**1. Low-Level vs. High-Level Programming Languages**

* **Low-Level Languages:**
  + **Definition:** Low-level programming languages are closer to machine code and provide less abstraction from the hardware. These languages allow direct control over hardware resources.
  + **Characteristics:**
    - More hardware-dependent.
    - Difficult to read and write for humans.
  + **Examples:**
    - **Assembly language**: A symbolic representation of machine code, specific to a processor architecture.
    - **Machine language**: The binary code understood directly by a computer's central processing unit (CPU).
* **High-Level Languages:**
  + **Definition:** High-level programming languages are designed to be easy to read and write, abstracting away hardware details.
  + **Characteristics:**
    - More user-friendly and easier for humans to understand.
    - Not tied to any specific hardware.
  + **Examples:**
    - **Python**
    - **Java**
    - **C++**
    - **JavaScript**

**2. Interpreted vs. Compiled Programming Languages**

* **Interpreted Languages:**
  + **Definition:** Interpreted languages are executed directly by an interpreter, which reads and executes the code line-by-line.
  + **Characteristics:**
    - Slower execution speed compared to compiled languages.
    - Easier to debug and test since you can execute code directly without waiting for a compilation process.
  + **Examples:**
    - **Python**
    - **JavaScript**
    - **Ruby**
    - **PHP**
* **Compiled Languages:**
  + **Definition:** Compiled languages are first translated into machine code by a compiler before being executed by the computer.
  + **Characteristics:**
    - Faster execution speed.
    - The compilation step happens before running the program.
    - Once compiled, the code can run on any machine that supports the compiled binary.
  + **Examples:**
    - **C**
    - **C++**
    - **Go**
    - **Rust**

**3. Programming vs. Scripting Languages**

* **Programming Languages:**
  + **Definition:** Programming languages are used to create software applications with a wide range of functionalities, typically compiled or interpreted, and often more complex.
  + **Characteristics:**
    - General-purpose.
    - Can be used for creating operating systems, databases, applications, etc.
  + **Examples:**
    - **C**
    - **C++**
    - **Java**
* **Scripting Languages:**
  + **Definition:** Scripting languages are typically interpreted and used for automating tasks, controlling software applications, or for web development.
  + **Characteristics:**
    - Often used for tasks like web development, automation, or controlling other software.
    - Generally easier to use for simple tasks or "scripts" that perform repetitive actions.
  + **Examples:**
    - **Python**
    - **JavaScript**
    - **Bash (Shell scripting)**
    - **Perl**

**4. Open Source vs. Not Open Source Programming Languages**

* **Open Source Languages:**
  + **Definition:** Open-source programming languages are developed and released under licenses that allow anyone to view, modify, and distribute the source code.
  + **Characteristics:**
    - Community-driven development.
    - Free to use and modify.
    - Often have large communities for support and contribution.
  + **Examples:**
    - **Python**
    - **Ruby**
    - **JavaScript**
    - **Go**
* **Not Open Source Languages:**
  + **Definition:** Not open source languages are proprietary, and their source code is not publicly available. Users must pay for licenses to use them.
  + **Characteristics:**
    - Typically come with commercial support and warranties.
    - Restricted modification rights.
    - Licensing fees might be required for use in commercial applications.
  + **Examples:**
    - **MATLAB**
    - **Objective-C (Apple's usage)**
    - **Visual Basic (older versions)**

**5. Support for Object-Oriented Programming (OOP) vs. Not Supporting OOP**

* **Languages Supporting OOP:**
  + **Definition:** These languages allow the creation and use of objects, classes, inheritance, polymorphism, and other OOP principles.
  + **Characteristics:**
    - Supports encapsulation, inheritance, and polymorphism.
    - Code is organized around objects and classes.
    - Easier to manage and scale large programs.
  + **Examples:**
    - **Java**
    - **C++**
    - **Python**
    - **Ruby**
    - **C#**
* **Languages Not Supporting OOP:**
  + **Definition:** These languages do not focus on or provide features for object-oriented programming.
  + **Characteristics:**
    - Lack built-in support for creating and managing objects.
    - Some can still be extended to have OOP features but are not inherently object-oriented.
  + **Examples:**
    - **C** (While C can simulate OOP to some extent, it’s not inherently object-oriented.)
    - **Assembly Language**