Python Proficiency Roadmap

A step-by-step roadmap to becoming a highly proficient and capable Python programmer.

1■■ Core Python Fundamentals

- Syntax & indentation rules
- Variables, constants, naming conventions (PEP 8)
- Data types: int, float, bool, str
- Input/output, print formatting (f-strings, .format)
- Operators: arithmetic, comparison, logical, assignment, bitwise
- Type conversion (int(), float(), str())
- Comments & docstrings

2■■ Data Structures & Built-ins

- Strings: indexing, slicing, methods
- Lists, tuples, sets, dictionaries
- Comprehensions (list/set/dict)
- Unpacking (*args, **kwargs)
- Iteration: for, while, enumerate, zip
- Membership testing: in, not in
- Built-ins: len, max, min, sum, sorted, any, all
- collections module: Counter, defaultdict, deque, namedtuple

3■■ Control Flow & Error Handling

- if / elif / else
- match (pattern matching, Python 3.10+)
- Loops: break, continue, else
- Exceptions: try/except/finally/else
- Custom exceptions

4■■ Functions & Functional Tools

- Define functions, default args
- Variable-length args (*args, **kwargs)
- Return values vs side effects
- First-class functions & closures
- Lambdas, map, filter, reduce

- Decorators (@staticmethod, @property, custom)
- Recursion

5■■ Object-Oriented Programming

- Classes, instances, attributes, methods
- __init__, __repr__, __str__
- Inheritance, composition, overriding
- Class vs static methods
- Encapsulation & private attributes
- Dunder methods (__len__, __iter__, __getitem__, __call__)
- Dataclasses (@dataclass)

6■■ Modules, Packages & Environments

- Creating & importing modules
- Package structure (__init__.py)
- pip & virtual environments (venv, pipenv, poetry)
- Dependency management (requirements.txt, pyproject.toml)
- Standard library: os, sys, pathlib, datetime, time, json, csv, re, math, statistics, random, logging, argparse

7■■ File & Data Handling

- Reading/writing text & binary files
- Context managers (with statement)
- CSV, JSON, XML parsing
- Pickle (serialization)
- Handling large files efficiently

8■■ Advanced Concepts

- Iterators, generators, yield
- Comprehensions with conditions
- Context managers (__enter__, __exit__)
- Descriptors & properties
- Abstract Base Classes (abc)
- Type hints & static analysis (mypy, typing)
- Dataclasses & attrs
- Async/await, asyncio
- Concurrency: threads, multiprocessing, futures

9■■ Testing & Quality

- Unit tests (unittest, pytest)
- Mocking & patching
- Test-driven development
- Code style & linting (flake8, pylint, black, isort)
- Profiling & optimization (cProfile, timeit, functools.lru_cache)
- Debugging (pdb, breakpoint())

■ Popular Libraries & Ecosystem

- Data: numpy, pandas, matplotlib, scikit-learn
- Web: Flask, FastAPI, Django
- Automation: requests, BeautifulSoup, selenium
- Cloud & DevOps: boto3, paramiko, docker SDK
- ML/AI: tensorflow, pytorch, transformers

11■■ Tooling & Workflow

- Git & GitHub (branching, PRs)
- Debuggers & IDEs (VS Code, PyCharm)
- Virtual environments per project
- Packaging & publishing (setuptools, twine)
- CI/CD basics (GitHub Actions)

12■■ Software Engineering Mindset

- Clean code (PEP 8, readability)
- DRY, KISS, YAGNI principles
- Big-O time/space complexity
- Reusable, modular code
- Documentation & docstrings (PEP 257)
- Design patterns (Singleton, Factory, Observer)
- Reading & understanding open-source code

■ Proficiency Checklist

- Build CLI tools, APIs, desktop/web apps
- Solve algorithmic problems efficiently
- Read & debug open-source Python code

- Write clean, tested, documented modules
- Contribute to or create libraries