1. What is Program?

a set of instructions that you give to a computer so that it will do a particular task

1. what is programming language?

A programming language is a formal language used to give instructions to a computer. It allows humans to communicate with computers by providing a set of rules and syntax for writing code that can be executed by the machine. Essentially, it's a bridge between human understanding and computer processing.

Key Characteristics:

**Formal Language:**

Programming languages have specific **rules and syntax** that must be followed to write valid code. These rules define how instructions are structured and expressed.

**Instructions for Computers:**

Programming languages provide instructions that computers can understand and execute. These instructions tell the computer what to do, from performing calculations to manipulating data or controlling other devices.

**Used for Software Development:**

Programming languages are essential for creating software, applications, websites, and other technology. They allow developers to build a wide range of digital products and services.

**Examples:**

Popular programming languages include Python, Java, JavaScript, C++, and Ruby.

How it Works:

**1. Humans Write Code:**

Programmers use a programming language to write code, which is a set of instructions.

**2. Compiler or Interpreter:**

The code is then processed by a compiler or interpreter.

**3. Machine Code:**

The compiler or interpreter translates the code into machine code, which is a format that the computer can understand and execute.

**4. Execution:**

The computer executes the machine code, performing the tasks specified by the programming instructions.

### What is a Compiler?

A **compiler** is a **software program** that translates code written in a **high-level programming language** (like C, C++, Java) into **machine code** (also known as **binary** or **assembly code**) that can be executed by a computer’s **CPU (hardware)**.

### 🧠 Software Role of a Compiler

The compiler performs these main tasks:

**Lexical Analysis**: Breaks code into tokens.

**Syntax Analysis**: Parses the structure according to grammar rules.

**Semantic Analysis**: Checks meaning and data types.

**Optimization**: Improves performance of the code.

**Code Generation**: Produces machine-level instructions.

**Code Linking**: Links with libraries and other modules.

All of these are done using **algorithms and data structures** — so it's a **software tool**, written in programming languages like C or C++.

### ⚙️ Interaction with Hardware

Even though a compiler is software:

The **output** it generates is designed to **run on specific hardware** (e.g., x86, ARM CPUs).

It considers **hardware architecture**, such as registers, instruction sets, memory hierarchy, etc., to optimize performance.

For example, compiling for a smartphone (ARM) vs. a laptop (x86) requires the compiler to generate different machine code.

### ❓ So Why the Confusion?

Some people associate compilers with hardware because:

The compiled output **runs directly on hardware**.

Compilers are often part of embedded systems development (close to hardware).

Some low-level tools (like assemblers, linkers) interact directly with hardware-level instructions.

But the compiler itself is **always software**.

### 🆚 Compiler vs Interpreter

| **Feature** | **Compiler** | **Interpreter** |
| --- | --- | --- |
| **Definition** | Translates the whole program into machine code before running it. | Translates and runs code **line by line**. |
| **Execution Speed** | **Faster** (after compilation) | **Slower** (executes code line-by-line) |
| **Error Handling** | Shows **all errors after compilation** | Stops and shows **errors one at a time** |
| **Output** | Generates a **separate executable file** | Does **not** create a separate executable file |
| **Examples of Languages** | C, C++, Go, Rust | Python, JavaScript, Ruby, PHP |
| **Compilation Time** | Takes time **before execution** | No compilation step; runs immediately |
| **Memory Usage** | Typically uses **more memory** | Uses **less memory**, runs in real-time |
| **Use Case** | Best for performance-critical applications | Best for scripting, quick testing, or learning |