Core Terms

Language- Means of communication between two people.
> Example: Hindi, English, and so on.

Programming language- Means of communication between human and computer.
> Example: Python, Java, c++, and so on.

Program- Set of instructions which give to computer to perform our specific tasks.
Set of instructions for the computer to follow and acheive certain goal and task.
> Example: print("This is python programming")

Here we are asking to computer to print this message on terminal.

```
print("This is python programming")
```

```
PROBLEMS 13  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Astha\Documents\Git\python_repo-1> python .\pp01.py
This is python programming
```

Written in: A Programming Language
Purpose: To solve problems or automate tasks

Programming- Process of developing program.

Programmer/Developer – Person who develops/creates program.

File Extension- The suffix at the end of a filename that indicates the file type.
Examples:

- .py → Python file
- .txt -> Text file
- .java → Java source file
- .cpp → C++ source file
- .html → HTML file

History of Evolution of Programming language

1. Machine Language (1940s)

- **Description:** Binary (0s and 1s) instructions directly executed by the computer.
- **Example:** 10110000 01100001
- **Limitation:** Extremely difficult for humans to write and debug.

2. Assembly Language (1950s)

- **Description:** Low-level language using symbolic code (mnemonics) for machine instructions.
- **Example:** MOV A, B
- **Tool:** Assembler (converts assembly to machine code)
- **Advantage:** Slightly easier for humans to understand.

3. High-Level Programming Languages (1950s–60s)

- **Goal:** Make programming easier and more like natural language/math.

Examples:

- **FORTRAN (1957):** For scientific and mathematical computing.
- **LISP (1958):** For symbolic computation and AI.
- **COBOL (1959):** For business data processing.
- **ALGOL (1958):** Introduced structured programming.

4. Structured Programming Languages (1970s)

- **Focus:** Code modularity, logic flow control (if, while, for).

Examples:

- **C (1972):** Portable and powerful system-level language.
- **Pascal (1970):** Focused on teaching structured programming.

◇ 5. Object-Oriented Programming (1980s–90s)

- **Key Concepts:** Classes, objects, inheritance, encapsulation.

Examples:

- **C++ (1983):** Extension of C with OOP features.
- **Java (1995):** Platform-independent, used widely for web and enterprise.
- **Python (1991):** Easy-to-read, versatile high-level language.

IDE (Integrated Development Environment) = One platform to write, run, debug, and manage your code efficiently.

Python programming language(Developed by Guido VAN Rossum in 1991)

- Characteristics:

1. Easy to Read, Write, and Learn

Python uses simple, English-like syntax.

2. High-Level Language

Python handles complex memory management and allows easy development of complex applications.

3. Interpreted Language

Python code is executed line by line, which helps with easier debugging.

4. Object-Oriented Programming (OOP)

Python supports classes and objects to encapsulate data and functions.

5. Dynamically Typed

You don't need to declare data types explicitly.

```
x = 10      # x is an integer
x = "Hello"  # now x is a string
```

6. Extensive Standard Library

Python comes with many modules and packages for different tasks.

7. Automatic Memory Management

Python handles memory allocation and garbage collection automatically.

8. Large Community and Ecosystem

A vast number of third-party libraries like NumPy, Pandas, TensorFlow, Django, etc.

9. Supports Functional Programming

Python supports features like lambda, map, filter, and reduce.

10. Cross-Platform

Python code runs on Windows, macOS, Linux, etc.

USE Cases:-

Python has a wide range of real-world **use cases** and **applications** across industries due to its simplicity, versatility, and rich ecosystem. Below are some of the most common and impactful **use cases of Python**, along with examples:

1. Web Development

Use Case: Building dynamic websites and web applications.
Tools/Libraries: Django, Flask, FastAPI

2. Data Science and Analytics

Use Case: Analyzing large datasets, statistical modeling, visualizing data.
Tools/Libraries: Pandas, NumPy, Matplotlib, Seaborn

3. Machine Learning & AI

Use Case: Predictive modeling, natural language processing, computer vision.
Tools/Libraries: TensorFlow, Scikit-learn, PyTorch, Keras

4. Automation & Scripting

Use Case: Automating repetitive tasks, file handling, reporting, web scraping.
Tools/Libraries: Selenium, BeautifulSoup, PyAutoGUI, os, shutil

5. DevOps and Cloud Automation

Use Case: Infrastructure as Code (IaC), deployment automation, monitoring.
Tools/Libraries: Boto3 (AWS), Google Cloud SDK, Ansible, Terraform plugins

6. Game Development

Use Case: Creating 2D and 3D games, game logic scripting.
Tools/Libraries: Pygame, Panda3D

7. Desktop GUI Applications

Use Case: Creating user-friendly desktop apps.
Tools/Libraries: Tkinter, PyQt, Kivy

8. Cybersecurity and Ethical Hacking

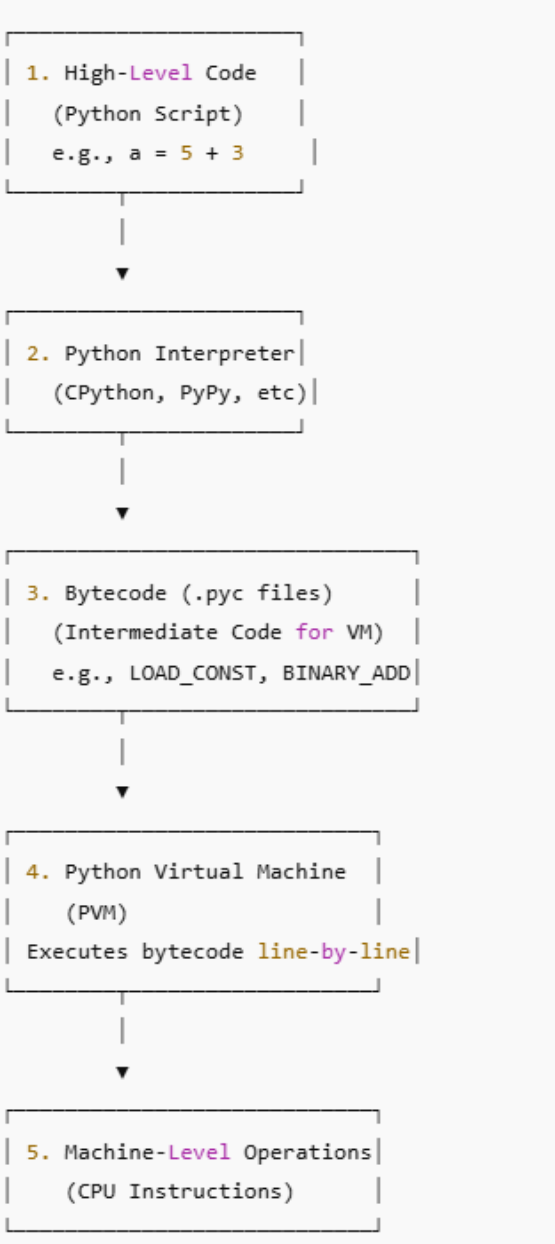
Use Case: Writing penetration testing scripts, analyzing network traffic.
Tools/Libraries: Scapy, Nmap, Paramiko

9. Scientific and Numeric Computing

Use Case: Research, simulations, modeling.
Tools/Libraries: SciPy, SymPy, Astropy

10. Finance and Fintech

Use Case: Algorithmic trading, financial analysis, risk modeling.
Tools/Libraries: QuantLib, Zipline, Pandas



Where Does the Code "Reside"?

◇ Before Execution:

- Your script file (e.g., script.py) **resides on disk** (SSD/HDD).
- When you run it using python script.py, the interpreter loads it from disk into **memory (RAM)**.

◇ During Execution:

- Python compiles it to **bytecode** and stores it in memory.
- Bytecode might also be written as .pyc files inside a __pycache__ folder for reuse.
- The **Python Virtual Machine (PVM)** interprets that bytecode **line-by-line**, keeping it in memory (RAM).
- CPU executes low-level instructions from RAM and uses **registers and caches** for high-speed operations.

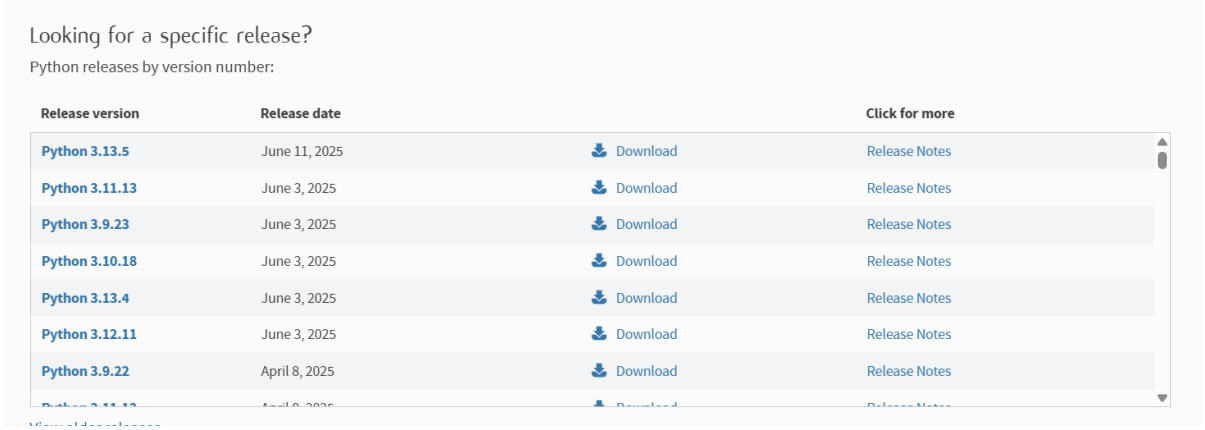
Python is **interpreted**, not directly compiled to machine code (like C/C++), so it goes through these stages:

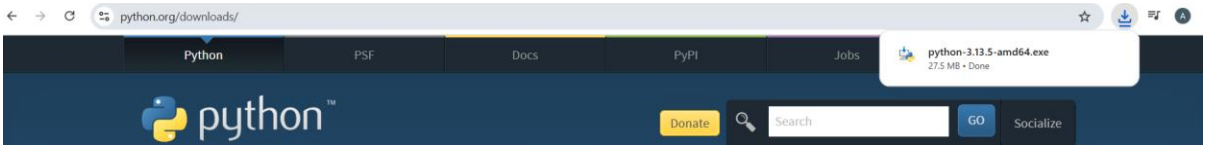
Step-by-Step Flow

Stage	Description	Where Code Resides
1. Write Code	You write .py script.	Stored on disk (e.g., main.py in filesystem).
2. Parse & Compile	Python interpreter parses and compiles code to bytecode (.pyc).	Bytecode is temporarily stored in memory and optionally saved to __pycache__/ folder.
3. Interpretation by PVM	Python Virtual Machine (PVM) reads and executes bytecode.	Bytecode is loaded into RAM.
4. Execution	Actual logic is executed via CPU instructions.	Operations happen in CPU registers and system

Steps to install python-

1. Download python version 3.13.1 from www.python.org

2. 

   Click on download after selecting 1st one.
3. 

   Run exe file and install it. Before clicking on Install tick on Add python.exe to PATH. Then check on cmd using python or python –-version to confirm it is installed or not.
4. Download and install vs code from google.