you still use the **open()** function, passing the filename and file mode parameter arguments.

You also still need to specify a variable that to assign the opened file to - this is done after the **as** keyword.

```
with open('my_file.txt', 'r') as file:  
    print(file.read())
```

Specify the file to open and the mode within open()

The 'with' method

When using with you still use the `open()` function, passing the filename and file mode parameter arguments.

You also still need to specify a variable that to assign the opened file to - this is done after the `as` keyword.

Python 3

```
with open('my_file.txt', 'r') as file:  
    print(file.read())
```

Specify the name of the variable that will be assigned the file object after the `as` keyword.

The 'with' method

Python 3

When using with you still use the `open()` function, passing the filename and file mode parameter arguments.

You also still need to specify a variable that to assign the opened file to - this is done after the `as` keyword.

```
with open('my_file.txt', 'r') as file:  
    print(file.read())
```

Write the code that processes the file's contents within the **indented** with statement.

Writing text files in Python

Python 3

The basic process of reading a file in Python involves:

1. **Opening** the file in **write** or **append** mode
2. **Assigning** the opened file to a variable
3. **Writing** data to the file with the `.write()` method
4. **Closing** the file once you are finished - this is a really important step!

```
# Steps 1 and 2
file = open('my_file.txt', 'w')

# Step 3
file.write("Hello, there!")

# Step 4
file.close()
```

Writing text files in Python

Python 3

The basic process of reading a file in Python involves:

1. **Opening** the file in **write** or **append** mode
2. **Assigning** the opened file to a variable
3. **Writing** data to the file with the `.write()` method
4. **Closing** the file once you are finished - this is a really important step!

```
# Steps 1 and 2
file = open('my_file.txt', 'w')

# Step 3
file.write("Hello, there!")

# Step 4
file.close()
```

Strange as it may seem, you still "open" the file for writing to even if it doesn't exist yet.

By "opening" a file with a new file name that file will be created on the computer, ready for your data to be written to.

Write vs. append mode

Python 3

There is a very important distinction between **writing** a file and **appending** data to a file.

- **Writing** a file totally replaces its contents with whatever data you write to it whilst the file is open.
- **Appending** data to a file adds new data to the end of the file, keeping whatever was there before intact.

Make sure you use the correct mode when saving data! You don't want to accidentally erase all your existing data!

```
# Example of writing to the file
```

```
with open('test_file.txt', 'w') as file:  
    file.write("This is going to replace  
                everything already in the file!")
```

```
# Example of appending to the file
```

```
with open('test_file.txt', 'a') as file:  
    file.write("This is going to be added  
                onto the end of the data already  
                in the file!")
```

Saving data in a file

Python 3

- Data is written to files using the file's `.write()` method.
- This is the case whether you've opened the file in write **or** append mode!
- Simply provide the **string data** that you want to write to the file within as the *argument* to `.write()` - i.e. put it inside the brackets.
- You can pass **variables** that have been assigned string data to the `.write()` method.

```
message_to_save = "This string data is  
assigned to a variable and will be  
written to the file."  
  
with open('test_file.txt', 'w') as file:  
    file.write("This data is being  
        written directly to the file.\n")  
    file.write(message_to_save)
```


Saving data in a file

Python 3

- Data is written to files using the file's `.write()` method.

The contents of **test_file.txt**:

This data is being written directly to the file.

This string data is assigned to a variable and will be written to the file.

```
message_to_save = "This string data is  
assigned to a variable and will be  
written to the file."
```

```
with open('test_file.txt', 'w') as file:  
    file.write("This data is being  
        written directly to the file.\n")  
    file.write(message_to_save)
```

Text files are made of string data

Python 3

- Because you are saving to a **text** file, all data that you write must be of the **string** data type
- If you want to save integers or floats they must be **cast** to a string first, just like using `print()`
- You can cast with `str()` or you can drop your integer and float values in to a **formatted string**

```
score = 100    # Integer data
player_name = "Mr. X"

# Casting score to string with str()
with open('high_scores.txt', 'w') as file:
    file.write(player_name + ":" +
               str(score))

# Including score within a formatted string
with open('high_scores.txt', 'w') as file:
    file.write("{0}:{1}".format
               (player_name, score))
```

Writing files with the 'with' method

Python 3

Recall that the **with** keyword allows us to open the file, work with its contents and then have the final **automatically closed** when we are done with it.

This is the **best practise** way to open files in Python as it is the **safest**.

However, the pseudocode that you will see in an exam will not use the with method, instead it will use `file = open(...)` and `file.close()`.

```
with open('my_file.txt', 'a') as file:  
    file.write("Hello")  
    file.write("This will also be  
                appended to the file, because  
                it's indented within the  
                'with' block")
```

Writing a list of data items to a file

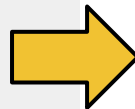
You will often want to **save a list of data items** to a file. You can easily do this using a for loop to iterate through all of the items in the list.

Python 3

```
to_do_list = ['wash car', 'make bed', 'buy groceries']

with open('todo.txt', 'w') as todo_file:
    todo_file.write("My To Do List:")
    todo_file.write("-----")
    for item in to_do_list:
        todo_file.write(item + "\n")
```

todo.txt



```
My To Do List:
-----
wash car
make bed
buy groceries
```

Writing a list of data items to a file

You will often want to save a list of data items to a file. You can easily do this using a for loop to iterate through all of the items in the list.

Python 3

```
to_do_list = ['wash car', 'make bed', 'buy groceries']
```

```
with open('todo.txt', 'w') as todo_file:
```

```
    todo_file.write("My To Do List:")
```

```
    todo_file.write("-----")
```

```
    for item in to_do_list:
```

```
        todo_file.write(item + "\n")
```

todo.txt

My To Do List:

wash car

make bed

Unlike `print()`, `write()` does not add a new line character (`\n`) to the end of each string that it writes to a file.

Therefore, if you want the next bit of data to be on a new line, you must add `\n` to the end of each string that is being written.

Using records to store data

Structured data

- Structured data refers to any data that resides in a fixed field within a record or file so that it can be easily entered, stored, queried and analyzed (source: [webopedia](#)).
- Structured data are stored as **records** made of up **fields**. For example a person could be represented as a record where the fields include their name, age and address.
- A file that contained a simple list of jobs to complete is an example of a basic form of structured data:
 - Each new "todo" item was written on a line
 - Each line comprised the whole of the "todo" item's description

Structured vs. unstructured data

Unstructured data could be any of the following:

- The text of a book
- An audio file
- Image data

Examples of structured data include:

- Lists
- Spreadsheets
- Databases
- HTML / XML documents

Complex structured data

Python 3

Imagine that instead of simply keeping a list of todo items, we wanted to store the following for each todo item:

- A description of the task
- The due date
- A priority value

We could do this in a text file by still storing all of the data about each todo item on a new line, but separating each individual item of data with a special character such as a comma.

```
Description,Due,Priority
```

```
Wash car,12/08/18,3
```

```
Clean room,12/08/18,2
```

```
Buy groceries,10/08/18,1
```

```
Do laundry,13/08/18,2
```

CSV files

The file on the right is an example of a very common file format for storing structured data - a **CSV** file.

C S V

Comma Separated Value

Python 3

```
Description,Due,Priority
```

```
Wash car,12/08/18,3
```

```
Clean room,12/08/18,2
```

```
Buy groceries,10/08/18,1
```

```
Do laundry,13/08/18,2
```

CSV file structure

Python 3

CSV files are ordinary text files that obey certain rules:

- Each new line contains all of the data about a single "thing" (**entity**) - this is a bit like a row in a spreadsheet or database
- Each line is called a **record**
- Each individual item of data that make up a record are called **fields** and are separated by **commas**
- The first row is often, but not always, used for a **header row** - basically the column or field names

```
Description,Due,Priority
```

```
Wash car,12/08/18,3
```

```
Clean room,12/08/18,2
```

```
Buy groceries,10/08/18,1
```

```
Do laundry,13/08/18,2
```

CSV file structure

Python 3

CSV files are ordinary text files that obey certain rules:

- Each new line contains about a single "thing" (**entity**) - this is a bit like a row in a spreadsheet
- Each line is called a **record**
- Each individual item of data that make up a record are called **fields** and are separated by **commas**
- The first row is often, but not always, used for a **header row** - basically the column or field names

The header row

```
Description,Due,Priority
```

```
Wash car,12/08/18,3
```

```
Clean room,12/08/18,2
```

```
Buy groceries,10/08/18,1
```

```
Do laundry,13/08/18,2
```

CSV file structure

Python 3

CSV files are ordinary text files that obey certain rules:

- Each new line contains all of the data about a single "thing" (**entity**) - this is a bit like a row in a spreadsheet
- Each line is called a **record**
- Each individual item of data that make up a record are called **fields** and are separated by **commas**
- The first row is often, but not always, used for a **header row** - basically the column or field names

A record

```
Description,Due,Priority
```

```
Wash car,12/08/18,3
```

```
Clean room,12/08/18,2
```

```
Buy groceries,10/08/18,1
```

```
Do laundry,13/08/18,2
```

CSV file structure

Python 3

CSV files are ordinary text files that obey certain rules:

- Each new line contains all of the data about a single "thing" (**entity**) - this is a bit like a row in a spreadsheet
- Each line is called a **record**
- Each individual item of data that make up a record are called **fields** and are separated by **commas**
- The first row is often, but not always, used for a **header row** - basically the column or field names

```
Description,Due,Priority
```

```
Wash car,12/08/18,3
```

```
Clean room,12/08/18,2
```

```
Buy groceries,10/08/18,1
```

```
Do laundry,13/08/18,2
```

A field

Pros and cons of CSV files

Pros

- Very simple, text-based format
- Can be read by many different applications
- Easy to use in programs

Cons

- Inefficient for large datasets
- No built-in means of sorting or searching data
- Cannot store data that cannot be represented as text (i.e. binary data such as video or audio data)
- Can be read by anyone (unless encrypted)

Processing a CSV file in Python

Processing a CSV file involves working through it line-by-line and splitting each line into a new list (or array) or items wherever a comma (,) is found.

Python 3

users.csv

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.csv', 'r')  
  
username = input("Enter your username: ")  
password = input("Enter your password: ")  
  
for line in file:  
    row_data = line.split(',')  
    if row_data[0] == username and row_data[1] == password:  
        logged_in = True  
  
file.close()
```


Processing a CSV file in Python

Processing a CSV file involves working through it line-by-line and splitting each line into a new list (or array) or items wherever a comma (,) is found.

users.csv

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.csv', 'r')
```

Open the 'users.csv' file in read mode

```
username = input("Enter your username: ")
```

```
password = input("Enter your password: ")
```

```
for line in file:
```

```
    row_data = line.split(',')
```

```
    if row_data[0] == username and row_data[1] == password:
```

```
        logged_in = True
```

```
file.close()
```

Processing a CSV file in Python

Processing a CSV file involves working through it line-by-line and splitting each line into a new list (or array) or items wherever a comma (,) is found.

users.csv

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.csv', 'r')
```

```
username = input("Enter your username: ")
```

```
password = input("Enter your password: ")
```

```
for line in file:
```

```
    row_data = line.split(',')
```

```
    if row_data[0] == username and row_data[1] == password:
```

```
        logged_in = True
```

```
file.close()
```

Get the user to enter their
username and password

Processing a CSV file in Python

Processing a CSV file involves working through it line-by-line and splitting each line into a new list (or array) or items wherever a comma (,) is found.

Python 3

users.csv

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.csv', 'r')
```

```
username = input()
```

```
password = input()
```

Work through each line in the
file, line-by-line

```
for line in file:
```

```
    row_data = line.split(',')
```

```
    if row_data[0] == username and row_data[1] == password:
```

```
        logged_in = True
```

```
file.close()
```

Processing a CSV file in Python

Processing a CSV file involves working through it line-by-line and splitting each line into a new list (or array) or items wherever a comma (,) is found.

Python 3

users.csv

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.csv', 'r')
```

```
username = input("Enter your username: ")
```

```
password = input("Enter your password: ")
```

Split each line wherever a comma (,) is found. Return the split parts in a list.

```
for line in file:
```

```
    row_data = line.split(',')
    if row_data[0] == username and row_data[1] == password:
```

```
        logged_in = True
```

```
file.close()
```

Processing a CSV file in Python

Processing a CSV file involves working through it line-by-line and splitting each line into a new list (or array) or items wherever a comma (,) is found.

Python 3

users.csv

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.csv', 'r')
```

```
username = input()
```

```
password = input()
```

Test to see if the data in the first part of the line is the same as the username that was entered...

```
for line in file:
```

```
    row_data = line.split(',')
```

```
    if row_data[0] == username and row_data[1] == password:
```

```
        logged_in = True
```

```
file.close()
```

... and check if the second part of the line is the same as the password that was entered.

Processing a CSV file in Python

Processing a CSV file involves working through it line-by-line and splitting each line into a new list (or array) or items wherever a comma (,) is found.

Python 3

users.csv

```
Bob,letmein  
Dave23,password  
Jenny99,iloveben  
Mark.Brown,s3cur3  
big_mike_xx,password123
```

```
file = open('users.csv', 'r')  
  
username = input("Enter your username: ")  
password = input("Enter your password: ")  
  
for line in file:  
    row_data = line.split(',')  
    if row_data[0] == username and row_data[1] == password:  
        logged_in = True  
  
file.close()
```

If the username and password entered match a row in the file, assign the value True to the logged_in variable.

Processing a CSV file in Python

Processing a CSV file involves working through it line-by-line and splitting each line into a new list (or array) or items wherever a comma (,) is found.

Python 3

users.csv

```
Bob,letmein
Dave23,password
Jenny99,iloveben
Mark.Brown,s3cur3
big_mike_xx,password123
```

```
file = open('users.csv', 'r')

username = input("Enter your username: ")
password = input("Enter your password: ")

for line in file:
    row_data = line.split(',')
    if row_data[0] == username and row_data[1] == password:
        logged_in = True

file.close()
```

Close the file once we are done with it.

Writing data to a CSV file

Pseudocode

The first line of data written to a CSV file should be the **header row** that states the names of the fields in each record (row).

Following this, each line of the CSV file should contain the **data values for each field** in the record, **separated by a comma**. A **newline character** must be written at the **end** of each record.

If records are being appended (added) to an existing CSV file then there is no need to write the header row again.

```
# Define data to store as a 2D array
people_records = [4,3]
people_records[0] = ['Dan',32,'M']
people_records[1] = ['Joey', 18, 'F']
people_records[2] = ['Mike', 42, 'M']
people_records[3] = ['Tammy', 24, 'F']
# Open the CSV file for writing
csv_file = openWrite('data.csv')
# Write the header row
csv_file.write("name,age,gender\n")
# Use a FOR loop to write each record in the 2D array to the CSV file
FOR i = 0 to 3
    csv_file.write(people_records[i][0] + "," +
        people_records[i][1] + "," + people_records[i][2] + "\n")
NEXT i
# Close the CSV file
csv_file.close()
```

Revisit two-dimensional arrays if this example doesn't make sense!

Writing data to a CSV file

Python 3

people_records[]

Pupil name	Age	Gender
Dan	32	M
Joey	18	F
Mike	42	M
Tammy	24	F

data.csv

```
name,age,gender
Dan,32,M
Joey,18,F
Mike,42,M
Tammy,24,F
```

```
# Define data to store as a 2D list
people_records = []
people_records.append(['Dan',32,'M'])
people_records.append(['Joey', 18, 'F'])
people_records.append(['Mike', 42, 'M'])
people_records.append(['Tammy', 24, 'F'])

# Open the CSV file for writing
csv_file = open('data.csv', 'w')

# Write the header row
csv_file.write("name,age,gender\n")

# Use a for loop and string formatting to write each record in
the 2D list to the CSV file
for person in people_records:
    csv_file.write("{0},{1},{2}\n".format(person[0],
                                           person[1], person[2]))

# Close the CSV file
csv_file.close()
```

Revisit two-dimensional arrays if this example doesn't make sense!

Searching for data with SQL

Databases

- A database is a **computerised system** for **storing data** in an **organised way**.
- Using a database makes it easier to search for data based on certain criteria.
- There are two basic types of database, **flat file** and **relational** databases.
- CSV files are a form of **flat file database** - where all the data is stored in a single file.

Relational Databases

- CSV files are **inefficient** for storing large amounts of data
- Organisations that manage a lot of data use **Relational Databases** instead
- A **Relational Database** stores data across different **tables**, each table storing data about a single type of **entity**
- For example, a library might have the following tables in its database:
BOOK, AUTHOR, LOAN, MEMBER

A database table

Table name: BOOK

ID	Title	Author	Genre	Format
1	Mary Berry: Foolproof Cooking	Mary Berry	4	2
2	Harry Potter and the Cursed Child Parts 1 & 2	J.K. Rowling	2	2
3	Me before You	Jojo Moyes	3	1
4	The Revenant	Michael Punke	2	3

A database table

Table name: BOOK

A **record** within the table -
each row is a record.

ID	Title	Author	Genre	Format
1	Mary Berry: Foolproof Cooking	Mary Berry	4	2
2	Harry Potter and the Cursed Child Parts 1 & 2	J.K. Rowling	2	2
3	Me before You	Jojo Moyes	3	1
4	The Revenant	Michael Punke	2	3

A database table

Table name: BOOK

A **field** within a record - each field stores an individual piece of data about the record.

ID	Title	Author	Genre	Format
1	Mary Berry: Foolproof Cooking	Mary Berry	4	2
2	Harry Potter and the Cursed Child Parts 1 & 2	J.K. Rowling	2	2
3	Me before You	Jojo Moyes	3	1
4	The Revenant	Michael Punke	2	3

A database table

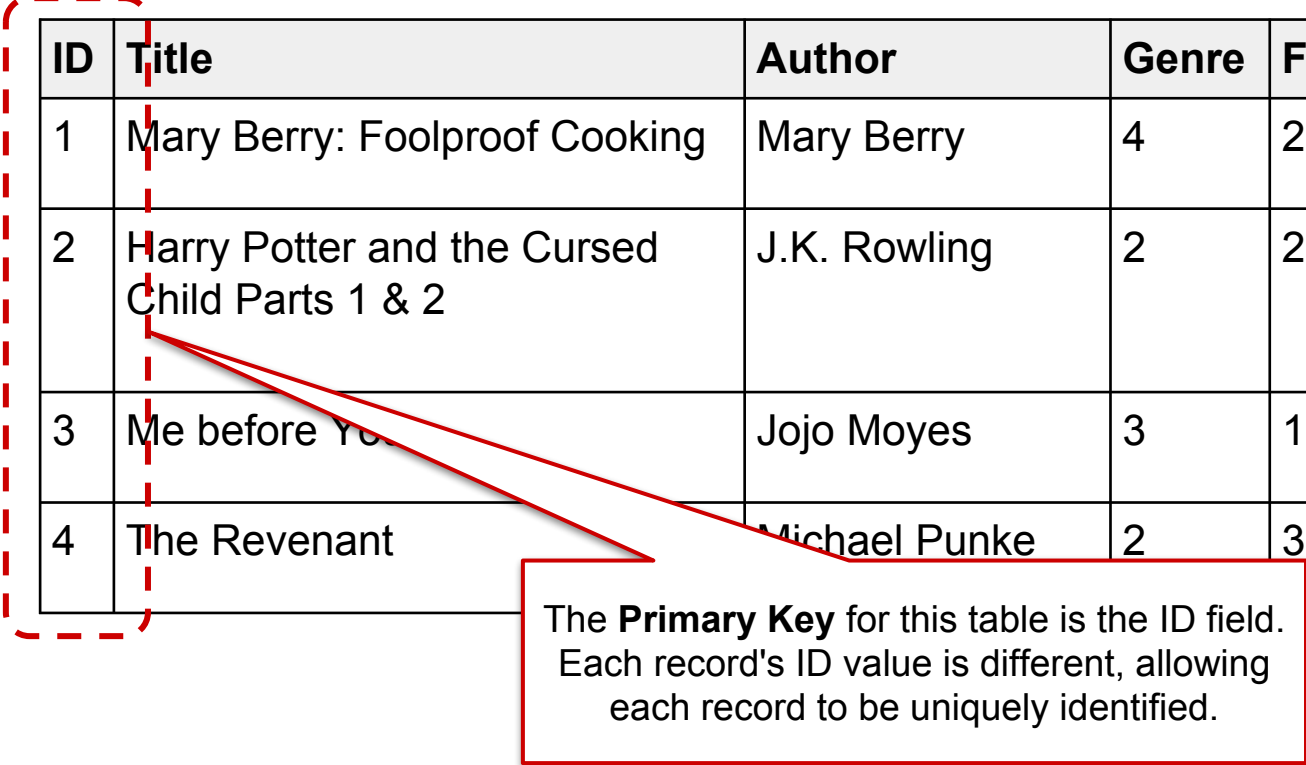
A **table** within a database - contains all of the records for that type of **entity** (thing)

Table name: BOOK

ID	Title	Author	Genre	Format
1	Mary Berry: Foolproof Cooking	Mary Berry	4	2
2	Harry Potter and the Cursed Child Parts 1 & 2	J.K. Rowling	2	2
3	Me before You	Jojo Moyes	3	1
4	The Revenant	Michael Punke	2	3

A database table

Table name: BOOK



ID	Title	Author	Genre	Format
1	Mary Berry: Foolproof Cooking	Mary Berry	4	2
2	Harry Potter and the Cursed Child Parts 1 & 2	J.K. Rowling	2	2
3	Me before You	Jojo Moyes	3	1
4	The Revenant	Michael Punke	2	3

The **Primary Key** for this table is the ID field. Each record's ID value is different, allowing each record to be uniquely identified.

A database table

Table name: BOOK

ID	Title	Author	Genre	Format
1	Mary Berry: Foolproof Cooking	Mary Berry	4	2
2	Harry Potter and the Cursed Child Parts 1 & 2	J.K. Rowling	2	2
3	Me before You	Jojo Moyes	3	1
4	The Fault in Our Stars	John Green	2	3

The Genre and Format fields are examples of **foreign keys** - their values are primary keys (ID numbers) in the Genre and Format tables in the database

Relationships between tables

- Every table in the database has a field (column) called the **Primary Key**.
- This field contains a value that **uniquely identifies each record** (row) within the table. Very often this is an ID number.
- The primary keys are used to **associate** records from different tables with each other.

Relationships between tables

BOOK table

ID	Title	Author	Genre	Format
1	Mary Berry: Foolproof Cooking	Mary Berry	4	2
2	Harry Potter and the Cursed Child Parts 1 & 2	J.K. Rowling	2	2
3	Me before You	Jojo Moyes	3	1
4	The Revenant	Michael Punke	2	3

The BOOK table is related to the FORMAT table by the BOOK.Format field and the FORMAT.ID field

FORMAT table

ID	Description
1	Paperback
2	Hardback
3	Audiobook
4	eBook

GENRE table

ID	Description
1	Fiction: Crime
2	Fiction: Adventure
3	Fiction: Romance
4	Non-fiction: Cookery

The BOOK table is related to the GENRE table by the BOOK.Genre field and the GENRE.ID field

Database keywords

- **Table** - a collection of all of the records about a type of thing (e.g. people, students, cars, etc)
- **Record** - all of the related data about a single thing (e.g. a person). Each row in a database table represents a record.
- **Field** - an individual piece of information that is stored about a thing, e.g. name or age for a person. Fields form the columns in a database table.
- **Primary Key** - a field that uniquely identifies each record within a table. This is often an auto-generated ID number.

Queries

- When we want to get data from the database we send the database a **query**.
- A query is like a question, for example "Can you show me all the books by J. K. Rowling?" or "Can you show me all of your audiobooks?".
- To run queries on a relational database we write commands in a special database language called **Structured Query Language (SQL)** for short).

Structured Query Language (SQL)

- SQL (pronounced "sequel" by some and "ess-cue-el" by others) is a language used by databases to tell them what to do.
- SQL can be used for selecting, inserting, updating, deleting data as well as creating, modifying and deleting tables.
- For GCSE you only need to know how to **select** (search for and retrieve) data across multiple tables.
- You can try out any of the following SQL SELECT statements on a live database here:
https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
 - Please note that you will have to adjust the SELECT statements that you write to reflect the tables and fields in the live database!

The SELECT statement

To select **all of the data** within a table in a database you can write a SELECT statement like this:

```
SELECT * FROM table_name
```

The * means "all fields" and *table_name* is the name of the table you'd like to get the data from.

For example:

```
SELECT * FROM book
```


SELECT specific fields

To select **specific fields** for each record within a table in a database you can write a SELECT statement like this:

```
SELECT field1, field2, field3... FROM table_name
```

For example, to see the ID numbers and titles of all books in the BOOK table, we could write:

```
SELECT ID, Title FROM book
```

Filtering results with WHERE

To filter the results of a SELECT statement we can add the WHERE statement like this:

```
SELECT field1, field2, field3... FROM table_name WHERE field=value
```

For example, to see all of the details for every book by J. K. Rowling we could write:

```
SELECT * FROM book WHERE author='J. K. Rowling'
```

Using WHERE with relational operators

We can use other relational operators such as $<$, $>$, $<=$, $>=$, $!=$ with WHERE statements.

For example, to see books that cost £5 or more, we could write:

```
SELECT * FROM book WHERE price >= 5.00
```

Using WHERE with multiple values

You can filter on multiple values using AND and OR after WHERE:

```
SELECT * FROM table_name WHERE field1=value AND field2=value
```

or

```
SELECT * FROM table_name WHERE field1=value OR field2=value
```

For example, to see the titles of books by J. K. Rowling or Mary Berry we could write:

```
SELECT title FROM book WHERE author='J. K. Rowling' OR  
author='Mary Berry'
```

Searching with WHERE...LIKE

Sometimes we don't know exact values to filter fields by. In this case we can use LIKE instead of = :

```
SELECT * FROM table_name WHERE field LIKE %value%
```

For example, to see all of the books that **contain** the word "the" in the title we could write:

```
SELECT * FROM book WHERE title LIKE '%the%'
```

The % symbol means "anything", so %the% will find titles with any characters before 'the' and any characters afterwards

Searching with WHERE...LIKE

Sometimes we don't know exact values to filter fields by. In this case we can use LIKE instead of = :

```
SELECT * FROM table_name WHERE field LIKE %value%
```

If we wanted to search for book titles that **started** with a particular letter we could use:

```
SELECT * FROM book WHERE title LIKE 'c%'
```

In this case, we would find all book titles that start with the character 'c'.

Sorting results with ORDER BY

You can sort results into ascending or descending order by adding the ORDER BY statement to the end of a SELECT statement:

```
SELECT * FROM table_name WHERE field LIKE %value% ORDER BY field  
ASC
```

or

```
SELECT * FROM table_name WHERE field LIKE %value% ORDER BY field  
DESC
```

Sorting results with ORDER BY

For example, to sort the list of books by author (A-Z) you would write:

```
SELECT * FROM book ORDER BY Author ASC
```

or to sort a list of book by price, with highest price first, write:

```
SELECT * FROM book ORDER BY price DESC
```


SELECT across multiple tables

To complete more complex searches, we often have to select across more than one table.

When we do this we need to use a WHERE statement to **join** the two tables on the fields that contain the primary and foreign keys:

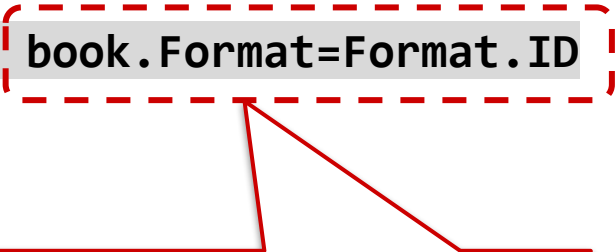
```
SELECT table1.field1, table2.field2 FROM table1, table2 WHERE  
table1.field=table2.id_field AND other_search_criteria
```

SELECT across multiple tables

```
SELECT table.field... FROM table1, table2 WHERE  
table1.field=table2.id_field AND other_search_criteria
```

For example, if we wanted to select the titles of all hardback books we could write the following statement:

```
SELECT book.Title FROM book, format WHERE book.Format=Format.ID  
AND Format.Description='Hardback'
```



This WHERE condition **joins** the two tables by looking for matching values in the Format field from the Book table and the ID (primary key) field from the Format table.

Comparing Relational and Flat File Databases

Remember that a Flat File Database is a **single file** that stores structured data, such as a CSV file or an Excel Spreadsheet.

	Flat File Databases	Relational Databases
Pros	✓ A simple format that is easy to share ✓ Can be read by many applications ✓ Easy to write programs to read and write data	✓ Data is split across tables ✓ This reduces data duplication and results in more efficient storage of data ✓ Database software provides features to allow data to be indexed, searched and extracted quickly via queries
Cons	– Not good for large collections of data as there is no built-in searching/filtering – There is potential for data duplication as data is not split into multiple tables, making them inefficient	– More complex to program for – Special database software is required – Need to learn SQL commands to access data



PRODUCING ROBUST PROGRAMS

CHAPTER 11

Defensive design considerations

What is defensive design?

Defensive design is an approach to software development that aims to produce **robust** and **reliable** programs by:

- **Anticipating misuse** of the program and planning for things that could go wrong
- Writing **maintainable code** that helps other diagnose problems if they arise
- **Testing the program** with suitable test data to identify and fix errors before it is put into general use

Adopting a defensive design approach can help prevent bugs in programs from causing unwanted consequences, such as deleting important data or being exploited by cybercriminals to take over (hijack) the host machine.

Robust vs. reliable programs

- **Robust programs** can handle unexpected inputs and data without crashing and without being tricked into malicious action.
- **Reliable programs** will work as intended under many different conditions or situations.
- We can confirm that programs are both robust **and** reliable through well-planned **testing**.

Video: Fallout from the British Airways system failure



Video by [CNNMoney](#)

Anticipating misuse

There are a number of ways that a system could be misused that a developer should consider:

- **Malicious inputs** could be used in an attempt to hijack a system - input sanitisation and validation helps to protect against this.
- An attempt to **remotely connect** to the system through remote-access systems (such as SSH). Using a **firewall** to prevent access from outsiders will help with this.
- Attempting to use **lower-level access rights** to gain access to sensitive information. User authentication should be used to combat against this.
- **Brute-force attacks** to gain access via passwords or PIN numbers - enforcing a maximum failed attempts policy will help protect against this.

Input sanitisation and validation

One of the biggest causes of error when running a program is users or other systems providing unexpected data.

There are two ways to help prevent this from causing errors:

- Input **sanitisation** - 'cleansing' input data by removing malicious characters or special command before passing it on to be processed by the program
- Input **validation** - checking that the input data meets certain criteria (length, format, range, etc) before being passed on for further processing.

Input sanitisation and validation are often **used together** to form an effective barrier against **anticipated misuse** of a system.

Input sanitisation

- Input sanitation involves the **removal of unwanted characters** from input data.
- This is a chief line of defence against SQL injection, where the characters ' , " , ; , = and - can all be used to manipulate the SQL query being processed by the webserver.
- Any data entered by users that becomes part of an SQL query (e.g. their username and password, credit card details or address) should undergo sanitisation to ensure these characters are removed.

Input validation

- Input validation is the process of checking that input data **meets certain criteria** before allowing it to be accepted by the program.
- If a user's input is rejected for being invalid they should be informed and asked to enter it again, ideally with some guidance on what criteria their input must meet.
- There are a number of **different validation checks** that can be made including a format check, range check, presence check, type check and length check.
- A classic example of a validation check is online services setting criteria for user passwords such as a minimum length or number of special characters or requiring that certain data is included within an online form.

Types of validation check

Type check

Tests the type of the data entered, for example whether it is integer, float (real), Boolean or string data.

Range check

Tests that the data falls within a permitted range. For numeric data, this might be "greater than or equal to 1 and less than or equal to 100". For string data this could be a range of permitted input options such as the a letter that corresponds to a menu option or "Y" or "N" at a "yes/no" prompt.

Presence check

Checks whether required data is present or not. This is often used on online forms to check for essential information such as a customer's credit card expiration date.

Types of validation check

Format check

Tests the data to check that it meets a set format. For example, if the input field was a customer's email address then a check would be made to ensure that it contained an @ symbol and at least one full stop (though not as the first or last character).

Length check

Tests that the number of characters entered is within a permitted range. Password validation often uses this to ensure that customer passwords are at least X characters long.

Authentication

- Authentication is the process of checking or confirming that a particular **person or system** (if accessing via an API) is **authorised** (permitted) to use a system and, if so, which parts of that system they can use.
- The most common form of authentication is to require that users enter a **username** and **password** when logging into a system.
- Once a user has logged into a system they should be able to see data relevant to themselves and no one else.
- Modern systems might implement other authentication methods other than a username and password such as **fingerprint scanning** or **facial recognition**. These are potentially more secure than a username/password combination and result in less user error (you cannot forget your fingerprint!).

Planning for contingencies

It is impossible to consider all of the different ways that users might misuse a program or interpret instructions, however there are ways to help reduce errors:

- Provide **helpful prompt messages** - for example, when you need the user to enter their name, present the message "Please enter your name:"
- Provide users with a way to **recover from problems**, for example a 'forgot password' link or an 'undo' feature.
- **Prevent access** to some parts of a system to **unauthorised users**.
- Use **exception handling** to trap and respond to unexpected inputs rather than allowing your program to crash.

Exception handling

Rather than leaving the Python interpreter to crash when it encounters a situation where it doesn't know what to do, we can use exception handling to give the interpreter another block of code to run in times of trouble. This is done using try and except blocks.

```
a = input("Enter a number: ")

try:
    a = int(a)
    b = a / 20
    print("The answer is:", b)
except:
    print("You didn't enter a number!")
```

```
Enter a number: b
You didn't enter a number!
>>> |
```

Program output - the code in the 'except' block is run when an input is given that cannot be cast to an Integer with int().

Exception handling

Exceptions are events that arise in a program when as it crashes. These can be 'caught' and 'handled' rather than allowing the program to crash.

If you have ever attempted to perform mathematical operations on string data - such as that which comes from `input()` in Python - you've probably encountered a "TypeError" exception that can be thrown when a mathematical operation is attempted on user input that is still in String form:

```
a = input("Enter a number")
Enter a number20
>>> b = a / 20
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    b = a / 20
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

Maintainability of code

Maintainability of code

A key aspect to defensive design is to ensure that all code is maintainable.

Maintainable code is code that can be read, understood and fixed by other developers. There are a few best-practise techniques to help write maintainable code, including:

- Using **comments** to explain complex parts of the code
- **indenting** blocks of code so that related functionality is grouped together
- Using **sensible identifiers** for variables, constants and subroutines
- **Structuring code** through the use of **subroutines**
- Using **subroutine parameters and return values** instead of global variables. This prevents one subroutine from changing the value of a global variable in a way that other subroutines hadn't anticipated.

Types of error

Types of error

There are three main types of error that occur when writing and running software:

- Syntax errors
- Logic errors
- Runtime errors (not on the specification)

Syntax errors

- Mistakes in the way the code is **written**.
- Due errors in the statements used in the code.
- Syntax errors **prevent programs from running** as a translator requires code to be syntactically correct.

```
4 # Syntax error
5 prnt("Hello there! This is a \
6 syntax error!")
```

```
Traceback (most recent call last):
  File "python", line 5, in <module>
NameError: name 'prnt' is not defined
> █
```

Logic errors

- Mistakes in the **logic** of the program.
- **Programs will run**, but their **outcome will not be as intended**.
- Difficult to spot as the translator will not detect them.

```
7 # Logic error
8 option = input("Would you like to quit
    (y/n)? ")
9 if option.lower() == "n":
10     print("Goodbye!")
11     quit()
12 else:
13     main_menu()
```

```
Python 3.6.1 (default, Dec 2015, 13:0
[GCC 4.8.2] on linux
>
Would you like to quit (y/n)? n
Goodbye!
```


Runtime errors

Not on the OCR specification

- Errors that occur when the **program is running**.
- Usually caused by **unhandled inputs**, receiving **unexpected or corrupted data** or because a system has **run out of memory**.
- These errors can only be detected **whilst a program is running**.

```
17 # Runtime error
18 num1 = int(input("Enter a number: "))
19 print("Twice the value of that number
    is {}".format(num1 * 2))
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
Enter a number: f
Traceback (most recent call last):
  File "python", line 18, in <module>
ValueError: invalid literal for int() with
base 10: 'f'
> █
```

Activity - identify the errors

Look at the code on the right

Can you spot the errors?

There are three to find!

```
01 num1 = input("Enter a number")
02 num2 = input("Enter another number)
03 sum = int(num1) + int(Num2)
04 print(summ)
```

Activity - identify the errors

Look at the code on the right

Can you spot the errors?

There are three to find!

```
01 num1 = input("Enter a number")
02 num2 = input("Enter another numberr)
03 sum = int(num1) + int(Num2)
04 print(summ)
```

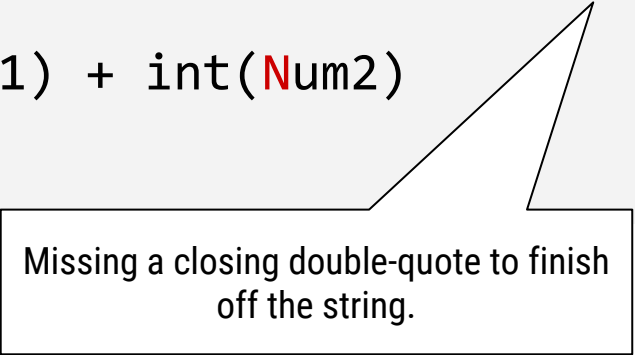
Activity - identify the errors

Look at the code on the right

Can you spot the errors?

There are three to find!

```
01 num1 = input("Enter a number")
02 num2 = input("Enter another numberr)
03 sum = int(num1) + int(Num2)
04 print(summ)
```



Missing a closing double-quote to finish off the string.

Activity - identify the errors

Look at the code on the right

Can you spot the errors?

There are three to find!

```
01 num1 = input("Enter a number")
02 num2 = input("Enter another numberr)
03 sum = int(num1) + int(Num2)
04 print(summ)
```

Typing error - a capital N instead of lowercase n for num2

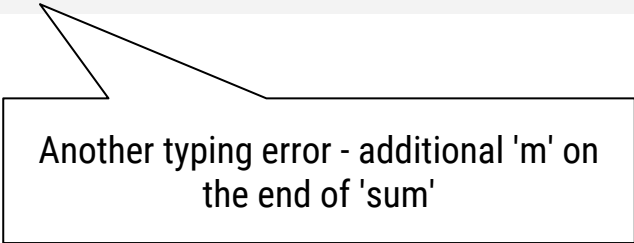
Activity - identify the errors

Look at the code on the right

Can you spot the errors?

There are three to find!

```
01 num1 = input("Enter a number")
02 num2 = input("Enter another numberr)
03 sum = int(num1) + int(Num2)
04 print(summ)
```



Another typing error - additional 'm' on the end of 'sum'

Testing

The purpose of testing

- Errors in software are often called **bugs**.
- Bugs are **mistakes** - they are not deliberate attempts to bring down systems; they should not be confused with **viruses** and other **malware**.
- The purpose of testing is to check that a program:
 - Works as intended under good conditions
 - Copes with errors when given bad data
- Different types of testing takes place at different points in the development and deployment of a solution including performance and load testing, usability testing, functionality testing and security testing.

Video: When systems aren't tested sufficiently



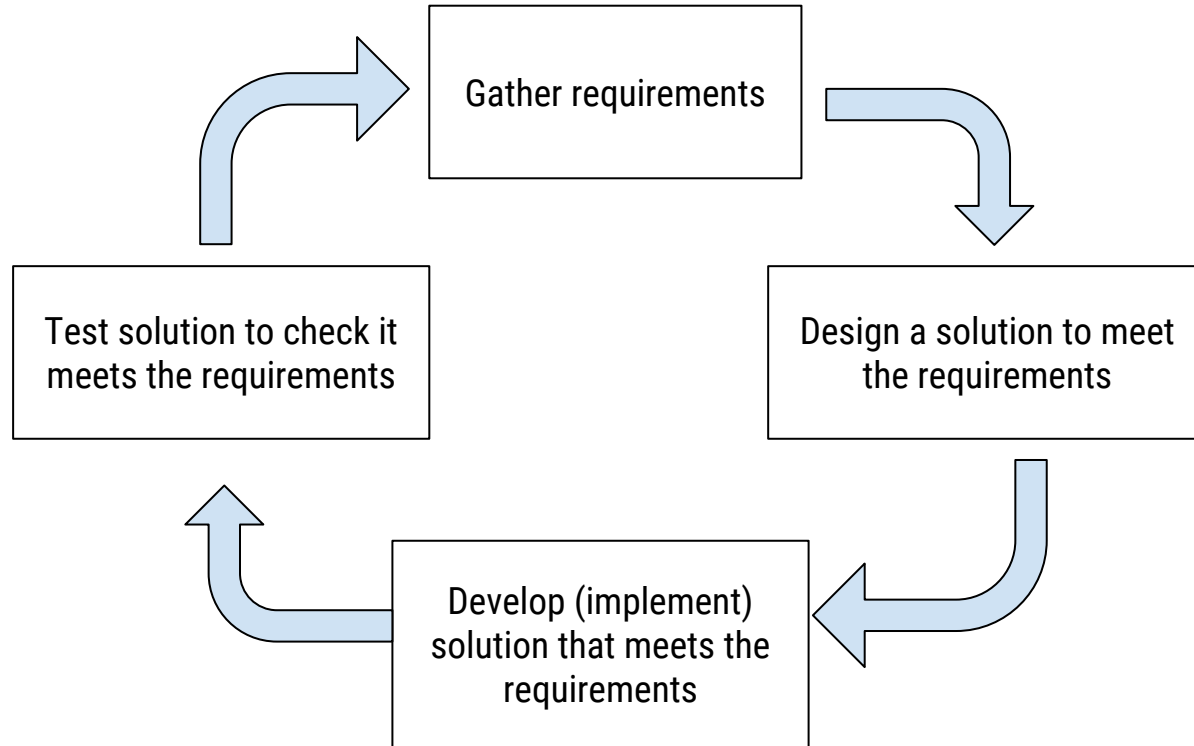
Video by [Mashable](#)

Types of testing

Testing should be performed both continually throughout the development of a product and once the product has been finished, prior to being released.

- **Iterative testing** involves testing code as it is written. The results of this continual testing feed back into the continued development of the solution.
- **Final** (or **terminal**) testing occurs at the end of development and is used to check how the software performs when given a predefined set of **test data**.

The iterative development cycle



Test data

To ensure that testing is thorough, both **expected** and **unexpected** data should be used in tests.

There are several types of test data:

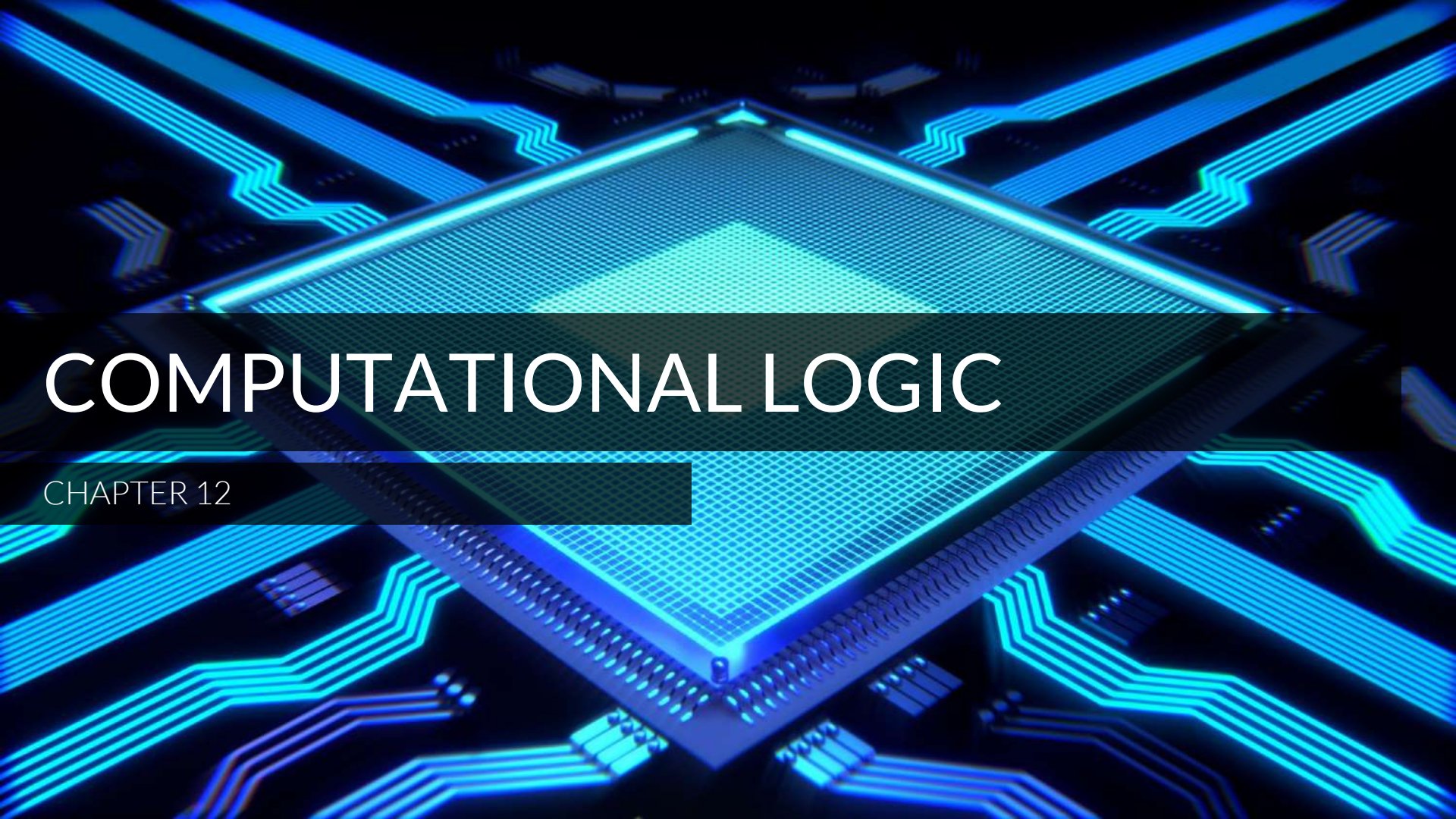
- **Valid data** (normal data) - data that is correct, within range and of the right format.
- **Extreme data** - data that is just on the edge of the valid range.
- **Invalid data** - data that is not within the acceptable range or of an incorrect format.
- **Null data** - when no data is entered to see what happens.
- **Erroneous data** - data that of the wrong type altogether, for example entering 'Bob' instead of a numerical ID number.

Test plans

- Test plans ensure that testing is effective by detailed exactly **which tests** will be performed, what **test data** will be used and what the system's **intended output/outcome** should be.
- A test plan should be **created before development** - it's part of the design stage.
- Each test on the test plan should refer to one of the **initial requirements** or **success criteria** identified during the analysis (requirements gathering) stage of the project.
- When testing is conducted, a "pass" or "fail" should be recorded against each test in the plan. Tests that fail should be followed up and the problems fixed before retesting to confirm that there is no longer an issue.

An example test plan

Test #	Purpose of test	Test data	Test data type	Expected outcome
1	Ensure that users can only select options 1, 2 or 3 at the Main Menu	2	Valid	Program will proceed to selected option's subroutine
2		3	Extreme	Program will proceed to selected option's subroutine
3		5	Invalid	Program will show error message and ask user to make their selection again
4		B	Erroneous	Program will show error message and ask user to make their selection again

A glowing blue microchip is centered on a circuit board, with intricate circuit traces radiating outwards. The chip has a grid-like pattern on its surface and is surrounded by a network of glowing blue lines representing the circuitry.

COMPUTATIONAL LOGIC

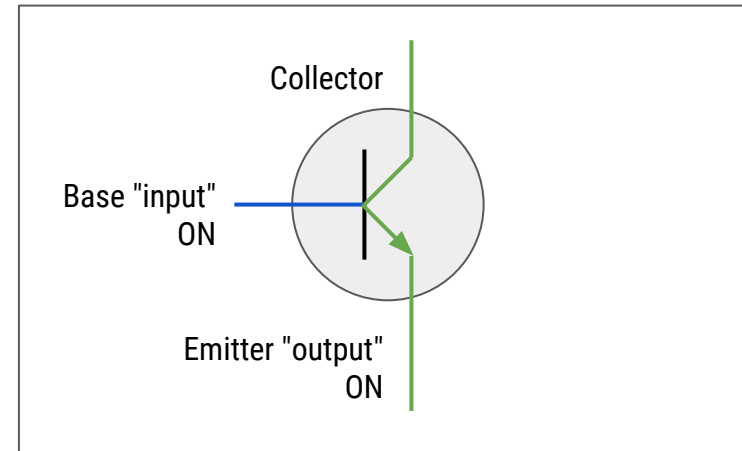
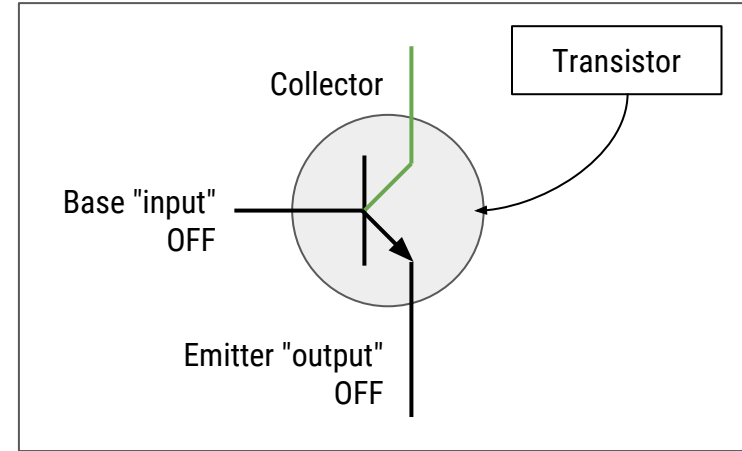
CHAPTER 12

Why is data represented in binary form?

- Computer systems store and process data as **binary digits** (represented as 0s and 1s)
- This is because, underneath the surface, computers are **electrical devices** that **process information** by **controlling the flow of electricity** through billions of wires and switches (called **transistors**) inside the CPU, memory chips and between components.
- The flow of electricity down a single cable or through a switch can be represented in binary as a 1, whereas a cable or switch without electricity flowing through it can be represented as a 0.
- As an example, an 8-bit binary number such as 01101101 represents the flow of electricity through a group of 8 wires or, if the data is stored in memory, the state of 8 individual transistors.
- So, **the bits in a binary number represent the flow of electricity through individual wires and switches** within the components of a computer system.

Transistors

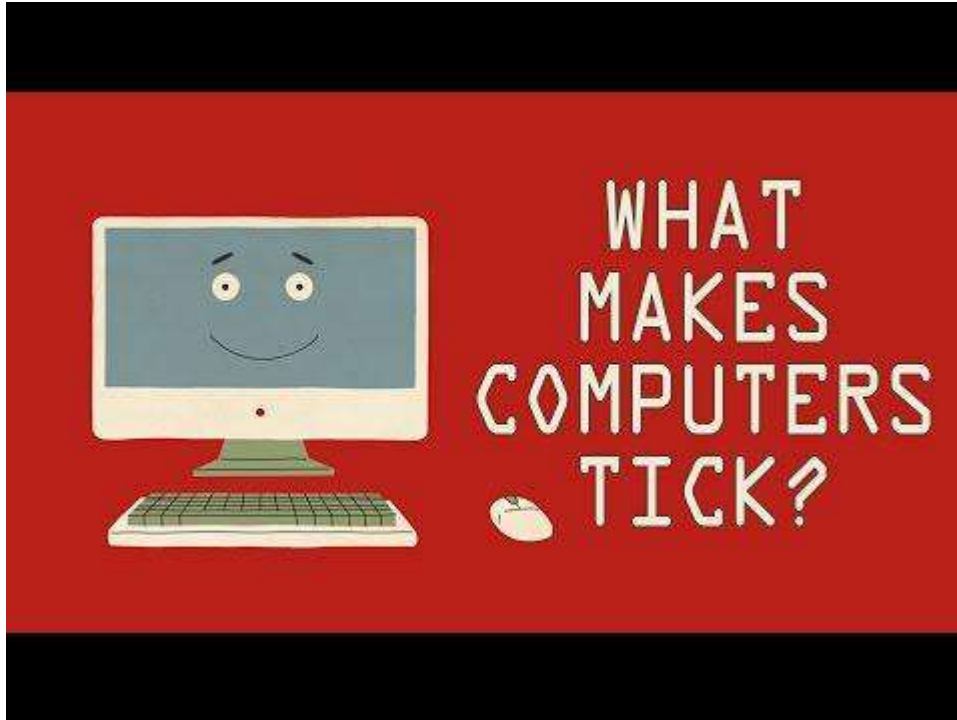
- Transistors are electrical components made from **semiconductor** materials that can act as **electrically-controlled switches**.
- A common way to think of transistors is that they are like taps. When a tap is turned on, water coming into the tap is allowed to flow through the tap and to come out.
- In a transistor, when the "base" input on a transistor has a small current flowing through it, the larger current entering the transistor from the "collector" can flow through to the "emitter".



Transistors in a CPU

- Inside a CPU, transistors are arranged and configured into **circuits** to **perform operations** such as calculations when electricity is passed through them.
- For example, transistors can be arranged to form an "adder" circuit that can accept two binary numbers and output the sum of their values.
- Modern CPUs contain over 3 billion transistors, allowing them to perform billions of calculations per second.
- **Moore's Law** famously stated that the number of transistors within an integrated circuit (or microchip) would double every 18-months to 2 years - a prediction that proved remarkably accurate for most of the history of the development of modern computer systems.

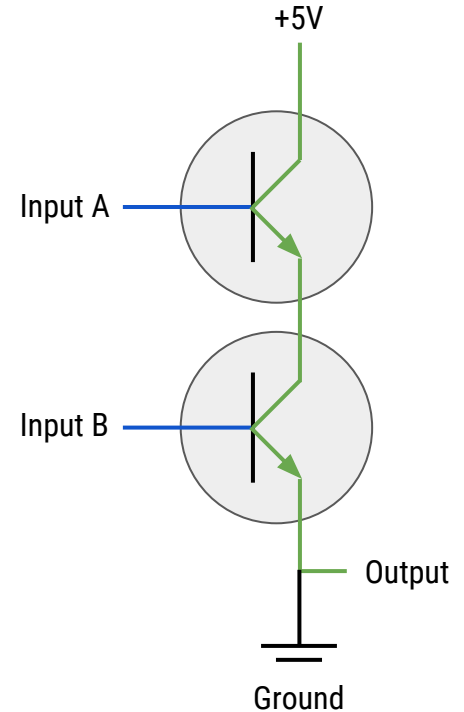
Video: What makes computers tick?



Video by [TED-Ed](#)

Logic gates

- Multiple transistors can be arranged together to form more complex "switches" that only allow electricity to flow through if all or some of the transistors are "on".
- Such arrangements of transistors are called **logic gates** - they act like gates that only open if certain logical conditions are met.
- There are three types of logic gate that you must be familiar with at GCSE - **AND**, **OR** and **NOT** logic gates.

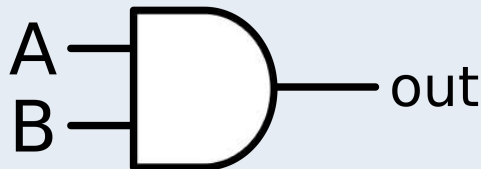


An AND logic gate made from combining two transistors. Electricity will only flow out of the output if both Input A **and** Input B are on.

AND gate

- AND gates have **two inputs** and **one output**
- An AND gate will **only output current** (represented as a 1 on the output) if **both inputs** are on (also represented as 1s)
- AND gates are represented in Boolean equations using the symbol $\wedge$ (like an upside-down V)
- The logical statement "A and B" is represented as $A \wedge B$

AND gate circuit symbol



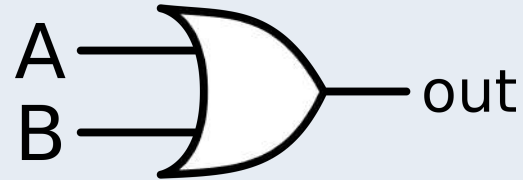
AND gate truth table

Input A	Input B	Output Q
OFF (0)	OFF (0)	OFF (0)
OFF (0)	ON (1)	OFF (0)
ON (1)	OFF (0)	OFF (0)
ON (1)	ON (1)	ON (1)

OR gate

- OR gates have **two inputs** and **one output**
- An OR gate will **output current** (represented as a 1 on the output) if **either or both inputs** are on (also represented as 1s)
- OR gates are represented in Boolean equations using the symbol $\vee$
- The logical statement "A or B" is represented as $A \vee B$

OR gate circuit symbol



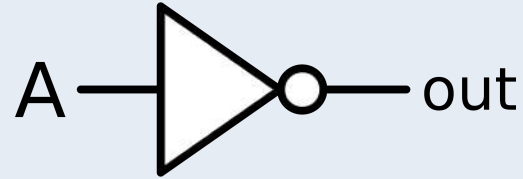
OR gate truth table

Input A	Input B	Output Q
OFF (0)	OFF (0)	OFF (0)
OFF (0)	ON (1)	ON (1)
ON (1)	OFF (0)	ON (1)
ON (1)	ON (1)	ON (1)

NOT gate

- NOT gates have **one inputs** and **one output**.
- NOT gates **invert** the input so their output is the opposite of the input given.
- NOT gates are represented in Boolean equations using the symbol $\neg$
- The logical statement "not A" is represented as $\neg A$

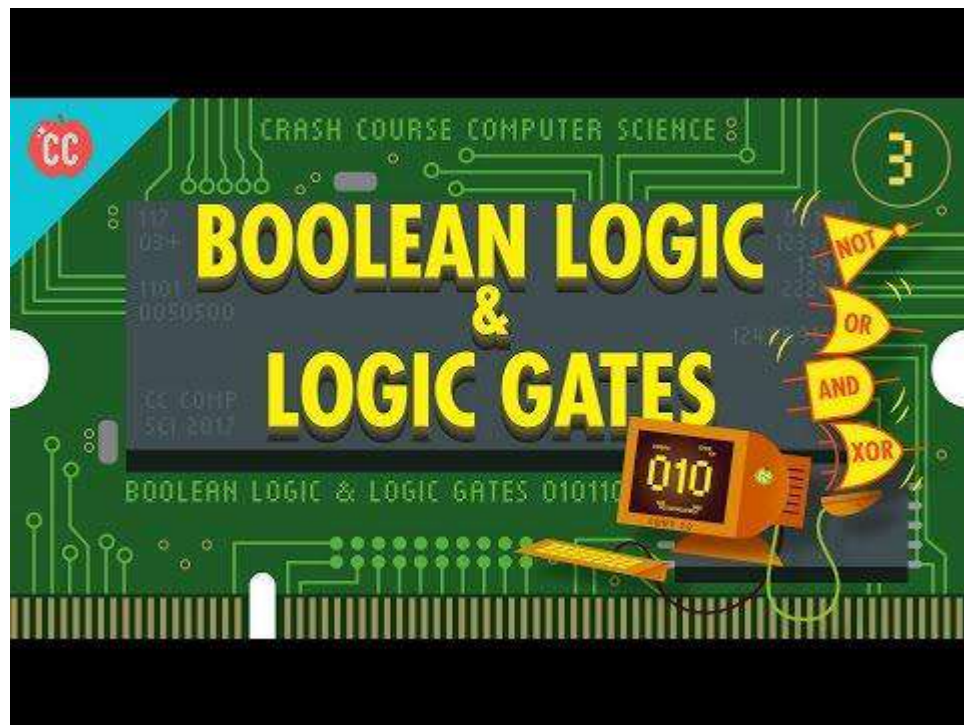
NOT gate circuit symbol



NOT gate truth table

Input A	Output Q
OFF (0)	ON (1)
ON (1)	OFF (0)

Video: Boolean Logic and Logic Gates

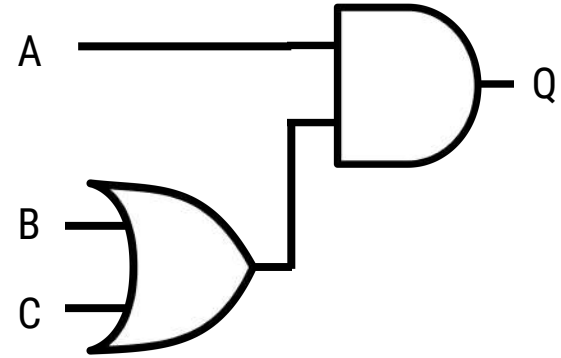


Video by [CrashCourse](#)

Evaluating Boolean Logic

Logic circuits

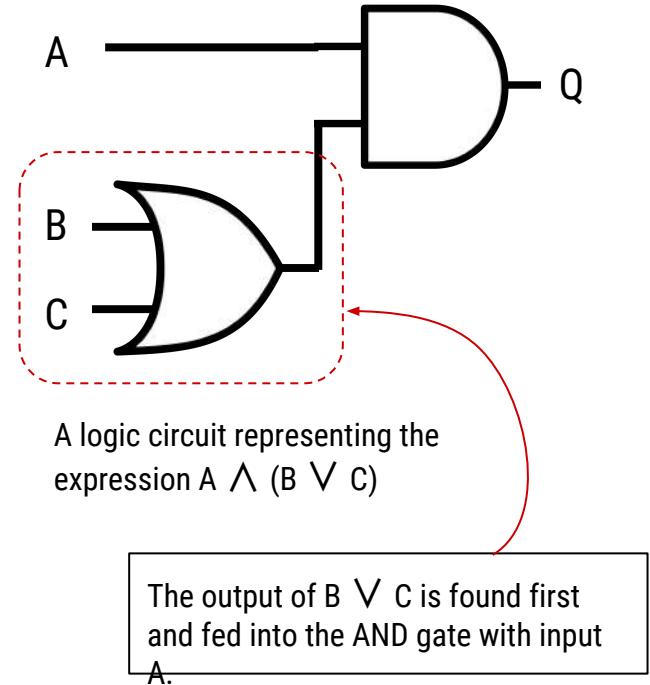
- Logic gates can be combined to form **logic circuits** that process complex logical statements such as A AND (B OR C).
- The output of logic statements in **brackets** become input to 'outer' logical operations.
- This is just the same as in maths where expressions in brackets must be evaluated first and the result used in a wider expression.
- Conventionally, **input labels start at A** (A, B, C, etc.) whereas **output labels start at Q** (Q, R, S, etc.).



A logic circuit representing the expression $A \wedge (B \vee C)$

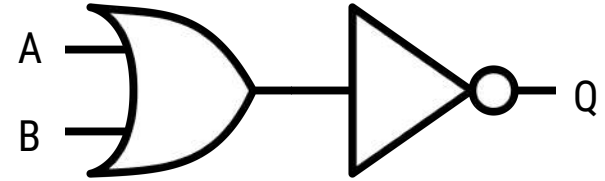
Logic circuits

- Logic gates can be combined to form **logic circuits** that process complex logical statements such as $A \text{ AND } (B \text{ OR } C)$.
- The output of logic statements in **brackets** become input to 'outer' logical operations.
- This is just the same as in maths where expressions in brackets must be evaluated first and the result used in a wider expression.
- Conventionally, **input labels start at A** (A, B, C, etc.) whereas **output labels start at Q** (Q, R, S, etc.).



Evaluating NOT expressions

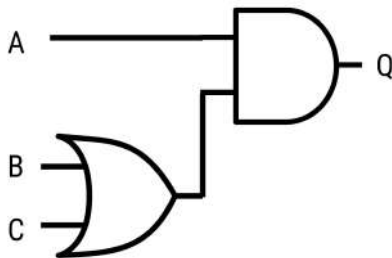
- NOT gates are applied to the output of the expression that they are inverting.
- For example, NOT (A OR B) is shown on the right.
- Again, the rule holds true: evaluate the expression in brackets and pass it as an input to the outer logic gate.



A logic circuit representing the expression $\neg(A \vee B)$

Evaluating logic circuits with truth tables

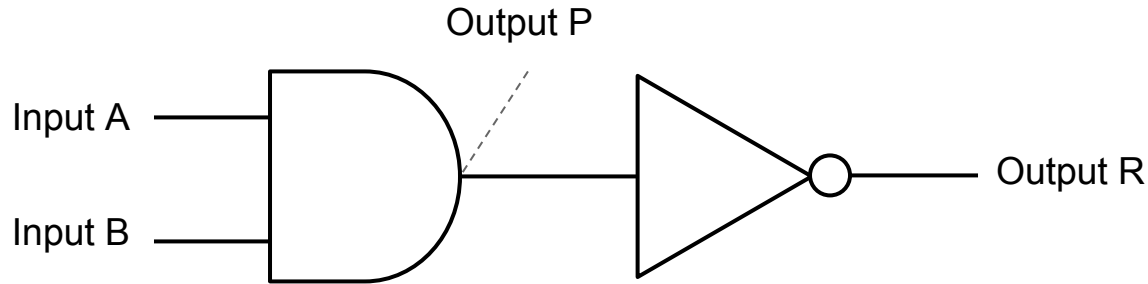
- Truth tables show **each possible combination of inputs and the resulting output**.
- In a complex circuit with multiple gates, it can be helpful to include a column for the output of each gate, as well as the overall output of the circuit.
- **Truth tables should always have 2^n rows**, where n is the number of inputs. For example, a circuit with three inputs would contain 8 (2^3) rows.
- On the right is a truth table for the expression $A \wedge (B \vee C)$



A	B	C	$B \vee C$	Output Q $A \wedge (B \vee C)$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Worked example

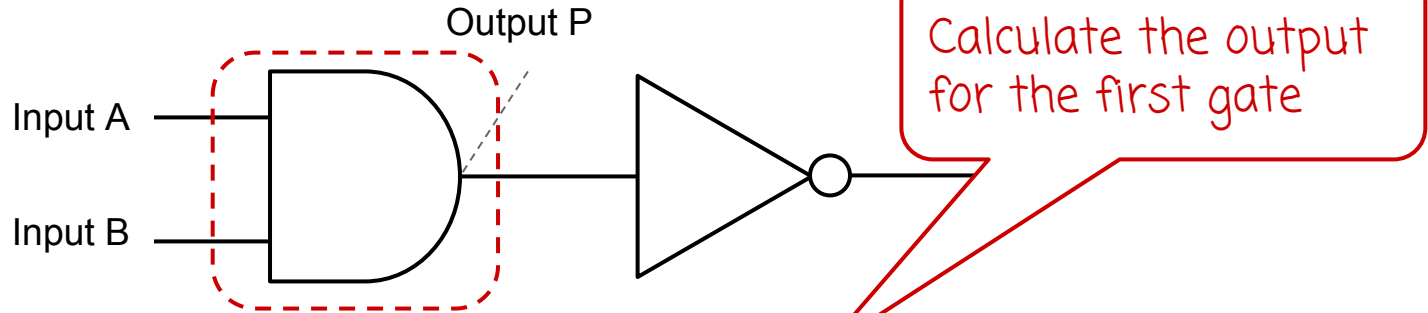
Complete the truth table to evaluate the expression NOT (A AND B)



Input A	Input B	Output P = A AND B	Output R = NOT P
0	0		
0	1		
1	0		
1	1		

Worked example

Complete the truth table to evaluate the expression NOT (A AND B)

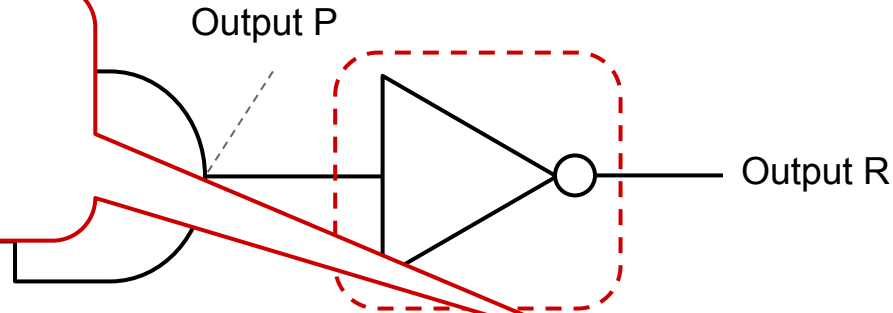


Input A	Input B	Output P = A AND B	Output R = NOT P
0	0	0	
0	1	0	
1	0	0	
1	1	1	

Worked example

Complete the truth table to evaluate the expression NOT (A AND B)

Then move onto
calculating the output
from the second gate



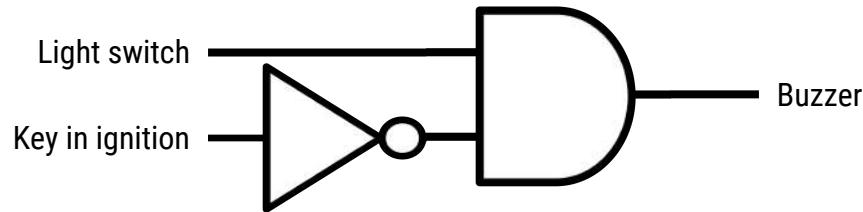
Input A	Input B	Output P = A AND B	Output R = NOT P
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Real-world logic questions

Logic questions are often presented in exams as real-world scenarios, such as:

"Draw a logic circuit to represent a warning buzzer in a car that activates if the lights are switched on and the key is not in the ignition."

In questions such as these, draw a logic circuit using the appropriate gates with the components from the question used as the inputs. For example:





TRANSLATORS AND FACILITIES OF LANGUAGES

CHAPTER 13

High and low level programming languages

- Most computer programs are written in **high-level programming languages** such as Python, Java or C++.
- High-level languages are made up of human-readable statements that make it easier for us to program.
- However, computers can only process instructions in the form of binary numbers. This is known as **machine code** and is an example of a **low-level programming language**.
- Programs written in high-level languages must be converted into low-level machine code by a type of software called a **translator**.

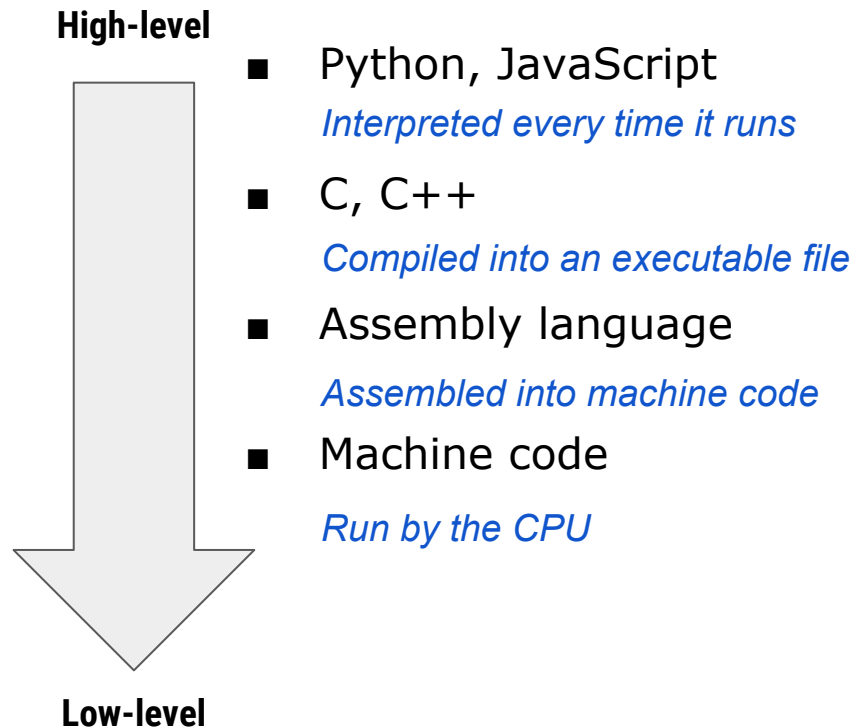
High and low level programming languages

High-level languages

- Human-readable.
- Must be translated for the CPU to execute.
- Often "portable", meaning it can run on different types of CPU.
- One statement represents many CPU instructions.

Low-level languages

- Hard to read by humans.
- Provides exact control over the CPU.
- Will only run on one type of CPU.
- Faster for computers to run as no translation is required.



High and low level programming languages

```
ask_again = True

while ask_again == True:

    cont = input("Would you like me\
                to ask again?")

    if cont == "no":
        ask_again = False
```

High level language (Python)

- Easy to read and understand by humans

```
q00003,    AND fnc026    ;Address of q00003
fnc026,    CLE          ;Function q00003
           BSA pop       ;Getting return address
           STA temp1
           BSA pop       ;Processing arg q00028
           STA q00028
           LDA temp1
           BSA push      ;Putting return address back
           CLE          ;{WORD} q00027 = q00032
           LDA q00032
           BSA push
           BSA pop
           STA q00027

           CLE          ;{WORD} q00035 = q00030
           LDA q00030
           BSA push
           BSA pop
           STA q00035
```

Low level language (assembly code)

- Hard to read and understand by humans

The image shows an open notebook with two pages. The left page contains assembly code for a function named `_start`. The right page contains C code for a function named `fib.c`. A hand is holding a pen and pointing at the assembly code on the left page.

Left Page (Assembly Code):

```

% cat00L -tv fib
fib:
    (.TEXT, __text) section
    .align 1
00000000:000000f20  pushq   %rbp
00000000:000000f21  movq    %rsp, %rbp
00000000:000000f24  subq    $0x20, %rsp
00000000:000000f29  movl    $0x0, -0x4(%rbp)
00000000:000000f2f  movl    $0x0, -0x8(%rbp)
00000000:000000f36  movl    $0x1, -0xc(%rbp)
00000000:000000f3d  movl    $0x56(%rip), %rdi
00000000:000000f44  movl    -0x8(%rbp), %eax
00000000:000000f47  movb    %eax, -0x4(%rbp)
00000000:000000f49  movl    $0x0, -0x8(%rbp)
00000000:000000f50  movl    $0x0, -0xc(%rbp)
00000000:000000f57  movl    $0x0, -0x10(%rbp)
00000000:000000f5e  movl    $0x0, -0x14(%rbp)
00000000:000000f65  movl    $0x0, -0x18(%rbp)
00000000:000000f6c  movl    $0x0, -0x1c(%rbp)
00000000:000000f73  movl    $0x0, -0x20(%rbp)
00000000:000000f7a  movl    $0x0, -0x24(%rbp)
00000000:000000f81  movl    $0x0, -0x28(%rbp)
00000000:000000f89  movl    $0x0, -0x2c(%rbp)
00000000:000000f90  movl    $0x0, -0x30(%rbp)
00000000:000000f99  movl    $0x0, -0x34(%rbp)
00000000:000000fa0  movl    $0x0, -0x38(%rbp)
00000000:000000fa9  movl    $0x0, -0x3c(%rbp)
00000000:000000fb0  movl    $0x0, -0x40(%rbp)
00000000:000000fb9  movl    $0x0, -0x44(%rbp)
00000000:000000fc0  movl    $0x0, -0x48(%rbp)
00000000:000000fd1  movl    $0x0, -0x4c(%rbp)
00000000:000000fda  movl    $0x0, -0x50(%rbp)
00000000:000000fe1  movl    $0x0, -0x54(%rbp)
00000000:000000fe9  movl    $0x0, -0x58(%rbp)
00000000:000000ff0  movl    $0x0, -0x5c(%rbp)
00000000:000000ff9  movl    $0x0, -0x60(%rbp)
00000000:000001000  movl    $0x0, -0x64(%rbp)
00000000:000001009  movl    $0x0, -0x68(%rbp)
00000000:000001010  movl    $0x0, -0x6c(%rbp)
00000000:000001019  movl    $0x0, -0x70(%rbp)
00000000:000001020  movl    $0x0, -0x74(%rbp)
00000000:000001029  movl    $0x0, -0x78(%rbp)
00000000:000001030  movl    $0x0, -0x7c(%rbp)
00000000:000001039  movl    $0x0, -0x80(%rbp)
00000000:000001040  movl    $0x0, -0x84(%rbp)
00000000:000001049  movl    $0x0, -0x88(%rbp)
00000000:000001050  movl    $0x0, -0x8c(%rbp)
00000000:000001059  movl    $0x0, -0x90(%rbp)
00000000:000001060  movl    $0x0, -0x94(%rbp)
00000000:000001069  movl    $0x0, -0x98(%rbp)
00000000:000001070  movl    $0x0, -0x9c(%rbp)
00000000:000001079  movl    $0x0, -0xa0(%rbp)
00000000:000001080  movl    $0x0, -0xa4(%rbp)
00000000:000001089  movl    $0x0, -0xa8(%rbp)
00000000:000001090  movl    $0x0, -0xac(%rbp)
00000000:000001099  movl    $0x0, -0xb0(%rbp)
00000000:0000010a0  movl    $0x0, -0xb4(%rbp)
00000000:0000010a9  movl    $0x0, -0xb8(%rbp)
00000000:0000010b0  movl    $0x0, -0xbc(%rbp)
00000000:0000010b9  movl    $0x0, -0xc0(%rbp)
00000000:0000010c0  movl    $0x0, -0xc4(%rbp)
00000000:0000010c9  movl    $0x0, -0xc8(%rbp)
00000000:0000010d0  movl    $0x0, -0xcc(%rbp)
00000000:0000010d9  movl    $0x0, -0xd0(%rbp)
00000000:0000010e0  movl    $0x0, -0xd4(%rbp)
00000000:0000010e9  movl    $0x0, -0xd8(%rbp)
00000000:0000010f0  movl    $0x0, -0xdc(%rbp)
00000000:0000010f9  movl    $0x0, -0xe0(%rbp)
00000000:000001100  movl    $0x0, -0xe4(%rbp)
00000000:000001109  movl    $0x0, -0xe8(%rbp)
00000000:000001110  movl    $0x0, -0xec(%rbp)
00000000:000001119  movl    $0x0, -0xf0(%rbp)
00000000:000001120  movl    $0x0, -0xf4(%rbp)
00000000:000001129  movl    $0x0, -0xf8(%rbp)
00000000:000001130  movl    $0x0, -0xfc(%rbp)
00000000:000001139  movl    $0x0, -100(%rbp)
00000000:000001140  movl    $0x0, -104(%rbp)
00000000:000001149  movl    $0x0, -108(%rbp)
00000000:000001150  movl    $0x0, -10c(%rbp)
00000000:000001159  movl    $0x0, -110(%rbp)
00000000:000001160  movl    $0x0, -114(%rbp)
00000000:000001169  movl    $0x0, -118(%rbp)
00000000:000001170  movl    $0x0, -11c(%rbp)
00000000:000001179  movl    $0x0, -120(%rbp)
00000000:000001180  movl    $0x0, -124(%rbp)
00000000:000001189  movl    $0x0, -128(%rbp)
00000000:000001190  movl    $0x0, -12c(%rbp)
00000000:000001199  movl    $0x0, -130(%rbp)
00000000:0000011a0  movl    $0x0, -134(%rbp)
00000000:0000011a9  movl    $0x0, -138(%rbp)
00000000:0000011b0  movl    $0x0, -13c(%rbp)
00000000:0000011b9  movl    $0x0, -140(%rbp)
00000000:0000011c0  movl    $0x0, -144(%rbp)
00000000:0000011c9  movl    $0x0, -148(%rbp)
00000000:0000011d0  movl    $0x0, -14c(%rbp)
00000000:0000011d9  movl    $0x0, -150(%rbp)
00000000:0000011e0  movl    $0x0, -154(%rbp)
00000000:0000011e9  movl    $0x0, -158(%rbp)
00000000:0000011f0  movl    $0x0, -15c(%rbp)
00000000:0000011f9  movl    $0x0, -160(%rbp)
00000000:000001200  movl    $0x0, -164(%rbp)
00000000:000001209  movl    $0x0, -168(%rbp)
00000000:000001210  movl    $0x0, -16c(%rbp)
00000000:000001219  movl    $0x0, -170(%rbp)
00000000:000001220  movl    $0x0, -174(%rbp)
00000000:000001229  movl    $0x0, -178(%rbp)
00000000:000001230  movl    $0x0, -17c(%rbp)
00000000:000001239  movl    $0x0, -180(%rbp)
00000000:000001240  movl    $0x0, -184(%rbp)
00000000:000001249  movl    $0x0, -188(%rbp)
00000000:000001250  movl    $0x0, -18c(%rbp)
00000000:000001259  movl    $0x0, -190(%rbp)
00000000:000001260  movl    $
```

Video by [Ben Eater](#)

Assembly language

- Every CPU has a set of instructions that it can interpret to perform operations. This set of instructions is, helpfully, called its **instruction set**.
- Each instruction is made up of an opcode and optional operand, for example 0x0A (0000 1010) might be the opcode for an addition operation.
- Assembly language is a **low-level language** that is **easier to work with** for human than the binary numbers of machine code.
- Assembly language uses short **mnemonics** in place of each opcode, for example ADD in place of 0x0A.
- This makes it easier for humans to use, but still requires a full understanding of how the CPU and memory within a system works in order to write programs directly using nothing but the CPU's built-in instructions.

Example of assembly language

```
100
101
102
103
104
105
106
107 00000030 B9FFFFFF
108
109
110 00000035 41
111 00000036 803C0800
112
113
114 0000003A 75F9
115
116
117
118
119
120
121
122 0000003C C3
```

```
-----
; zstr_count:
; Counts a zero-terminated ASCII string to determine its size
; in:  eax = start address of the zero terminated string
; out: ecx = count = the length of the string

zstr_count:                ; Entry point
    mov  ecx, -1           ; Init the loop counter, pre-decrement
                           ; to compensate for the increment

.loop:
    inc  ecx               ; Add 1 to the loop counter
    cmp  byte [eax + ecx], 0 ; Compare the value at the string's
                           ; [starting memory address Plus the
                           ; loop offset], to zero
    jne  .loop             ; If the memory value is not zero,
                           ; then jump to the label called '.loop',
                           ; otherwise continue to the next line

.done:
                           ; We don't do a final increment,
                           ; because even though the count is base 1,
                           ; we do not include the zero terminator in the
                           ; string's length
    ret                   ; Return to the calling program
```

Equivalent machine code

Assembly language showing mnemonics and comments to explain the code.

Video: High-level and Low-level languages



Video by [Banchory Academy](#)
[Computing Science](#)

The purpose of translators

- Translators are a special type of software that **translate** (convert) computer programs from a **higher level language to low-level machine code** that the CPU can **execute** (run).
- Translators are essential because a CPU can only process instructions in the **form of binary numbers** and that are a part of its own **instruction set**.
- There are three types of translator, each of which works slightly differently:
 - Interpreters
 - Compilers
 - Assemblers

Interpreters

- Interpreters translate high-level source code into low-level machine code **one instruction at a time**.
- Because of this, **running an interpreted program is slower** than a compiled program, however it is **quicker to get started** executing the program as there is no time spent waiting for the whole program to be compiled first.
- Interpreters **do not produce a compiled program file**, so the process of translation must be **repeated every time** the program is run.
- Interpreters will **stop as soon as they encounter an error** in a program, making them very useful for debugging.
- Interpreted languages are often "**portable**" meaning that they can be **run on many different types of CPU** as long as there is an interpreter for that CPU platform.
- Python and Javascript are examples of interpreted languages.

Compilers

- Compilers translate the whole of a program written in a high-level language into machine code **in one go**.
- The process of compiling a program **can be slow** (especially if it is a large program) however **the resulting compiled machine code runs much more quickly**.
- Because compilers produce an **executable program file** there is **no need to repeat the compilation process** each time a program is run.
- As an aside, compiled programs **hide the original source code** which can help protect developers' intellectual property.
- A compiler will present **a list of errors once it has attempted the compilation process**, making it harder to find and fix each error.
- C, C++, Visual Basic and Swift are all examples of compiled languages.

Video: Comparing interpreters and compilers

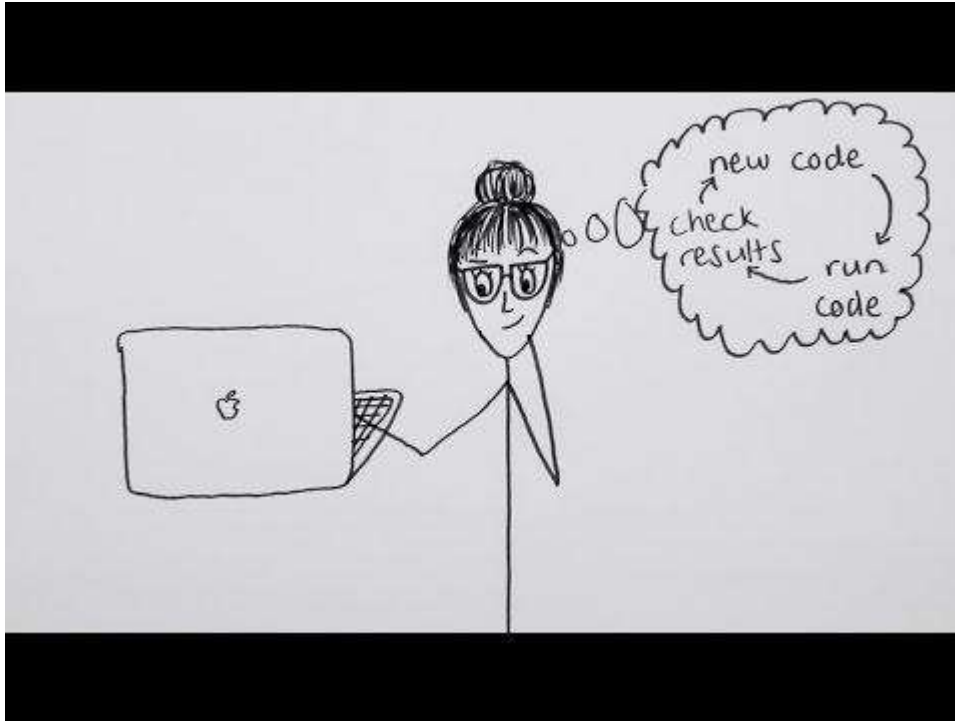


Video by: [Bits and Bytes TVO](#)

Assemblers

- Assemblers **translate low-level assembly code into machine code**.
- Each assembly statement is written as a **mnemonic** that **corresponds directly** to one of the instructions in the CPU's **instruction set**.
- Writing programs in assembly code is **very difficult** as it requires a **deep technical knowledge of the CPU's structure and how it accesses memory** on the type of system that the program will run on.
- However, writing programs in assembly code provides developers with **exact control** over how the CPU functions, often leading to **very efficient code** that takes up **little space** in memory and is **quick to execute**.
- Low-level software engineers might use assembly code to write software drivers and boot-loader code for computer systems. A lot of software for embedded systems is written in assembly code as it has to run with very little resources.

Video: How CPUs process instructions



Video by [All About Software](#)

Integrated Development Environments (IDEs)

Integrated Development Environments (IDEs)

- Integrated Development Environments (or "IDEs" as they are commonly known) are **software packages** that **facilitate software development**.
- IDEs provide a number of **features** to make **writing programs easier**.
- They differ from a simple code editor (like Geany or Notepad++) and text editor (such as Notepad) as they **do more** than just allowing you to write code.

Features of IDEs

IDEs offer many features to help with writing programs, including:

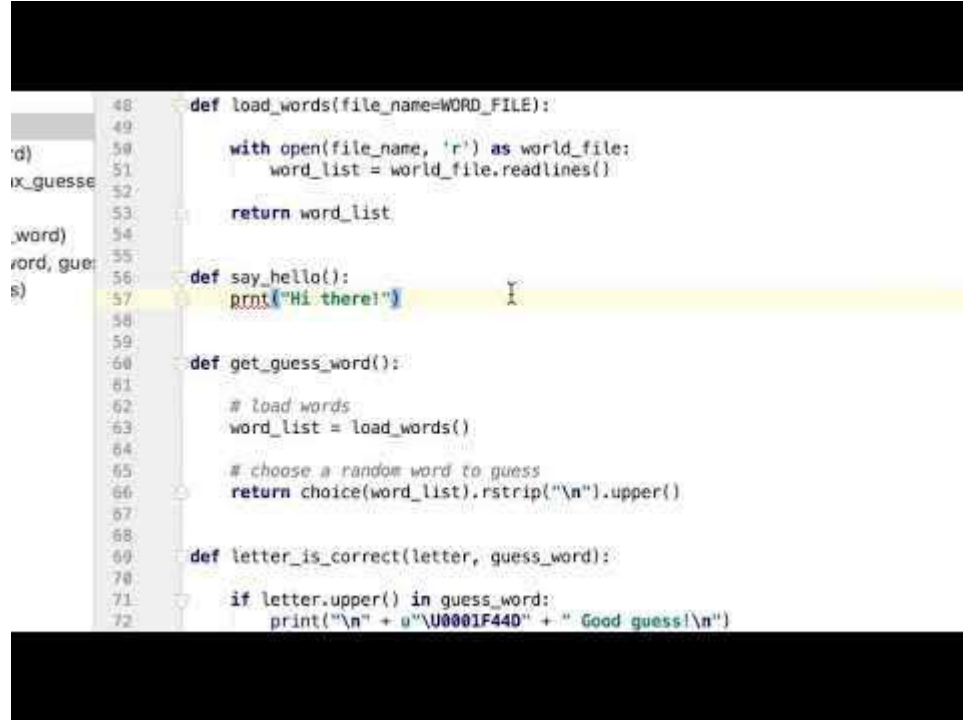
- A **code editor** where you can write your program. The code-editor often includes features to help write code including **syntax highlighting** and **code-completion**.
- **Error diagnostics** such as underlining syntax errors in statements or flagging them with a warning sign.
- A **debugger**, which allows you to run your code **line-by-line** so that you can check the values of variables at any point and **step through** the logic in your programs. Debuggers also allow you to set **breakpoints** - lines in your program where execution will **pause** so that you can see what is going on in more detail.

Features of IDEs

IDEs offer many features to help with writing programs, including:

- **Translators** - i.e. an interpreter, compiler or assembler that is appropriate to the high-level language being used. Source code can often be translated into machine code with the click of a "Run" button, rather than typing complex commands at a command line interface.
- **Run-time environment** - this allows you to run your programs and see how they work. Some IDEs allow you to test your program as if it were running on a range of different devices and operating systems. For example, Apple's Xcode includes an iPhone simulator so you can test iPhone apps on a Mac.

Video: Comparing code editors and an IDE



```
48 def load_words(file_name=WORD_FILE):
49
50     with open(file_name, 'r') as world_file:
51         word_list = world_file.readlines()
52
53     return word_list
54
55
56 def say_hello():
57     print("Hi there!")
58
59
60 def get_guess_word():
61
62     # load words
63     word_list = load_words()
64
65     # choose a random word to guess
66     return choice(word_list).rstrip("\n").upper()
67
68
69 def letter_is_correct(letter, guess_word):
70
71     if letter.upper() in guess_word:
72         print("\n" + u"\U0001F440" + " Good guess!\n")
```

This video compares the features of a good code editor, SublimeText, with those of the Pycharm IDE for Python programming.

Which features of the IDE do you think you would find most useful?

Video by [Mr A W Dimmick](#)

DATA REPRESENTATION

CHAPTER 14

Data units

Units of data - bits and bytes

Recap from the 1.3 Storage unit

- The smallest unit of data that a computer can represent is a **bit (b)**, it's value can be either **0** or **1**.
- A group of four bits is a **nibble** and a group of eight bits is a **byte (B)**.
- Because these units are very small, larger units are typically used to describe the size of data, including kilobytes (KB, 1KB = 1024 bytes), megabytes (MB, 1MB = 1024KB), gigabytes (GB, 1GB = 1024MB) and terabytes (1TB = 1024GB).
- Note that each unit is 1024 times larger than the next smallest unit and not 1000 times. This is because 1024 is more easily represented in binary than 1000.

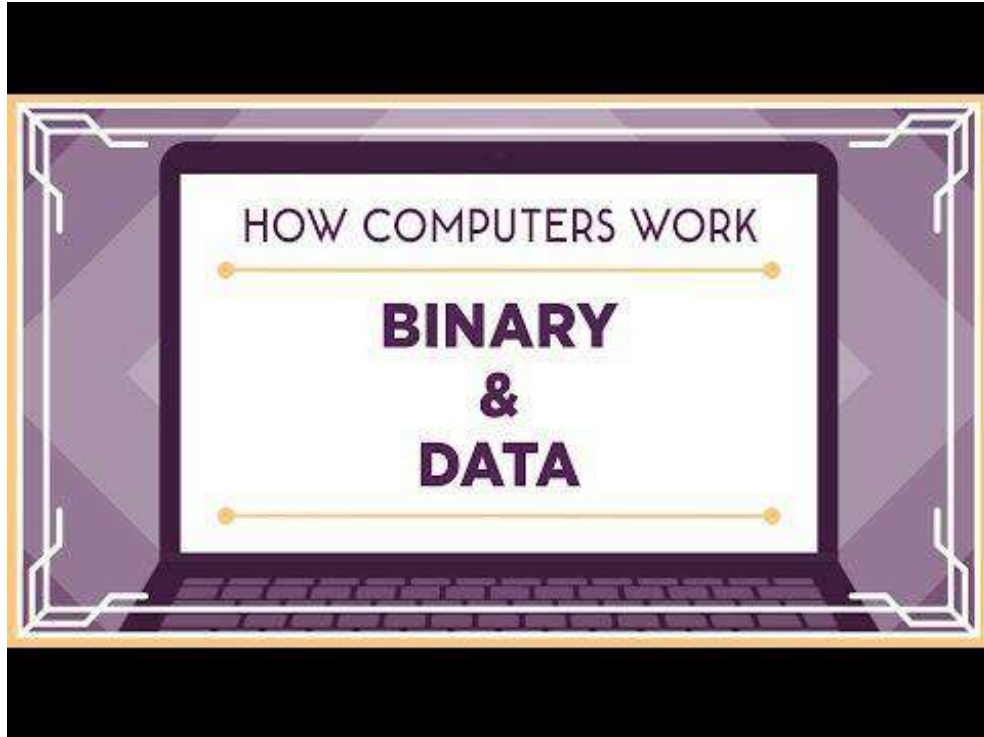
Comparing data units

Unit	Abbreviation	Equivalent to	What sort of things are in this size range?
Bit	b	0 or 1	A single binary value or a True or False boolean value
Byte	B	8 bits	A single ASCII character or a number between 0 and 255
Kilobyte	KB	1,024 bytes	A text file
Megabyte	MB	1,024 kilobytes	Digital photographs and MP3 files
Gigabytes	GB	1,024 megabytes	Videos and software applications
Terabytes	TB	1,024 gigabytes	Backups of whole systems

Everything is in binary!

- Every single thing that a computer stores or processes has be converted into **binary format**.
- This is because computers are **electrical devices** and **binary numbers represent the flow of electricity** through the switches and wires within the CPU and memory chips.
- Secondary storage devices may **represent the binary data in other forms**, such as positively or negatively charged magnetic particles, but these are ultimately converted into electrical signals for processing.
- All instructions that are **processed by a CPU**, all data that is **stored on secondary storage** or transmitted across a **network**, everything within a computer system is **encoded as a sequence of binary bits, either 0s or 1s**.

Video: How computers work (Binary & Data)



Video by [Code.org](https://code.org)

Representing numbers

Number systems

- A number system is a mechanism for representing any number using a set of symbols or digits.
- The number system that we are most used to using is the **denary number system**. This number system uses a combination of **ten symbols** to represent every number. Those symbols are: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.
- Computers represent data using the **binary number system**. This number system uses only **two symbols** to represent all numbers - 0 and 1.
- Computer programmers often write numbers using the **hexadecimal number system**. This number system uses sixteen number symbols to represent any number - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

Column values

- When we represent numbers, the **column** that a digit is placed within determines its **value**
- For example, in the denary number system the symbols 2, 4 and 9 can be combined to represent the number two-hundreds and forty-nine:

2 hundreds 4 tens 9 units (1s)

2 4 9

Column values

- We don't need 249 symbols to represent the number, instead we can represent *any* number from a combination of just ten symbols

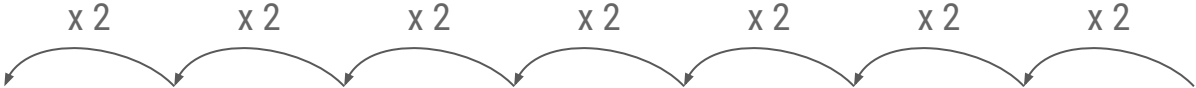
2 hundreds 4 tens 9 units (1s)

The diagram illustrates the place value of the digits in the number 249. The digit 2 is in the hundreds place, the digit 4 is in the tens place, and the digit 9 is in the units place. Red arrows point from the handwritten labels to their respective digits: '2 hundreds' points to the 2, '4 tens' points to the 4, and '9 units (1s)' points to the 9.

Binary numbers

Understanding binary numbers

- Evaluating the numeric value of a binary number involves exactly the same principles as in denary, except that the column values are different.
- The rightmost column value is 1s (units), the column to its left is 2s, the next column to its left is 4s, then 8s, then 16s and so on.
- In fact, every column is worth double that of the column to its right.
- The contents of each column is then either a 0 or a 1.



Column value	128	64	32	16	8	4	2	1
Contents	0	1	0	0	1	1	1	0

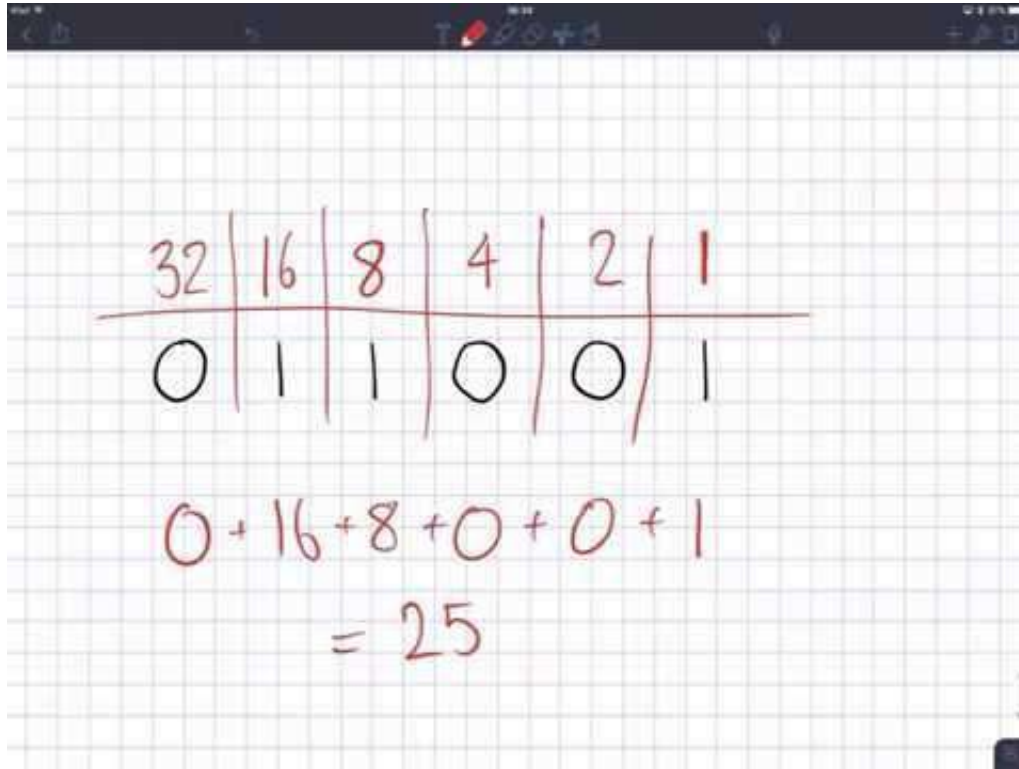
Understanding binary numbers

Column value	128	64	32	16	8	4	2	1
Contents	0	1	0	0	1	1	1	0

- Once each binary digit is placed within its appropriate column, evaluating the value of the number is simply a case of adding up all of the column values that contain a 1.
- In the example above, we can see that 0100 1110 in binary represents the value 78 in denary ($64+8+4+2 = 78$).

Exam tip: You will not be asked to interpret binary numbers any larger than 8-bits!

Video: How to convert binary numbers to denary



A screenshot of a digital notepad showing the conversion of the binary number 011001 to the denary number 25. The binary digits are placed in a table with columns for powers of 2 (32, 16, 8, 4, 2, 1). The calculation 0 + 16 + 8 + 0 + 0 + 1 = 25 is written below the table.

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

$$0 + 16 + 8 + 0 + 0 + 1 = 25$$

Video by [Mr A W Dimmick](#)

Converting denary numbers into binary

To convert a denary number into binary:

1. Write out the column values for binary numbers. You are only expected to work with 8-bit or smaller numbers, so use the column values 128, 64, 32, 16, 8, 4, 2 and 1.
2. Starting with the largest (leftmost) column, place a 1 in the column if the value of that column is smaller than the denary number being converted, otherwise place a 0 in that column.
3. If you wrote a 1 in that column, subtract the value of that column from the number being converted.
4. Move onto the next column, repeating steps 2 to 4 but comparing the column values to the remainder of the original number each time after subtracting each column value.
5. Keep going until there is no more of the original number left. When this happens, place 0s in each remaining column.

Example: converting denary to binary

Write the denary value 86 as an 8-bit binary number.

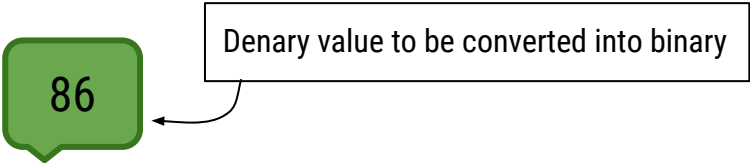
Firstly, write out the column values for an 8-bit binary number (starting at 128 on the left).

Column value	128	64	32	16	8	4	2	1
Contents								

Example: converting denary to binary

Write the denary value 86 as an 8-bit binary number.

Starting at the leftmost column, can 128 be subtracted from 86? No, so place a 0 in the column and move on to the right.

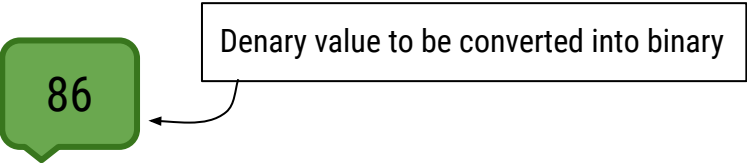


Column value	128	64	32	16	8	4	2	1
Contents	0							

Example: converting denary to binary

Write the denary value 86 as an 8-bit binary number.

Moving onto the next column, can 64 be subtracted from 86? Yes! So place a 1 in the column and subtract 64 from 86 to get a remainder of **22** and move to the right.

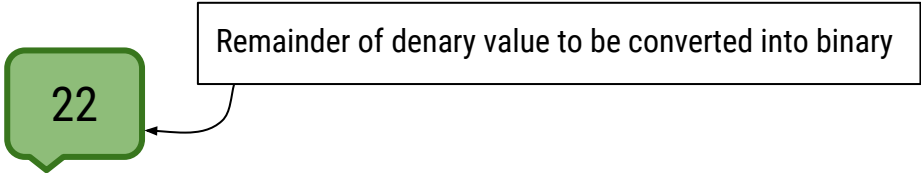


Column value	128	64	32	16	8	4	2	1
Contents	0	1						

Example: converting denary to binary

Write the denary value 86 as an 8-bit binary number.

Can 32 be subtracted from 22? No, so place a 0 in the 32s column and move on to the right.



Column value	128	64	32	16	8	4	2	1
Contents	0	1	0					

Example: converting denary to binary

Write the denary value 86 as an 8-bit binary number.

Can 16 be subtracted from 22? Yes, so place a 1 in the 16s column and subtract 16 from 22 to calculate the new remaining value of **6**. Move onto to the next column.

Remainder of denary value to be converted into binary								
22								
Column value	128	64	32	16	8	4	2	1
Contents	0	1	0	1				

Example: converting denary to binary

Write the denary value 86 as an 8-bit binary number.

Can 8 be subtracted from 6? No, so place a 0 in the 8s column and move on to the right.

Remainder of denary value to be converted into binary

6

Column value	128	64	32	16	8	4	2	1
Contents	0	1	0	1	0			

Example: converting denary to binary

Write the denary value 86 as an 8-bit binary number.

Can 4 be subtracted from 6? Yes, so place a 1 in the 4s column and subtract 4 from 6 to gain a new remainder of 2. Move on to the right.

Remainder of denary value to be converted into binary

6

Column value	128	64	32	16	8	4	2	1
Contents	0	1	0	1	0	1		

Example: converting denary to binary

Write the denary value 86 as an 8-bit binary number.

Can 2 be subtracted from 2? Yes, so place a 1 in the 2s column and subtract 2 from 2 to gain a new remainder of 0 and move onto the right.

Remainder of denary value to be converted into binary

2

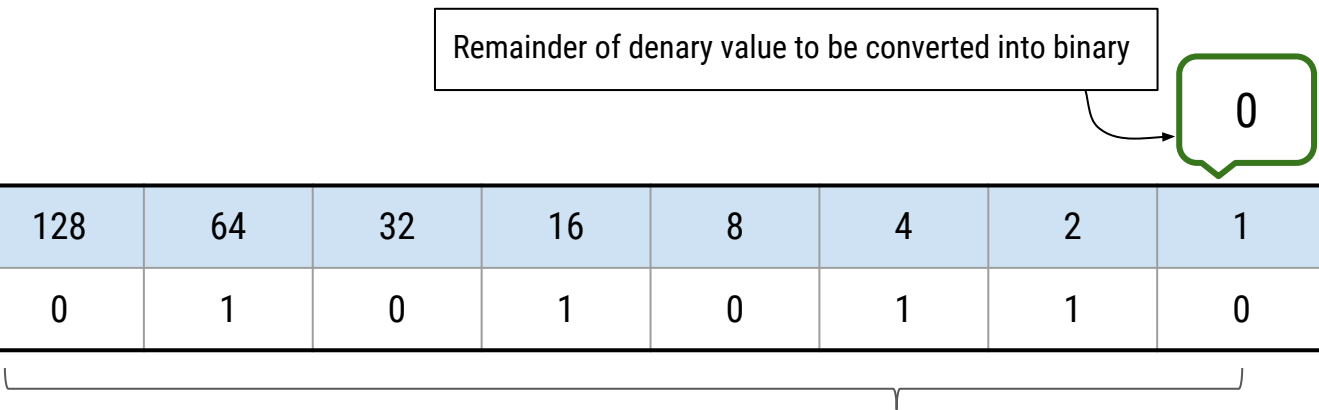
Column value	128	64	32	16	8	4	2	1
Contents	0	1	0	1	0	1	1	

Example: converting denary to binary

Write the denary value 86 as an 8-bit binary number.

There's no more of the original value to convert, so place 0s in the remaining columns - we're done!

Remainder of denary value to be converted into binary								
Column value	128	64	32	16	8	4	2	1
Contents	0	1	0	1	0	1	1	0



The denary number 86 as an 8-bit binary number is **0101 0110**.

Video: converting denary to binary numbers

Denary to binary worked example

- Show the number 57 in binary

32	16	8	4	2	1
1	1	1	1		

4 does not go into 1, so put a 0 in the 4 column and move on

Note, in this video the result isn't given as an 8-bit binary number.

To convert the result into 8 bits just add enough 0s to the **front** of the binary number to make it 8 digits long.

Video by [Mr A W Dimmick](#)

Binary addition

Adding two 8-bit binary numbers

To add two binary numbers we use the same [column addition method](#) that we are familiar with to add denary numbers but will the following rules applied:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 1 = 1\ 0$ (write a 0 and carry 1 to the next column)
- $1 + 1 + 1 = 1\ 1$ (write 1 and carry 1 to the next column)

Always start at the rightmost column and carry to the left.

Tip: when performing binary addition, it is really important to ensure that you clearly line up each digit within the correct column!

Worked example: Binary addition

Add together 1000 1101 and 0100 1000 using binary addition.

$$\begin{array}{rcccccccc} & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ + & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline & & & & & & & & \\ \hline \end{array}$$

Worked example: Binary addition

Add together 1000 1101 and 0100 1000 using binary addition.

	1	0	0	0	1	1	0	1
+	0	1	0	0	1	0	0	0
								1
								1

Rule being applied: $1 + 0 = 1$

Worked example: Binary addition

Add together 1000 1101 and 0100 1000 using binary addition.

	1	0	0	0	1	1	0	1
+	0	1	0	0	1	0	0	0
							0	1

Rule being applied: $0 + 0 = 0$

Worked example: Binary addition

Add together 1000 1101 and 0100 1000 using binary addition.

	1	0	0	0	1	1	0	1
+	0	1	0	0	1	0	0	0
						1	0	1

Rule being applied: $1 + 0 = 1$

Worked example: Binary addition

Add together 1000 1101 and 0100 1000 using binary addition.

	1	0	0	0	1	1	0	1
+	0	1	0	0	1	0	0	0
					0	1	0	1

1

Rule being applied: $1 + 1 = 10$ (write 0 in the column, carry 1 to the left column)

Worked example: Binary addition

Add together 1000 1101 and 0100 1000 using binary addition.

	1	0	0	0	1	1	0	1
+	0	1	0	0	1	0	0	0
				1	0	1	0	1
				1				

Rule being applied: $0 + 0 + 1$ (the carry value) = 1

Worked example: Binary addition

Add together 1000 1101 and 0100 1000 using binary addition.

	1	0	0	0	1	1	0	1
+	0	1	0	0	1	0	0	0
			0	1	0	1	0	1

A red dashed box highlights the third column from the right, containing the digits 0 and 0 from the two addends and the resulting digit 0. A green '1' is positioned below the box, indicating a carry-out to the next column.

Rule being applied: $0 + 0 = 0$

Worked example: Binary addition

Add together 1000 1101 and 0100 1000 using binary addition.

	1	0	0	0	1	1	0	1
+	0	1	0	0	1	0	0	0
		1	0	1	0	1	0	1
				1				

Rule being applied: $0 + 1 = 1$

Worked example: Binary addition

Add together 1000 1101 and 0100 1000 using binary addition.

	1	0	0	0	1	1	0	1
+	0	1	0	0	1	0	0	0
	1	1	0	1	0	1	0	1
				1				

Rule being applied: $1 + 0 = 1$

Worked example: Binary addition

Add together 1000 1101 and 0100 1000 using binary addition.

$$\begin{array}{rcccccccc} & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ + & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline \end{array}$$

1

Answer: 1101 0101 (213 in denary)

Overflow errors

- An **overflow error** occurs if the result of a binary addition is **too large** to represent **within the number of bits available** within the system.
- For example, in an 8-bit system, any result larger than 255 will cause an overflow error.
- Overflow errors can lead to inaccurate results and software **crashes** if they are not handled correctly by programmers.
- Ways to deal with an overflow error include **storing the extra (overflow) bits within another location in memory** and to **set a special "flag" bit to indicate that the error has occurred**.
- The CPU, OS or application can detect the status of this flag and respond accordingly, for example by breaking the sum down into two smaller sums that can be computed and combined together.

Worked example: Binary addition with overflow error

Add together 1011 1011 and 1000 0010 using binary addition.

	1	0	1	1	1	0	1	1
+	1	0	0	0	0	0	1	0
								1
								1

Rule being applied: $1 + 0 = 1$

Worked example: Binary addition with overflow error

Add together 1011 1011 and 1000 0010 using binary addition.

	1	0	1	1	1	0	1	1
+	1	0	0	0	0	0	1	0
							0	1

1

Rule being applied: $1 + 1 = 10$ (0, carry 1 to the left)

Worked example: Binary addition with overflow error

Add together 1011 1011 and 1000 0010 using binary addition.

	1	0	1	1	1	0	1	1
+	1	0	0	0	0	0	1	0
						1	0	1
						1		

Rule being applied: $0 + 0 + 1$ (carried from the right) = 1

Worked example: Binary addition with overflow error

Add together 1011 1011 and 1000 0010 using binary addition.

	1	0	1	1	1	0	1	1
+	1	0	0	0	0	0	1	0
					1	1	0	1

A red dashed box highlights the 5th column from the left, containing the digits 1 and 0 in the top two rows, and the result 1 in the row below. A green '1' is written below the box, indicating a carry-out to the next column.

Rule being applied: $1 + 0 = 1$

Worked example: Binary addition with overflow error

Add together 1011 1011 and 1000 0010 using binary addition.

	1	0	1	1	1	0	1	1
+	1	0	0	0	0	0	1	0
				1	1	1	0	1
						1		

Rule being applied: $1 + 0 = 1$

Worked example: Binary addition with overflow error

Add together 1011 1011 and 1000 0010 using binary addition.

	1	0	1	1	1	0	1	1
+	1	0	0	0	0	0	1	0
			1	1	1	1	0	1
						1		

Rule being applied: $1 + 0 = 1$

Worked example: Binary addition with overflow error

Add together 1011 1011 and 1000 0010 using binary addition.

	1	0	1	1	1	0	1	1
+	1	0	0	0	0	0	1	0
		0	1	1	1	1	0	1
						1		

Rule being applied: $0 + 0 = 0$

Worked example: Binary addition with overflow error

Add together 1011 1011 and 1000 0010 using binary addition.

	1	0	1	1	1	0	1	1
+	1	0	0	0	0	0	1	0
	0	0	1	1	1	1	0	1
1						1		

Rule being applied: $1 + 1 = 10$ (0, carry 1 to the left)

Worked example: Binary addition with overflow error

Add together 1011 1011 and 1000 0010 using binary addition.

There is **no space** for this overflow bit within an 8-bit system!

A binary addition diagram showing two 8-bit numbers being added. The first number is 1011 1011 and the second is 1000 0010. The result shown is 0011 1101. A red dashed box highlights the overflow bit (1) that occurs at the 9th position. A green '1' is also shown below the 7th position of the result. A curved arrow points from the text box above to the overflow bit.

	1	0	1	1	1	0	1	1
+	1	0	0	0	0	0	1	0
	1	0	0	1	1	1	1	0
	1	0	0	1	1	1	1	0

The answer **should** be 317, but this number cannot be represented with only 8 bits.

Result: An overflow error has occurred! The extra bit will be lost, giving an answer of 0011 1101 (61 in denary)

Video: A step-by-step guide to binary addition

$$\begin{array}{r} 01001110 \quad (78) \\ + 10111010 \quad (186) \\ \hline 000 \\ \hline 11 \end{array}$$

$$\left\{ \begin{array}{l} 0+0 = 0 \\ 0+1 = 1 \\ \underline{1+1} = 0 \text{ carry } 1 \text{ (10)} \\ 1+1+1 = 1 \text{ carry } 1 \text{ (11)} \end{array} \right.$$

Video by [Mr A W Dimmick](#)

Exam tips!

- You will not be asked to work with binary numbers larger than 8-bits.
- Always write out the column values, even if you are a well-practised pro at this!
- The largest denary value that can be represented by an 8-bit binary number is 255 (1111 1111) so if your answer is larger than this then something has gone wrong!
- Make sure that your columns are really clearly laid out, especially when doing binary addition exercises.

Binary shifts

- A binary shift is a technique for performing **multiplication** or **division** on a binary number.
- When performing a binary shift, each digit is moved ("shifted") **one column to the left or to the right**, depending on whether a left shift or right shift is being performed.
- Extra 0 bits are added to the start or end of the binary number to fill any missing spaces.
- Each **shift to the left** has the effect of **multiplying the binary number by 2**.
- Each **shift to the right** has the effect of **dividing the binary number by 2**.

Worked example: Multiplication by left-shifting

Perform a two-place left shift on the binary number 0001 1100. Explain the effect that this has had.

0001 1100 left-shifted twice = 0111 0000.

The number has been multiplied by 4 (x2 for each left shift). 0001 1100 is 28 in denary and 0111 0000 is 112 in denary.

Overflows and drop-offs following binary shifts

- Just as with binary addition, performing a multiplication through a binary left shift can result in an **overflow error** if the result is greater than the number of bits available within the system.
- For example, 1010 0000 (160 in denary) left shifted results in 0100 000 (64 in denary) with the left-most bit being lost.
- Equally, when dividing with a binary right shift, bits can be lost off the right-hand side resulting in a **loss of accuracy** in the result. For example 0010 0011 (35 in denary) left-shifted one place results in 0001 0001, which is 17 in denary rather than 17.5.
- In fact, left-shifting is how **integer division** is performed when the DIV operator is used.

Worked example: Integer division by right-shifting

Perform a right shift on the binary number 0100 0101. Explain the effect that this has had.

0100 0101 right-shifted = 0010 0010

The number has been integer divided by 2, leading to a loss of accuracy in the result as a bit has been lost on the right-hand side. 0100 0101 is 69 in denary and 0010 0010 is 34 in denary but the result of $69/2$ should be 34.5.

Worked example: Overflow error following a left shift

Perform a left shift on the binary number 1000 0010. Explain the effect that this has had.

1000 0010 left-shifted = 0000 0100.

An overflow error has occurred leading to an inaccurate result. 1000 0010 is 130 in denary. A left-shift should multiply a number by 2, however the leftmost bit has been lost, resulting in the value 0000 0100, which is just 4 in denary.

Hexadecimal numbers

Hexadecimal numbers

The hexadecimal number system ("hex" for short) uses **sixteen** number symbols within each column: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

Programmers often use the hexadecimal system to represent numbers because:

- They can be easily converted to and from binary
- They use fewer character to write down
- Using hexadecimal significantly reduces the likelihood of errors when copying numbers down
 - For example, it's much easier to write 5D in hex than 0101 1101 in binary, even though they both represent the same number (93 in denary).

Hexadecimal digit values

Denary	Binary	Hex	Denary	Binary	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Converting 2-digit hexadecimal values to denary

Converting from numbers in hexadecimal form to denary is simple:

1. Multiply the value of the first digit by 16
2. Add the value of the second digit

For example, to convert the hex value D4 into denary:

1. $D = 13$; $13 * 16 = 208$
2. Add the value of the second digit; $208 + 4 = 212$

You will not be asked to convert hexadecimal values with more than two digits at GCSE!

Converting 2-digit hexadecimal values to denary

Converting from numbers in hexadecimal form to denary is simple:

1. Multiply the value of the first digit by 16
2. Add the value of the second digit

For example, to convert the hex value 32 into denary:

1. $3 * 16 = 48$
2. Add the value of the second digit; $48 + 2 = 50$

You will not be asked to convert hexadecimal values with more than two digits at GCSE!

Converting from denary to hexadecimal

Converting from denary to hexadecimal is slightly more complicated:

1. Divide the denary number by 16 using integer (whole number) division. Use this as the value of the first hexadecimal digit.
2. Multiply the value of the first digit by 16 and subtract this from the original denary number. This is the value for the second hex digit.

For example, convert 170 in denary to hexadecimal.

1. $170 \text{ DIV } 16 = 10$; first hex digit is A
2. $10 * 16 = 160$; $170 - 160 = 10$; second hex digit is A
3. 170 in denary is **AA** in hexadecimal form.

Converting from binary to hexadecimal form

- Every group of **four binary bits** (a nibble) equates to **one hex digit**.
- To convert from binary to hex, simply break the number in to groups of four bits and replace with the equivalent hex digit.

For example, convert 1101 1100 into hexadecimal form.

- First digit is 1101, which is 13 in denary and **D** in hex
- Second digit is 1100, which is 12 in denary and **C** in hex
- So, 1101 1100 in hex form is **DC**.

Converting from hexadecimal to binary form

- Converting from hexadecimal to binary is equally simple.
- Again, each hex digit simply needs to be replaced with the equivalent four binary bits.

For example, write the hex value A5 in binary form.

- First digit is A, which is 10 in denary and 1010 in binary.
- Second digit is 5, which is 5 in denary and 0101 in binary.
- So, A5 in binary form is **1010 0101**.

Check digits

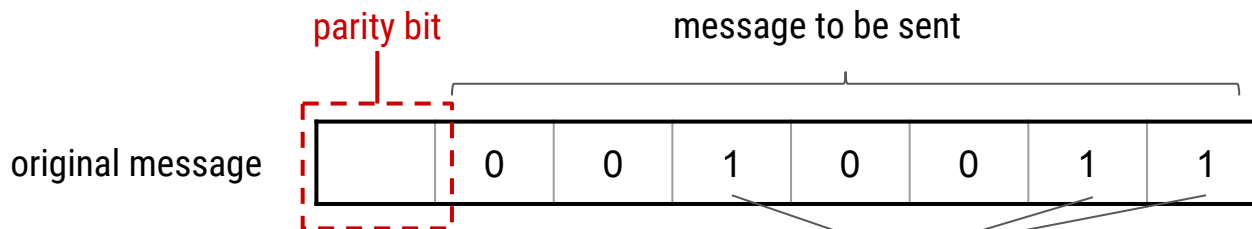
Check digits

- Check digits are extra digits that are **appended to a number** in order to confirm that the number has been **transmitted or retrieved correctly**.
- The value of a check digit depends on the algorithm being applied. For example, the last digit of an ISBN number is a check digit that is calculated by performing a combination of multiplication and addition calculations on each of the other digits in the number.
- When an ISBN number is read, for example through a barcode scanner, the same algorithm is applied to the non-check digits. If the outcome of this calculation matches the check digit then we can be confident that the value has been read correctly. However, **if the values do not match then an error has occurred** and the number can be rescanned.

Parity bits

- A parity bit is a simple check digit that is added to a binary number to help check that it has been transmitted correctly.
- The value of the parity bit depends on whether **odd** or **even parity** is being enforced.
- In an **even parity** system, there should be an **even number of 1s** in a binary number.
- In an **odd parity** system, there should be an **odd number of 1s** in a binary number.
- The value of the **parity bit** is set to either 0 or 1 as required to satisfy the requirements of the odd or even parity system being used.
- For example, in an even parity system, the value of the parity bit added to the 7-bit binary number 100 0011 would be 1 so that, overall, there are an even number of 1s within the number. By contrast, the value of the parity bit added to the binary number 110 0011 would be 0 because there is already an even number of 1s in the number.

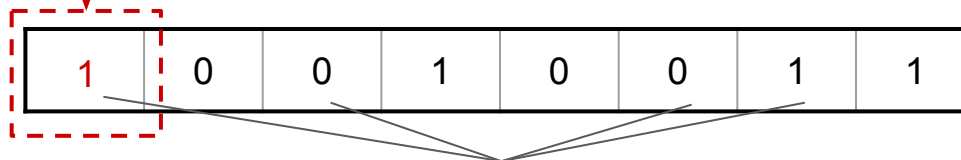
Worked example: Even parity enforced



The parity bit is set to 1 and added to the original message in order to **enforce an even number of 1s** in the binary sequence.

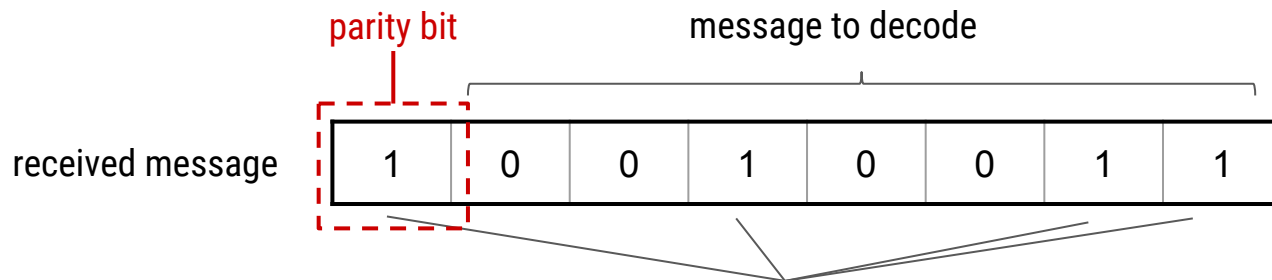
message contains three 1s (odd number)

transmitted message



transmitted message contains four 1s (even number)

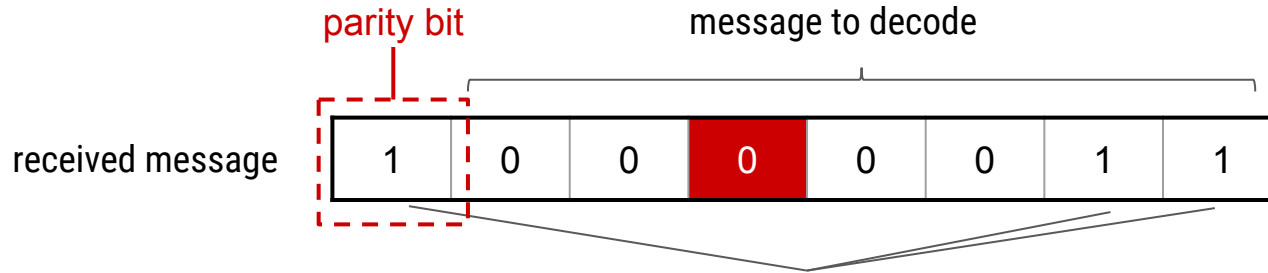
Worked example: Message received with correct parity



The received message contains four 1s, which is an even number, therefore we can assume that this message has become correctly received.

The parity bit is now ignored and the remainder of the message is decoded.

Worked example: Message received with incorrect parity

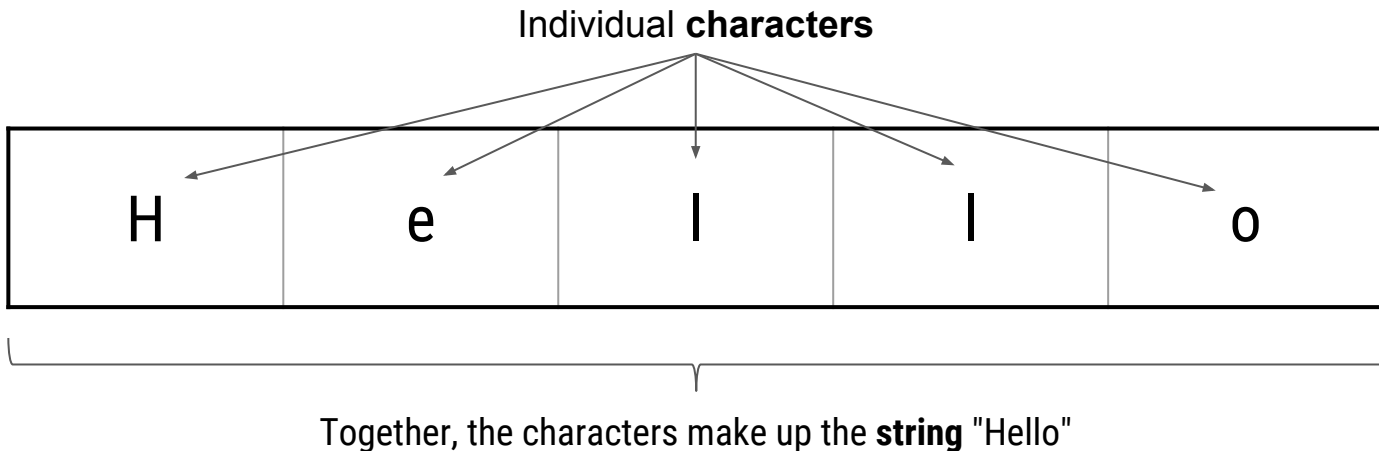


In this case, the received message contains three 1s, but it should contain an even number! This message has become corrupted and the sender is asked to re-send the message as a result.

Representing text and characters

Characters

- Text data is made up of characters. Each character is an individual letter, number or symbol.
- **Strings** are collections of characters.



Character sets

- Every character has its own **character code** - a binary number that represents that character within a computer system.
- A **character set** is a collection of **all the characters and their binary character codes** that a computer **recognises**.
- Character sets include **alphanumeric** characters and also **special** characters.
 - Alphanumeric characters include letters, numbers and symbols.
 - Special characters perform actions such as starting a new line or moving a cursor along the screen.

The ASCII character set

- The American Standard Code for Information Interchange character set (ASCII for short) is the most common character set.
- Each character in ASCII is represented by a **7-bit** binary code, giving it a maximum of 128 characters.
 - 000 0000 to 111 1111 = 128 unique values
- ASCII includes all upper- and lower-case letters in the **English language**, plus numeric digits, punctuation symbols (such as @, ".! ? etc) and special **command characters**.
- When used in 8-bit systems, the extra bit can be used as a parity bit to ensure correct transmission between computer systems.

ASCII Character Codes

Character	Denary	Binary
A	65	1000001
B	66	1000010
C	67	1000011
D	68	1000100
E	69	1000101
...	...	...
Y	89	1011001
Z	90	1011010

Character	Denary	Binary
a	97	1100001
b	98	1100010
c	99	1100011
d	100	1100100
e	101	1100101
...	...	...
y	121	1111001
z	122	1111010

Notice that the uppercase and lowercase form of each letter has its own binary representation and that there is only 1 bit difference between the binary representation of a letter's uppercase and lowercase form.

See the whole ASCII character set here: <https://theasciicode.com.ar/>

Limitations of ASCII

- Due to using only **7-bits per character**, ASCII can not represent more than **128 characters**, most of which are taken up with characters from the English language.
- This **limits its usefulness** in other countries that use other symbols such as French, Spanish, Russian, Arabic and Japanese.
- To help address these issues, the **Extended ASCII character set** was introduced that used **the full 8-bits** to provide up to **256 characters**.
- However, 256 characters is **still not enough** to represent all of the symbols used by the world's languages.
- To overcome this, Extended ASCII came in **different versions** that contained **different sets of symbols** to cater for different languages. This became **confusing** and led to **documents being unreadable** if the incorrect **character encoding** was used by a text-reading application.

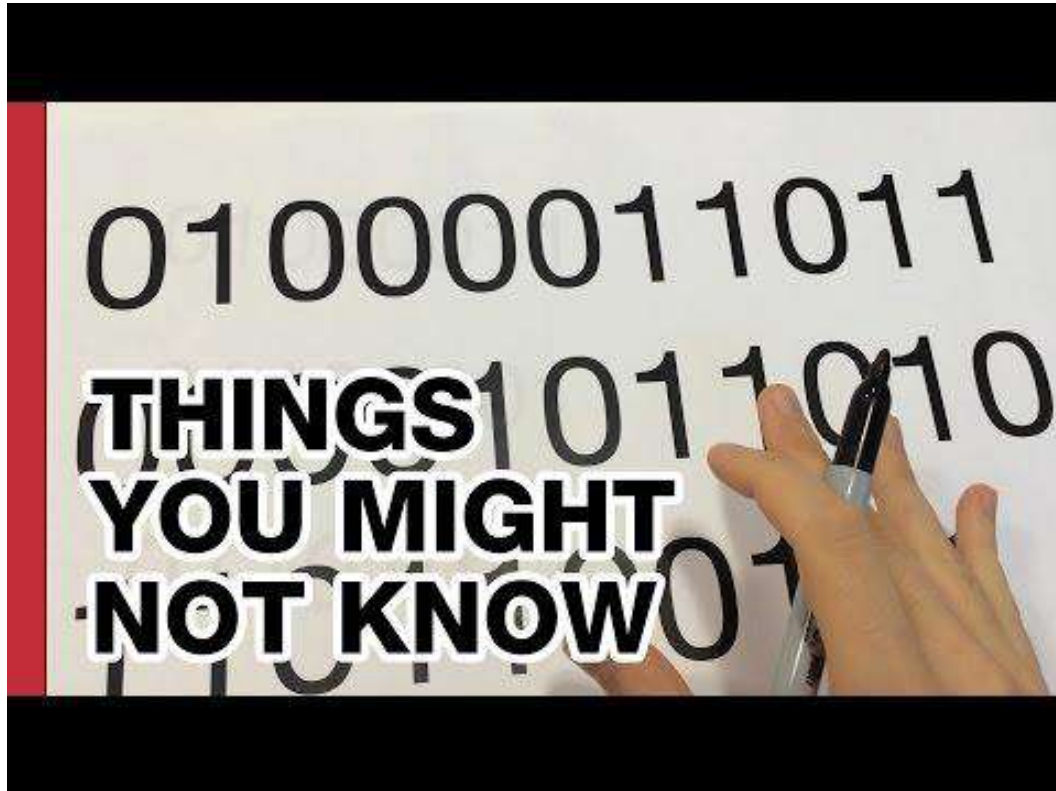
Unicode

- Unicode is a modern character encoding scheme that aims to represent **every possible character** in the world.
- The most common form of Unicode (UTF-8) uses between 8- and 48-bit binary codes to represent each character.
- Crucially, the first 256 characters within Unicode are the same as Extended ASCII, making it backwards-compatible with text documents that have been encoded using these older character sets.
- As well as presenting characters from all of the world's languages, Unicode is also the character set that defines **emoji characters**! ☐

Representing text: Key points

- ✓ Every **character** is represented by a computer using a **unique number** called **character code**.
- ✓ Each character's number is defined by a **character set**.
- ✓ **ASCII**, **Extended-ASCII** and **Unicode** are examples of common character sets.
- ✓ The **number of bits** used to represent each character determines **how many characters** can be represented in a set.

Video: Reading messages in binary



Video by [Tom Scott](#)

Representing images

Representing images

- Most digital images are stored as **bitmaps** - a 'map' of tiny squares called **pixels**.
- The **colour of each pixel** is stored as a **binary value**.
- The **greater the number of bits** used to **represent each pixel**, the **greater the number of unique colours** that are available for each pixel.
- The number of bits used to represent each pixel is called the **colour-depth** of the image:
 - A 1-bit image can contain only two colours, usually black and white.
 - An 8-bit image can be made up of 256 colours.
 - A 24-bit image can be made up of 16.7 million colours!

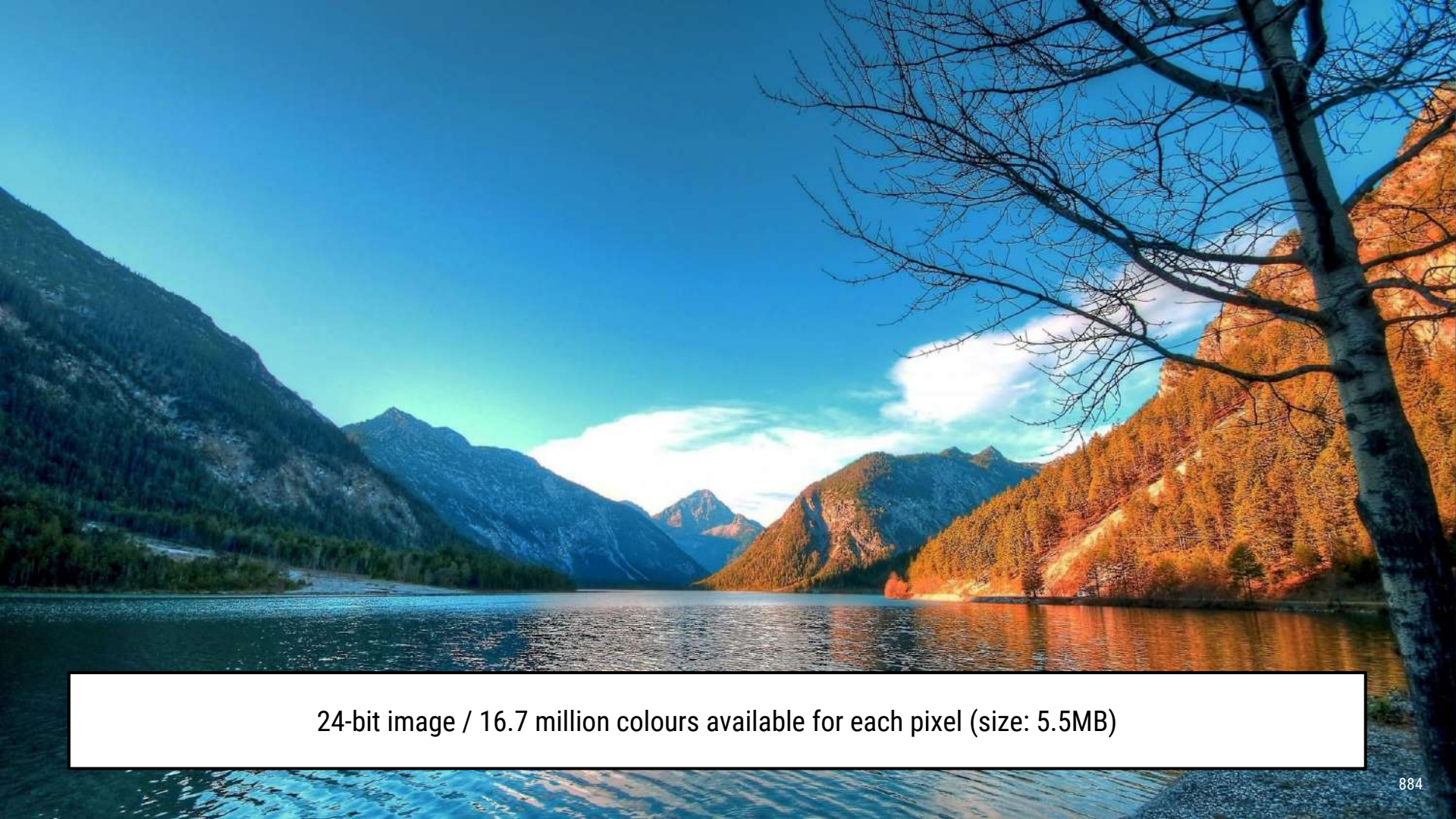
10	10	10	11	10	10	10	10
10	10	11	11	10	10	11	10
10	11	10	10	11	10	10	10
10	10	10	01	01	01	10	10
11	10	01	00	00	00	01	10
01	01	00	00	00	00	00	01
01	00	00	00	00	00	00	01
01	00	00	00	00	00	00	01

a bitmap image

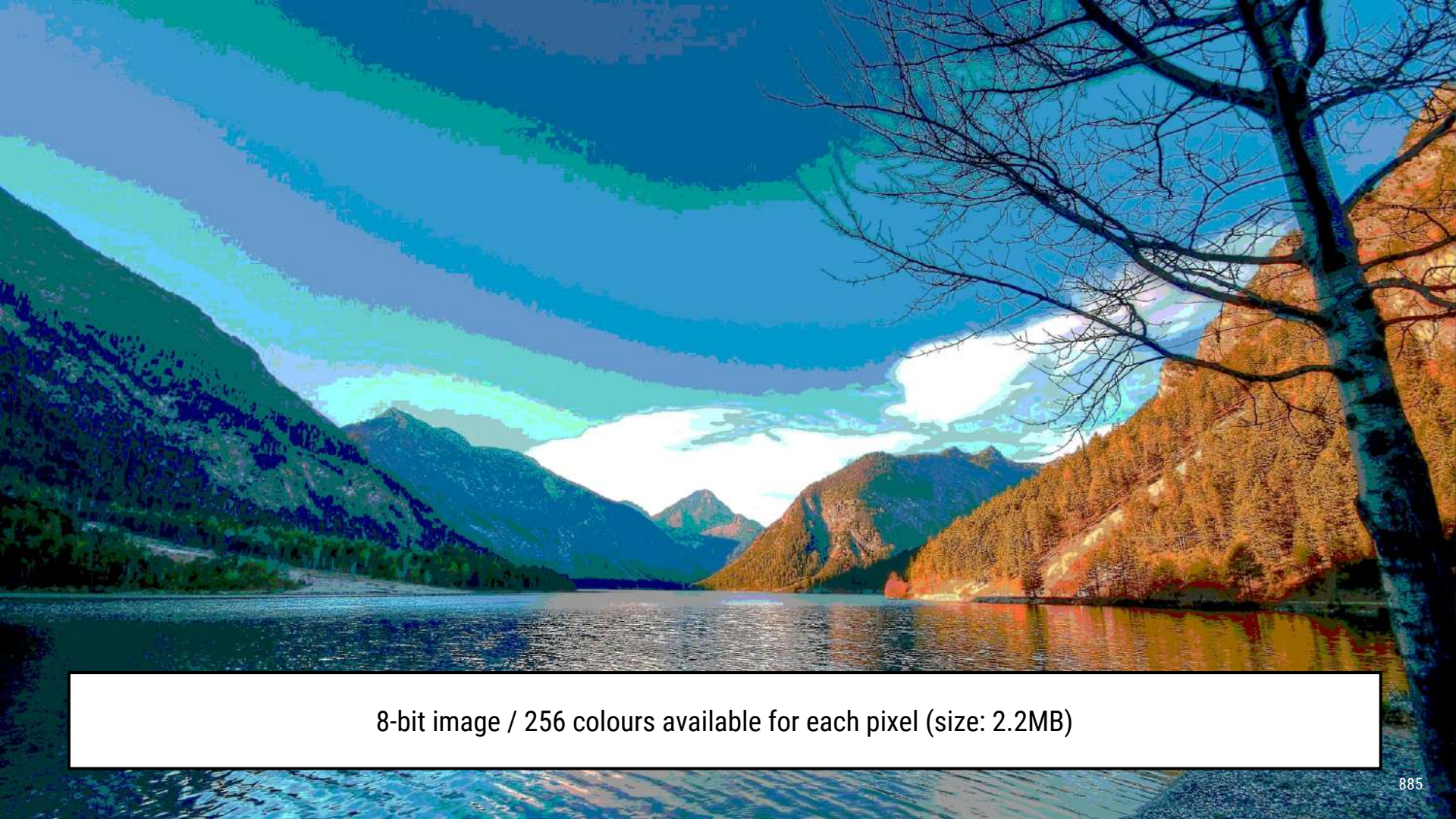
a pixel

Comparing colour-depths

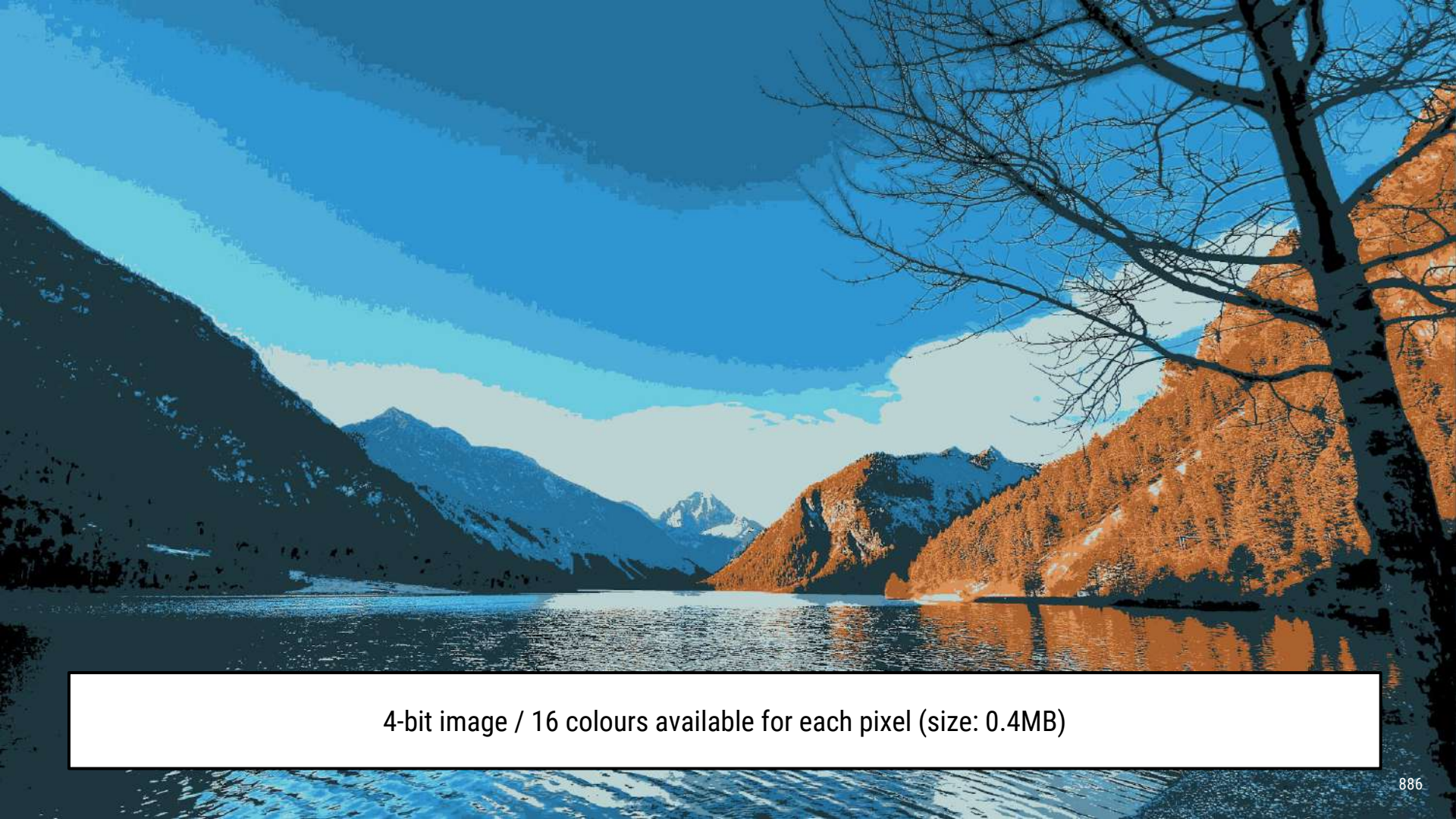
Colour-depth	Number of colours available for each pixel	Space required to encode each pixel
1-bit	2	1 bit
2-bit	4	2 bits
4-bit	16	4 bits / a nibble
8-bit	256	1 byte
16-bit	65,536	2 bytes
24-bit	16,777,216	3 bytes



24-bit image / 16.7 million colours available for each pixel (size: 5.5MB)



8-bit image / 256 colours available for each pixel (size: 2.2MB)



4-bit image / 16 colours available for each pixel (size: 0.4MB)



1-bit image / 2 colours available for each pixel (size: 0.07MB)

How many colours?

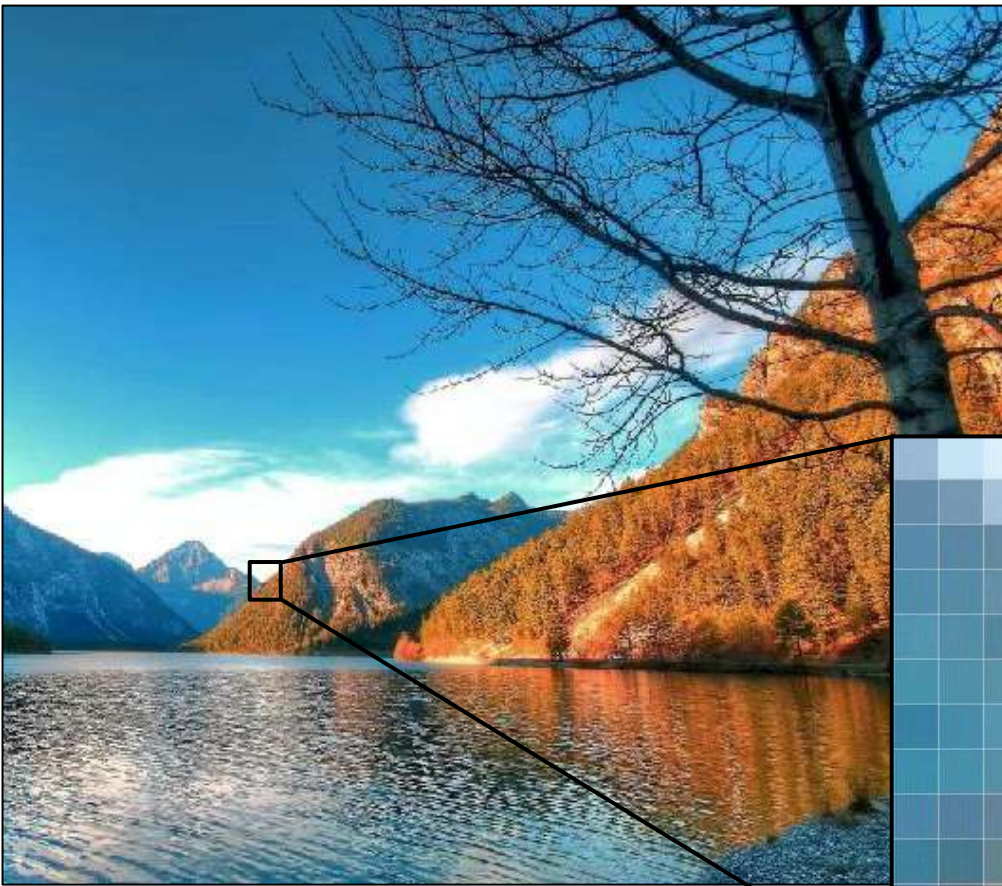
The number of colours available for each pixel can be worked out using the following formula:

$$\text{number of colours} = 2^n$$

where n is the bit-depth of the image

24-bit colour depth

- Most devices such as digital cameras and computer monitors use **24-bit** colour depth.
- **8-bits** are used to indicate how much of each of **red**, **green** and **blue** are needed in each pixel (8-bit per colour x 3 colour 'channels' = 24 bits).
- The human eye can see about **10 million colours**, so 24-bits (16.7 million colours) should cover all the colours we can see.
- The **greater the colour depth** of an image, **the more colours it can represent** and therefore the more accurately it can reflect the real world. However, **using more bits per pixel increases the file size of the image**.



If you zoom in on a bitmap image (such as a photograph) you will see the individual pixels that make up the image.

Notice that each pixel is a single colour.

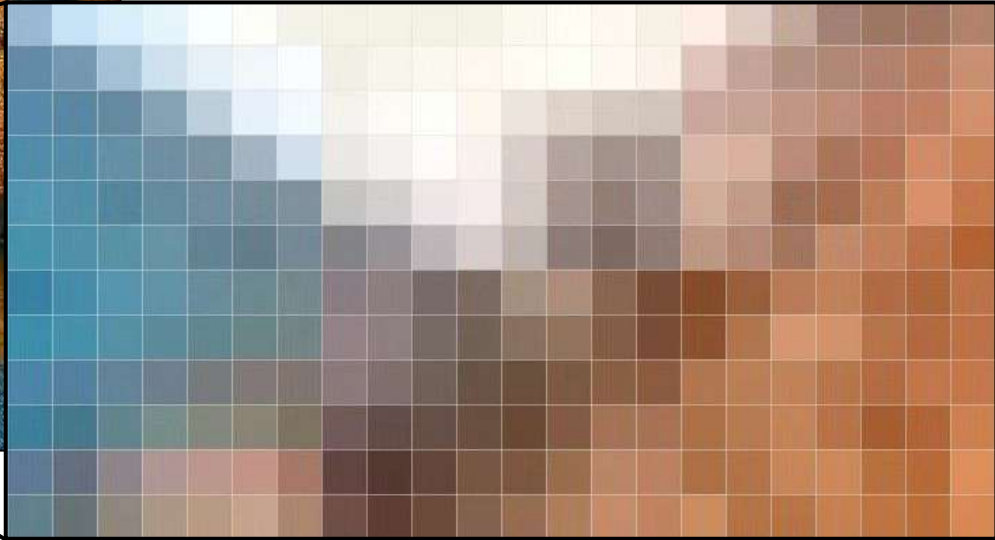
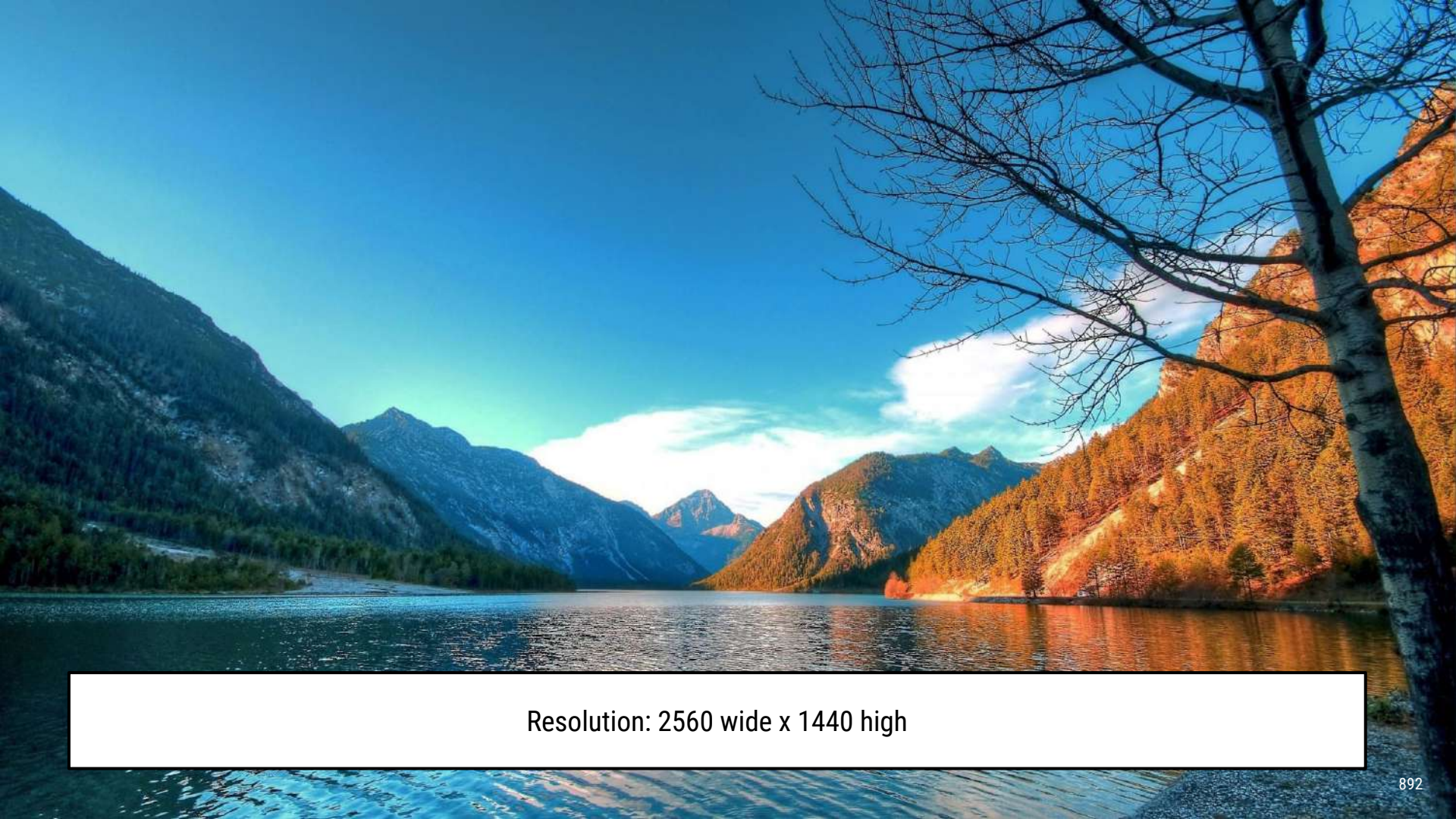


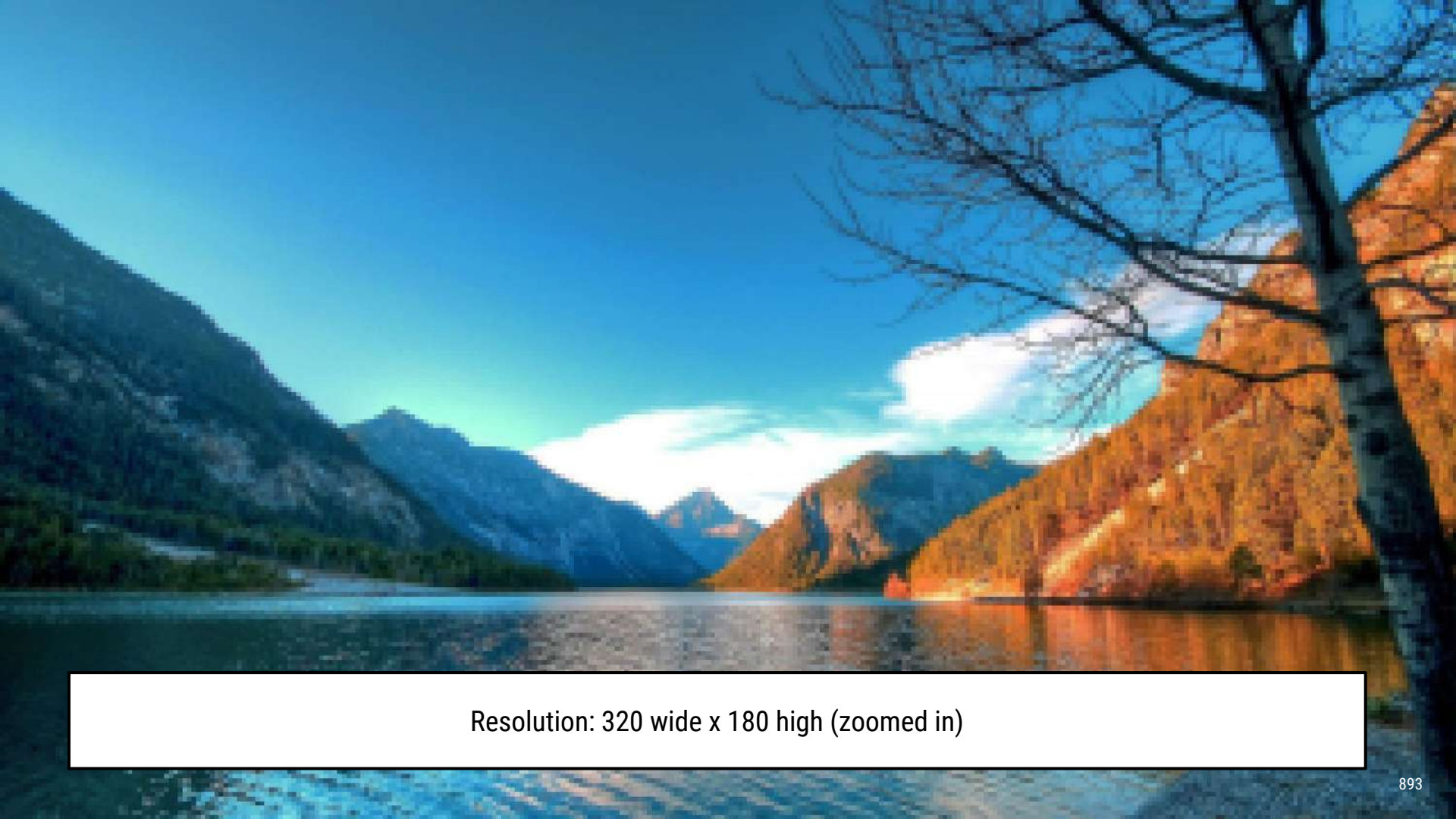
Image from www.desktopbackground.org

Resolution

- As well as the number of colours being used, the quality of a digital image is affected by its **resolution**.
- Exactly what is meant by "resolution" depends on what is being referred to:
 - The resolution of a **display** is the number of pixels that it contains within a given area, sometimes measured in **pixels-per-inch** or PPI. A "Retina display" on a MacBook is around 230 ppi, whereas a traditional computer monitor typically has a resolution of 72 to 96 ppi.
 - The resolution of a **bitmap image** is **number of pixels that make up the image**. This can be calculated by multiplying the height and width of the image. For example, an image that is 3264 pixels wide by 2448 pixels high has a resolution of 7,990,272 pixels (or 8 "megapixels").



Resolution: 2560 wide x 1440 high



Resolution: 320 wide x 180 high (zoomed in)

The effect of colour depth and resolution on file size

- The size of an image file is **directly affected** by the resolution and colour depth of the image:

Higher resolution = more pixels = larger file size

Higher colour depth = more bits per pixel = larger file size

Calculating file size

To calculate the size of an image, use this formula:

$$\text{file size} = \text{image height} \times \text{image width} \times \text{colour-depth}$$

For example a 1024x768, 24-bit image would be:

$$18,874,368 \text{ bits} = 2,359,296 \text{ bytes} = \mathbf{2.25 \text{ MB}}$$

Metadata

- Metadata is **extra information** that is added to a file such as a photo, video or Word document.
- In image files, metadata is used by image-viewing and editing applications to ensure that it is correctly displayed and to provide users with useful information such as:
 - The **resolution** of the image
 - The **bit-depth** of the image
 - The **encoding format** of the image
 - The **time** and **date** that the image was created or last edited
 - The **GPS** location where the image was taken
 - The **camera** used to take the image

Metadata shown in Apple Photos for this photo of a marina, apparently taken in California!

Info

No Title

IMG_0034.jpg

19 August 2016 02:51:37

Apple iPhone 5s

iPhone 5s back camera 4.15mm f/2.2

2551 × 1535 931 KB

JPEG

ISO 32

4.15mm

0 ev

f/2.2

1/1466

No Description

No Keywords

Looking up location info...

HERMOSA BEACH

REDONDO BEACH

Flagler Ln

ridge Ln

W 190th St

Beryl St

Henrietta St

Victor St

Legal

—

+

Image representation: Key points

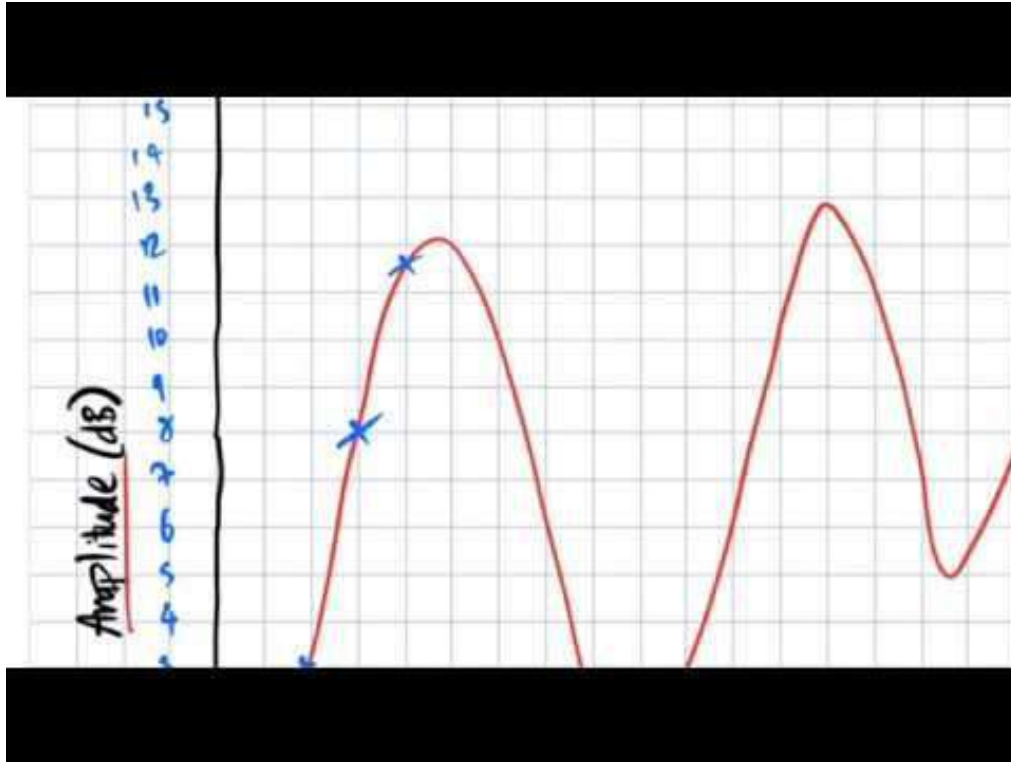
- ✓ **Bitmap images** are made up of squares called **pixels**.
- ✓ The **colour** of each pixel is stored as a **binary value**.
- ✓ **More colour** requires **more bits** per pixel.
- ✓ **Resolution** is a measure of **how many pixels** are used to make up an image - more pixels = higher quality.
- ✓ **Metadata** tells the computer **how to display** an image file properly.

Representing sound

Sampling sound

- All sound travels as **analogue waves**. The **amplitude** of the wave determines the **volume** of the sound and the **frequency** of the wave determines the pitch of the sound.
- When sound is recorded by a computer its volume (amplitude) is **recorded** at **regular intervals**, thousands of times each second. This process is called **sampling**.
- The **value of the amplitude** at each sample is stored as a **binary value**.
- The sampling of the analogue sound wave is done by a device called an **analogue-to-digital converter** (ADC).
- A digital audio file contains the **binary representations of the amplitudes** of the original sound wave that were captured **each time it was sampled**.

Video: How an analogue wave is converted to binary



Video by [Mr A W Dimmick](#)

Sample size and sampling frequency

- The number of **bits** used to store each sample is called the audio's sample size.
- The sample size determines the range of **values** that are available to represent the amplitude, and therefore how accurately (or with what **definition**) the audio is captured (the more bits the better).
- The number of times the amplitude is sampled each second is called the **sampling frequency** or sample rate of the sound file and is measured in **hertz** (Hz) or kilohertz (kHz).
- The amount of **time between samples** is called the **sampling interval**.
- Decreasing the sampling interval will increase the sampling frequency and therefore increase both the quality of the sound and also its file size.
- CD-quality digital audio files use 16-bit samples and a sampling frequency of 44.1kHz.

Video: The effect of sampling frequency on audio quality



Video by [Mr A W Dimmick](#)

Bit-rates and file size

- The **bit-rate** of an audio file is the **number of bits** required to store **one second** of audio.
- The bit-rate of a digital audio recording is calculated using the formula:

$$\text{bit rate} = \text{sampling frequency} \times \text{sample size}$$

- The bit-rate of a recording can be used to calculate the file size of the recording.
- For example, a CD uses two audio channels (for stereo playback), each of which is sampled 44,100 times per second (44.1kHz), using 16-bit samples. Therefore the bit-rate is $2 * 44100 * 16 = 1,411,200$ bits per second.
- A three minute song (180 seconds) would therefore require 254,016,000 bits, or 30MB of space.

Improving the quality of the audio

The quality of a digital audio recording can be improved by:

- Increasing the **sample size**
 - Using **more bits per sample** allows for greater accuracy when converting the amplitude of the sound wave to a digital value.
- Increasing the **sampling frequency** (sample rate)
 - Taking samples **more often** means that the digital representation will more closely match the original recording
 - This is the same as **reducing the sampling interval**.
- However, increasing either the sample size or sample rate will increase the **bit-rate** of the audio file, resulting in a **larger** file.

Compression

Why do we need compression?

- Representing things such as audio, images and particularly videos in digital form often results in **very large files**.
- These files can **use up limited storage capacity** on smartphones and SSD drives and take a **long time to transmit** over the internet.
- Compression is the process of **reducing** the size of a file by applying an **algorithm** to the binary data within the file.
- Depending on the type of compression used, the **compressed file** should either usefully represent its original information but with some loss of quality or it should be possible to perfectly restore the original data.

Types of compression - Lossy and Lossless

There are two types of compression:

- **Lossy** - an algorithm is applied to **remove unnecessary detail** from the original file. Some of the original data is **permanently lost** but **enough remains for the file to still be useful**. For example, compressed audio, video and images where your brain 'fills in the gaps' to make up for the loss in quality.
- **Lossless** - **none of the original data is lost** but instead an algorithm is used to **perfectly restore the original file**. This is used for **executable program files** where all instructions exact and present for a CPU to process them correctly. Running the lossless decompression algorithm adds CPU **overhead** and makes using the files slower.

Comparing lossy and lossless compression

- **Lossy compression** results in a **permanent loss of information** and a **lower quality** digital file, however the loss of quality is often **unnoticeable** and the file size is **much smaller** than the original. Lossy compression is suitable for information that people take in with their senses - audio, images, videos, etc.
- **Lossless compression** transforms the data within a file so that it can be **reconstructed** when it is needed, leading to **perfect restoration** (and no loss in quality). Lossless compression often results in **less reduction in file size** compared to lossy compression, however lossless compression is the **only form suitable for compressing software programs and text data** where no information can afford to be lost.

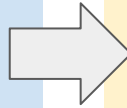
Lossless compression with Run Length Encoding (RLE)

Run-length encoding is a form of lossless compression that replaces repeating sequences of 0s and 1s with more efficient representations.

Original

File size = 12 bytes

```
00110101 00110101 00110101 00110101  
11100101 10110101 10110101 10110101  
01101100 01101100 01101100 01101100
```



Compressed

File size = 8 bytes

```
00000100 00110101  
00000001 11100101 00000011 10110101  
00000100 01101100
```

This line means print 0110110 four times

Compression in images

- **JPEG** is a very common image format that uses **lossy** compression.
- Detail in the image is removed by **averaging over adjacent pixels** and, once removed, the **details cannot be put back**.
- The amount (**strength**) of compression affects the quality and file size.
- The **PNG** and **TIFF** image file formats use **lossless compression**.
- PNG image file sizes are typically **larger** than JPEG but are of **higher quality**.



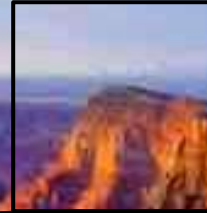
Lossless compression: PNG (size: 20MB). High quality, pin-sharp image with accurate colour.



Lossy compression: JPEG 85% Quality (size: 235KB). Less detail and some loss in colour accuracy.



Lossy compression: JPEG 10% Quality (size: 40KB). A significant loss in quality, but still enough detail for a viewer to know what the picture is of.



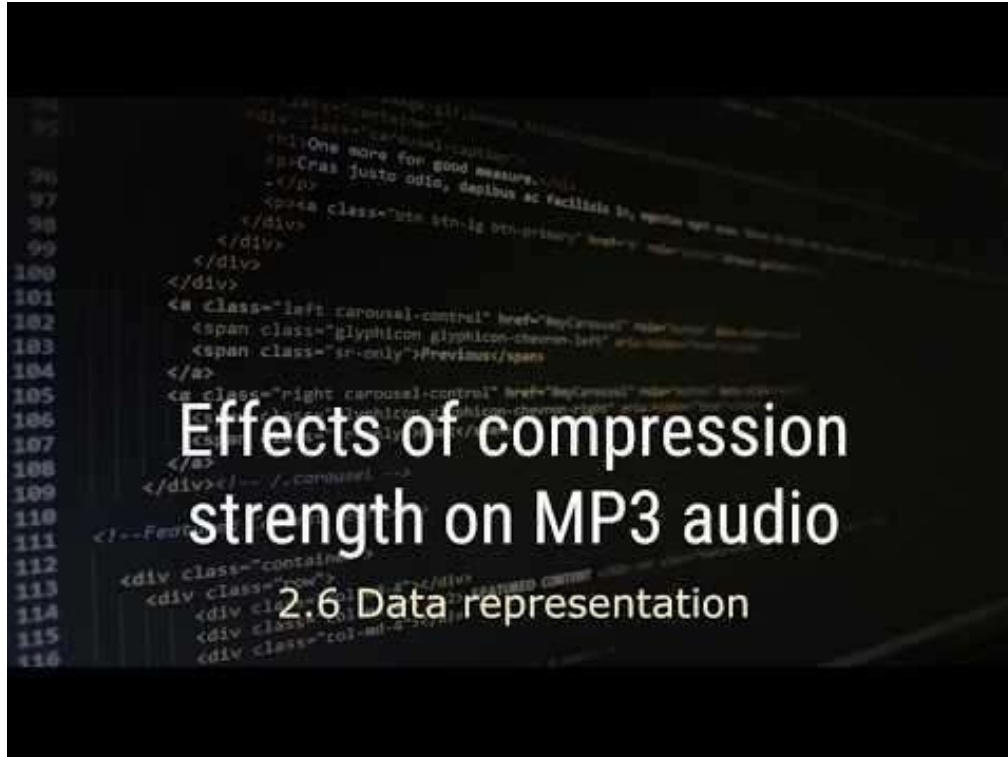
High compression reduces the clarity of the image as many pixels have been merged together, resulting in "blotchy" patches.

Lossy compression: JPEG 10% Quality (size: 40KB)

Compression in audio

- **MP3** is a common digital audio format that uses **lossy** compression to maintain small file sizes.
- It works by first **removing** audio **frequencies** that humans cannot easily **hear**.
- It then looks for **adjacent samples** with similar **values** and replaces them with average values.
- The **strength of the compression** that is applied affects the audio quality. This is measured in kilobits per second as the application of compression ultimately reduces the **bit-rate** of the audio recording.

Video: Example of lossy compression applied to audio



Video by [Mr A W Dimmick](#)



IMAGE SOURCES

Image sources

SYSTEM ARCHITECTURE

Page title	Description	Author	Source URL
The purpose of the CPU	Processor image	D-Kuru	https://upload.wikimedia.org/wikipedia/commons/4/41/AMD_Phenom_II_X4_840_%28HDX840WFK42GM%29_CPU-top_oblique_PNr%C2%B00373.jpg
The Von Neumann CPU architecture	John Von Neumann portrait	US Department of Energy	https://upload.wikimedia.org/wikipedia/commons/d/d6/JohnvonNeumann-LosAlamos.jpg
Factors affecting CPU performance	Speedometer	OpenClipart-Vectors	https://pixabay.com/en/speedometer-tachometer-speed-148960/

Image sources

MEMORY

Page title	Description	Author	Source URL
Main memory: RAM and ROM	RAM module	Laserlicht	https://commons.wikimedia.org/wiki/File:Ram_chip.jpg

Image sources

STORAGE

Page title	Description	Author	Source URL
Magnetic Hard Disk Drives (HDD)	HDD diagram	Surachit	https://en.wikipedia.org/wiki/Hard_disk_drive#/media/File:Hard_drive-en.svg
Magnetic tape	Magnetic tape cassette	Austinmurphy at English Wikipedia	https://en.wikipedia.org/wiki/Linear_Tape-Open#/media/File:LTO2-cart-purple.jpg
Pits	Pits on the surface of a CD-ROM	freiermensch	https://en.wikipedia.org/wiki/File:Afm_cd-rom.jpg
Different types of optical media	Comparison of optical media pits and wavelengths	Cmglee	https://commons.wikimedia.org/wiki/File:Comparison_CD_DVD_HDDVD_BD.svg

Image sources

WIRED AND WIRELESS NETWORKS

Page title	Description	Author	Source URL
Roles of computers in a client-server network	Client computer icon	Amazon Web Services	https://www.shareicon.net/non-specific-client-copy-service-92209
	Server computer icon	Amazon Web Services	https://commons.wikimedia.org/wiki/File:AWS_Simple_Icons_Non-Service_Specific_Traditional_Server.svg
Examples of peer-to-peer networks	Smartphone icon	To Uyen	https://commons.wikimedia.org/wiki/File:Smartphone_icon_-_Noun_Project_283536.svg
	Printer icon	MGalloway (WMF)	https://commons.wikimedia.org/wiki/File:00js_UI_icon_printer-ltr.svg
	Client computer icon	Amazon Web Services	https://www.shareicon.net/non-specific-client-copy-service-92209

Image sources

WIRED AND WIRELESS NETWORKS

Page title	Description	Author	Source URL
Network Interface Controller (NIC)	Network Interface Card	blickpixel	https://pixabay.com/en/network-card-map-pci-riser-board-568043/
	Motherboard	Jonathan Zander	https://commons.wikimedia.org/wiki/File:MicroATX_Motherboard_with_AMD_Athlon_Processor_2_Digon3.jpg
Switches	Path cables in switch	Jon 'ShakataGaNai' Davis	https://commons.wikimedia.org/wiki/File:Network_switches.jpg
	Network switch	Geek2003	https://commons.wikimedia.org/wiki/File:2550T-PWR-Front.jpg
Wireless Access Points	Linksys WAP	Macic7 at English Wikipedia	https://commons.wikimedia.org/wiki/File:Linksys_WAP54G.JPG

Image sources

WIRED AND WIRELESS NETWORKS

Page title	Description	Author	Source URL
Twisted-pair copper cable	Unsheathed twisted pair cable	Baran Ivo	https://commons.wikimedia.org/wiki/File:FTP_cable.jpg
	RJ45 plug underside	Leon Brooks	https://pixnio.com/objects/electronics-devices/computer-components-pictures/network-utp-unshielded-twisted-pair-cable
Fibre-optic cable	Fibre-optic cable ends	blickpixel	https://pixabay.com/en/fiber-optic-cable-glass-fiber-it-502894/

Image sources

NETWORK TOPOLOGIES, PROTOCOLS AND LAYERS

Page title	Description	Author	Source URL
The star topology	Client computer icon	Amazon Web Services	https://www.shareicon.net/non-specific-client-copy-service-92209
	Central switch icon	Open Security Architecture; OSA	http://www.opensecurityarchitecture.org/cms/library/icon-library
The mesh network topology	Client computer icon	Amazon Web Services	https://www.shareicon.net/non-specific-client-copy-service-92209
WiFi channels within the 2.4 GHz frequency band	Overlapping channels diagram	Michael Gauthier, Wireless Networking in the Developing World	https://en.wikipedia.org/wiki/File:2.4_GHz_Wi-Fi_channels_(802.11b,g_WLAN).svg

Image sources

NETWORK TOPOLOGIES, PROTOCOLS AND LAYERS

Page title	Description	Author	Source URL
Private and Public IP addresses	Server computer icon	Amazon Web Services	https://commons.wikimedia.org/wiki/File:AWS_Simple_Icons_Non-Service_Specific_Traditional_Server.svg
Brute-force attacks	Clipart	OpenClipart-Vectors	https://pixabay.com/en/sledgehammer-smash-angry-broken-151228/

Image sources

SYSTEM SECURITY

Page title	Description	Author	Source URL
Example of a malware lure within Skype	Screenshot	Unknown	http://i.imgur.com/Q3F6yMW.png
	Client computer icon	Amazon Web Services	https://www.shareicon.net/non-specific-client-copy-service-92209
	Smartphone icon	To Uyen	https://commons.wikimedia.org/wiki/File:Smartphone_icon_-_Noun_Project_283536.svg

Image sources

SYSTEM SOFTWARE

Page title	Description	Author	Source URL
User Interfaces	DOS screenshot	Public domain	https://upload.wikimedia.org/wikipedia/commons/f/f9/PC_DOS_1.10_screenshot.png
DOS: An OS with a Command Line Interface	MS DOS packaging	Firstversions.com	https://www.firstversions.com/2014/12/ms-dos.html
CLIs today	Windows PowerShell screenshot	Phillip Stewart	https://flic.kr/p/9BYfvZ
	macOS Terminal screenshot	Public domain	https://commons.wikimedia.org/wiki/File:Tcsh_ejecut%C3%A1ndose_en_escritorio_Mac_OSX.png

Image sources

SYSTEM SOFTWARE

Page title	Description	Author	Source URL
Peripheral (device) management	Keyboard and mouse	Public domain	http://www.publicdomainfiles.com/show_file.php?id=13533955827656
	Webcam	Zephyris	https://commons.wikimedia.org/wiki/File:AdventWebcam.jpg
	Printer	Joydeep	https://commons.wikimedia.org/wiki/File:HP_LaserJet_1020_printer.jpg
Device drivers	Printer icon	Tango Desktop Project	https://commons.wikimedia.org/wiki/File:Printer.svg
	Floppy disk icon	Kisspng.com	https://www.kisspng.com/png-floppy-disk-disk-storage-hard-drives-clip-art-file-821478/preview.html

Image sources

SYSTEM SOFTWARE

Page title	Description	Author	Source URL
Types of utility software	Defragmentation screenshot	Unknown	https://i2.wp.com/www.pctechguide.com/wp-content/uploads/2015/11/disk-defrag.gif?ssl=1
	Encryption - document icon	TheNounProject	https://commons.wikimedia.org/wiki/File:Document_icon_(the_Noun_Project_27904).svg
	Encryption - padlock	AzaToh	https://en.m.wikipedia.org/wiki/File:Padlock-gold.svg
	Time Machine	Duncan Hull 	https://www.flickr.com/photos/dullhunk/31563758166
	Compression vice	OpenClipart-Vectors	https://pixabay.com/en/compression-archiver-compress-149782/
Defragmentation utilities	Broken glass	Unknown	http://suptg.thisisnotatrueending.com/archive/13946714/images/1298045425390.jpg

Image sources

SYSTEM SOFTWARE

Page title	Description	Author	Source URL
SSDs and data defragmentation	SSD photograph	Shakib Saifi	https://www.flickr.com/photos/126850594@N06/14714714978
Professional and home backup media	Stack of magnetic backup tapes	Unknown (via www.fixitmobile.com.au)	https://www.fixitmobile.com.au/uploads/89/backup.jpg
	Hard Disk Drive	Darkone	https://commons.wikimedia.org/wiki/File:Hard_disk_Western_Digital_WD1000_1_(dark1).jpg
	USB Flash Drive	Unknown	https://commons.wikimedia.org/wiki/File:SanDisk_Cruzer_Micro.png
	Optical media	Unknown	https://commons.wikimedia.org/wiki/File:CD-ROM.png
	Cloud	95C	https://pixabay.com/en/cloud-isolated-cumulus-transparent-2421760/

Image sources

SYSTEM SOFTWARE

Page title	Description	Author	Source URL
Encryption utilities	Encrypted disk icon	icons.png.com	https://www.soluzionecomputer.it/wp-content/uploads/2017/05/Bitlocker.png

Image sources

ETHICAL, LEGAL, CULTURAL AND ENVIRONMENTAL CONCERNS

Page title	Description	Author	Source URL
The environmental impact of Computing	Trees background	Tama66	https://pixnio.com/nature-landscapes/forest/forest-l-eaf-green-grass-landscape-environment-wood-summer-tree-nature
This is e-waste / The growing problem of e-waste	Dumped computer monitors	techbirmingham	https://c1.staticflickr.com/1/139/345897594_272bf58cc9_b.jpg
Natural resources and raw material	Skiriotissa Copper Mine	MeanStreets	https://sv.m.wikipedia.org/wiki/Fil:Geology_of_Cyprus-SkiriotissaMine.jpg
How much energy does it take to run the Internet?	Energy sources	music4life	https://pixabay.com/en/energy-current-electricity-2344443/

Image sources

ETHICAL, LEGAL, CULTURAL AND ENVIRONMENTAL CONCERNS

Page title	Description	Author	Source URL
Positive impacts of technology on the environment	Sunflower field	Carissa R. Cano	https://i0.wp.com/www.sparkassess.com/wp-content/uploads/2018/01/aesthetic-flower-blooming-free-images-imaiges-our-httpswwwetsycomshop-our-field-of-sunflowers-aesthetic-sunflower-httpswwwetsycomshop-group-all-about-group-field-of-sunflowers.jpg?zoom=2&fit=1900%2C1237
Main source of news by age group	Graph	Reuters Institute, University of Oxford	https://reutersinstitute.politics.ox.ac.uk/sites/default/files/Digital%20News%20Report%202017%20web_0.pdf
The Computer Misuse Act (1990)	Hacker	sik-life	https://pixabay.com/en/anonymous-hacker-network-mask-2821433/

Image sources

ALGORITHMS

Page title	Description	Author	Source URL
Examples of abstraction	Tube map	Transport for London	https://tfl.gov.uk/cdn/static/cms/images/tube-map.gif
Tube map abstraction vs real paths and distances	Realistic tube map	Jc86035	https://commons.wikimedia.org/wiki/File:London_Underground_Zone_1_Highlighted.svg
Decomposition	Animal skeleton	Hbreton19	https://commons.wikimedia.org/wiki/File:Example_of_a_pig_carcass_in_the_dry_decay_stage_of_decomposition.jpg
Binary search - efficient but data must be stored	Phonebook image	Tomasz Sienicki	https://commons.wikimedia.org/wiki/File:Telefonbog_ugt-1.JPG

Image sources

PROGRAMMING TECHNIQUES

Page title	Description	Author	Source URL
Arrays are a little like pigeon holes...	Pigeon holes	Owen Massey McKnight	https://www.flickr.com/photos/addedentry/3273096118/

PRODUCING ROBUST PROGRAMS

Page title	Description	Author	Source URL
Arrays are a little like pigeon holes...	Pigeon holes	Owen Massey McKnight	https://www.flickr.com/photos/addedentry/3273096118/

Image sources

TRANSLATORS AND FACILITIES OF LANGUAGE

Page title	Description	Author	Source URL
Example of assembly language	Comparison of machine and assembly code	Public domain	https://commons.wikimedia.org/wiki/File:Zstr_count_x86_assembly.png

DATA REPRESENTATION

Page title	Description	Author	Source URL
Comparing colour depths	Mountain and river landscape	Unknown	https://www.desktopbackground.org/wallpaper/desktop-hi-res-nature-pics-download-jpg-445114

Version history

Version	Date published	Changes made
1.3	10th November 2018	Corrected CSV user authentication worked example on pages 663-670, adding <code>line.rstrip("\n")</code> before splitting each csv line into a list of values.
1.2	10th October 2018	Corrected algorithms on page 446/447 (pseudocode vending machine algorithm). Added TEDEd video <i>What's in a smartphone</i> on page 349.
1.1	14th September 2018	Corrected number of binary searches required to find an item within a dataset of 1 million and 8 billion (page 476)
1.0	11th July 2018	Original version

Thank you for reading.