SOFTWARE DEVELOPMENT AND ENGINEERING PRACTICES

Python Programming

Level: Practitioner • 2 days (expandable to 3) • Virtual, In-person

Overview

Python has become the shared language of automation, data work, and AI, which means it is now the language people from every technical background end up needing. Its reputation for being easy is half true: the syntax is friendly, but writing Python that is genuinely good, idiomatic, well structured, and robust when things go wrong, is a different skill from making a snippet run. The gap between copy-paste Python and real Python is where this course lives.

This is a hands-on, practitioner course. It builds the language in dependency order: core syntax and Python's way of doing things first, then the data structures everything else is made of, then functions, files, error handling, and organizing real programs with modules and packages. Rather than survey every corner of a very large ecosystem, it goes deep on the core you will use in every script and program you ever write. Every module ends with a lab and builds on the one before, and the course finishes with a complete, useful program you built yourself.

Who Should Attend

- Developers who know another language and need to become productive in Python
- Testers, analysts, and operations engineers automating their work with Python
- Developers who use snippet-level Python already and want real fluency

Learners who have never programmed should take *Introduction to Programming* first.

Prerequisites

- Working proficiency in at least one programming language, or solid scripting experience
- Understanding of variables, control flow, and functions
- No prior Python experience required

What You Will Learn

- Write clean, idiomatic Python using its core types, control flow, and conventions
- Choose and use the right data structure: lists, dictionaries, tuples, and sets
- Design functions with Python's argument model and write comprehensions that stay readable
- Read and write files and handle errors with exceptions so programs fail gracefully
- Organize code into modules and packages and manage dependencies with virtual environments
- Build a complete command-line program from a problem statement to working code

Course Outline

Day one: the language and its data structures

- Python Bearings
  - Running Python: the interpreter, scripts, and a proper editor setup
  - Core types, variables, and dynamic typing without surprises

  - Control flow, truthiness, and what "Pythonic" actually means
  - Lab: set up your environment and write a script that already does something useful
- The Data Structures That Do the Work
  - Lists and tuples: sequences, slicing, and iteration
  - Dictionaries: the data structure behind most Python programs
  - Sets, and choosing the right structure for the job
  - Lab: load a realistic dataset into the right structures and answer questions about it
- Functions and Idiomatic Python
  - Defining functions: positional, keyword, and default arguments, and *args and* *kwargs
  - Scope, return values, and functions as values
  - Comprehensions: powerful, and knowing when they stop being readable
  - Lab: refactor your dataset code into clean, reusable functions with comprehensions where they help

Day two: real programs

- Files and Errors
  - Reading and writing text and CSV files with context managers
  - Working with JSON: the format your programs will meet everywhere
  - Exceptions: try, except, raise, and designing for failure
  - Lab: build a script that processes a messy real-world file and survives its defects
- Modules, Packages, and Environments
  - Splitting a program into modules, and how import really works
  - Virtual environments and installing packages with pip
  - The standard library tour that saves you from reinventing wheels: pathlib, datetime, collections
  - Lab: restructure your project into a package with pinned dependencies in a virtual environment
- Classes, and Building a Complete Program
  - Classes in Python: when a dictionary stops being enough
  - Structuring a command-line program: entry points, arguments, and output
  - Debugging and a first look at testing your code
  - Lab: plan and build a complete command-line tool that reads, processes, and reports on real data

Extended Version

The three-day version keeps the same gradient and adds depth and range:

- Deeper object-oriented Python: dunder methods, dataclasses, and inheritance done sparingly
- Testing with pytest and writing code that is easy to test
- Working with APIs: calling HTTP services with requests and processing the results
- A capstone: build a complete automation tool for a realistic scenario and present how it works