🐍 Python – Inner Working of Python 🐍

🕵️ Problem & Curiosity Trigger

When we wrote a basic script like:

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

👀 We noticed some **strange files & folders** popping up:

- `__pycache__/`
- `.pyc` files
- Random filenames like: `hello_chai.cpython-312.pyc`

🧠 So the real question is:

- What are these `.pyc` files?
- What is `__pycache__`?
- Will they regenerate on change?
- Is Python compiled or interpreted? 🤔

🔧 Python's Inner Working – Behind the Scenes

Here's what **actually happens** when we run Python code:

☑️ Step 1: Compilation to Bytecode

➡️ Python source code (`.py`) ⬇️ Compiled to **Bytecode** (`.pyc`) ⬇️ Executed by **Python Virtual Machine (PVM)**

⚠️ **Important**: Even though the word "compile" is used, **Python is still an interpreted language**. Compilation here refers to **conversion to intermediate bytecode**, not machine code.

💡 What is Bytecode?

📜 Bytecode is:

- An **intermediate, low-level** platform-independent code
- NOT machine instructions
- NOT assembly
- ☑️ Optimized for the **PVM (Python Virtual Machine)**

⚡ **Runs faster** than source because:

- Parsing & syntax checks are already done

- Bytecode is lightweight and quick to interpret

🗔 Understanding `__pycache__/` & `.pyc` Files

🗀 Python auto-creates:

- A `__pycache__/` folder
- Inside it: `.pyc` files named like `module_name.cpython-38.pyc`

📌 Why?

- To **store compiled bytecode**
- Avoid re-compilation on every run
- Optimize execution time ⚡

🌀 These files get **reconstructed or updated** when:

- Your source code changes
- Python detects diffs via internal diffing algorithms (like Git diff!)

🤖 PVM – Python Virtual Machine

💻 What is the PVM?

- A tiny software engine that:
  - Continuously runs in a **loop**
  - Feeds on **bytecode**
  - Executes it line by line

🛠 You can also call it:

- **Python Runtime Engine**
- **Python Interpreter**

💬 Think of it like:

> "No engine, no car." 🚗 So, **No PVM, no Python execution!** 😷

🚀 Why `.pyc` Files Matter?

- They're generated **only when a module is imported**
- They don't appear with just a single script
- So in small "Hello World" files — you often won't notice them

📦 They help in:

- Optimization
- Faster imports
- Managing multiple modules efficiently

🛠️ Different Python Implementations

By default, you're using **CPython** — the standard Python interpreter.

But Python has many other versions:

🧪 Version	🔍 Description
CPython	Default & most widely used
Jython	Python on Java Virtual Machine
IronPython	Python for .NET Framework
Stackless Python	Good for concurrency-heavy applications
PyPy	Faster, performance-optimized version of Python

🧠 Interview Tips – Bytecode vs Machine Code

📚 Remember:

- **Bytecode ≠ Machine Code**
- Bytecode is NOT executable by hardware directly
- Needs a **runtime/interpreter (PVM)** to execute

This is often misunderstood. Bytecode is platform-independent; machine code is platform-specific!

📝 Bonus Analogy

🎮 **Think of Python Execution Like This**:

1. You write your code (🎨)
2. Python compiles it to a special game disk (💿 Bytecode)
3. PVM is like the console (🎮) — it reads the disk and runs the game
4. Machine doesn't know the game rules — **only the console does!**

📚 Assignment (Optional but Fun!)

📝 Write a short blog/article on:

> "How Python Executes Code Behind the Scenes"

🔗 Post it on hashnode.com or your personal blog 🎥 Embed this video 📣 Share it with your dev friends!

💖 Final Words

💬 If this video helped you:

- ✅ Comment & Share with friends
- ✅ Ask creators for more inner-working videos
- ☕ Keep sipping chai & coding
- 💖 Stay curious, stay motivated!

🔁 TL;DR – Quick Summary

🔍 **Concept**	💡 **Explanation**
`.py` file	Your original Python script
`.pyc` file	Compiled bytecode version
`__pycache__`	Folder that stores `.pyc` files
Bytecode	Optimized, intermediate low-level code
PVM	Executes bytecode (Python's runtime engine)
CPython	Default Python interpreter
Bytecode ≠ Machine code	Needs PVM to execute
Multi-Implementations	CPython, Jython, PyPy, etc.
