JavaScript vs Python - Compilation & Execution

JavaScript

Compilation Phase (Before Execution)

1. Parsing: JS engine parses the code and converts it into an Abstract Syntax Tree (AST).
2. Compilation (Just-In-Time):
   - Compiled to bytecode and optimized to machine code by engines like V8.
   - Uses JIT (Just-In-Time) compilation - combines interpretation + compilation.
3. Hoisting: Declarations are hoisted. Global Execution Context is created.
4. Memory Allocation: Variables/functions are stored in memory.

Execution Phase

1. Runs Line-by-Line (single thread).
2. Creates Execution Contexts for functions.
3. Event Loop for async tasks:
   - setTimeout, fetch, Promises handled by Web APIs.
   - Callbacks go to Callback Queue, then executed.

Event Loop:

  Call Stack Web APIs Callback Queue

Python

Compilation Phase

1. Source code (.py) is parsed to an AST.

2. Compiled into Bytecode (.pyc in __pycache__).

Execution Phase

1. PVM (Python Virtual Machine) reads bytecode and executes.

2. Executes line-by-line (blocking by default).

3. Async via asyncio, threading, multiprocessing.

Python Async Flow with asyncio:

async def Event Loop Tasks Coroutine results

Key Differences

Aspect	JavaScript	Python
Compilation	JIT compilation	Bytecode compilation (.pyc)
Execution	Non-blocking, single-threaded	Blocking, single-threaded
Async Default	Yes	No
Virtual Machine	V8	PVM
Needs Async Setup	Optional	Mandatory (asyncio)

Real-World Analogy

- JS: Restaurant with waiters (non-blocking).

- Python: Single chef making one dish at a time.

- Python + asyncio: Restaurant using waiters and prep-chefs.