Introduction to Python

Md. Aminul Islam Shazid

Outline

1. Introduction and History
2. Language Features
3. Programming Paradigms
4. Why Learn Python?
5. Ecosystem and Applications
6. Present and Future

Introduction and History

What is Python?

- High-level, interpreted, general-purpose programming language
- Emphasizes readability and simplicity
- Python philosophy: ”Readability counts”

The Zen of Python

- import this
- Key principles:
  - Beautiful is better than ugly
  - Simple is better than complex
  - Readability counts
  - There should be one– and preferably only one –obvious way to do it

History of Python

- Created by Guido van Rossum, 1991
- Influences: ABC, Modula-3, Unix shell scripting
- Designed for scripting, prototyping, and general programming

Original Goals and Evolution

- Original goals: ease of use, readability, ”batteries included”
- Evolution:
  - From scripting to web development, data science, AI
  - Strong ecosystem and community growth

Language Features

Basic Language Features

- Interpreted, dynamically typed
- High-level, cross-platform
- Large standard library

Interpreters and Implementations

- CPython (default, reference implementation)
- PyPy (JIT compilation)
- Cython, Numba (performance optimization)
- MicroPython (embedded)
- Jython (Java), IronPython (.NET)
- Pyodide (CPython ported to WebAssembly), Brython (Python interpreter written in JavaScript)

Interpreted vs Compiled

- Interpreted: executes directly, easier debugging, slower
- Compiled: precompiled to machine code, faster, less flexible
- While Python is primarily an interpreted language, one can use both approaches via JIT or AOT compilers

High-Performance Compilers and Tools

- PyPy, Cython, Numba for speed

Interoperability with Other Languages

- C, C++, Fortran extensions
- Java (via Jython), R (via rpy2)
- Integration for high-performance or legacy code

Programming Paradigms

Object-Oriented Programming

- Classes, inheritance, polymorphism
- Encapsulation of data and behavior
- Widely used in web frameworks, GUI apps, simulations

Functional Programming

- First-class functions
- map(), filter(), reduce()
- Comprehensions and generators

Why Learn Python?

Why Learn Python in General?

- Simplicity & readability → low entry barrier
- Versatility:
  - Web development: Django, Flask
  - Backend & APIs: FastAPI, Flask
  - Desktop applications: GUI apps, utilities
  - Mobile development: Kivy, BeeWare (smaller adoption)
  - Automation & scripting: one-off scripts, data cleaning, DevOps
- Community & ecosystem → global adoption, extensive libraries

Python in Action — Famous Apps & Services

- Web: Instagram, Pinterest, Reddit, Spotify, Netflix, YouTube, Quora
- Desktop: Dropbox, BitTorrent, Calibre, Anki, Blender (Python scripting)

Why Learn Python for Data Science?

- Core libraries: NumPy, Pandas, Polars
- ML/AI frameworks: Scikit-learn, TensorFlow, PyTorch
- Visualization: Matplotlib, Seaborn, Plotly
- Big Data & cloud: PySpark, Dask
- Dominant language for AI research and production

Python vs Other Languages

- Python vs C: simplicity vs raw performance
- Python vs R: general-purpose vs domain-specific (statistics)
- Complementary use cases; Python bridges prototyping & production

Ecosystem and Applications

PyPI and the Library Ecosystem

- Over 500k packages available
- Domains: web dev, scientific computing, ML/AI, automation
- Easy to install and manage via pip

Solving the Two-Language Problem

- Rapid prototyping in Python
- Production optimization via Cython, Numba, Rust, or C++ integration
- Combines speed of compiled languages with ease of Python

Present and Future

Current State of Python

- Widespread adoption: academia, enterprise, startups
- Strong ecosystem of libraries and tools

Future Prospects and Innovation

- Performance improvements: PEP 659 specialization, sub-interpreters
- Rust integration, WebAssembly, cloud-native tools
- Expansion into embedded and edge devices
- Continued growth in AI, data science, and scientific computing