

IG CS Topic 4.4-4.6 Programming Languages

created by Hardy Wen

Types of Programming language

■ Definitions

■ High-level language (HLL)

- uses human-language style words
 - e.g. if, while, output, print, input
- examples include Python and C++

■ Low-level language (LLL)

- included **machine code** and **assembly language**
- **Machine code**
 - binary code, an example of a low-level language
 - **non portable**
 - cannot be run on different types and manufacturers of computers
- **Assembly language**
 - code written in mnemonics that allows direct manipulation of the hardware
 - must be converted into binary code to run
 - using an **assembler**

■ Difference

- Remember to refer to the context when answering the questions

High-level language	Low-level language
Easier to understand, read, and write	Harder to understand, read, write
Portable, machine independent	Not Portable, machine dependent
Explicit - one statement can represent many low-level instructions	Several instructions are needed for each high-level statement
Cannot directly manipulate the hardware	Can directly manipulate the hardware
Easier to debug	Harder to debug
Slower execution: translation process is slow	More efficient in speed and memory usage

Translators

- a type of software that converts encode written in one programming language into another, usually a high-level language into a low-level language

- **Assembler**

- converts the assembly language into machine code

- **Compiler and Interpreter**

- Both of these translators convert a high-level language into a low-level language

Interpreters	Compilers
translates and executes line by line	translates and executes as a whole file
reports the syntax error one by one until it is fixed	reports the all the errors at once
codes need to be translated each time running, so needs the source code and the interpreter software to run	produces an executable file that can be run without the source code and the compiler
tests can occur without completing the whole code	have to finish a section of code before testing it

- **Model Answer: Usage of *Interpreters***

- Remember to refer to the context when answering the questions
 - An interpreter is useful when developing the code. This is because an interpreter runs the code line by line and stops until the current error has been fixed, which offers it the ability to show syntax error one by one and make people to comprehend the error and fix it easily. In addition, you can test a section of the whole code while developing using an interpreter, which also makes it easier to debug.
 - However, an interpreter might not be helpful when the code has already been developed and is ready for sell or distribution. This is because the source code and the interpreter software must exist in order to run the code, and the end users must wait for the program to be interpreted before run it.

- **Model Answer: Usage of *Compilers***

- Remember to refer to the context when answering the questions
 - A compiler is useful when the code is finished and is ready for sell. This is because a compiler creates an executable file which allows the program to be ruined without the compiler software and the source code. Therefore, the users would not need to download the software in advance to run the program, and the no-requirement for the source code protects the intellectual property and decreases the risk of being hacked as sometimes people can find out bugs through the source code and operate some malice attacks based on the bugs.
 - However, a compiler might not be helpful when developing the code. This is because compilers run the code as a whole, and hence would reveal all the errors at once, making it hard to debug as so much bugs might mess up.

Integrated Development Environment (IDE)

- a piece of software that allows a user to write, test and run program code
- **Parts of an IDE**
 - **Editor**
 - a feature of an IDE that allows the user to enter and amend program code
 - **Auto-completion**
 - a feature of an editor that gives the user options when they start typing a command
 - **Auto-correction**
 - a feature of an editor that identifies spelling error and changes them
 - **Prettyprint**
 - a feature of an editor that changes the color of the text such as highlight key words
 - Block minimising
 - a feature of an editor that minimizes a section of code
 - **Translator**
 - a feature of an IDE that provides a relevant translator for the programming language
 - **Run-time environment**
 - a feature of an IDE that allows a program to be run and lets the user interact with the program
 - features that help debugging
 - *Break points*
 - the user can set break points and the program will stop running on this line
 - to check outputs and values
 - *Variable watch window*
 - the window which reveals the variables in the program and its current value
 - *Stepping*
 - the program runs line by line, and the user needs to tell the program to move on
 - allows the program to be checked line by line