# One-Month Advanced Python Training Schedule

## Week 1: Pythonic OOP

### Day 1: The Single-Responsibility Principle

Exercise: Refactor a complex class that handles multiple responsibilities into multiple classes, each handling a single responsibility.

### Day 2: Inheritance

Exercise: Create a class hierarchy for a simulation (e.g., animals in a zoo) demonstrating inheritance and method overriding.

### Day 3: Refactoring For Inheritance

Exercise: Refactor a codebase that uses repeated code for similar classes into a clean inheritance structure.

### Day 4: Pythonic Interfaces

Exercise: Implement a custom interface for a set of classes representing different types of data storage (e.g., JSON, XML, Database).

### Day 5: Methods And Inheritance

Exercise: Develop a class hierarchy where multiple methods are overridden in subclasses to modify behaviors.

### Day 6: Access Control

Exercise: Implement a class with private attributes and methods, and provide controlled access through public methods.

### Day 7: Review and Practice

Exercise: Develop a small project (e.g., a library management system) that incorporates all concepts from the week.

## Week 2: Pythonic OOP (continued) & Test-Driven Python

Day 8: Properties

Exercise: Create a class representing a bank account with properties for balance, and interest rate, and validate changes through property setters.

Day 9: Properties And Refactoring

Exercise: Refactor an existing class to replace getter and setter methods with properties.

Day 10: Factories, Class Methods, And Static Methods

Exercise: Implement factory methods to create instances of different subclasses based on input parameters.

Day 11: Pythonic Design Patterns

Exercise: Implement the Singleton, Factory, and Strategy patterns in a practical scenario (e.g., logging system).

Day 12: Foundations of Automated Testing

Exercise: Write basic unit tests for a small module (e.g., a calculator).

Day 13: Types of Tests: Unit Tests, Integration Tests, and More

Exercise: Create unit and integration tests for a small web application (e.g., a simple API).

Day 14: Review and Practice

Exercise: Refactor and test an existing small project, ensuring all principles and testing practices are applied.

## Week 3: Test-Driven Python & Scaling Python With Generators

Day 15: Test-Driven Development

Exercise: Develop a small feature (e.g., a todo list) using TDD.

Day 16: Detailed Test Assertion Types & Strategies

Exercise: Write a variety of tests using different assertion types for a data processing function.

Day 17: Fixtures

Exercise: Create fixtures for a test suite that involves setting up and tearing down a database.

Day 18: Parameterized Tests (And Subtests)

Exercise: Implement parameterized tests for a function that processes multiple types of input data.

Day 19: Mock Objects

Exercise: Use mocks to test a service that makes external API calls.

Day 20: Patching With Mocks

Exercise: Patch a function in a test to simulate different environments (e.g., different configurations).

Day 21: Review and Practice

Exercise: Develop a fully tested small application (e.g., a URL shortener) using all testing principles covered.

## Week 4: Scaling Python With Generators, Higher Order Python, Python Code Walkthroughs

Day 22: Foundations of Generators

Exercise: Write a generator to lazily load and process large data files.

Day 23: Generator Design Patterns

Exercise: Implement a pipeline of generators to process a data stream.

Day 24: The Iteration Protocol in Python

Exercise: Create a custom iterable class that implements the iteration protocol.

Day 25: List Comprehensions and Generator Comprehensions

Exercise: Refactor a piece of code using loops to use list comprehensions and generator comprehensions.

Day 26: Passing Data Into Generators (Coroutines)

Exercise: Develop a coroutine that processes user input interactively.

Day 27: Overview of AsyncIO

Exercise: Write an async program that performs multiple I/O-bound tasks concurrently.

Day 28: Review and Practice

Exercise: Combine generators and async programming to build a small data processing pipeline that handles real-time data.

Day 29: Higher Order Python: Variable Arguments and Argument Unpacking

Exercise: Write functions that use *args and **kwargs for flexible argument passing and processing.

Day 30: Functions As Objects, Writing Simple Decorators

Exercise: Develop simple and advanced decorators, including those that add logging and caching to functions.

Day 31: Review and Comprehensive Project

Exercise: Develop a comprehensive project (e.g., a RESTful API service) that integrates OOP, testing, generators, higher-order functions, and practical engineering practices.

## Ongoing: Python Code Walkthroughs & Practical Python Engineering

Daily (Optional):

Exercise: Review real-world codebases and write summaries and refactor suggestions.

Exercise: Apply logging, error handling, and dependency management practices to an existing project.