Introduction to Programming Language

What is Programming?

A **program** is a set of instructions that tells a computer what to do. Since computers don't understand human languages, we use programming languages to communicate with them.

Programming Languages:

- **Low-Level Languages** – Closer to machine code (like Assembly).
- **High-Level Languages** – Easier to understand (like Java, Python, C++).

Java is a **high-level, object-oriented programming language** that is widely used for software development, web applications, mobile apps, and more.

What is a Programming Language?

A **programming language** is a set of instructions used to communicate with a computer. Since computers only understand **binary (0s and 1s)**, programming languages act as a bridge between humans and machines.

Types of Programming Languages

There are two main types of programming languages:

1. Low-Level Languages (Difficult for Humans, Easy for Computers)

- These languages are closer to how computers work.
- They are **fast** but **hard to read** for humans.
- Examples: **Machine Language, Assembly Language**

A. Machine Language (Binary Code - 0s and 1s)

- The only language a computer understands directly.
- Written in 0s and 1s (binary).
- **Example:** 10101101 11001010 (very hard for humans to understand).

B. Assembly Language (Uses Short Words Instead of Numbers)

- A little easier than machine language.
- Uses short words (called mnemonics) like MOV, ADD, SUB.
- Needs an **Assembler** to convert it into machine language.

2. High-Level Languages (Easy for Humans, Needs Translation for Computers)

- These are closer to human languages, so they are easier to learn.
- They need a **compiler or an interpreter** to convert them into machine code.
- Examples: **Java, Python, C, C++, JavaScript**

How Does a Programming Language Work?

When you write a program, it goes through three steps:

1. **Writing the Code:** You type your instructions in a programming language (like Java).
2. **Translating the Code:** A special tool (compiler or interpreter) changes your code into machine language (0s and 1s).
3. **Executing the Code:** The computer runs the translated code and gives output.

Why Are There So Many Programming Languages?

Different languages are made for different jobs:
- C – Good for making operating systems (like Windows).
- Java – Used for websites, mobile apps (Android), and big companies.
- Python – Used for data science, AI, and automation.
- JavaScript – Used to make websites interactive.

High-Level Programming Languages

A **high-level programming language** is a type of language that is easy for humans to read and write. It allows programmers to focus on logic instead of hardware details. These languages need a **compiler** or **interpreter** to translate them into machine language (binary) that the computer can understand.

Key Features of High-Level Languages:

1. **Human-Readable** – Uses English-like words (e.g., print, if, while).
2. **Portable** – Can run on different types of computers.
3. **Error Handling** – Easier to find and fix mistakes.
4. **Faster Development** – Programs can be written quickly.
5. **Memory Management** – Many high-level languages handle memory automatically.

Types of High-Level Programming Languages

1. Procedural Programming Languages

- Based on **step-by-step instructions** (like a recipe).
- Follows a **top-down approach** (executed line by line).

- Uses functions to organize code.
- Series of well-structured steps and procedure to compose a program.
- Contains a systematic order of statements functions and commands to complete a task.

Example: C, Fortran, Pascal
Interview Example: "C is a procedural language where execution starts from the main() function and follows a structured approach."

Key Point: Functions are the building blocks of procedural languages.

---

2. Object-Oriented Programming (OOP) Languages

- Uses **objects** to structure programs.
- Objects represent **real-world entities** (e.g., Car, Student, Employee).
- Follows principles: **Encapsulation, Inheritance, Polymorphism, and Abstraction**.

Example: Java, C++, Python, C#
Interview Example: "Java is an object-oriented language where real-world objects interact

Key Point: OOP makes code **modular, reusable, and easier to maintain**.

---

3. Scripting Languages

- Used for **automation, web development, and system scripting**.
- Executed line by line using an **interpreter** (no compilation required).

Example: Python, JavaScript, Ruby, PHP
Interview Example: "Python is an interpreted scripting language used for web development, automation, and AI applications."

Key Point: Scripting languages are **lightweight and dynamic**, making them useful for quick development.

---

4. Functional Programming Languages

- Focuses on **mathematical functions** rather than sequences of steps.
- Functions do not change data (immutable state).

Example: Haskell, Lisp, Scala
Interview Example: "Functional programming avoids changing state and uses pure functions to ensure predictability."

Key Point: Functional programming is used in **AI, big data, and high-performance computing**.

---

5. Markup Languages

- Used for **structuring and formatting content**, not for logic-based programming.

Example: HTML, XML, Markdown
Interview Example: "HTML is a markup language used to define the structure of a webpage, such as headings, paragraphs, and links."

Key Point: Markup languages **do not have loops, conditions, or functions** like programming languages.

---

6. Query Languages

- Used for **retrieving and managing data in databases**.

Example: SQL (Structured Query Language)
Interview Example: "SQL is used to query and manipulate databases, allowing operations like selecting, inserting, and deleting records."

Key Point: Query languages **do not perform calculations or logic-based operations** but are essential for databases.

Interview Summary

Common Questions & Answers

Q1: What is a high-level programming language?
"A high-level programming language is designed for humans to easily read and write code. It needs a compiler or interpreter to convert it into machine language."

Q2: What is the difference between procedural and object-oriented languages?
"Procedural languages execute code in a step-by-step manner (e.g., C), while object-oriented languages use objects and classes to structure code (e.g., Java)."

Q3: Why is Java considered an object-oriented language?
"Java follows OOP principles like encapsulation, inheritance, polymorphism, and abstraction, allowing modular and reusable code."

Q4: What is the difference between compiled and interpreted languages?
"Compiled languages (e.g., C, Java) convert code into machine language before execution. Interpreted languages (e.g., Python, JavaScript) execute code line by line."

Is Python a scripting or functional language?
"Python is **primarily a scripting language** because it is interpreted and used for automation.

However, it also supports **functional programming** with higher-order functions and immutability features.

Type	Examples	Used For
Machine Language	0s and 1s	Direct communication with computer
Assembly Language	MOV, ADD	Low-level programming, hardware control
Procedural	C, Pascal	Step-by-step execution
Object-Oriented	Java, C++	Organizing code into objects
Scripting	Python, JavaScript	Web, automation
Markup	HTML, XML	Web page structure
Query	SQL	Database management
Functional	Haskell, Lisp	AI, data science

Programming languages can be classified based on **when variables' data types are checked**:

Feature	Static Languages (Java, C, C++)	Dynamic Languages (Python, JavaScript)
Type Checking	At **compile time** (before execution)	At **runtime** (while executing)
Developer Control	**More control** (strict typing, explicit declarations)	**Less control** (language decides types dynamically)
Flexibility	Less flexible, but safer	More flexible, but riskier
Error Detection	Errors are **caught early** (before execution)	Errors **occur at runtime** (may cause crashes)
Memory Management	**Manual in some languages (C, C++)**	**Automatic (garbage collection)**
Performance	**Faster execution** (compiler optimizes code)	**Slower execution** (checks types at runtime)
Use Case	Large projects, high-performance applications	Quick development, scripting, prototyping

Static Languages (C, C++, Java)

Dynamic Language: Python, JavaScript

Static vs. Dynamic Language

Q1: What is the difference between a static and a dynamic language?
Answer: "A static language checks variable types **at compile time**, meaning errors are caught early (e.g., Java, C). A dynamic language checks types **at runtime**, making it more flexible but prone to runtime errors (e.g., Python, JavaScript)."

Q2: Which is better: Static or Dynamic Typing?
Answer: "It depends on the use case. **Static typing** is better for performance and error prevention, while **dynamic typing** is more flexible and speeds up development."

Q3: Can a language be both statically and dynamically typed?

Answer: "Yes! Some languages, like **TypeScript**, allow both. TypeScript is based on JavaScript but adds static typing, letting developers **choose** whether to use strict types or not."

Q4: Who has more control over the program – a developer using static or dynamic languages?
Answer: "In **statically typed languages (Java, C, C++)**, the developer has **more control** because they must declare variable types, manually manage memory (in some cases), and handle strict type checking before execution. In contrast, **dynamically typed languages (Python, JavaScript)** allow the language to handle types automatically, reducing control but increasing flexibility."

Q5: Which type of language is better for AI and ML, Static or Dynamic?

Answer: "Dynamic languages like **Python** are better for AI and ML because they allow **quick testing, flexibility, and easy integration** with libraries like TensorFlow and PyTorch. However, for high-performance AI in real-time applications (like robotics), **C++ is used** for speed and control."

Q6: Why do large companies use Java for backend services instead of Python?

Answer: "Java is statically typed, meaning it provides **better performance, scalability, and security** for handling **millions of users**. Python is easier to write but slower, making it less suitable for high-load applications like Amazon and Netflix."

Q7: Why do startups prefer Python and JavaScript?

Answer: "Startups need to develop products **quickly** with minimal effort. **Python and JavaScript** allow fast prototyping without worrying about strict rules. This helps startups launch faster and iterate based on user feedback."

Memory Management

What is Memory Management?

Memory management is the process of **allocating and freeing memory** so that a program runs efficiently **without wasting resources or crashing** due to memory issues.

Think of memory as a **hotel**:

- Each **guest (variable, object, function, etc.)** needs a room (memory space).
- The hotel manager (**memory manager**) decides where to place guests and when to free rooms.
- If guests don't leave after checkout (**memory leaks**), the hotel runs out of rooms, just like a program may **crash due to memory exhaustion**.

Two Types of Memory – Stack & Heap

Every program uses **two main types of memory**:

Memory Type	Characteristics	Example
Stack Memory	Small & Fast, Stores local variables, function calls	**Local variables inside a function**
Heap Memory	Large & Slow, Stores objects & dynamically allocated data	**Objects created with new in Java**

Stack Memory – Fast but Limited

- Stack memory is like a **pile of plates** (Last In, First Out – LIFO).
- It automatically allocates and deallocates memory.
- Used for storing **local variables, function calls, and method parameters**.
- **Fast but limited in size** (stack overflow happens if it runs out of memory).

Example (Java – Stack Memory)

```java
public class StackExample {
    public static void main(String[] args) {
        int x = 10;  // Stored in stack
        int y = 20;  // Stored in stack
        add(x, y);
    }

    static void add(int a, int b) {
        int sum = a + b;  // sum is stored in stack memory
        System.out.println(sum);
    }
}
```

Here, x, y, a, b, and sum are stored in the **stack**. When the method ends, their memory is automatically freed.

Heap Memory – Large but Slower

- Heap memory is like a **storage room**—large but messy.
- Used for storing **objects and dynamically allocated data**.
- The **developer must manage memory manually (in some languages).**
- Java automatically manages heap memory using **Garbage Collection (GC)**.

Example (Java – Heap Memory)

```java
class Student {
    String name;
    int age;

    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

public class HeapExample {
    public static void main(String[] args) {
        Student s1 = new Student("Ruchi", 20);  // Stored in heap
        Student s2 = new Student("Verma", 22);  // Stored in heap
    }
}
```

Here, s1 and s2 are objects stored in **heap memory**.

How Memory is Managed in Different Languages

Language	Memory Management Type	Who Controls Memory?	Garbage Collection?
C	Manual (Explicit)	Developer (via malloc & free)	No
C++	Manual (Explicit)	Developer (via new & delete)	No (unless using smart pointers)
Java	Automatic (Implicit)	JVM	Yes (Garbage Collector)
Python	Automatic (Implicit)	Python Interpreter	Yes (Reference Counting & GC)
JavaScript	Automatic (Implicit)	JS Engine (e.g., V8)	Yes

Garbage Collection (GC) – Automatic Cleanup (Java & Python)

- **Garbage Collector (GC)** automatically removes unused objects from heap memory.
- Prevents **memory leaks** (unused objects wasting memory).
- Java's **JVM runs garbage collection in the background**.

Example of Memory Leak (Before GC)

```java
class MemoryLeakExample {
    static List<int[]> list = new ArrayList<>();

    public static void main(String[] args) {
        while (true) {
            list.add(new int[1000000]); // Keeps adding large arrays
        }
    }
}
```

Problem: This keeps adding data to heap memory, causing an **OutOfMemoryError**.

Solution: Rely on Garbage Collection

```java
public class GarbageCollectorExample {
    public static void main(String[] args) {
        String str = new String("Hello World");
        str = null; // Eligible for Garbage Collection
        System.gc(); // Suggests GC to run (not guaranteed)
    }
}
```

Java's **GC will remove the unused Hello World object** to free memory.

Key Differences Between Manual & Automatic Memory Management

Feature	**Manual (C, C++)**	**Automatic (Java, Python, JS)**
Developer Control	High – must free memory	Low – GC does it automatically
Risk of Memory Leaks	High (if memory is not freed)	Low (GC prevents leaks)
Performance	Faster but risky	Slower but safer
Ease of Use	Harder (developer must manage memory)	Easier (GC handles cleanup)

Q1: What is memory management, and why is it important?

Answer: "Memory management ensures that a program **uses memory efficiently** without wasting resources. It helps prevent issues like **memory leaks (unused memory not being freed)** and **stack overflow (running out of stack memory).** Java and Python handle memory **automatically using garbage collection**, but in languages like C/C++, developers must manually allocate and free memory."

Q2: What is the difference between Stack and Heap memory?

Answer: "Stack is **fast but small** and stores **local variables and function calls**. Heap is **large but slower** and stores **objects and dynamically allocated memory**. Stack memory is automatically managed, while heap memory requires **manual or garbage collection management**."

Q3: What is a memory leak?

Answer: "A memory leak happens when a program **forgets to free memory** it no longer needs. Over time, this **wastes RAM**, making the program **slow or crash**. Java prevents memory leaks using **Garbage Collection**, but in C/C++, developers must free memory manually using free() or delete."

✓ **Stack Memory (Fast, Small)** – Stores **local variables & function calls**.
✓ **Heap Memory (Slow, Large)** – Stores **objects & dynamically allocated data**.
✓ **Garbage Collection (GC)** – Java, Python, and JS **automatically clean up unused memory**.
✓ **Manual Memory Management (C, C++)** – Developers must use malloc() & free().
✓ **Memory Leak** – When memory is **not freed, leading to crashes**.
