## Operating System fundamentals

# Purpose of an OS

## What is an operating system (OS)?

- An operating system (**OS**) is software that **provides an interface between the user and the hardware** in a computer system
- An operating system **hides the complexities of the hardware** from the user, for example:
  - A user **does not need to know** 'where' on secondary storage data is kept, just that it is saved for when they want it again
- To help achieve this an OS provides a **user interface**

### User interface

- A user interface is **how the user interacts with the operating system**
- Examples of user interfaces include:
  - **Command Line Interface (CLI)**
  - **Graphical User Interface (GUI)**
  - **Menu**
  - **Natural language (NLI)**

### What is a command line interface (CLI)?

- A Command Line Interface (**CLI**) requires users to interact with the operating system using **text-based commands**
- CLIs are more commonly used by **advanced users**
- Examples of CLIs are MS-DOS (Microsoft Disk Operating System) and Raspbian (for Raspberry Pi)

## What is a graphical user interface (GUI)?

- A Graphical User Interface (**GUI**) requires users to interact with the operating system using **visual elements** such as windows, icons, menus & pointers (**WIMP**)
- GUIs are **optimised for mouse and touch gesture input**
- Examples of GUIs are Windows, Android and macOS

## What is a menu interface?

- A menu interface is **successive menus** presented to a user with a **single option at each stage**
- Often performed with **buttons** or a **keypad**
- Examples include:
  - **Chip and pin machines**
  - **Vending machines**
  - **Entertainment streaming services**

## What is a natural language interface (NLI)?

- A natural language interface (**NLI**) uses the spoken word to respond to spoken or textual inputs from a user
- Examples include:
  - **Virtual assistants** - Amazon Alexa, Google Assistant, Siri
  - **Search engines**
  - **Smart home devices**

## OS management tasks

An operating system has five key **management tasks**:

- **Memory management**
- **File management**
- **Security management**
- **Hardware management**
- **Process management**

## Memory management

- Memory management is a process carried out by the operating system **allocating main memory (RAM) between different programs** that are **open at the same time**
- The OS is responsible for **copying programs and data from secondary to primary storage** as it is needed

- **Programs and data require different amounts of RAM** to operate efficiently and the OS manages this process
- **RAM is allocated** based on **priority and fairness**, for example, system applications (essential) may have a higher priority than user applications
- The OS **dynamically manages the memory**, adjusting allocation as needed to maintain optimal system performance
- Memory management makes **multitasking** possible
  - Prevents data from different programs overwriting each other in RAM
  - Decides which processes and data should be kept in main memory at any given time

## File management

- File management is a process carried out by the operating system **creating, organising, manipulating and accessing files and folders on a computer system**
- The OS **manages where data is stored** in both **main memory** and **secondary storage**
- File management gives the user the ability to:
  - **Create files/folders**
  - **Name files/folders**
  - **Rename files/folders**
  - **Copy files/folders**
  - **Move files/folders**
  - **Delete files/folders**
- The OS allows users to control who can access, modify and delete files/folders (**permissions**)
- The OS provides a **search facility** to find specific files based on various criteria

## Security management

- Operating systems provide various security features such as **password-protected system accounts**, a **firewall**, virus scanning and **file encryption**
- Password-protected system accounts are a very common feature in operating systems
- A system administrator is able to **allocate** different **access rights** for different users on a network
- The OS is able to **maintain settings** for individual users, such as desktop backgrounds, icons and colour schemes
- The OS **audits** (keeps a log of) files created by users, accesses, edits and deletes

## Hardware management

- Peripheral management is a process carried out by the operating system **managing the way peripherals (hardware) interact with software**
- The OS **allocates system resources** to peripherals to ensure **efficient operation**
- Peripheral management makes **plug-and-play** (**PnP**) functionality possible, **automatically detecting and configuring new peripherals** without the need for manually installing device drivers or power cycling the system
- A device driver is **a piece of software used to control a piece of hardware**
- Input/output devices **require device drivers** in order to be used by the operating system
- The OS has **generic device drivers built in** which makes **basic compatibilit**y possible
- In order for hardware to be used to its **maximum capacity**, often **a separate device driver must be downloaded** from the manufacturer
- Device drivers are **OS specific** and are **regularly updated**

## Process management

- Process management is a process carried out by the operating system **dividing time** (**time slicing**) into small chunks and **allocating them to different processes**
- The CPU executes **one instruction at a time**, but rapid context switching allows **multiple processes** to share processor time
- The OS uses a **scheduling algorithm** to **prioritise** processes
- Processes are **placed in queue** whilst waiting to be carried out, they return to the back of the queue when their time is up
- The goal of process management is to **share resources** (**CPU & main memory**)
- Allows processes to be **paused and resumed** using context switching
- Decides **which process is executed next** using a scheduling algorithm

### Worked Example

Identify four key management tasks that the Operating System will perform. **[4]**

**Answer**

- Memory management [1 mark]
- File management [1 mark]
- Security management [1 mark]
- Hardware / device / peripheral / resources management [1 mark]
- Input/output management [1 mark]
- Process management [1 mark]
- Error checking and recovery [1 mark]
- Provision of a platform for software [1 mark]

## Utility software

## Utility software

### What is utility software?

- Utility software is a **collection of tools** designed to help **maintain** a computer system
- Utility software is designed to **perform a limited number of tasks**
- Utility software **interacts with the computers hardware**, for example, secondary storage devices
- Some utility software comes **installed with the operating system**
- Examples of utility software include:
  - **Disk formatter**
  - **Virus checker**
  - **Defragmentation software**
  - **Disk contents analysis/disk repair software**
  - **File compression**
  - **Back-up software**

### Disk formatter

- A disk formatter **prepares a storage device** (like a hard drive or USB stick) for use by **creating a file system** and **organising the space into sectors and tracks**
- It is needed:
  - To **wipe** and **re-initialise** a disk before use
  - To change the **file system format** (e.g. FAT32, NTFS)
  - To **remove all data** and errors before installing a new OS or reusing the drive
- For example:
  - **Formatting** a new external drive before saving files
  - **Reformatting** a corrupted USB stick to make it usable again

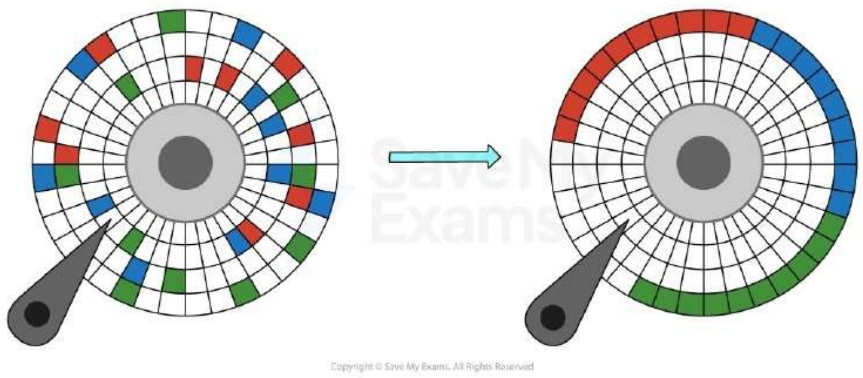
### Virus checker

- Virus checking software is a **combination of different software to prevent** computers from being susceptible to **viruses** and other **malicious software**
- It typically includes:
  - **Anti-virus**

  - **Anti-spam**
  - **Anti-spyware**
- A virus checker **scans** through **email** attachments, **websites** and downloaded **files** to search for issues
- Two ways virus checkers can approach the task are:
  - Use a list of known **unique malware fingerprints** (**signatures**) to block immediately if they try to access your device in any way
  - **Monitor** the **behaviour** of programs to identify suspicious activities that might indicate malware such as;
    - **Rapid deletion/modification of files**
    - **Attempts to access sensitive data/resources**
    - **Communicating with known malicious servers**
- Virus checkers will also perform **checks for updates** to ensure the database of signatures is **up to date**

## Defragmentation software

- Defragmentation software **groups fragmented files back together** in order to **improve the access speed** of a hard disk drive **(HDD)**
- As programs and data are added to a new hard disk drive, it is added in order, **over time as files are deleted this leaves gaps**
- As programs and data are added over time, **these gaps get filled and data becomes fragmented**
- Defragmentation can only be used on **magnetic storage**



## Disk contents analysis/disk repair software

- This utility checks the **structure and health** of a disk, including **files, folders, sectors,** and the **file allocation table**
  - It is needed:
    - To **diagnose and fix errors** on the disk (e.g. bad sectors, file system corruption)
    - To **recover lost or damaged files**
    - To improve system performance by identifying and repairing issues
  - For example:
    - Running **Check Disk (chkdsk)** on Windows to repair file system errors
    - Using a **disk utility** on macOS to verify and repair drive problems

## File compression

- Compression **reduces the amount of secondary storage** required by performing an algorithm on the original data
- **Lossy compression** physically **removes data** from the original data to reduce its size, **the original file can not be re-created**
- **Lossless compression** uses mathematics to **order data** more efficiently reducing its size, **the original files can be re-created as no data is lost**

## Back-up software

- Back-up software is used to **create copies of personal data** in order to keep it safe in the event of:
  - **Accidental loss**
  - **Data theft**
- Backups can be **automated** and **scheduled** to happen at less busy periods of the day, to not take up valuable system resources (e.g. overnight etc.)
- Backups can be made in two ways:
  - **Full** - all files are backed up (saftest, slow)
  - **Incremental** - only files that have been added/modified since the last backup are backed up (faster, less secure)
- Backups can be stored **locally** (secondary storage) or **remotely** (cloud)
- Backup software can be **purchased** or come as a standalone application **bundled** with an **operating system**

**Worked Example**

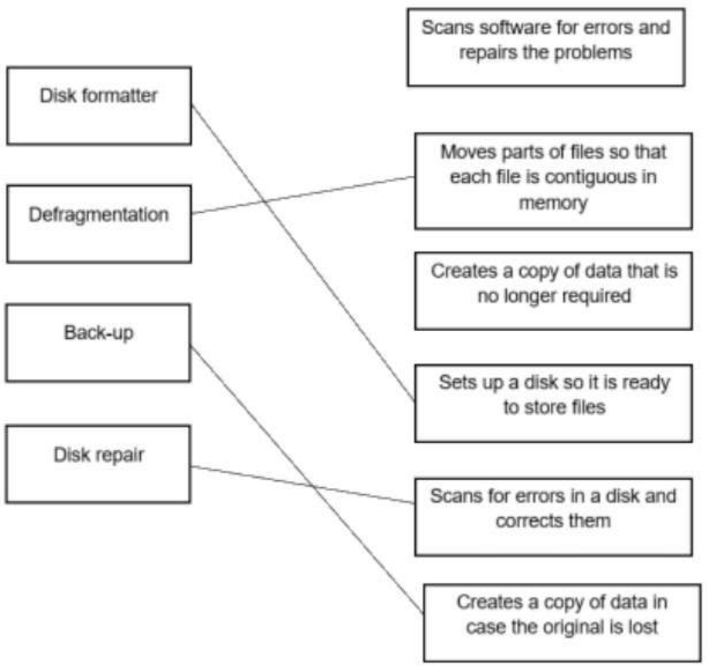
Draw one line from each utility software to its correct description.

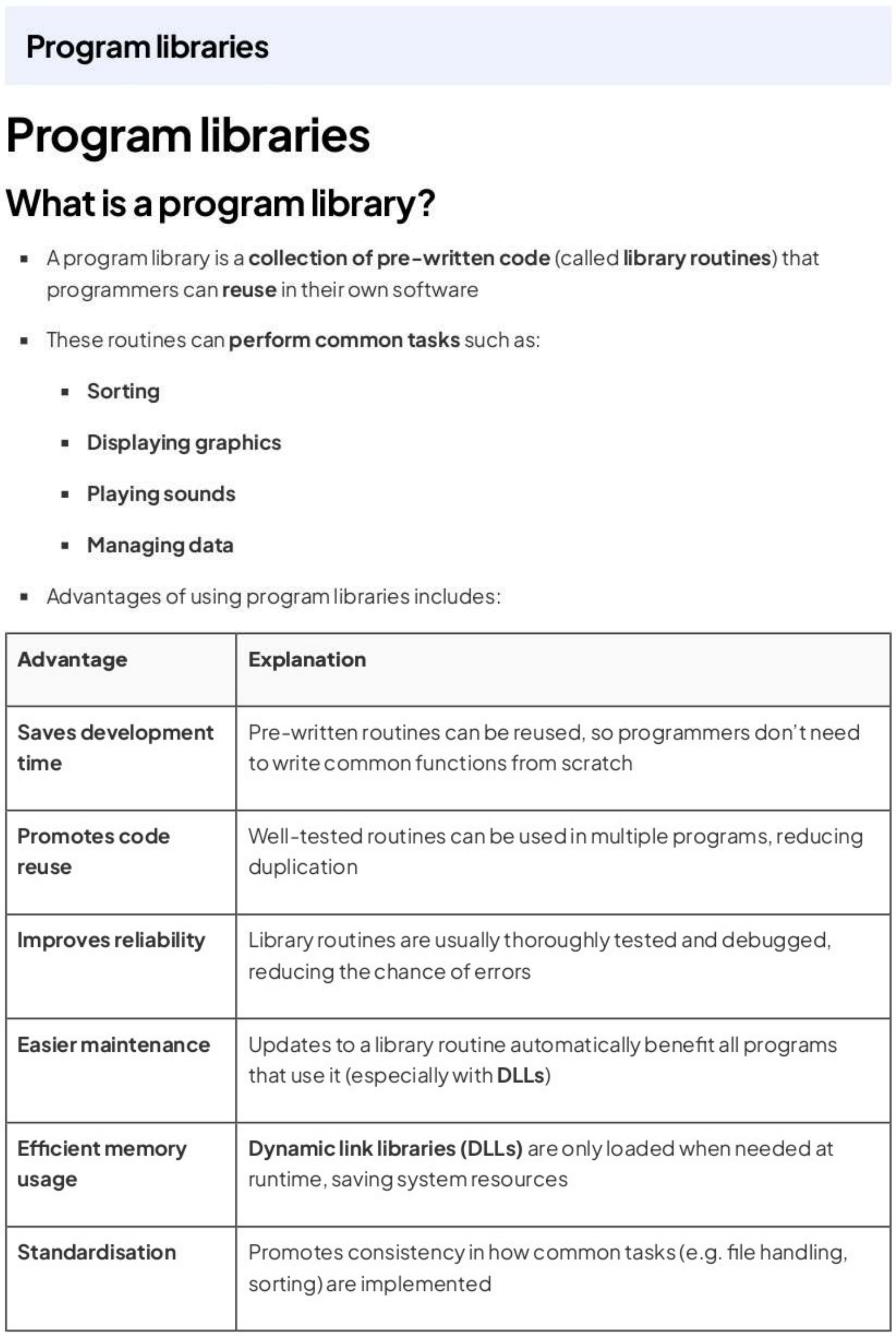
**Utility software Description**

| | |
|---|---|
| | Scans software for errors and repairs the problems |
| Disk formatter | Moves parts of files so that each file is contiguous in memory |
| Defragmentation | Creates a copy of data that is no longer required |
| Back-up | Sets up a disk so it is ready to store files |
| Disk repair | Scans for errors in a disk and corrects them |
| | Creates a copy of data in case the original is lost |

**[4]**

**Answer**


Disk formatter
Defragmentation
Back-up
Disk repair
Scans software for errors and repairs the problems
Moves parts of files so that each file is contiguous in memory
Creates a copy of data that is no longer required
Sets up a disk so it is ready to store files
Scans for errors in a disk and corrects them
Creates a copy of data in case the original is lost



## Program libraries

## Program libraries

### What is a program library?

- A program library is a **collection of pre-written code** (called **library routines**) that programmers can **reuse** in their own software
- These routines can **perform common tasks** such as:
  - **Sorting**
  - **Displaying graphics**
  - **Playing sounds**
  - **Managing data**
- Advantages of using program libraries includes:

| **Advantage** | **Explanation** |
|---|---|
| **Saves development time** | Pre-written routines can be reused, so programmers don't need to write common functions from scratch |
| **Promotes code reuse** | Well-tested routines can be used in multiple programs, reducing duplication |
| **Improves reliability** | Library routines are usually thoroughly tested and debugged, reducing the chance of errors |
| **Easier maintenance** | Updates to a library routine automatically benefit all programs that use it (especially with **DLLs**) |
| **Efficient memory usage** | **Dynamic link libraries (DLLs)** are only loaded when needed at runtime, saving system resources |
| **Standardisation** | Promotes consistency in how common tasks (e.g. file handling, sorting) are implemented |

## What is a DLL?

- A **Dynamic Link Library (DLL)** is a file that contains **pre-written code**, such as **functions or routines**, that can be **used by multiple programs** at **runtime**

| Aspect | Advantages | Disadvantages |
|---|---|---|
| **Program Size** | Reduces program size by keeping reusable code separate | - |
| **Memory Usage** | Saves memory by sharing DLLs between programs | - |
| **Code Reuse** | Encourages reuse of common routines (e.g. sorting, sound) | - |
| **Modularity** | Supports modular design for easier development and maintenance | - |
| **Ease of Updates** | Update a DLL once to improve all programs that use it | Can cause **version conflicts** if newer DLLs aren't compatible |
| **Dependencies** | External DLLs can be reused by many applications | Program may **fail to run** if a required DLL is **missing** |
| **Security** | - | Malicious or altered DLLs can pose **security risks** |
| **Debugging** | - | Errors in DLLs can be **hard to trace** across multiple programs |

### Worked Example

Jennifer is writing a computer program for her A Level homework.

Jennifer uses a program library to help her write her computer program.

Describe how a program library can be used while writing a computer program. **[2]**

**Answer**

- Program libraries store pre-written functions and routines [1 mark]
- The program library can be referenced/imported [1 mark]
- The functions/routines can be called in her own program [1 mark]

## Translator types

## What is a translator?

- A translator is a program that **translates** program **source code** into **machine code** so that it can executed directly by a processor
- **Low-level languages** such as **assembly code** are translated using an **assembler**
- **High-level languages** such as **Python** are translated using a **compiler** or **interpreter**

## Assembler, compiler, interpreter

### What is an assembler?

- An assembler translates **mnemonics written in assembly language** (low-level) in **to machine code**
- **Each lime** of assembly language is **assembled** into a **single machine code** instruction
- Assemblers have been used less and less since high-level languages were introduced

| Advantages | Disadvantages |
|---|---|
| Speed of execution | Difficult to write due to limited and hard to understand commands |
| Optimises the code | Changes mean it must be reassembled |
| Original source code will not be seen | Designed solely for one specific processor |

### What is a compiler?

- A compiler translates high-level languages into machine code **all in one go**
- Compilers translate inputs into machine code **directly**
- Compilers are generally used when **a program is finished** and has been checked for syntax errors
- Compiled code can be **distributed (creates an executable)** and run **without the need for translation software**
- If compiled code contains any errors, after fixing, it **will need re-compiling**

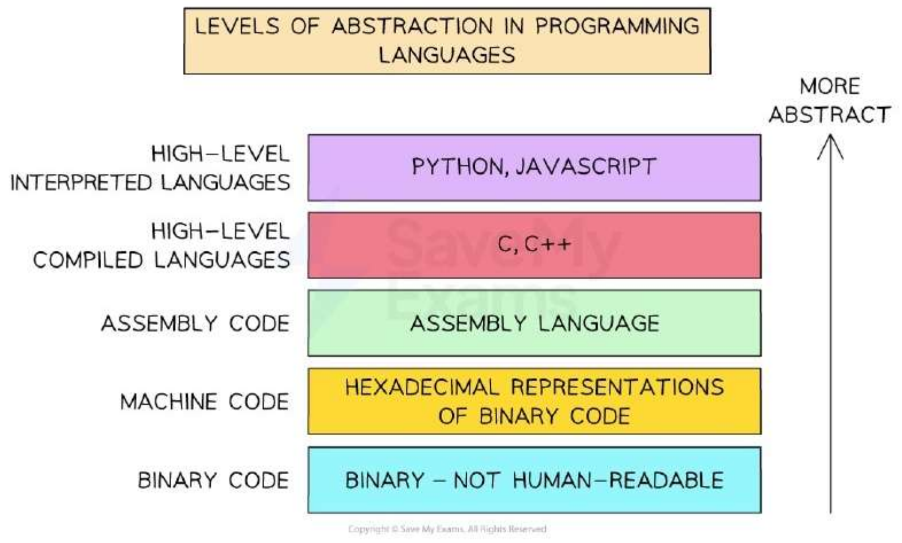
| Advantages | Disadvantages |
|---|---|
| Speed of execution | Can be memory intensive |

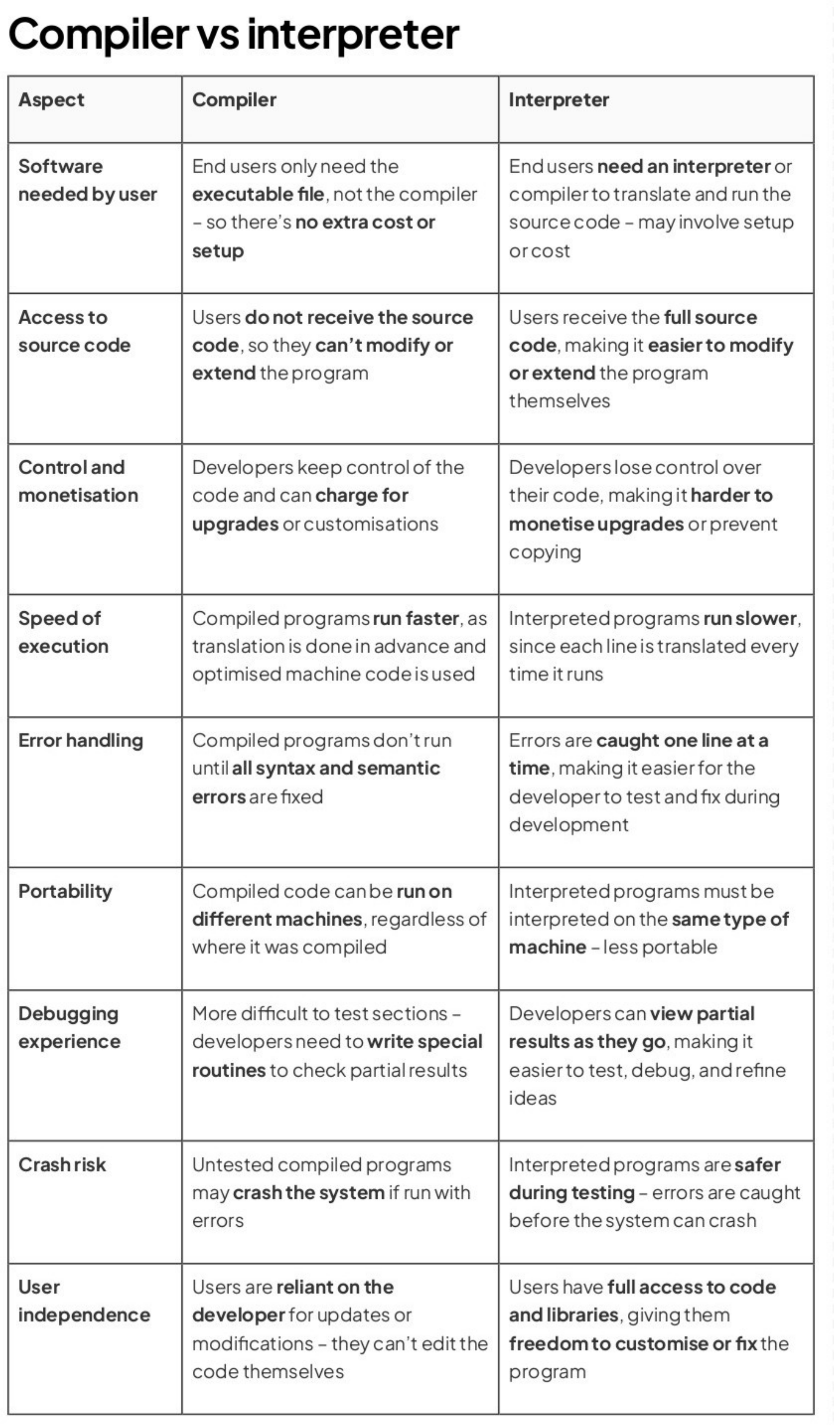
| | |
|---|---|
| Optimises the code | Difficult to debug |
| Original source code will not be seen | Changes mean it must be recompiled |
| | It is designed solely for one specific processor |

## What is an interpreter?

- An interpreter translates high-level languages into machine code **one line at a time**
- Each line is executed after translation and if **any errors are found**, the **process stops**
- Interpreters **do not** generate machine code **directly**, appropriate machine code subroutines are called
- Interpreters are generally used when a program **is being written** in the development stage
- Interpreted code is more **difficult to distribute** as **translation software is needed for it to run**

| **Advantages** | **Disadvantages** |
|---|---|
| Stops when it finds a specific **syntax error** in the code | Slower execution |
| Easier to debug | Every time the program is run it has to be translated |
| Require less RAM to process the code | Executed as is, no optimisation |


LEVELS OF ABSTRACTION IN PROGRAMMING LANGUAGES
MORE ABSTRACT
HIGH-LEVEL INTERPRETED LANGUAGES
PYTHON, JAVASCRIPT
HIGH-LEVEL COMPILED LANGUAGES
C, C++
ASSEMBLY CODE
ASSEMBLY LANGUAGE
MACHINE CODE
HEXADECIMAL REPRESENTATIONS OF BINARY CODE
BINARY CODE
BINARY - NOT HUMAN-READABLE
Copyright © Save My Exams. All Rights Reserved



## Compiler vs interpreter

| Aspect | Compiler | Interpreter |
|---|---|---|
| **Software needed by user** | End users only need the **executable file**, not the compiler – so there's **no extra cost or setup** | End users **need an interpreter** or compiler to translate and run the source code – may involve setup or cost |
| **Access to source code** | Users **do not receive the source code**, so they **can't modify or extend** the program | Users receive the **full source code**, making it **easier to modify or extend** the program themselves |
| **Control and monetisation** | Developers keep control of the code and can **charge for upgrades** or customisations | Developers lose control over their code, making it **harder to monetise upgrades** or prevent copying |
| **Speed of execution** | Compiled programs **run faster**, as translation is done in advance and optimised machine code is used | Interpreted programs **run slower**, since each line is translated every time it runs |
| **Error handling** | Compiled programs don't run until **all syntax and semantic errors** are fixed | Errors are **caught one line at a time**, making it easier for the developer to test and fix during development |
| **Portability** | Compiled code can be **run on different machines**, regardless of where it was compiled | Interpreted programs must be interpreted on the **same type of machine** – less portable |
| **Debugging experience** | More difficult to test sections – developers need to **write special routines** to check partial results | Developers can **view partial results as they go**, making it easier to test, debug, and refine ideas |
| **Crash risk** | Untested compiled programs may **crash the system** if run with errors | Interpreted programs are **safer during testing** – errors are caught before the system can crash |
| **User independence** | Users are **reliant on the developer** for updates or modifications – they can't edit the code themselves | Users have **full access to code and libraries**, giving them **freedom to customise or fix** the program |

D

## Mixed mode translation

## What is mixed mode translation?

- Mixed mode translation **combines features** of both a **compiler** and an **interpreter**
- Instead of fully compiling or interpreting the code, the program is **partially compiled into an intermediate form**, which is then **interpreted at runtime**
- It works by:
    - The source code is **compiled into intermediate code**, not full machine code
    - This **intermediate code** (e.g. bytecode) is **interpreted by a virtual machine** (e.g. the Java Virtual Machine – JVM)
    - Optionally, some parts may later be **just-in-time (JIT) compiled** into machine code for better performance
- **Java** uses mixed mode translation:
    - Java source code is **compiled** to **bytecode** (.class file)
    - Bytecode is **interpreted** (or JIT compiled) by the **JVM** on any platform

| **Feature** | **Compiler** | **Interpreter** | **Mixed mode translation** |
|---|---|---|---|
| Translation | Entire program at once | One line at a time | Compiles to intermediate code |
| Speed of execution | Fastest (fully compiled) | Slowest | Medium (JIT makes it faster over time) |
| Portability | Not portable | Not portable | Highly portable (across platforms) |
| Errors shown | After full compilation | One at a time | During compile or runtime |
| Requires runtime system | No | Yes | Yes (e.g. JVM) |
| Example languages | C, C++ | Python, JavaScript | Java, C# (with .NET CLR) |

**Worked Example**

Jennifer uses an Integrated Development Environment (IDE) to write her computer program.

The IDE allows Jennifer to use both an interpreter and a compiler while creating her computer program.

Describe the ways in which Jennifer can use both a compiler and an interpreter while developing the program.**[4]**

**Answer**

**Interpreter:**

- Use an interpreter while writing the program [1 mark]
- ... to test/debug the partially completed program [1 mark]
- ... because errors can be corrected and processing continue from where the execution stopped // errors can be corrected in real time // errors are identified one at a time [1 mark]

**Compiler:**

- Use the compiler after the program is complete [1 mark]
- ... to create an executable file [1 mark]
- Use the compiler to repeatedly test the same (completed) section [1 mark]
- ... without having to re-interpret every time // compiler not needed at runtime [1 mark]

## Development tools

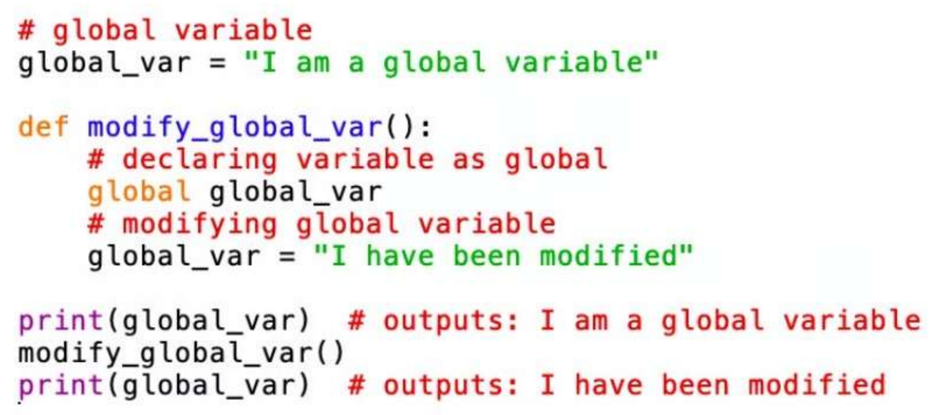
## What is an IDE?

- An **Integrated Development Environment (IDE)** is a software tool that provides programmers with a comprehensive and integrated platform to **write**, **edit**, **compile**, **debug**, and **manage** their code efficiently
- IDEs are designed to **streamline the software development process** by offering a wide range of features and tools that can help programmers
- IDEs can be an **app to download** or can be **web-based**

## IDE features

### Syntax highlighting (pretty printing)

- IDEs use syntax highlighting to **visually distinguish** different elements of code based on their syntax
- The IDE assigns **distinct colours or styles** to keywords, strings, comments, variables, functions, and other language constructs
- Syntax highlighting helps programmers **quickly identify and differentiate code** components, reducing the chance of syntax errors and improving code comprehension
- In this example Python code, you can see that the code comments are displayed in red, the strings are displayed in green and the print commands are displayed in purple



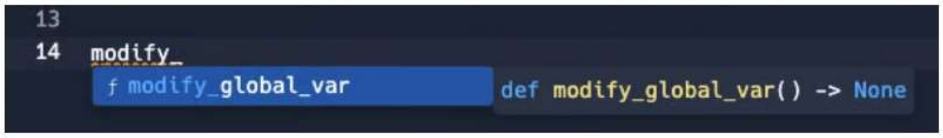
```python
# global variable
global_var = "I am a global variable"

def modify_global_var():
    # declaring variable as global
    global global_var
    # modifying global variable
    global_var = "I have been modified"

print(global_var)  # outputs: I am a global variable
modify_global_var()
print(global_var)  # outputs: I have been modified
```

Syntax highlighting in an IDE

### Autocomplete

- Autocomplete will **suggest code completions** as programmers type
- For example, as developers start typing a variable, function, or method name, the IDE displays a list of available options

- Another example is when programmers use opening brackets and the autocomplete facility will automatically add closing brackets
- Code completion **improves productivity** by reducing typing effort and preventing typing errors and naming inconsistencies
- IDEs can automatically correct common mistakes or typing errors made during coding. When an error-prone sequence is detected, **the IDE can suggest a correction**, helping developers fix minor errors as they type their code

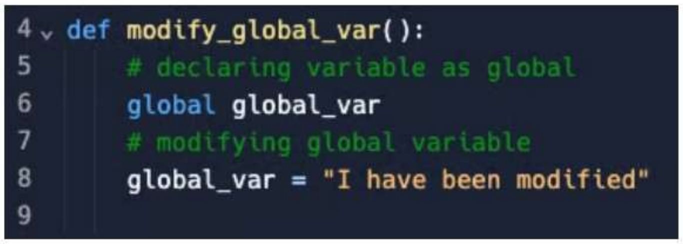


```
modify_
  ƒ modify_global_var        def modify_global_var() -> None
```

Code completion in an IDE

## Auto indent

- Auto indent will **automatically indent the programming code** when you start a new line of code within a code block
- They are commonly used in **selection** and **iteration** programming constructs
- In the example below, the indentation is automatic after starting to define the code block on line 04 and is clearly marked to show which lines of code are included within the block.



```
4  def modify_global_var():
5      # declaring variable as global
6      global global_var
7      # modifying global variable
8      global_var = "I have been modified"
9
```

Code formatting with automatic indentation

## Code comments

- Comments are **lines of text within the code that are not executed** but serve as documentation and explanations for programmers
- IDEs enable developers to add single-line or multi-line **comments to describe the code's purpose**, algorithmic steps, and any relevant information that will be useful to the maintenance of the program
- Comments enhance code understanding, making it easier for developers to revisit and modify code at a later stage
- Comments are in green on the image above

## Syntax error highlighting

- IDEs can automatically highlight **syntax errors in the code editor** as developers type
- This feature **helps catch mistakes instantly**, allowing programmers to address syntax-related issues promptly
- Syntax errors are highlighted with distinct colours, making them easily noticeable, even before executing the code

## Variable watch window

- The watch window is a debugging tool that allows developers to **monitor the values of specific variables** during **runtime**
- Developers can add variables to the watch window and observe their values change as the program executes
- This tool is particularly valuable for understanding how variables evolve throughout the program's execution and diagnosing issues related to variable values

## Breakpoints

- Breakpoints are **markers** that developers can set at specific lines of code to **halt program execution** during **runtime**
- When the program reaches a breakpoint, it pauses, allowing developers to inspect the program's state and variable values and identify the cause of potential bugs
- Breakpoints provide a controlled way to analyse code step-by-step

## Error message list

- The error message list **displays a collection of errors**, **warnings**, and **messages** generated during the compilation and execution of the program
- IDEs present detailed error descriptions and relevant code locations, enabling programmers to address and correct issues systematically

## Stepping mode

- Stepping mode is a debugging mode that allows developers to **execute the program one line at a time**
- Developers can choose between different step options, such as step into, step over, and step out, to examine function calls and control flow
- Step mode offers a granular view of the program's execution, making it easier to identify the exact location of potential bugs

## Crash dump/post-mortem report

- When a program crashes, IDEs can **generate** a crash dump or post-mortem report
- This report **captures the program's state** and **memory** contents **at the time of the crash**, helping developers diagnose the root cause of the crash after the program has terminated unexpectedly

## Stack contents

- The stack contents tool allows developers to **inspect the contents of the program's call stack** during debugging
- It shows the order in which functions were called and the parameters and local variables of each function
- Understanding the call stack, aids in understanding the program's flow and diagnosing issues related to function calls

### Worked Example

Identify two debugging tools that a typical IDE can provide.**[2]**

**Answer**

e.g.

- Breakpoints [1 mark]
- Single stepping [1 mark]
- Report windows [1 mark]