Course Outline – Advanced Python

Document Summary

This document details the course outline for Advanced Python training

SUMMARY	The client is keen on equipping a team of 15 - 20 intermediate Python programmers with necessary advanced topics in Python programming through a hands-on instructor led online training.
PROGRAM/ DURATION	20 Hrs (4 hrs per day)
LOCATION	Online / Remote
CLIENT NAME	COMCAST, Chennai
SUBMITTED BY	Arunkumar Krishnamoorthy
SUBMITTED ON	30-July-2024
PREPARED BY	Arunkumar Krishnamoorthy
VERSION #	1.0
EXPECTED START DATE	5th August 2024

TIMINGS	09:30 to 13:30

Objective of the training program

By achieving the following objectives, the training program aims to provide a comprehensive, hands-on learning experience that prepares participants who could be beginners in Python programming (learnt as part of self-learning doing assignments or course work) to tackle complex challenges and excel in their Python programming endeavors writing production ready, enterprise grade applications.

1. **Master Advanced Python Syntax and Constructs**
    - Objective: Ensure participants can confidently write and understand complex Python code, utilizing advanced syntax and programming constructs.
2. **Develop Proficiency in Object-Oriented Programming**
    - Objective: Enable participants to design and implement Python programs using object-oriented principles, including inheritance, polymorphism, and encapsulation.
3. **Leverage Key Python Libraries and Frameworks**
    - Objective: Equip participants with the skills to effectively use essential Python libraries and frameworks for data analysis, web development, and machine learning.
4. **Implement Robust File Handling and Error Management**
    - Objective: Teach participants to manage file operations and handle exceptions gracefully, ensuring robust and error-free code execution.
5. **Utilize Advanced Python Features for Efficiency**
    - Objective: Introduce participants to advanced Python features like decorators, generators, and context managers to write efficient and maintainable code.
6. **Deploy and Manage Python Applications**
    - Objective: Provide hands-on experience in deploying Python applications to various cloud platforms and managing their lifecycle.
7. **Conduct Thorough Testing and Debugging**
    - Objective: Train participants in using testing frameworks and debugging tools to ensure code quality and facilitate troubleshooting.

8. **Collaborate Effectively Using Version Control**
    - Objective: Foster collaboration skills by teaching participants to use Git and GitHub for version control, including branching, merging, and conflict resolution.
9. **Apply Python Skills to Real-World Projects**
    - Objective: Enable participants to apply their Python knowledge to real-world scenarios through mini-projects and a comprehensive capstone project.
10. **Promote Continuous Learning and Problem-Solving**
    - Objective: Encourage participants to engage in continuous learning and enhance their problem-solving abilities by utilizing online resources and coding platforms.

Pre-requisites

Candidate should be comfortable with following topics

- Procedural /Structural, Object oriented, functional programming constructs
- Python language syntax
- installing Python, setting up Python environment, running and debugging python code from IDE
- Installing external libraries using PIP, Importing python modules in a python file
- Version control systems like Git, clone, check-out, check-in processes
- Writing automated unit test cases

LinkedIn Course that will cover the basics - https://www.linkedin.com/learning/python-essential-training-18764650

Learning Outcome

Upon completion of the advanced Python training program, participants will be able to:

1. **Proficiently Use Python Syntax and Data Structures**
    - Demonstrate mastery of Python syntax and effectively use various data types such as lists, dictionaries, sets, and tuples.
    - Perform type conversion and understand the nuances of mutable and immutable types.

2. **Implement Control Structures and Functions**
    - Utilize control structures (if, elif, else, for, while) to create complex, logical flows in programs.
    - Define and invoke functions, including the use of lambda functions, with an understanding of scope and argument passing.
3. **Manage Files and Exceptions**
    - Read from and write to files using Python’s file handling mechanisms.
    - Employ context managers (with statement) for resource management.
    - Handle exceptions gracefully using try, except, finally blocks, and create custom exceptions.
4. **Apply Object-Oriented Programming (OOP) Principles**
    - Design and implement classes and objects, encapsulating attributes and methods.
    - Leverage inheritance, polymorphism, and encapsulation to create reusable and modular code.
    - Utilize advanced OOP concepts like class methods, static methods, and decorators.
5. **Work with Advanced Python Features**
    - Implement decorators and generators to write efficient and clean code.
    - Develop custom context managers for managing resources.
6. **Utilize Key Libraries and Frameworks**
    - Use essential libraries for data manipulation (NumPy, pandas), data visualization (Matplotlib, Seaborn), and web development (Flask, Django).
    - Implement basic machine learning models using scikit-learn and understand the basics of deep learning frameworks like TensorFlow or PyTorch.
    - Perform web scraping using BeautifulSoup and Scrapy.
7. **Develop and Deploy Applications**
    - Build and deploy web applications using Flask or Django.
8. **Test and Debug Python Applications**

    - Write and execute unit tests using unittest and pytest.
    - Utilize debugging tools (pdb, logging) to troubleshoot and resolve issues in Python code.

9. **Collaborate Using Version Control**
    - Use Git for version control, including branching, merging, and resolving conflicts.
    - Collaborate on projects using GitHub or similar platforms.

By achieving these outcomes, participants will be equipped with the knowledge and skills required to tackle complex programming challenges and develop robust Python applications in various domains.

Modules

1. Functional Programming with Python

2. Modular and Object Oriented Programming in Python

3. Asynchronous Programming in Python

4. Dependency management, packaging and deployment in Python along with setting up of internal PyPi repository for release management

6. Web application development with Python, DJango, Flask and MySQL

7. REST API based web application with Python Fast API and Angular / React web app

8. Building microservices with Python

9. PyTest and Behaviour Driven Development in Python

10. Data analysis using Python and Pandas

10. Computer Vision in Python using OpenCV

11. NLP in Python using spacy, nltk

12. PDF document processing using PyPDF

13. Content extraction from unstructured documents using Tesseract OCR

14. ML in Python - Regression models

Customer Requirements & Expectation

- Train 15 - 20 intermediate Python programmers on advanced python topics
- Conduct an online instructor-led classroom training program
- Program should cover assessments, learning design, learning delivery

- Training should be immersive, and interactive
- Individuals should be competent to write, test and deploy high performant, robust, production ready enterprise applications using Pythons
- Duration of the program to be decided after discussion with client

Course Curriculum

Day 1 – Session 1

Module 1	1. Functional Programming with Python 2. Modular and Object Oriented Programming in Python
Experience	Write small code snippets using each / some of the constructs
Topics Covered	**1. Advanced Python Concepts** • Iterators and Generators • Decorators • Context Managers • Metaclasses • Abstract Base Classes **Functional Programming** • Higher-Order Functions • Lambdas • Map, Filter, and Reduce • Functional Libraries (e.g., functools, itertools) **Object-Oriented Programming** • Advanced OOP Techniques • Inheritance and Polymorphism • Magic Methods • Mixins and Composition

Day 2 – Session 1

Module 1	3. Asynchronous Programming in Python 4. Dependency management, packaging
Experience	Implement few data structures and algorithms
Topics Covered	**Data Structures and Algorithms** • Advanced Data Structures (e.g., heaps, balanced trees) • Algorithms (e.g., sorting, searching) **Dependency Management** • Managing Dependencies with pip

	• Virtual Environments • Dependency Resolution **Packaging** • Creating Packages • Distributing Packages **Deployment** • Deploying Python Applications • Continuous Integration/Continuous Deployment (CI/CD) **Internal PyPI Repository** (stretch goal) • Setting Up an Internal PyPI Repository • Managing Releases • Using the Internal Repository

Day 3 – Session 1

Module 1	5. Asynchronous programming in Python 6. Web application development with Python, DJango, Flask and MySQL 7. REST API based web application with Python Fast API and Angular / React web app
Experience	Build a mini end-to-end web application
Topics Covered	**Concurrency and Parallelism** • Threading • Multiprocessing • Asyncio • Concurrent Futures **Python in Web Development** • Advanced Flask/Django • REST APIs • WebSockets • Asynchronous Programming with Aiohttp

Day 4 – Session 1

Module 1	8. Building microservices with Python 9. PyTest and Behaviour Driven Development in Python 10. Data analysis using Python and Pandas

Experience	Setup a mini microservices based Python application Write unit and bdd test cases for individual services Implement small data processing application using Pandas
Topics Covered	**Microservices** • Designing Microservices • Communication Between Services • API Gateway • Deployment Strategies **Testing and Debugging** • Unit Testing with unittest and pytest • Mocking and Patching • Code Coverage • Debugging Techniques • **Behavior-Driven Development (BDD)** ○ Introduction to BDD ○ Tools (e.g., Behave, Lettuce) ○ Writing and Running BDD Tests **Performance Optimization** • Profiling • Cython • PyPy • Memory Management

Day 5 – Session 1

Module 1	10. Computer Vision in Python using OpenCV 11. NLP in Python using spacy, nltk 12. PDF document processing using PyPDF 13. Content extraction from unstructured documents using Tesseract OCR 14. ML in Python - Regression models
Experience	Implement basic algorithms in each of the techniques using 3rd party libraries
Topics Covered	**Data Science and Machine Learning** • Numpy and Pandas Advanced Techniques • Scikit-learn Advanced Usage • TensorFlow/PyTorch • Data Visualization with Matplotlib/Seaborn **Linear Regression**

	• Theory and Concepts • Implementation with Scikit-learn • Model Evaluation **Natural Language Processing with SpaCy** • Tokenization, Lemmatization, POS Tagging • Named Entity Recognition • Text Classification **PDF Processing** • Extracting Text from PDFs • Creating and Manipulating PDFs • Libraries (e.g., PyPDF2, ReportLab) **Optical Character Recognition (OCR)** • Introduction to OCR ○ What is OCR? ○ Use Cases for OCR • Using Tesseract for OCR ○ Installing Tesseract and pytesseract ○ Extracting Text from Scanned PDFs ○ Preprocessing Images for Better OCR Results • Advanced OCR Techniques ○ Handling Noisy and Low-Quality Scans ○ Extracting Structured Data from Scanned Documents

Extra Session (if time permits)

Module 1	Best Practices and Design Patterns Networking and Security Deployment and DevOps
Topics Covered	**Best Practices and Design Patterns** • Code Quality and PEP8 • Design Patterns (e.g., Singleton, Factory, Observer) • Refactoring **Networking and Security** • Network Programming with Sockets • Security Practices • Encryption with PyCrypto • Web Security **Deployment and DevOps** • Dockerizing Python Applications • CI/CD with Jenkins/GitHub Actions

	• Monitoring and Logging • Cloud Services (e.g., AWS, Azure)

Detailed Curriculum (tentative)

1. Advanced Python Concepts

1.1 Iterators and Generators

- **Iterators**
    - Introduction to Iterators
    - Building Custom Iterators
    - The Iterator Protocol
- **Generators**
    - Generator Functions
    - Generator Expressions
    - Using yield
    - Comparing Generators with Iterators
    - Generator Use Cases (e.g., handling large data streams)

1.2 Decorators

- **Function Decorators**
    - Basics of Function Decorators
    - Decorating Functions with Parameters
    - Stacking Decorators
- **Class Decorators**
    - Decorating Classes
    - Use Cases for Class Decorators
- **Practical Examples**
    - Caching/Memoization
    - Logging
    - Access Control

1.3 Context Managers

- **Using with Statements**
    - Basics of with Statements
    - Built-in Context Managers (e.g., file handling)
- **Creating Custom Context Managers**
    - Using __enter__ and __exit__
    - The contextlib Module
    - Practical Examples (e.g., database connections)

1.4 Metaclasses

- **Introduction to Metaclasses**
  - What Are Metaclasses?
  - Use Cases for Metaclasses
- **Creating Metaclasses**
  - The type Function
  - Customizing Class Creation
  - Practical Examples

1.5 Abstract Base Classes

- **Introduction to ABCs**
  - What Are Abstract Base Classes?
  - Why Use ABCs?
- **Creating and Using ABCs**
  - The abc Module
  - Defining Abstract Methods
  - Practical Examples

1.6 Functional Programming

- **Higher-Order Functions**
  - Definition and Examples
  - Common Higher-Order Functions (e.g., map, filter, reduce)
- **Lambda Functions**
  - Syntax and Use Cases
  - Differences from Regular Functions
- **Functional Libraries**
  - Overview of functools
  - Using itertools for Efficient Iteration
  - The operator Module for Functional Programming

1.7 Object-Oriented Programming

- **Advanced OOP Techniques**
  - Inheritance and Composition
  - Polymorphism
- **Magic Methods**
  - Common Magic Methods (e.g., __init__, __str__, __repr__)
  - Overriding Magic Methods
- **Mixins**
  - Introduction to Mixins
  - Creating and Using Mixins
  - Use Cases for Mixins

2. Data Structures and Algorithms

2.1 Advanced Data Structures

- **Heaps**
  - Introduction to Heaps
  - Implementing Heaps with heapq
  - Use Cases (e.g., Priority Queues)
- **Balanced Trees**
  - Introduction to Balanced Trees
  - Types of Balanced Trees (e.g., AVL Trees, Red-Black Trees)
  - Implementing Balanced Trees
- **Graphs**
  - Graph Representations (e.g., Adjacency Matrix, Adjacency List)
  - Implementing Graphs
  - Graph Traversal Algorithms (e.g., BFS, DFS)
- **Trie (Prefix Tree)**
  - Introduction to Trie
  - Implementing Trie
  - Use Cases (e.g., Autocomplete, Spell Checking)

2.2 Algorithms

- **Sorting Algorithms**
  - Quick Sort
  - Merge Sort
  - Heap Sort
  - Comparison of Sorting Algorithms
- **Searching Algorithms**
  - Binary Search
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)
- **Graph Algorithms**
  - Shortest Path Algorithms (e.g., Dijkstra's, A*)
  - Minimum Spanning Tree (e.g., Kruskal's, Prim's)
- **Dynamic Programming**
  - Introduction to Dynamic Programming
  - Common Dynamic Programming Problems (e.g., Knapsack Problem, Longest Increasing Subsequence)
- **Greedy Algorithms**
  - Introduction to Greedy Algorithms
  - Common Greedy Problems (e.g., Huffman Coding, Activity Selection)

2.3 Complexity Analysis

- **Big O Notation**
  - Introduction to Big O Notation

    - Analyzing Time Complexity
    - Analyzing Space Complexity
- **Common Time Complexities**
    - Constant Time (O(1))
    - Logarithmic Time (O(log n))
    - Linear Time (O(n))
    - Linearithmic Time (O(n log n))
    - Quadratic Time (O(n^2))
    - Exponential Time (O(2^n))
- **Amortized Analysis**
    - Introduction to Amortized Analysis
    - Example: Dynamic Arrays

3. Concurrency and Parallelism

3.1 Threading

- **Introduction to Threading**
    - What is Threading?
    - The Global Interpreter Lock (GIL)
- **Thread Management**
    - Creating and Starting Threads
    - Synchronizing Threads (Locks, Semaphores)
    - Thread Pools

3.2 Multiprocessing

- **Introduction to Multiprocessing**
    - What is Multiprocessing?
    - Differences Between Threading and Multiprocessing
- **Process Management**
    - Creating and Managing Processes
    - Inter-Process Communication (Pipes, Queues)
    - Process Pools

3.3 Asyncio

- **Introduction to Asyncio**
    - What is Asyncio?
    - Asyncio vs. Threading and Multiprocessing
- **Asyncio Basics**
    - Creating Async Functions
    - Running Async Tasks
    - Awaiting Coroutines
- **Advanced Asyncio**
    - Task Scheduling
    - Handling Exceptions in Asyncio

    - Asyncio Event Loop

3.4 Concurrent Futures

- **Introduction to Concurrent Futures**
    - What is Concurrent Futures?
    - Using ThreadPoolExecutor
    - Using ProcessPoolExecutor
- **Advanced Techniques**
    - Managing Futures
    - Handling Exceptions
    - Combining Futures with Asyncio

11. Dependency Management, Packaging, and Deployment

11.1 Dependency Management

- **Managing Dependencies with pip**
    - Installing Packages
    - Requirements Files
    - Freezing and Recreating Environments
- **Virtual Environments**
    - Creating Virtual Environments with venv
    - Managing Environments with virtualenv
    - Using pipenv for Environment and Dependency Management
- **Dependency Resolution**
    - Understanding Dependency Conflicts
    - Using pip-tools to Manage Dependencies

11.2 Packaging

- **Creating Packages**
    - Structuring Your Project
    - Creating setup.py Files
    - Including Package Metadata
    - Creating Source Distributions and Wheels
- **Distributing Packages**
    - Using twine to Upload Packages to PyPI
    - Managing Package Versions
    - Best Practices for Versioning

11.3 Deployment

- **Deploying Python Applications**
    - Overview of Deployment Strategies
    - Packaging Applications with pyinstaller
    - Creating Executable Files
- **Continuous Integration/Continuous Deployment (CI/CD)**

    - Introduction to CI/CD
    - Setting Up CI/CD Pipelines with Jenkins
    - Using GitHub Actions for CI/CD
    - Deploying to Cloud Platforms (e.g., AWS, Azure)

11.4 Internal PyPI Repository

- **Setting Up an Internal PyPI Repository**
    - Introduction to Internal Repositories
    - Using devpi to Set Up a Local PyPI Server
    - Configuring devpi for Your Organization
- **Managing Releases**
    - Publishing Packages to the Internal Repository
    - Version Management and Release Cycles
    - Access Control and Security
- **Using the Internal Repository**
    - Configuring pip to Use the Internal Repository
    - Best Practices for Managing Internal Dependencies

4. Python in Web Development

4.1 Advanced Flask/Django

- **Flask**
    - Advanced Routing
    - Blueprint Architecture
    - Middleware and Hooks
    - Flask Extensions (e.g., Flask-Login, Flask-Migrate)
    - Testing Flask Applications
- **Django**
    - Advanced ORM Queries
    - Custom Managers and QuerySets
    - Middleware
    - Django Signals
    - Testing Django Applications

4.2 REST APIs

- **Designing RESTful APIs**
    - REST Principles
    - Resource Design
    - Versioning
- **Implementing REST APIs**
    - Using Flask-RESTful
    - Using Django REST Framework
    - Authentication and Authorization
    - Pagination, Filtering, and Sorting

- **Testing REST APIs**
    - Writing Unit Tests
    - Using Postman for API Testing
    - Automated Testing with pytest

4.3 WebSockets

- **Introduction to WebSockets**
    - WebSocket Protocol
    - Use Cases for WebSockets
- **Implementing WebSockets**
    - Using websockets Library in Python
    - Integrating WebSockets with Flask
    - Integrating WebSockets with Django Channels
- **Practical Examples**
    - Real-Time Chat Application
    - Real-Time Data Dashboard

4.4 Asynchronous Programming with Aiohttp

- **Introduction to Aiohttp**
    - Overview of Aiohttp
    - When to Use Aiohttp
- **Building Async Web Applications**
    - Setting Up Aiohttp Server
    - Routing and Handling Requests
    - Middleware and Error Handling
- **Advanced Aiohttp**
    - WebSockets with Aiohttp
    - Integrating with Databases
    - Testing Aiohttp Applications

4.5 Microservices

- **Designing Microservices**
    - Principles of Microservices Architecture
    - Identifying Service Boundaries
    - Designing APIs for Microservices
- **Communication Between Services**
    - Synchronous Communication (REST, gRPC)
    - Asynchronous Communication (Message Queues, Event-Driven Architecture)
- **API Gateway**
    - Introduction to API Gateways
    - Implementing API Gateway with Kong or Nginx
    - Handling Service Discovery and Load Balancing

- **Deployment Strategies**
    - Containerization with Docker
    - Orchestration with Kubernetes
    - Monitoring and Logging Microservices

5. Data Science and Machine Learning

5.1 Numpy and Pandas Advanced Techniques

- **Numpy**
    - **Advanced Array Operations**
        - Broadcasting and Vectorization
        - Universal Functions
        - Structured Arrays
    - **Performance Optimization**
        - Memory Layout and Strides
        - Working with Large Datasets
- **Pandas**
    - **Advanced DataFrame Operations**
        - GroupBy Operations
        - Pivot Tables and Crosstabs
        - Time Series Analysis
    - **Data Merging, Joining, and Concatenation**
        - Merge vs. Join
        - Concatenation Techniques
        - Handling Duplicates
    - **Handling Missing Data**
        - Imputation Techniques
        - Dropping Missing Values
    - **Performance Optimization**
        - Efficient Data Manipulation
        - Working with Large DataFrames

5.2 Scikit-learn Advanced Usage

- **Advanced Model Selection**
    - **Cross-Validation Techniques**
        - K-Fold Cross-Validation
        - Stratified Cross-Validation
    - **Hyperparameter Tuning**
        - Grid Search
        - Random Search
        - Bayesian Optimization
- **Pipeline and Feature Engineering**
    - **Building Pipelines**

        - Creating and Using Pipelines
        - Custom Pipeline Components
    - **Custom Transformers**
        - Implementing Custom Transformations
        - Integrating with Pipelines
    - **Feature Selection and Extraction**
        - Univariate Feature Selection
        - Recursive Feature Elimination
        - Principal Component Analysis (PCA)

5.3 TensorFlow/PyTorch

- **TensorFlow**
    - **Advanced TensorFlow Concepts**
        - TensorFlow 2.x Basics
        - Eager Execution vs. Graph Execution
    - **Building and Training Complex Models**
        - Custom Layers and Models
        - Training Loops with tf.GradientTape
    - **TensorFlow Extended (TFX) for End-to-End ML Pipelines**
        - TFX Components
        - Data Validation and Transformations
        - Model Deployment with TFX
- **PyTorch**
    - **Advanced PyTorch Concepts**
        - Autograd and Computational Graphs
        - Custom Loss Functions and Layers
    - **Implementing Complex Models**
        - Recurrent Neural Networks (RNNs)
        - Generative Adversarial Networks (GANs)
    - **Using PyTorch Lightning for High-Level Model Management**
        - Introduction to PyTorch Lightning
        - Structuring Code with Lightning Modules
        - Training and Evaluation with Lightning

5.4 Data Visualization with Matplotlib/Seaborn

- **Matplotlib**
    - **Advanced Plotting Techniques**
        - Subplots and GridSpec
        - Customizing Axes and Ticks
    - **Customizing Plots**
        - Customizing Plot Aesthetics
        - Creating Custom Plot Styles

    - **Interactive Visualizations**
        - Using matplotlib Widgets
        - Interactive Plots with mpl_interactions
- **Seaborn**
    - **Advanced Seaborn Plots**
        - PairPlots and PairGrids
        - FacetGrids
    - **Customizing Seaborn Visualizations**
        - Custom Palettes and Styles
        - Annotating Plots
    - **Integrating Seaborn with Matplotlib**
        - Combining Seaborn and Matplotlib Elements
        - Advanced Plot Customizations

5.5 Linear Regression

- **Theory and Concepts**
    - **Introduction to Linear Regression**
        - Simple Linear Regression
        - Multiple Linear Regression
    - **Assumptions of Linear Regression**
        - Linearity, Independence, Homoscedasticity, Normality
    - **Interpretation of Coefficients**
        - Understanding Model Coefficients
        - Interpreting Statistical Significance
- **Implementation with Scikit-learn**
    - **Fitting a Linear Regression Model**
        - Preparing Data
        - Training and Evaluating the Model
    - **Model Evaluation Metrics**
        - R-squared, Adjusted R-squared
        - Mean Squared Error, Mean Absolute Error
    - **Handling Multicollinearity**
        - Detecting Multicollinearity
        - Techniques to Address Multicollinearity
- **Model Evaluation**
    - **Residual Analysis**
        - Plotting Residuals
        - Analyzing Residual Patterns
    - **Cross-Validation**
        - K-Fold Cross-Validation
        - Leave-One-Out Cross-Validation

    - **Regularization Techniques (Ridge, Lasso)**
        - Understanding Ridge and Lasso Regression
        - Implementing Regularization with Scikit-learn

5.6 Natural Language Processing with SpaCy

- **Introduction to SpaCy**
    - **SpaCy Basics**
        - Tokenization, Lemmatization, POS Tagging
        - Named Entity Recognition
    - **SpaCy vs. Other NLP Libraries**
- **Core NLP Tasks**
    - **Tokenization, Lemmatization, POS Tagging**
        - Custom Tokenization
        - Custom Pipeline Components
    - **Named Entity Recognition**
        - Training Custom NER Models
        - Evaluating NER Performance
    - **Dependency Parsing**
        - Understanding Dependency Trees
        - Visualizing Dependency Trees
- **Advanced NLP with SpaCy**
    - **Text Classification**
        - Rule-Based Matching
        - Training Text Classification Models
    - **Customizing SpaCy Pipelines**
        - Adding Custom Components
        - Modifying Built-in Components
    - **Training Custom Models**
        - Data Preparation and Annotation
        - Using SpaCy's Training API

5.7 PDF Processing

- **Extracting Text from PDFs**
    - **Using PyPDF2 for Text Extraction**
        - Basic Text Extraction
        - Handling Complex PDF Structures
    - **Using pdfminer.six for Detailed Extraction**
        - Extracting Text and Metadata
        - Handling Encrypted PDFs
- **Creating and Manipulating PDFs**
    - **Creating PDFs with ReportLab**
        - Basic PDF Creation

- Benefits of Measuring Code Coverage

- **Using coverage.py**
  - Installing and Configuring coverage.py
  - Running Tests with Coverage
  - Generating Coverage Reports
  - Analyzing Coverage Data
- **Improving Code Coverage**
  - Identifying Uncovered Code
  - Writing Additional Tests
  - Best Practices for Code Coverage

6.4 Debugging Techniques

- **Introduction to Debugging**
  - Common Debugging Approaches
  - Debugging Tools in Python
- **Using pdb (Python Debugger)**
  - Basics of pdb
  - Setting Breakpoints
  - Stepping Through Code
  - Inspecting Variables and State
- **Using IDE Debugging Tools**
  - Debugging in PyCharm
  - Debugging in VSCode
- **Advanced Debugging**
  - Remote Debugging
  - Debugging Multithreaded and Multiprocess Code
  - Using ipdb for IPython Integration

6.5 Behavior-Driven Development (BDD)

- **Introduction to BDD**
  - What is BDD?
  - Differences Between BDD and TDD
  - Benefits of BDD
- **Using Behave**
  - Installing Behave
  - Writing Features and Scenarios
  - Implementing Step Definitions
  - Running BDD Tests
- **Using Lettuce**
  - Overview of Lettuce
  - Writing and Running Tests with Lettuce
- **Integrating BDD with CI/CD Pipelines**

    - Automating BDD Tests
    - Reporting and Analyzing Test Results

7. Performance Optimization

7.1 Profiling

- **Introduction to Profiling**
    - Importance of Profiling
    - Types of Profiling (CPU, Memory)
- **Using cProfile**
    - Basics of cProfile
    - Running and Interpreting Profiles
    - Visualizing Profile Data with snakeviz
- **Using line_profiler and memory_profiler**
    - Line-by-Line Profiling with line_profiler
    - Memory Profiling with memory_profiler

7.2 Cython

- **Introduction to Cython**
    - What is Cython?
    - Benefits of Using Cython
- **Writing Cython Code**
    - Cython Syntax and Basics
    - Compiling Cython Code
    - Integrating Cython with Python Projects
- **Performance Optimization with Cython**
    - Profiling and Identifying Bottlenecks
    - Optimizing Critical Code Paths

7.3 PyPy

- **Introduction to PyPy**
    - What is PyPy?
    - Differences Between PyPy and CPython
- **Using PyPy**
    - Installing and Running PyPy
    - Compatibility with Python Libraries
    - Performance Benchmarks

7.4 Memory Management

- **Understanding Python's Memory Model**
    - Memory Allocation in Python
    - Reference Counting and Garbage Collection
- **Optimizing Memory Usage**
    - Identifying Memory Leaks
    - Using tracemalloc for Memory Profiling

    - Techniques for Reducing Memory Usage

8. Best Practices and Design Patterns

8.1 Code Quality and PEP8

- **Introduction to Code Quality**
    - Importance of Code Quality
    - PEP8 Coding Standards
- **Using Linters**
    - flake8
    - pylint
    - Integrating Linters with IDEs and CI/CD Pipelines
- **Code Formatting Tools**
    - black
    - isort

8.2 Design Patterns

- **Introduction to Design Patterns**
    - What Are Design Patterns?
    - Benefits of Using Design Patterns
- **Creational Patterns**
    - Singleton
    - Factory Method
    - Abstract Factory
- **Structural Patterns**
    - Adapter
    - Composite
    - Decorator
- **Behavioral Patterns**
    - Observer
    - Strategy
    - Command

8.3 Refactoring

- **Introduction to Refactoring**
    - What is Refactoring?
    - Benefits of Refactoring
- **Common Refactoring Techniques**
    - Extract Method
    - Rename Variable
    - Simplify Conditionals
- **Tools for Refactoring**
    - rope for Python
    - Refactoring in IDEs (e.g., PyCharm, VSCode)

6. Testing and Debugging

6.1 Unit Testing with unittest and pytest

- **Introduction to Unit Testing**
    - Importance of Unit Testing
    - Overview of Testing Frameworks in Python
- **Using unittest**
    - Creating Test Cases
    - Assertions in unittest
    - Running and Organizing Tests
- **Using pytest**
    - Basics of pytest
    - Writing Simple Tests
    - Fixtures in pytest
    - Parametrizing Tests
    - Comparing unittest and pytest

6.2 Mocking and Patching

- **Introduction to Mocking**
    - What is Mocking?
    - Use Cases for Mocking
- **Using unittest.mock**
    - Creating Mocks
    - Patching Objects
    - Mocking Side Effects
    - Asserting Calls and Call Counts
- **Advanced Mocking Techniques**
    - Mocking Classes and Methods
    - Using pytest-mock Plugin

6.3 Code Coverage

- **Introduction to Code Coverage**
    - What is Code Coverage?
    - Benefits of Measuring Code Coverage
- **Using coverage.py**
    - Installing and Configuring coverage.py
    - Running Tests with Coverage
    - Generating Coverage Reports
    - Analyzing Coverage Data
- **Improving Code Coverage**
    - Identifying Uncovered Code
    - Writing Additional Tests
    - Best Practices for Code Coverage

6.4 Debugging Techniques

- **Introduction to Debugging**
    - Common Debugging Approaches
    - Debugging Tools in Python
- **Using pdb (Python Debugger)**
    - Basics of pdb
    - Setting Breakpoints
    - Stepping Through Code
    - Inspecting Variables and State
- **Using IDE Debugging Tools**
    - Debugging in PyCharm
    - Debugging in VSCode
- **Advanced Debugging**
    - Remote Debugging
    - Debugging Multithreaded and Multiprocess Code
    - Using ipdb for IPython Integration

6.5 Behavior-Driven Development (BDD)

- **Introduction to BDD**
    - What is BDD?
    - Differences Between BDD and TDD
    - Benefits of BDD
- **Using Behave**
    - Installing Behave
    - Writing Features and Scenarios
    - Implementing Step Definitions
    - Running BDD Tests
- **Using Lettuce**
    - Overview of Lettuce
    - Writing and Running Tests with Lettuce
- **Using pytest-bdd**
    - Introduction to pytest-bdd
    - Installing pytest-bdd
    - Writing Features and Scenarios
    - Implementing Step Definitions with pytest
    - Running BDD Tests with pytest
- **Integrating BDD with CI/CD Pipelines**
    - Automating BDD Tests
    - Reporting and Analyzing Test Results

7. Performance Optimization

7.1 Profiling

- **Introduction to Profiling**
    - Importance of Profiling
    - Types of Profiling (CPU, Memory)
- **Using cProfile**
    - Basics of cProfile
    - Running and Interpreting Profiles
    - Visualizing Profile Data with snakeviz
- **Using line_profiler and memory_profiler**
    - Line-by-Line Profiling with line_profiler
    - Memory Profiling with memory_profiler

7.2 Cython

- **Introduction to Cython**
    - What is Cython?
    - Benefits of Using Cython
- **Writing Cython Code**
    - Cython Syntax and Basics
    - Compiling Cython Code
    - Integrating Cython with Python Projects
- **Performance Optimization with Cython**
    - Profiling and Identifying Bottlenecks
    - Optimizing Critical Code Paths

7.3 PyPy

- **Introduction to PyPy**
    - What is PyPy?
    - Differences Between PyPy and CPython
- **Using PyPy**
    - Installing and Running PyPy
    - Compatibility with Python Libraries
    - Performance Benchmarks

7.4 Memory Management

- **Understanding Python's Memory Model**
    - Memory Allocation in Python
    - Reference Counting and Garbage Collection
- **Optimizing Memory Usage**
    - Identifying Memory Leaks
    - Using tracemalloc for Memory Profiling
    - Techniques for Reducing Memory Usage

8. Best Practices and Design Patterns

8.1 Code Quality and PEP8

- **Introduction to Code Quality**

    - Importance of Code Quality
    - PEP8 Coding Standards
- **Using Linters**
    - flake8
    - pylint
    - Integrating Linters with IDEs and CI/CD Pipelines
- **Code Formatting Tools**
    - black
    - isort

8.2 Design Patterns

- **Introduction to Design Patterns**
    - What Are Design Patterns?
    - Benefits of Using Design Patterns
- **Creational Patterns**
    - Singleton
    - Factory Method
    - Abstract Factory
- **Structural Patterns**
    - Adapter
    - Composite
    - Decorator
- **Behavioral Patterns**
    - Observer
    - Strategy
    - Command

8.3 Refactoring

- **Introduction to Refactoring**
    - What is Refactoring?
    - Benefits of Refactoring
- **Common Refactoring Techniques**
    - Extract Method
    - Rename Variable
    - Simplify Conditionals
- **Tools for Refactoring**
    - rope for Python
    - Refactoring in IDEs (e.g., PyCharm, VSCode)

9. Networking and Security

9.1 Networking Basics

- **Introduction to Networking**

    - Understanding Network Protocols
    - OSI Model Overview
- **Sockets Programming**
    - Introduction to Sockets
    - Creating TCP and UDP Sockets
    - Socket Programming with socket Library
- **Advanced Sockets**
    - Non-Blocking Sockets
    - Handling Multiple Connections
    - Using selectors for Efficient I/O Multiplexing

9.2 HTTP and Web Scraping

- **HTTP Protocol Basics**
    - Understanding HTTP Requests and Responses
    - Working with HTTP Headers and Status Codes
- **Web Scraping with requests and BeautifulSoup**
    - Fetching Web Pages with requests
    - Parsing HTML with BeautifulSoup
    - Handling Pagination and AJAX Requests
- **Advanced Web Scraping**
    - Using Selenium for Dynamic Content
    - Scrapy Framework for Large-Scale Scraping
    - Handling Anti-Scraping Techniques

9.3 RESTful APIs

- **Consuming APIs**
    - Making API Calls with requests
    - Handling Authentication (API Keys, OAuth)
    - Parsing JSON Responses
- **Creating APIs**
    - Building APIs with Flask
    - Building APIs with Django REST Framework
    - Best Practices for API Design

9.4 Security Practices

- **Common Security Vulnerabilities**
    - Introduction to OWASP Top 10
    - Understanding SQL Injection, XSS, CSRF
- **Securing Python Applications**
    - Input Validation and Sanitization
    - Using Secure Coding Practices
    - Implementing Authentication and Authorization
- **Encryption and Cryptography**

        - Using cryptography Library
        - Symmetric vs Asymmetric Encryption
        - Hashing and Digital Signatures

9.5 Network Security

- **Introduction to Network Security**
    - Basics of Firewalls and VPNs
    - Intrusion Detection Systems (IDS)
- **Securing Network Communications**
    - Using SSL/TLS for Secure Communication
    - Setting Up HTTPS with Flask and Django
    - Implementing Secure WebSocket Connections
- **Monitoring and Logging**
    - Logging Network Activities
    - Analyzing Network Logs
    - Using Tools like Wireshark for Network Analysis

10. Cloud Computing and Big Data

10.1 Cloud Platforms

- **Introduction to Cloud Computing**
    - Benefits of Cloud Computing
    - Overview of Cloud Service Models (IaaS, PaaS, SaaS)
- **Using AWS**
    - Setting Up AWS EC2 Instances
    - Working with AWS S3
    - Using AWS Lambda for Serverless Computing
- **Using Azure**
    - Creating and Managing Azure VMs
    - Working with Azure Blob Storage
    - Using Azure Functions
- **Using Google Cloud**
    - Setting Up Google Compute Engine
    - Working with Google Cloud Storage
    - Using Google Cloud Functions

10.2 Big Data Processing

- **Introduction to Big Data**
    - Understanding Big Data Concepts
    - Overview of Big Data Technologies
- **Hadoop Ecosystem**
    - Introduction to Hadoop
    - Working with HDFS
    - Using MapReduce for Data Processing

- **Spark**
    - Introduction to Apache Spark
    - Spark RDDs, DataFrames, and Datasets
    - Writing Spark Applications with PySpark
- **Big Data Storage Solutions**
    - Using NoSQL Databases (e.g., MongoDB, Cassandra)
    - Working with HBase
    - Using Amazon Redshift for Data Warehousing

Conclusion and Additional Resources

12.1 Conclusion

- **Review of Key Concepts**
    - Recap of Major Topics Covered
    - Key Takeaways from Each Section
- **Next Steps for Continued Learning**
    - Identifying Areas for Further Study
    - Resources for Advanced Topics

12.2 Additional Resources

- **Books and Online Courses**
    - Recommended Reading
    - Online Courses and Tutorials
- **Communities and Conferences**
    - Joining Python Communities (e.g., PyCon, local meetups)
    - Staying Updated with Python News and Developments
- **Practice and Projects**
    - Contributing to Open Source Projects
    - Building Personal Projects to Apply Learned Concepts