Programming Languages, Compilers, Interpretors & Assemblers.

Introduction

(1) We are using Python as a Programming Language.

(2) Python is a type of High Level Language.

(3) There are numerous other High Level Languages.
Examples include Java, Visual Basic, C, C++, C#, R.

(4) Now, computers do not understand High Level Languages.

(5) Computers understand Machine Code/Language or another name for it is Binary Code (and Binary Code is made of 1s' and 0s'.

(6) And Machine Code/Language (or Binary Code) is a type of Low Level Language.

(7) So High Level Languages have to be translated to Machine Code.

Translators

(1) There are 3 types of translators:

(i) Compiler

(ii) Interpretor

(iii) Assembler

(2) First of all, we shall look at Compilers and Interpretors.

Compilers

(1) Compilers work by taking a Source Program/Code written in High Level Language

(2) and converting all of it to an <u>Object Program/Code.</u>

(3) The Object Program/Code is a binary file (meaning it contains 1s' and 0s') of machine instructions.

(4) The Object Program/Code can be executed (which means it can be run) , And the code can be executed swiftly (quickly).

(5) So, the Object Program/Code is also sometimes referred to as executable code.

(6) Now, if there is a syntax error in the Source Program/Code, the Object Program/Code (or executable code) will not form.
Instead the Compiler will report all of the Syntax Errors in the Source Code.

Interpretor

(1) Interpretors work in a different fashion to Compilers.

(2) Interpretors work by taking a Source Program written in High Level Language and converting it Machine Code one line at a time.

(4) Therefore, as the Interpretor converts/translates the lines, the lines are executed/run.

(5) If there is an error in the line, then the Interpretor will stop and report the error message.

(6) Every time the program is run it will have to be converted/translated by the Interpreter.

Compilers and Interpretors Compared

		Compiler	Interpretor
1	**Mode of Operation**	**The Compiler works by translating all of the source code into an object code/ executable form <u>all at once</u>**	**Interpretors work by taking a Source Program written in High Level Language and converting it Machine Code <u>one line at a time.</u>**
2	More on Mode of Operation	The way the Compiler works means that it is needed <u>only once t</u>o create the object code/executable file.	The fact that the Interpretor works by converting/translating one line at a time means that the program will have to be interpreted every time it is run. So, the Interpretor is needed <u>every time</u> you run the program.
3	Speed of execution of the Program	A Compiled Program executes faster as it is already in machine code/executable code.	Interpreted Programs take more time to execute/run as each statement/ instruction is translated before it is executed.
4	Dealing with Errors	A whole list of errors is produced once compilation is complete.	The Interpretor, as it works by translating one line at a time, will stop as soon as it encounters an error and consequently only report one error. This facility makes it very useful for debugging.

Compilers & Interpretors Compared - Contd.

		Compiler	Interpretor
1	**Commercial/ Business**	**Customers/Clients need only see the executable program when the program is distributed and they need not see the source code and so cannot copy it.**	Customers/Clients can see the source code and can copy it.
2	Usage	This is especially good for software which is copyright software as the customer/client cannot see the source code and therefore cannot copy it.	Good for Program Development (when designing programs).
3	Software requirements	Customers/Clients do not need the compiler installed on their computer to run the software.	**Customers/Clients need to have the interpretor installed on their computer and they can see the source code.**

More on Programming Languages - both High Level Languages & Low Level Languages

(1)Programming Languages have evolved over time and now there are 5 generations of Programming languages, since the 1940s'.

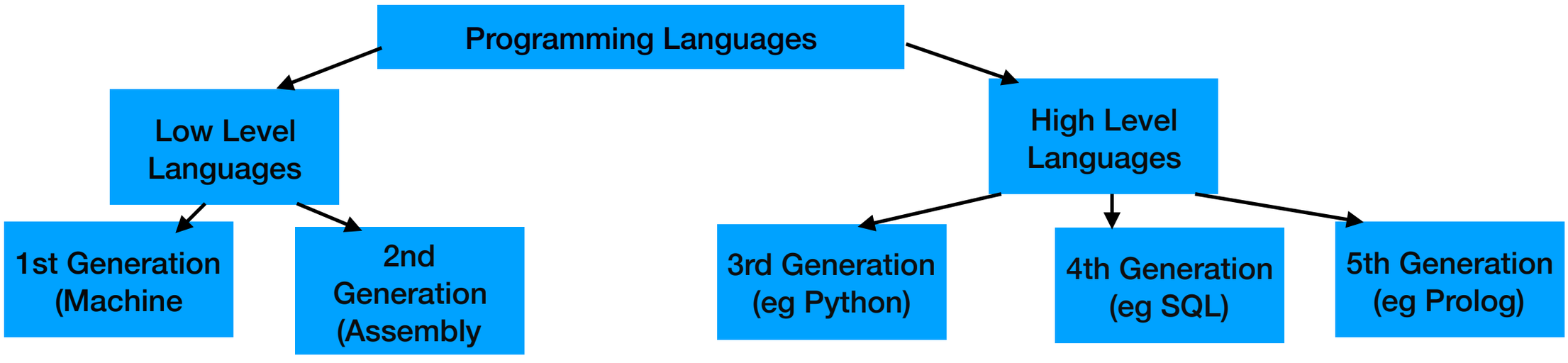
(2) All of the generation of languages are in current use and they all have advantages and disadvantages.

(3) The 1st and 2nd generation of languages are known as Low Level Languages.

(4) The 3rd, 4th, 5th generation of languages are known as High Level Languages.

(5) Low Level Languages are defined as programming languages which contain basic instructions recognised by the computer.

(6) High Level Languages are defined as programming languages which resemble natural languages and are more or less independent of a type of computer.



(7) Let us look at the different generation of languages.

1st Generation (Low Level Languages)

(1) The first programs were written in machine code/language.

(2) Machine Code/Language is written in Binary (1s' and 0s').

(3) Machine Code/Language is a type of Low Level Language.

(4) Being able to program Machine Code required intricate knowledge of the internal workings o the computer i.e they require knowledge of the memory cells (registers) of the computer.

(5) So, a program in Machine Code to add two numbers may look like this:

Code	Meaning
0110000001000000	Load the A-Register from
0100000001000000	Add the contents of
011100000100001	Store the A-Register in

(6) Now, Programming in Machine Code is both time-consuming and prone to errors.

(7) These issues led to the develop of the Assembly Languages - the 2nd Generation, Which are also known as Low Level Languages.

2nd Generation - Assembly Language (Low Level Language)

(1) As mentioned the issues with Machine Code led to the develop of the Assembly Languages - the 2nd Generation of Programming Languages.

(2) In Assembly Language some parts of the Machine Codes instructions are replaced by mnemonics (i.e words)

(3) Please see the figure below to see the equivalent program to add two numbers in assembly language.

Program in Machine Code		Equivalent Program in Assembly Language	
0110000010000000	Load the A-Register from 64	LDA 100	// Load the A-Register from 100 (64)
0100000010000001	Add the contents of 65	ADA 101	// Add to the A-reg the contents of 101 (65)
0111000001000010	Store the A-Register in 66	STA 102	// Store the A-Register contents in 102 (66)

(4) In the example below, LDA, ADA & STA are the

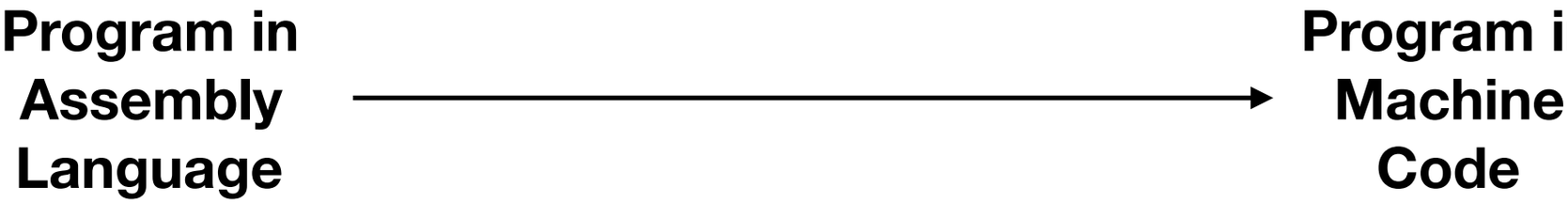
(5)You can see a 1:1 correspondence between Machine Code and Assembly Language instructions.

(6) Mnemonics (pronounced ne-mo-nics) are easier to remember than binary code.

Assembly Language & The Assembler - 3rd type of translator

(1) Note that programs in Assembly language have to be converted to Machine Code before they can be executed.

Assembler - which translates from Assembly Language to Machine Code



Assembly Language Usage

Assembly language is still in use by programmers.

Programmers use Assembly Language when:-

(1) Programs that require details of the computer hardware.

(2) Programs that must be optimised for speed and memory requirements.

3rd Generation Languages (High Level Languages)

(1) Now, Assembly Languages, although easier to program in than Machine Language, still require knowledge of the internal workings of the computer, and still could be improved upon.

(2) This led to the generation of 3rd Generation Languages.

(3) In 1954, FORTRAN, the first 3rd generation language, was devised.

(4) This Programming language was made up of English-like and mathematical statements.

(5) These features should make it easier for a programmer to program.

(6) And the code to add two numbers can look like this:-

X = Y + Z

0110000010000000	**LDA 100**
0100000010000001	**ADA 101**
0111000001000010	**STA 102**

A Comparison of High Level & Low Level Languages

(1) We see there is 1:1 Correspondence between Machine Language and Assembly Language

(2) We see that 1 statement in a High Level Language corresponds to many statements in a Machine Code.

1st Generation -Machine Code	2nd Generation -Assembly Language	3rd Generation - High Level Language
0110000010000000	LDA 100	
0100000010000001	ADA 101	X = Y + Z
0111000001000010	STA 102	

4th & 5th Generation Languages

Fourth Generation Languages :
These are languages that consist of statements that are similar to statements in the human language. These are used mainly in database programming and scripting. Example of these languages include Perl, Python, Ruby, SQL, MatLab(MatrixLaboratory).

Fifth Generation Languages :
These are the programming languages that have visual tools to develop a program. Examples of fifth generation language include Mercury, OPS5, and Prolog.

https://www.geeksforgeeks.org/generation-programming-languages/

(Information provided here for your interest and education)

High Level & Low Level Languages Compared

No.		High Level Language	Low Level Language
1	**Definition**	**High Level Languages are defined as programming languages which resemble natural languages and are more or less independent of a type of computer.**	**Low Level Languages are defined as programming languages which contain basic instructions recognised by the computer.**
2	**Dealing with the Code**	As these programs are written in English-like statements, Code/Programs are easy to read, understand, debug, learn and modify.	Code is very difficult to read, understand, debug, learn and modify.
3	**Writing the Code**	The Programmer does not need to know about the internal structure of the CPU and how it manages memory.	The Programmer needs to know about the internal structure of the CPU and how it manages memory.
4	**Computers/ machines where the programs will work**	The same program/code will work for many different machines and processors.	This is usually written for one type of machine or processor and won't work on any others. So, the code is specific to a particular machine.
5	**Use of Translators**	High Level Language must be translated into Machine Code (Binary Code) before the computer is able to understand it. There are 2 types of translators:- Compilers & Interpretors.	Low Level Languages can be divided into two categories:- (1) Assembly Language and this needs to be translated into Machine Code before execution. (2) Machine Code (or Binary Code) and this does not need any translators.

High Level & Low Level Languages Compared

No.		High Level Language	Low Level Language
1	**Definition**	**High Level Languages are defined as programming languages which resemble natural languages and are more or less independent of a type of computer.**	**Low Level Languages are defined as programming languages which contain basic instructions recognised by the computer.**
6	**Relative Speeds of Running the Programs**	You do not have much control over the what the CPU, so programs can be less memory efficient and slower compared to those written in Low level languages.	As you can control what the CPU does and how it uses memory, programs can be more memory efficient and faster than programs written in High Level Languages.
7	**Relationship between High Level and Low Level Languages**	One instruction of High Level Code represents Many Instructions of Machine Code (Low Level Language).	Low Level Languages can be categorised into Assembly Language & Machine Code. One instruction of Assembly Code equates to One Instruction of Machine Code.
8	**Programming Constructs Available**	High Level Languages have access to programming constructs that low level languages do not have. These include:- 1) The ability to store data in arrays or lists. 2) Usage of IF…THEN…ELSE, FOR…NEXT, WHILE…ENDWHILE 3) Use of Boolean Operators such as:- AND , OR, NOT.	Low Level Languages have a much limited number of programming constructs.

That's all folks for now!!