PYTHON

Introduction

Python is a high-level, general-purpose, interpreted programming language created by Guido van Rossum and first released in 1991. “High-level” means it hides most of the complex detail of the computer's hardware — memory addresses, processor registers — so a programmer can focus on solving the problem, not managing the machine.

It is dynamically typed (you never declare a variable's type) and automatically garbage-collected (you never manually free memory), which removes two of the biggest sources of bugs in older languages like C and C++.

Python also supports multiple programming styles — procedural, object-oriented, and functional — in the same codebase, so learners can start with simple top-to-bottom scripts and grow into classes and objects at their own pace.

- Guided by “The Zen of Python” — 19 design principles favoring simplicity
- Dynamically typed with automatic memory management (garbage collection)
- Multi-paradigm: procedural, object-oriented, and functional in one language

Why Learn Python — and Where It's Used

Python's gentle learning curve makes it the most common first language taught in university computer-science courses, while its depth and huge ecosystem of libraries mean the same skills scale up to professional, production-grade software. That combination is why it consistently ranks as the most in-demand language on job boards and the most popular language overall on the TIOBE and PYPL indices.

- Web Development — Django & Flask power backends for Instagram, Pinterest, and Spotify.
- Data Science & AI — Pandas, NumPy, TensorFlow, and PyTorch drive most modern machine learning.
- Automation & Scripting — Replacing repetitive manual tasks with short, reliable scripts.
- Scientific Computing — Used in physics, biology, and astronomy research (e.g., NASA).

History

Origins: 1989–1991

In December 1989, Dutch programmer **Guido van Rossum** was looking for a hobby project to keep him occupied over the Christmas holidays at CWI (Centrum Wiskunde & Informatica) in the Netherlands. He decided to design a successor to the ABC language — one that kept ABC's readability but fixed its technical shortcomings, such as poor extensibility and weak support for larger programs.

Python 0.9.0 was released publicly in February 1991. Even in that first version, it already had functions, exception handling, core data types (list, dict, str), and a module system for reusing code — the essential building blocks that are still central to Python today.

Where the name comes from: Despite the snake logo, Python is not named after the reptile. Guido van Rossum was reading scripts from Monty Python's Flying Circus, the British comedy series, while writing the language — and wanted a name that was short, unique, and slightly mysterious.

Growth to the Present Day

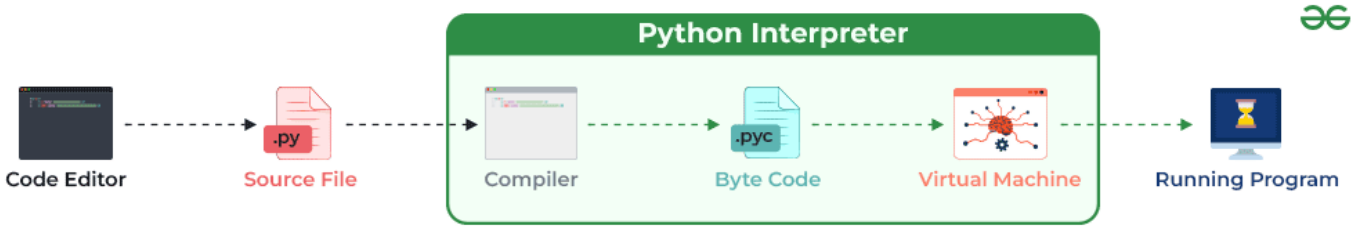
2000 — Python 2.0: Added a cycle-detecting garbage collector, list comprehensions, and Unicode support — and introduced the community-driven PEP (Python Enhancement Proposal) process still used to evolve the language today.

2008 — Python 3.0: A deliberate, non-backward-compatible redesign that fixed early design flaws — print became a function, integer division was corrected, and text was Unicode by default.

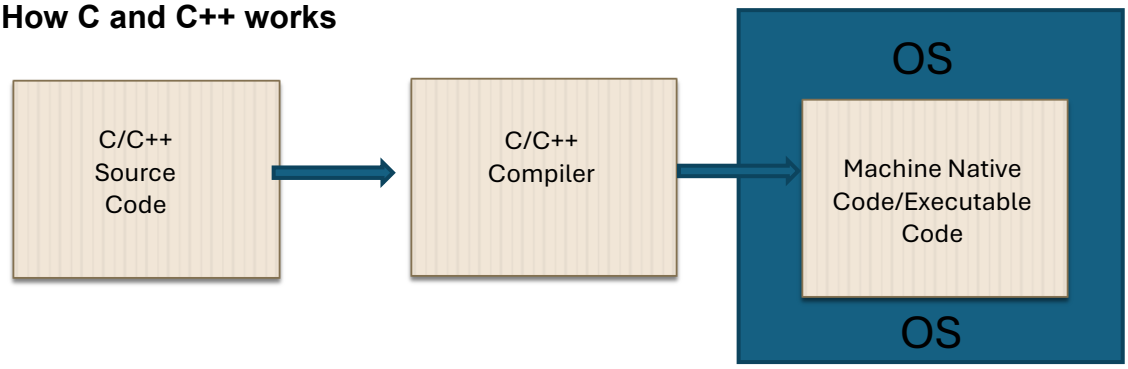
2020 — Python 2 Retires: Python 2 reached its official end-of-life on January 1, 2020, completing the community's decade-long migration to Python 3.

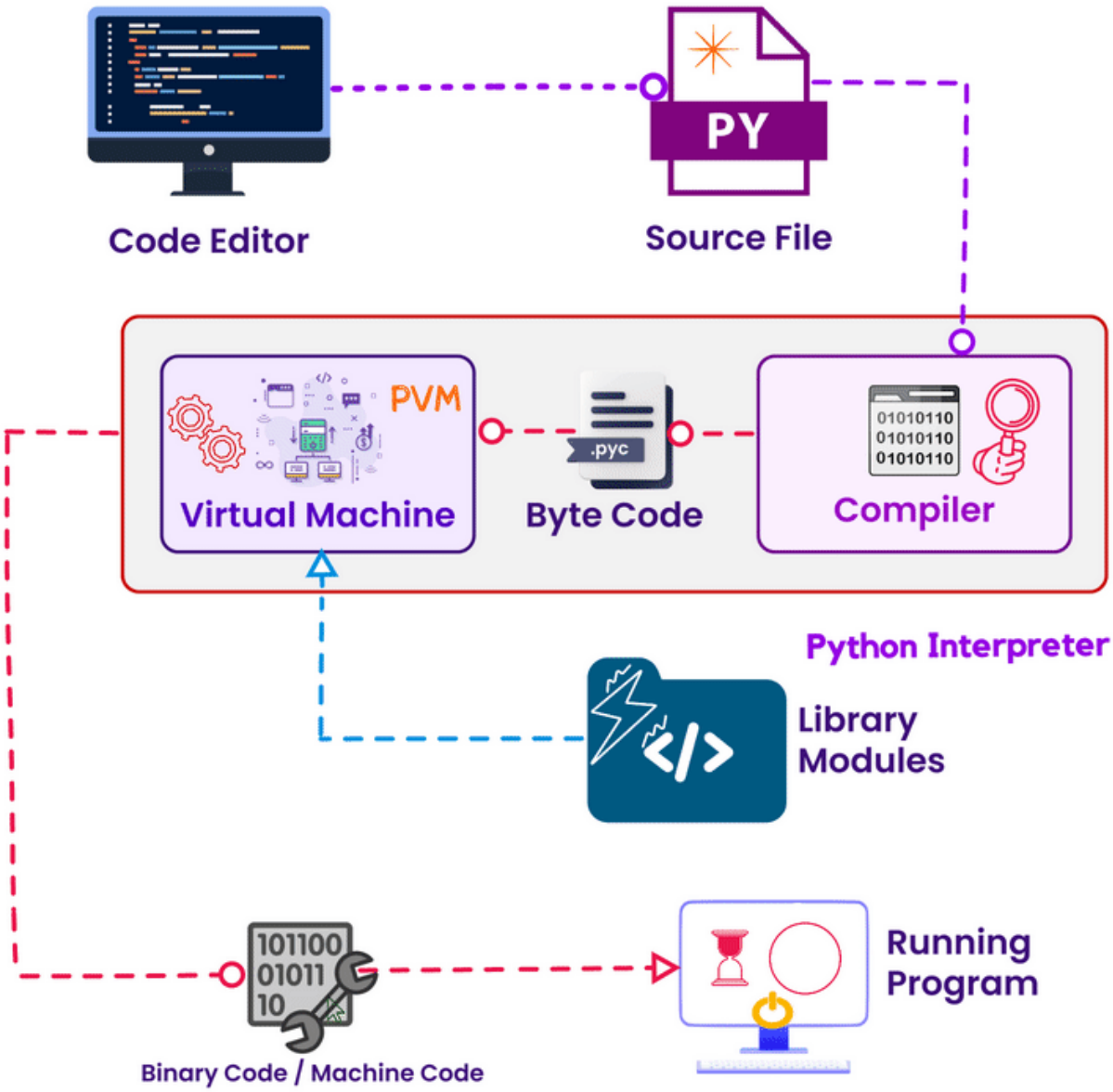
Today — Global Standard: Python tops the TIOBE and PYPL indices and powers Google, Netflix, Instagram, and NASA, and it is the primary language behind the modern AI boom via PyTorch and TensorFlow.

How python works



How C and C++ works





Process of Python

 Source Code: You write standard Python code in a .py file.

 Compilation: The standard CPython interpreter parses and converts your code into numeric bytecodes.

 Caching: Python saves this compiled bytecode inside **.pyc files** within a hidden **__pycache__** folder. This allows modules to load faster on subsequent runs by skipping the compilation step.

 Execution: The PVM reads the bytecode instructions one by one and translates them into CPU-specific machine instructions.

Bytecode :

Python bytecode is a low-level, platform-independent intermediate representation of your source code. When you run a Python script, the software does not execute your code directly. Instead, the internal **Python compiler translates** your .py source file into **bytecode instructions**, which are then processed by the **Python Virtual Machine (PVM)**

File extension of python bytecoe is .pyc. Bytecode store in a __pycache__ folder. A .pyc file contains **Python bytecode**, which is **not specific to Windows, Linux, or macOS**. It means pyc file is platform independent.

- Bytecode are store in .pyc file in __pycache__ folder
- Bytecode is machine independent code which can be run on any machine.
- .pyc file can be run independently without having python code.
- .pyc files can often be decompiled back into readable Python.
- Pyc file can be distributed independently .

Bytecode file format

__pycache__/hello.cpython-311.pyc

Difference between machine code and byhtecode (.pyc)

Machine Code	.pyc
Executed directly by CPU	Executed by Python Virtual Machine
CPU specific	Python version specific
OS dependent	Largely OS independent
Very difficult to reverse	Easier to decompile

Python Virtual Machine (PVM)

Python Virtual Machine (PVM) is the runtime engine that interprets and executes Python bytecode, acting as an abstraction layer between your code and the underlying computer hardware. It is a core component of the Python interpreter (most commonly CPython).

The PVM loads this bytecode, processes it through a continuous interpretation loop (ceval.c in CPython), and converts it into low-level machine instructions that your specific CPU can execute.

Cross-Platform Portability: It enables Python's "Write Once, Run Anywhere" capability. As long as a system has a compatible PVM installed, it can read and execute the exact same .pyc bytecode files.

Security & Abstraction: Because Python code runs inside a controlled virtual sandbox environment, it protects the host operating system from rogue, direct memory manipulations.

Python Interpreter

The official, standard, and most widely used Python interpreter is CPython. When you download Python from the official Python website and run your code using the python you are using CPython. It is written in C and serves as the reference implementation for the language.

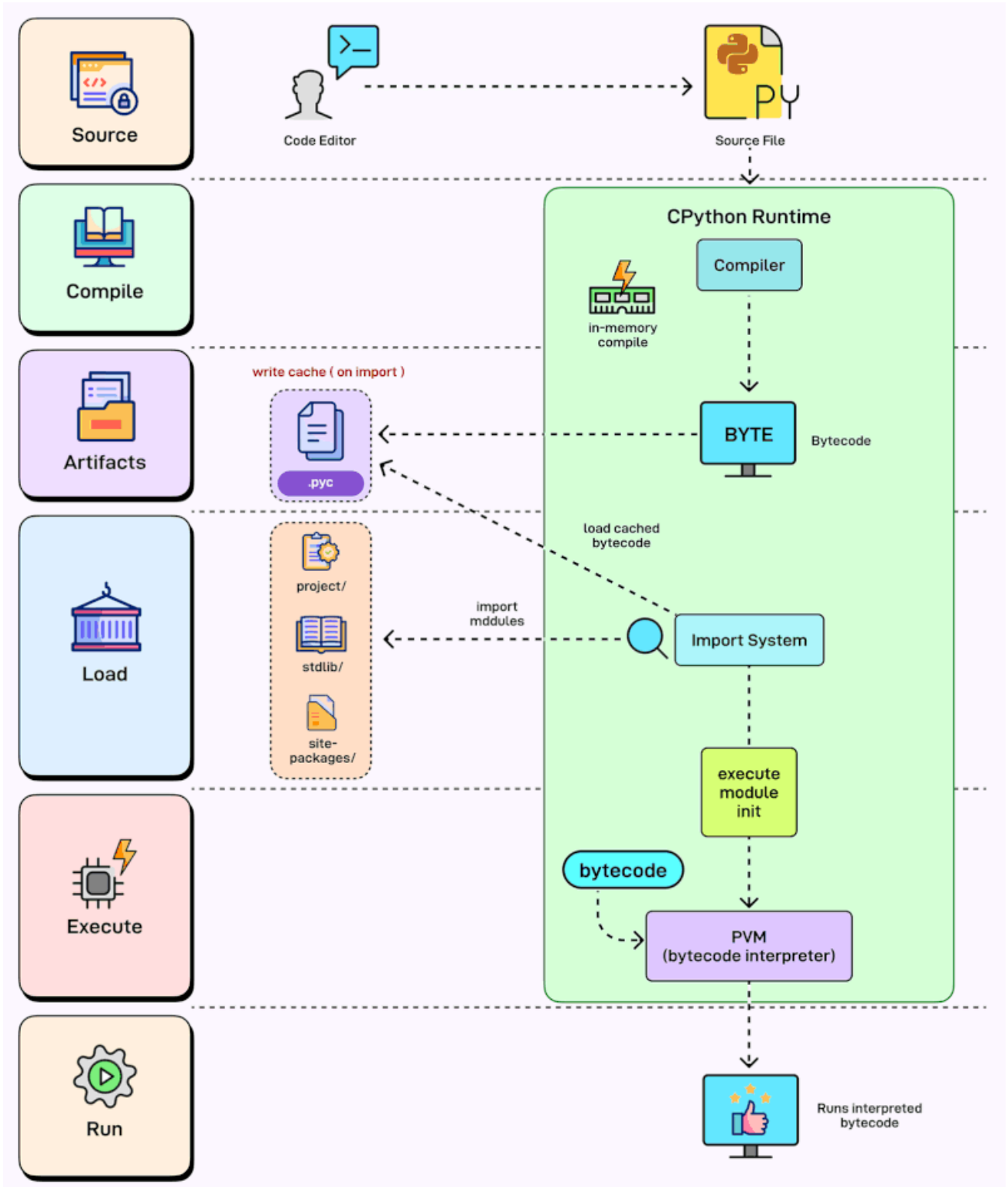
Other Python Interpreter

PyPy: A fast alternative interpreter that uses a Just-In-Time (JIT) compiler to make Python code run significantly faster.

Jython: An interpreter designed to run on the Java platform, compiling Python code straight into Java bytecode.

IronPython: An implementation designed for the Microsoft .NET framework.

MicroPython: A lightweight, stripped-down interpreter optimized to run on tiny microcontrollers and embedded systems.



Artifact is a compiled package file containing your Python code, metadata, and dependencies. These are built so your code can be easily installed by others.

Features

Key Features

 Dynamically Typed: You do not need to declare variable types (like int or string). Python determines the data type at runtime while executing the script.

 Automatic Memory Management: Python uses built-in tools like garbage collection to allocate memory for variables and clean it up automatically once they are no longer needed.

 Indentation-Based Structure: Instead of using curly brackets {} or semicolons ; to structure blocks, Python strictly relies on whitespace indentation to define functions, loops, and conditions.

1. Easy to Learn & Read

Python was deliberately designed so that code reads almost like plain English and ordinary mathematical notation. Instead of curly braces and semicolons, Python uses indentation itself to mark where a block of code begins and ends — so the visual shape of the code always matches its logical structure.

This removes an entire category of beginner mistakes (mismatched braces, stray semicolons) and lets new programmers focus on learning logic and problem-solving rather than fighting the language's punctuation.

```
#Check if a number is even — no braces, no semicolons
If number % 2 == 0:
  print("Even number")
```

- No semicolons or curly braces — indentation defines every block
- Keywords read like English: and, or, not, in, is
- Consistently the most-recommended first language for beginners

2. Interpreted Language

Python code is executed line by line by an interpreter (CPython, by default) rather than being compiled ahead of time into a standalone machine-code program, the way C or C++ are. That means you can run a script the instant you save it, and test small snippets interactively in a REPL or a Jupyter notebook.

The trade-off is raw execution speed: an interpreted language is typically slower than a compiled one. In practice this rarely matters for students, because performance-critical libraries like NumPy are written in C underneath and called from Python.

- Immediate feedback — no separate build or compile step
- Interactive REPL and notebooks make experimenting fast
- Slower raw speed than compiled languages, offset by C-optimized libraries

3. Cross-Platform Compatibility

Because Python programs are interpreted rather than compiled into a platform-specific binary, the exact same script runs unmodified on Windows, macOS, and Linux — and even on embedded devices like the Raspberry Pi — as long as a compatible interpreter is installed.

For students, this means a program written on a laptop at home behaves identically on a lab computer or a classmate's machine. For professional teams, it means one codebase can be developed on any operating system and still deploy cleanly to Linux-based cloud servers.

- One codebase runs on Windows, macOS, and Linux without changes
- No recompilation needed when moving between machines
- Simplifies collaboration across different student and team setups

4. Extensive Libraries & Ecosystem

Python ships with a large standard library covering file handling, regular expressions, networking, and dozens of other everyday tasks — often summarized as “batteries included,” because many programs need zero extra installs to get started.

Beyond the standard library, the Python Package Index (PyPI) hosts more than 500,000 third-party packages. A single command — pip install — brings in tools like NumPy and Pandas for data analysis, Django and Flask for web apps, or TensorFlow and PyTorch for machine learning.

```
pip install pandas       # data analysis

pip install Django       # web development

pip install torch       # machine learning
```

- Standard library covers most everyday programming tasks
- 500,000+ third-party packages available on PyPI
- One-command installs with pip make adding tools effortless

5. Versatility Across Domains

Many languages are optimized for a single niche — JavaScript for the browser, R for statistics.

Python instead performs well as a general-purpose tool across an unusually wide span of domains, from REST APIs and websites to scientific research, office automation, game scripting, and deep learning.

For a student, this versatility is a practical advantage: the core language skills learned in one course — variables, functions, loops, classes — transfer directly into whichever domain a future project or career happens to need.

- Web development, data science, and AI/ML all use the same core syntax
- Also used for automation scripts, game logic, and cybersecurity tooling
- Skills transfer directly across academic projects and future careers

6. Free & Open Source

Python is released under an OSI-approved license through the Python Software Foundation, meaning anyone can download, use, modify, and redistribute it — including in commercial products — at no cost and with no license fees.

Development happens transparently on GitHub through community-authored PEPs (Python Enhancement Proposals). Students can read the interpreter's own source code, follow how new features are debated, and even contribute a fix or improvement themselves.

- Free for personal, academic, and commercial use alike
- Governed by an open, public PEP proposal process
- Backed by a large global community and events like PyCon