Programming Fundamentals

-Introduction to Programming-

Introduction to Programming Languages

- **Program**: A program is a set of instructions given to a computer to perform a specific operation.
- The computer only understands binary language/**low-level language**.
- The programs we are going to write are in a **high-level language** which is almost similar to human language.

Why learn programming?

- Can boost **problem solving and logic skills**
- Teaches us combine technical skills and creativity.
- Improves interpersonal skills.
- Can lead to software development jobs
- Helps us understand how software works

Software= program+ documentation (source code/ requirement specification documentation/ software design document, test plans, user manuals etc)

Types of Programming Languages

There are **two types of programming languages**, which can be categorized into the following ways:

1. Low level language
2. High level language

Low level languages

a) Machine Language

- Consists of strings (combination of characters) of binary numbers (i.e. 0s and 1s)
- 011000111 1111000110
- It is the only one language, the processor directly understands.

b) Assembly Language

- Programmer requires detailed knowledge of hardware specification.
- This language uses mnemonics code in place of 0s and 1s.
- ADD,MUL, MOV, POP/PUSH.. (comp ogra, microprocessor)
- The program is converted into machine code by assembler. The resulting program is referred to as an object code.

High level language

- Instructions of this language **closely resembles to human language**.
- Uses mathematical notations to perform the task.
- Easier to learn.
- Requires less time to write and is easier to maintain the errors.
- Converted into machine language by one of the two different languages translator programs; **interpreter or compiler.**

High level language

Differences between Interpreter and Compiler	
Interpreter translates just one statement of the program at a time into machine code.	Compiler scans the entire program and translates the whole of it into machine code at once.
An interpreter takes very less time to analyze the source code. However, the overall time to execute the process is much slower.	A compiler takes a lot of time to analyze the source code. However, the overall time taken to execute the process is much faster.
An interpreter does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory.	A compiler always generates an intermediary object code. It will need further linking. Hence more memory is needed.
Keeps translating the program continuously till the first error is confronted. If any error is spotted, it stops working and hence debugging becomes easy.	A compiler generates the error message only after it scans the complete program and hence debugging is relatively harder while working with a compiler.
Interpreters are used by programming languages like Ruby and Python for example.	Compliers are used by programming languages like C and C++ for example.

a) Procedural-Oriented language

- Designed to express the logic and the procedure of a problem to be solved.
- This programing executes a series of statements that lead to an outcome.
- Usually, it uses heavy loops, multiple variables and some other elements, that is also a major difference between procedural and functional languages.
- Includes languages such as Pascal, COBOL, C, FORTAN, etc.

b) Functional Programming

- Designed on the concept of mathematical functions that use conditional expressions and recursion to perform computation.
- Does not support flow controls like loop statements and conditional statements like If-Else and Switch Statements. They directly use the functions and functional calls.
- Uses Immutable data.
- Avoids the concept of shared state -does not depend on a local or global state, value output depends only on the arguments passed to the function.
- Example: Scala, Haskell

c) Object-Oriented Programming

- Refers to languages that uses objects in programming.
- Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc. in programming.
- Binds together the data and the functions that operate on them so that no other part of the code can access this data except that function.
- Example: Java, C++, Python

Batch (class)

B2 b5 b7 a3 (objects)

Properties of a class (class id, no. of students)

Student

Rekha jaya meena (objects)

Properties (roll no, marks..)

d) **Scripting Programming**

- Supports scripts which are programs written exclusively for a special runtime environment to automate the execution of a specific action/function.
- Scripting languages are written in one language and interpreted within another program, for instance, JavaScript has to be incorporated within HTML which will then be interpreted by the Internet browser.
- Example: JavaScript, VBScript
- <script> starting bracket
- </script> closing bracket