

Software

IGCSE Computer Science ■ Topic 4

IGCSE Computer Science

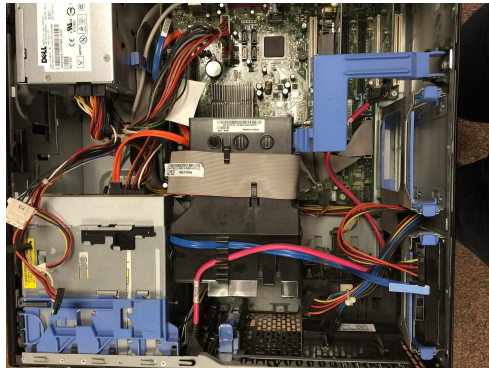
applications

operating system

hardware

What we will cover

- 1 Hardware & software
- 2 System vs application
- 3 The operating system
- 4 Interrupts
- 5 Languages & translators
- 6 Recap

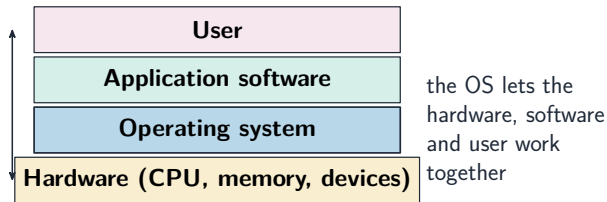




Hardware and software

- **hardware** : the **physical** parts you can touch
- **software** : the **programs** that tell the hardware what to do

Hardware alone does nothing useful; software needs hardware to run on. **Both** are needed.



Hardware is the physical parts; software is

hardware
(physical parts)

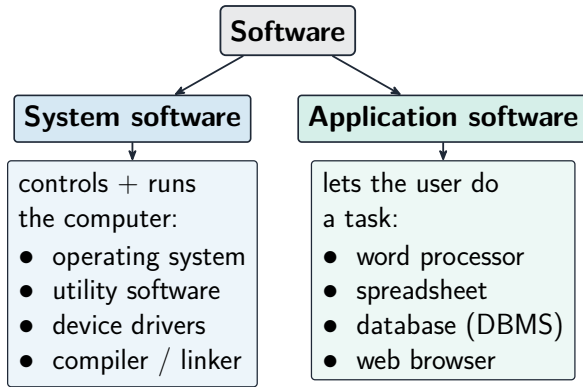
CPU, keyboard, RAM

software
(programs)

apps, the OS

Hardware is the physical parts; software is the programs

System vs application software



- **word processing** software (writing documents);
- **spreadsheet** software (working with numbers in a grid);
- **database management system** (storing and searching data);
- **web browser** (viewing web pages).

The layers of software

application software

does a job for the **user** – word processor, browser, game

the operating system

manages the machine: memory, processes, input and output, security, and **managing files** – saving, opening, naming, moving and deleting

utility software

small tools that **look after the computer** – anti-virus, backup, disk defragmenter, compression

device drivers

let the OS talk to a particular piece of hardware

firmware

sits **below** the operating system – software stored permanently on ROM or flash that runs the hardware at start-up

the bootloader

the first firmware to run; it loads the operating system into RAM

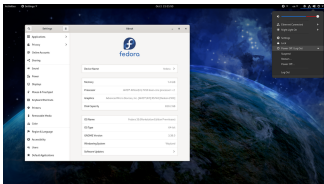
Sort by **who it serves**: the user, the machine, or one device. That is what separates application, system and utility software.

The operating system

The **operating system** runs the whole computer:

- manages files, memory, **peripherals**
- handles interrupts & **multitasking**
- provides a **user interface** and a platform
- manages security – accounts, passwords

It is the **layer between** the hardware and your programs – neither talks to the other directly.



What an operating system does

Providing a user interface

Providing a user interface : the GUI or command line through which a person gives instructions.

Managing peripherals

Managing peripherals : talking to printers, drives and screens through device drivers, so applications do not have to.

Managing multitasking

Managing multitasking : sharing processor time between programs so each appears to run at once.

Providing a platform

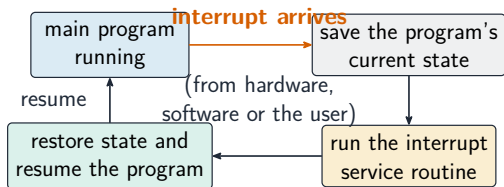
Providing a platform : a common set of services every application can call – which is why a program is written for an OS, not for a machine.

Memory, files, security and users complete the list. Each one is a resource that would otherwise be fought over.

Interrupts

An **interrupt** is a signal that something needs attention **now**. It can come from:

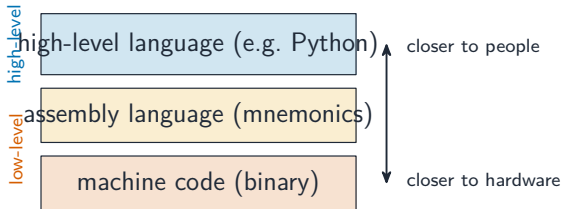
- **hardware** – a key pressed, printer out of paper
- **software** – a divide-by-zero
- **the user** – a cancel key



The processor **saves its place**, services the interrupt, then **resumes** exactly where it stopped.



High- and low-level languages



- **high-level** : English-like (Python); easy to read; runs on **any** computer
- **low-level** : close to the machine; **fast**, but tied to one CPU

High-level is portable and readable; low-level trades that away for speed and direct hardware control.

Compiler vs interpreter

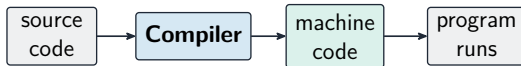
Both are **translators** turning high-level code into machine code:

- **compiler** : translates **all at once** into a file you keep and re-run
- **interpreter** : translates & runs **line by line**, stopping at the first error

A compiler makes a file you re-run; an interpreter stops at the first error – useful while writing. An **IDE** bundles editor, translator and debugger.

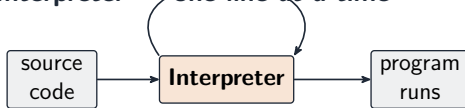
Compiler vs interpreter, in detail

Compiler — translate the whole program once



translated once; the machine code then runs many times

Interpreter — one line at a time



reads, translates and runs one line at a time (every run); stops at the first error

A **compiler** makes a file you re-run; an **interpreter** stops at the first error, which is useful while writing.

Assembler

An **assembler** translates a **low-level assembly-language** program (written in mnemonics such as LDA, ADD) into machine code.

Each mnemonic becomes **one** machine instruction – a direct one-to-one translation, unlike a compiler or interpreter, which work on high-level code.

Choosing a translator, and what an IDE adds

a compiler

translates the **whole** program at once; the finished program runs fast and needs no translator

an interpreter

translates line by line, and **stops at the first error it finds** – which helps while testing

an assembler

translates **assembly language** into machine code, one line to one instruction

the CPU

all computers process data in it, and it executes **machine code only**

auto-completion and auto-correction

the IDE finishes or fixes code as you type

prettyprinting

lays the code out neatly with colour and indentation, so it is easy to read

Use an **interpreter while developing** – errors surface one at a time – and a **compiler for the released program**, which then runs without one.

The translators, and where the linker fits

High-level language

A **high-level language** is close to English and portable; a low-level one is close to the **central processing unit** and machine-specific .

Compiler

Translates the whole **programming language** source at once into an executable, reporting every error at the end. Fast to run afterwards.

Interpreter

Translates and runs line by line, stopping at the first error. Slower, and far easier to debug.

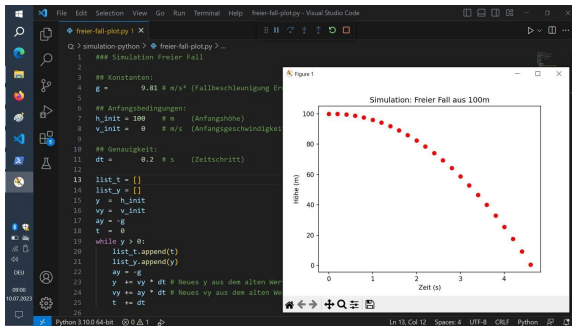
Linker

The **linker** joins the compiled code to the library modules it calls, producing the single file that actually runs.

Development favours the interpreter; delivery favours the compiler. Most real toolchains use both at different stages.

Integrated development environment (IDE)

- a **code editor** – for typing in your program;
- a **run time environment** – for running the program to test it;
- a **translator** – a compiler or interpreter to turn your code into machine code;
- **error diagnostics** – messages that help you find and understand mistakes;



Worked example – which translator, at which stage?

A team wants errors found quickly while developing, and a fast program that hides its source when they ship.

- **While developing:** an **interpreter** – it runs one line at a time and stops at the first error, so a mistake is easy to locate.
- **To ship:** a **compiler** – it translates the whole program once into machine code, which runs fast with no translator present.
- The customer receives only the executable, so the source stays private.

Two different jobs, two different tools. Most real toolchains use both, at different stages of the same project.

Exam tips

- **Hardware** is the physical parts; **software** is the programs. **System software** runs the computer (OS, drivers, utilities); **application software** does a task for the user.
- A **compiler** translates the whole program at once (the finished program runs fast, and all errors are reported at the end); an **interpreter** translates line by line (it stops at the first error, runs slower, and is needed every time).
- **High-level** languages are close to English and run on any computer; **low-level** languages (assembly, machine code) are tied to one type of CPU.
- An **interrupt** is a signal that makes the CPU stop, save its place, service the event, then carry on where it left off.
- Learn the operating system's roles: managing files, memory, peripherals, multitasking, security and the user interface.



Topic 4 – key takeaways

1 Two types

system runs the computer; application serves the user

2 OS

the layer between hardware and applications

3 Interrupt

save place, service, resume

4 Translate

compiler all at once; interpreter line by line

Check yourself

- 1 Is a web browser system or application software?
- 2 What does the processor do first on an interrupt?
- 3 Which translator stops at the first error?
- 4 Which language type runs on any computer?

apps

OS

hardware

Answers 1. application 2. saves its place 3. the interpreter 4. high-level