Chapter 1

Different Types Programming Languages:

Based on Hardware abstraction Programming Languages can be classified in two broad categories

- Low Level Programming Languages
- High-level Programming Languages

Low Level Programming Language:

A **low-level programming language** is a programming language that provides little or no abstraction from a computer's instruction set architecture—commands or functions in the language map closely to processor instructions. Generally, this refers to either machine code or assembly language. The word "low" refers to the small or nonexistent amount of abstraction between the language and machine language; because of this, low-level languages are sometimes described as being "close to the hardware". Low level Languages can be of two types:

Machine Code:Machine code is the only language a computer can process directly without a previous transformation. Currently, programmers almost never write programs directly in machine code, because it requires attention to numerous details that a high-level language handles automatically. Furthermore it requires memorizing or looking up numerical codes for every instruction, and is extremely difficult to modify.

Assembly Code:An **assembly language** (or **assembler language**), often abbreviated **asm**, is any low-level programming language in which there is a very strong correspondence between the instructions in the language and the architecture's machine code instructions. Assembly language may also be called *symbolic machine code*. It provides one abstraction level on top of the machine code

Assembly code is converted into executable machine code by a utility program referred to as an *assembler*. The conversion process is referred to as *assembly*, as in *assembling* the source code. Assembly language usually has one statement per machine instruction.

Advantages of low level languages

1. Programs developed using low level languages are fast and memory efficient.
2. Programmers can utilize processor and memory in better way using a low level language.

3. There is no need of any compiler or interpreters to translate the source to machine code. Thus, cuts the compilation and interpretation time.
4. Low level languages provide direct manipulation of computer registers and storage.
5. It can directly communicate with hardware devices.

Disadvantages of low level languages

1. Programs developed using low level languages are machine dependent and are not portable.
2. It is difficult to develop, debug and maintain.
3. Low level programs are more error prone.
4. Low level programming usually results in poor programming productivity.
5. Programmer must have additional knowledge of the computer architecture of particular machine, for programming in low level language.

High Level Programming Languages:

High-level Languages can be classified as follows

Procedural oriented programming (pop):-

A program in a procedural language is a list of instruction where each statement tells the computer to do something. It focuses on procedure (function) & algorithm is needed to perform the derived computation.

When program become larger, it is divided into function & each function has clearly defined purpose. Dividing the program into functions & module is one of the cornerstones of structured programming.

E.g.:- c, basic, FORTRAN.

Characteristics of Procedural oriented programming:-

1. It focuses on process rather than data.
2. It takes a problem as a sequence of things to be done such as reading, calculating and printing. Hence, a number of functions are written to solve a problem.
3. A program is divided into a number of functions and each function has clearly defined purpose.
4. Most of the functions share global data.
5. Data moves openly around the system from function to function.

Drawback of Procedural oriented programming (structured programming):-

1. It emphasis on doing things. Data is given a second class status even through data is the reason for the existence of the program.

2. Since every function has complete access to the global variables, the new programmer can corrupt the data accidentally by creating function. Similarly, if new data is to be added, all the function needed to be modified to access the data.
3. It is often difficult to design because the components function and data structure do not model the real world.

Object oriented programming : The main idea behind object oriented approach is to combine process (function) and data into a unit called an object. Hence, it focuses on objects rather than procedure.

Characteristics of Object Oriented Programming :

a. Objects:-

Any physical or logical units having specific characteristics which match to the real word are called as object.

Object oriented approach views a problem in terms of objects rather than procedure for doing it.

Objects can be classified below:-

1. Physical objects
2. Elements of the computer user environment
3. Collection of data
4. User defined data types
5. Components in computer games

b. Class:-

A class is a collection of similar objects. E.g. ram, sita, hari are the member of class student.

c. Inheritance:-

Inheritance is the capability of one class to inherit the properties from another class. The child or derived class inherits the characteristics of base or parent class. The child class not only inherits the properties from the base but also have some additional property of its own.

The original class is called bas class and the classes which share its characteristics are called derived class.

Importance of inheritance in Object Oriented Programming :-

In Object Oriented Programming , the concept of inheritance provides an important extension to the idea of re usability. A programmer can take an existing class and without modifying it, add additional features and capabilities to it. This is done by deriving a new class from the existing one. The new class will inherit the capabilities of the old one but also have some additional features of its own.

d. Polymorphism:-

Th word polymorphism is derived from Greek word polymorphism where poly means many and morph means form.

Polymorphism means the ability to take more than one form. It allows different objects to respond to the same message in different ways. It is the ability for a message or data to be processed in more than one form. The same operation is performed differently depending upon the data type it is working upon.

E.g. Consider the operation of addition for two numbers; the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation. Overloading is a kind of polymorphism. It is also an important feature of Object Oriented Programming .

e. Encapsulation:-

It is the way of wrapping both data and functions under a single unit called class. This prevents the data from accidental alternations. It means that data is hidden so that it can’t be accessed mistakenly by functions outside the class.

Ex:C++,Simula,Java

Features / advantages of Object Oriented Programming :-

1. It emphasis in own data rather than procedure.
2. It is based on the principles of inheritance, polymorphism, encapsulation and data abstraction.
3. It implements programs using the objects.
4. Data and the functions are wrapped into a single unit called class so that data is hidden and is safe from accidental alternation.
5. Objects communicate with each other through functions.
6. New data and functions can be easily added whenever necessary.

The disadvantages of object oriented programming language are as follow :

- Sometimes, the relation among the classes become artificial in nature.
- Designing a program in OOP concept is a little bit tricky.
- The programmer should have a proper planning before designing a program using OOP approach.

- Since everything is treated as objects in OOP, the programmers need proper skill such as design skills, programming skills, thinking in terms of objects etc.
- The size of programs developed with OOP is larger than the procedural approach.
- Since larger in size, that means more instruction to be executed, which results in the slower execution of programs.

Object Based Language:

An Object Based language also known as Prototype Language can build actual objects from a constructor function and it has almost any feature that any object could have:

- Constructor.
- Methods
- Properties
- Instances.

But it has no feature that fits the requirements of the definition of object-oriented programming like:

- Polymorphism
- Inheritance
- Encapsulation

Ex:JavaScript

OOP	**POP**
OOP takes a bottom-up approach in designing a program.	POP follows a top-down approach.
Program is divided into objects depending on the problem.	Program is divided into small chunks based on the functions.
Each object controls its own data.	Each function contains different data.
Focuses on security of the data irrespective of the algorithm.	Follows a systematic approach to solve the problem.
The main priority is data rather than functions in a program.	Functions are more important than data in a program.
The functions of the objects are linked via message passing.	Different parts of a program are interconnected via parameter passing.
Data hiding is possible in OOP.	No easy way for data hiding.
Inheritance is allowed in OOP.	No such concept of inheritance in POP.
Operator overloading is allowed.	Operator overloading is not allowed.

C++, Java.	Pascal, Fortran, C.

Note:Although many other types of languages exist like Functional Language,System Language,Scripting Language based on different classification criteria I don't want to delve into it you can explore on your own.

Compiler:

A compiler is a computer program that transforms code written in a high-level programming language into the machine code. It is a program which translates the human-readable code to a language a computer processor understands (binary 1 and 0 bits). The computer processes the machine code to perform the corresponding tasks.
A compiler takes entire program and converts it into object code which is typically stored in a file. The object code is also refereed as binary code and can be directly executed by the machine after linking. Examples of compiled programming languages are C and C++.
A compiler should comply with the syntax rule of that programming language in which it is written. However, the compiler is only a program and cannot fix errors found in that program. So, if you make a mistake, you need to make changes in the syntax of your program. Otherwise, it will not compile.

Interpreter:

An interpreter is a computer program, which coverts each high-level program statement into the machine code. This includes source code, pre-compiled code, and scripts. Both compiler and interpreters do the same job which is converting higher level programming language to machine code. However, a compiler will convert the code into machine code (create an exe) before program run. Interpreters convert code into machine code when the program is run.
Examples of interpreted languages are Perl, Python and Matlab.

Following are some interesting facts about interpreters and compilers.

1) Both compilers and interpreters covert source code (text files) into tokens, both may generate a parse tree, and both may generate immediate instructions. The basic difference is that a compiler system, including a (built in or separate) linker, generates a stand alone machine code program, while an interpreter system instead performs the actions described by the high level program.

2) Once a program is compiled, its source code is not useful for running the code. For interpreted programs, the source code is needed to run the program every time.

3) In general, interpreted programs run slower than the compiled programs.

4) Java programs are first compiled to an intermediate form, then interpreted by the interpreter.

Interpreter	**Compiler**
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.