What is a Computer?

A computer is an electronic device that can perform a variety of tasks based on a set of instructions (called a program). It processes data and converts it into meaningful information

Definition:

A computer is a programmable machine that receives input, processes it, stores data, and produces output.

Basic Operations of a Computer:

1. **Input** – Accepts data from the user (e.g., keyboard, mouse)
2. **Processing** – Manipulates data as instructed
3. **Storage** – Saves data permanently or temporarily (e.g., hard drive, RAM)

4.Output – Displays the result to the user (e.g., monitor, printer)

5. Control – Directs the manner and sequence in which all of the above operations are carried out

What is Software?

Software is a collection of programs, procedures, and related data that provides instructions to the computer on how to perform specific tasks.

Definition:

Software is a set of instructions that tell a computer how to perform operations. It enables the hardware to function and helps users interact with the system.

Without software, hardware is non-functional.

Hardware	Software
Physical parts of a computer	Set of instructions or code
Tangible (can be touched)	Intangible (cannot be touched)
Examples: CPU, keyboard	Examples: Windows, MS Word

Types of Software

Software is broadly categorized into the following types:

A. System Software

System software is responsible for managing the internal operations of a computer and controlling the hardware.

Purpose:

To serve as a platform for other software and manage hardware resources.

Sub-types of System Software:

1. **Operating System (OS):**

- Acts as an interface between user and hardware.
- Manages files, memory, processes, devices, and input/output.
- Examples: Windows, Linux, macOS, Unix

2. **Device Drivers:**
    - Special programs that enable communication between the OS and hardware devices.
    - Example: Printer driver, Graphics driver
3. **Utility Software:**
    - Performs specific tasks to manage system resources.
    - Examples: Antivirus software, Disk cleanup tools, File management tools

---

B. Application Software

Application software is developed to help users perform specific tasks such as document creation, calculation, image editing, and browsing the internet.

Purpose:

To fulfill specific user-oriented tasks.

Types of Application Software:

1. General-Purpose Software:
    - Examples: Microsoft Word (word processing), Excel (spreadsheets), PowerPoint (presentations)
2. Custom Software:
    - Developed for a specific organization or task.

- Examples: Hospital management systems, School ERP software

3. Web Applications:
    - Software accessed via web browsers.
    - Examples: Google Docs, Gmail, Facebook

What is Programming?

Definition:
Programming is the process of writing instructions for a computer to perform specific tasks.

Explanation:
Computers cannot think on their own. They need exact instructions to perform any task. Programming involves writing those instructions using a programming language, which the computer can then understand and execute.

Why Do We Need Different Types of Programming Languages?

Definition:
Different types of programming languages exist to bridge the gap between human thinking and computer understanding.

Explanation:
Computers understand only binary (0s and 1s), while humans prefer languages that are readable and easy to write. Different levels of languages — machine, assembly, and high-level — help in this translation and make programming more efficient and accessible.

Types of Programming Languages

Machine Language

Definition:
Machine language is the lowest-level programming language, consisting only of binary code (0s and 1s) that the computer's hardware can directly execute.

Explanation:
This is the only language that the CPU can understand without any translation. Every instruction is written in binary format. Writing in machine language is extremely difficult and error-prone, which is why programmers rarely use it directly. However, understanding it is important for grasping how a computer fundamentally operates.

Assembly Language

Definition:
Assembly language is a low-level programming language that uses symbolic names (mnemonics) instead of binary code to represent machine instructions.

Explanation:
Assembly language is a thin layer above machine language. It uses short codes like `MOV`, `ADD`, `SUB`, which are easier for humans to read and write. However, each assembly language is specific to a particular processor architecture. A tool called an **assembler** converts assembly code into machine code. It gives the programmer more control over hardware and is used in system-level and embedded programming.

High-Level Language

Definition:
A high-level language is a programming language that uses English-like syntax, making it easier to read, write, and maintain.

Explanation:
High-level languages abstract away the complexities of hardware and provide features that help developers focus on solving problems rather than managing memory or processor instructions. These languages require a **compiler or interpreter** to convert the code into machine language. Examples include Python, C++, Java, JavaScript, etc. They are widely used in software development because they are efficient, portable, and easy to understand.

Compiler and Interpreter

Definition:

- **Compiler:** A program that converts the entire high-level code into machine code before execution.
- **Interpreter:** A program that translates and executes code line by line.

Explanation:
High-level languages cannot be directly understood by computers. Compilers and interpreters act as translators. For example, C uses a compiler, while Python typically uses an interpreter. Compilers are generally faster for large programs since the translation happens once. Interpreters are useful for scripting and dynamic execution.

Summary Table:

Language Type	Readability	Execution Speed	Human-Friendly	Computer-Friendly	Examples
Machine Language	Very Low	Very High	No	Yes	Binary code
Assembly Language	Low	High	Somewhat	Yes	MOV A, B
High-Level	High	Medium	Yes	No (needs compiler/interpreter)	Python, Java, C++

Procedural Programming vs. Object-Oriented Programming (OOP)

---

1. Procedural Programming

Definition:
Procedural programming is a programming paradigm based on the concept of

procedures or routines (also known as functions), where the program is divided into a set of instructions that are executed in sequence.

Explanation:
In procedural programming, the focus is on writing procedures or functions that perform operations on data. The code is structured into functions, and data is typically passed between these functions. It follows a **top-down** approach.

- Data is separate from functions.
- Code is written step-by-step, in a logical flow.
- Commonly used in smaller or less complex applications.
- Examples of procedural languages: C, Pascal, FORTRAN.

---

2. Object-Oriented Programming (OOP)

Definition:
Object-Oriented Programming is a programming paradigm based on the concept of "objects", which contain both data (attributes) and methods (functions) that operate on the data.

Explanation:
OOP organizes programs around **objects**, which are instances of **classes**. A class defines the blueprint for an object. OOP allows for **data abstraction, encapsulation, inheritance, and polymorphism**, which help in building scalable and maintainable applications.

- Data and functions are bundled together as a single unit (object).
- Encourages code reuse and modularity.
- Follows a **bottom-up** approach.
- Examples of OOP languages: Java, C++, Python, C#.

Key Differences: Procedural vs. OOP

Feature	Procedural Programming	Object-Oriented Programming
Approach	Top-down	Bottom-up
Focus	Functions and procedures	Objects and classes
Data	Data is global and passed to functions	Data is encapsulated within objects
Code Reuse	Limited reuse via functions	Reuse through inheritance and classes
Modularity	Function-based modularity	Class-based modularity
Security	Less secure; data is exposed	More secure; data is hidden (encapsulation)
Examples	C, Pascal, BASIC	Java, C++, Python, C#

When to Use

- Use **Procedural Programming** for:
    - Small or simple programs.
    - Tasks that are performed in a linear or sequential manner.
- Use **OOP** for:
    - Large-scale software systems.
    - Projects where modularity, reusability, and scalability are important.

Summary in One Line:

- **Procedural Programming** = Code organized by **functions**.

- **Object-Oriented Programming** = Code organized by **objects and classes**