# Python Inner Working Explained

**How Python works internally**, step by step — covering the **compiler**, **interpreter**, **bytecode**, **CPython**, **PVM**, and more. I'll keep it structured and easy to follow.

## □ Python Internal Architecture - Detailed Breakdown

---

### □ 1. Source Code (.py File)

You begin by writing your Python code like this:

```
print("Hello, World!")
```

This code is just plain text — nothing has been executed yet.

---

### ⚙ 2. Compilation Phase (to Bytecode)

When you run a Python program, it **first compiles** your .py file into **bytecode**.

#### ◆ What is Bytecode?

- A **low-level, platform-independent** set of instructions.
- It's **not machine code**, but more optimized than raw source code.
- Python does this to speed up execution for future runs.

#### ◆ Where is Bytecode Stored?

It is stored as .pyc files in the __pycache__ folder.

For example:

hello.py → Compiled → __pycache__/hello.cpython-313.pyc

---

### 3. CPython – The Default Implementation

- **CPython** is the **official and most widely used Python interpreter**, written in **C**.
- When you install Python from python.org, you're using **CPython**.

**CPython does two things:**

1. **Compiles** Python code to bytecode.
2. **Runs** that bytecode using the **PVM (Python Virtual Machine)**.

---

### □ 4. PVM – Python Virtual Machine

- The **PVM** is the heart of the Python interpreter.
- It **executes bytecode** line-by-line, translating it into actual machine instructions.
- It handles:
  - Memory management
  - Variable references
  - Control flow (loops, conditions)
  - Exception handling
  - Function calling and more

### 🔥 Example:

Let’s say this line is compiled:

```
print("Hello")
```

The bytecode may look something like:

```
LOAD_NAME    print
LOAD_CONST   "Hello"
CALL_FUNCTION
```

The **PVM** interprets and runs this sequence.

---

### 🔁 Summary of the Whole Process:

Step 1: You write code → hello.py
Step 2: Python compiles it to → Bytecode (.pyc file)
Step 3: Bytecode is fed into → Python Virtual Machine (PVM)
Step 4: PVM executes it → You see output on screen

---

### □ Optional: Other Python Implementations

| Implementation | Description |
|---|---|
| **CPython** | Default implementation, written in C. |
| **PyPy** | Fast Python with Just-In-Time (JIT) Compiler. |
| **Jython** | Runs on Java Virtual Machine (JVM). |
| **IronPython** | Runs on .NET/Mono platform. |
| **MicroPython** | Lightweight version for microcontrollers. |

---

### □ 5. Garbage Collection

Python has **automatic memory management** using a technique called **Reference Counting** and **Garbage Collection (GC)**.

- Objects are removed from memory when no variable refers to them anymore.
- The GC also handles *cyclic references*.

---

### ✅ Advantages of This System

- Easy to run Python code on any machine.
- Quick execution using bytecode + interpreter.
- Code reusability via .pyc files.
- Clean memory handling using GC.

---

**Visual Flow**

Your Code (.py)
↓ [Compiler]
Bytecode (.pyc)
↓ [Interpreter → PVM]
Executed Instructions
↓
Output + Memory Managed