Week 1: Setup, Basics & Object-Oriented Programming

Goal: Set up the environment and cover core syntax and object-oriented concepts.
Environment Setup:

- Install Python and configure a virtual environment (venv).
- Set up an IDE (**PyCharm**, VS Code, or Jupyter).
- Write and execute simple Python scripts.

Python Basics:

- Learn **data types** (int, float, str, list, dict, tuple, set).
- Use **operators** and perform **basic I/O** operations.
- Write **conditional statements** (if, elif, else) and **loops** (for, while).
- Understand **functions** (defining, calling, parameters, and returns).

Object-Oriented Programming (OOP):

- Learn **classes & objects**, **encapsulation**, **inheritance**, and **polymorphism**.
- Create programs using OOP concepts.
- Learn **memory management** using tracemalloc.

Hands-on:

- Write a simple class-based program and test memory management.
- **Exercise**: Build a class for a basic task (e.g., Inventory Management).

Week 2: Multithreading, Multiprocessing & Async Programming

Goal: Learn concurrency, parallelism, and asynchronous programming.
Concurrency and Parallelism:

- Learn **threading** (creating threads, managing threads, thread pools).
- Use **concurrent.futures** (ThreadPoolExecutor, ProcessPoolExecutor).
- Understand **multiprocessing** (parallel execution).

Async Programming:

- Learn **asyncio**, event loops, coroutines, and async/await.
- Practice asynchronous programming for concurrent tasks (e.g., web scraping or API calls).

  Hands-on:
- Write a program that downloads multiple web pages asynchronously.
- Develop a script that executes multiple tasks in parallel using threads and processes.

Week 3: Libraries, Data Processing, Visualization & Testing

Goal: Explore essential Python libraries and data manipulation techniques.
Package Management:

- Learn how to use **pip** and **setuptools** for package management.
- Understand the role of **virtual environments** (venv).

Core Libraries:

- **NumPy** and **Pandas**: Learn basic operations for data manipulation.
- **Matplotlib**: Understand how to create simple plots and charts.
- **Requests**: Make HTTP requests and interact with APIs.
- **Pydantic**: Use for data validation and schema definitions.

Testing:
- Introduction to **unit testing** using pytest.
- Learn basic testing principles and writing tests for the code.

Hands-on:
- Load data using **Pandas**, perform analysis, and visualize it with **Matplotlib**.
- Write unit tests for code.
- Fetch data from a public API and validate it using **Pydantic**.

Week 4: Docker, Logging & Real-World Application

Goal: Apply all concepts learned and build a small Python project with deployment and logging.

Docker:
- Learn how to package Python applications in a **Docker container**.
- Manage dependencies in a containerized environment.

Logging & Best Practices:
- Learn to use **logging** and **colouredlogs** for application monitoring.
- Format code using **black** and check for style with **flake8**.

Mini Project:
- Develop a **real-world application** using the concepts learned.
- It could be a command-line tool, web scraper, or data analysis application.
- Use **Docker** to package the project for deployment.

Hands-on:
- Build the mini project, incorporating Docker, logging, and unit tests.
- Review and refactor code for best practices (style, readability, and performance).

Steps to repeat:
- **Review** and revise any topics that need improvement.
- **Explore more advanced topics** if time permits, like advanced libraries or frameworks.

- Continue to practice by building small projects and contributing to our coding repositories.