**Chapter 1: Introduction to Python Basics**

Tasks:

1. Print "Hello, World!" using the `print()` function.

2. Perform basic arithmetic operations (+, -, \*, /).

3. Convert temperature from Celsius to Fahrenheit.

4. Calculate the area of a triangle given base and height.

5. Swap the values of two variables.

6. Check if a number is even or odd.

7. Calculate the sum of digits in a number.

8. Concatenate two strings.

9. Create a list of numbers and print their squares.

10. Generate a random number between 1 and 100.

**Chapter 2: Control Structures**

Tasks:

1. Write a program for a simple guessing game (use `if` statements).

2. Use a loop to print even numbers from 1 to 20.

3. Implement a basic calculator using conditional statements.

4. Print Fibonacci series up to a given number.

5. Check if a number is prime or not.

6. Use a loop to find the factorial of a number.

7. Create a program that prints the multiplication table of a given number.

8. Generate a list of numbers and filter out even numbers using a list comprehension.

9. Create a program that sums numbers from a given range using a `while` loop.

10. Implement a rock-paper-scissors game against the computer.

**Chapter 3: Functions and Modules**

Tasks:

1. Write a function to calculate the area of a rectangle.

2. Create a function to find the maximum of three numbers.

3. Build a function that checks if a string is a palindrome.

4. Implement a module to calculate the perimeter of common geometric shapes.

5. Write a recursive function to calculate the factorial of a number.

6. Create a program that uses the `math` module to calculate the square root.

7. Build a function to find the greatest common divisor (GCD) of two numbers.

8. Create a module that converts currency units.

9. Write a function that generates a list of random numbers.

10. Implement a basic calculator using functions for different operations.

**Chapter 4: Data Structures - Lists and Tuples**

Tasks:

1. Find the largest element in a list.

2. Count the occurrences of an element in a list.

3. Concatenate two lists.

4. Reverse a list without using the reverse function.

5. Implement a stack using a list (push, pop operations).

6. Create a program that sorts a list of numbers (without using built-in sorting functions).

7. Write a function that returns common elements between two lists.

8. Convert a list of tuples into a dictionary.

9. Use list comprehension to extract vowels from a list of words.

10. Remove duplicates from a list and preserve the order.

**Chapter 5: Data Structures - Dictionaries and Sets**

Tasks:

1. Create a dictionary of words and their meanings.

2. Implement a frequency counter using a dictionary for a list of elements.

3. Check if two dictionaries have a common key.

4. Remove a key from a dictionary and handle errors if the key is absent.

5. Build a set of unique prime numbers.

6. Perform set operations: union, intersection, difference.

7. Create a program that counts the frequency of characters in a string using a dictionary.

8. Implement a priority queue using a dictionary.

9. Write a function that removes duplicates from a list using a set.

10. Implement a simple spell checker using a set of valid words.

**Chapter 6: Data Structures - Lists and List Comprehensions**

Tasks:

1. Use list comprehension to generate a list of squares.

2. Flatten a nested list using list comprehension.

3. Filter out negative numbers from a list using list comprehension.

4. Find the intersection of two lists using list comprehension.

5. Create a matrix using a nested list comprehension.

6. Implement a basic text-based calendar using a list of lists.

7. Generate a list of prime numbers using the Sieve of Eratosthenes algorithm.

8. Implement a circular buffer using a list.

9. Use list comprehension to transform a list of temperatures from Celsius to Fahrenheit.

10. Create a program that finds the second-largest element in a list.

**Chapter 7: Object-Oriented Programming (OOP) Basics**

Tasks:

1. Create a class representing a basic bank account with deposit and withdrawal methods.

2. Implement inheritance with a base class and two derived classes with overridden methods.

3. Create a class hierarchy for different types of vehicles.

4. Implement encapsulation by creating a class with private attributes and methods.

5. Use class composition to model a university composed of departments and students.

6. Implement a simple class for a library catalog and books.

7. Use properties to validate and sanitize the attributes of a class.

8. Create a class for a basic quiz game with questions and multiple-choice answers.

9. Implement a class that simulates a basic shopping cart.

10. Create a class to represent a geometric shape and its properties.

**Chapter 8: File Handling and Exceptions**

Tasks:

1. Read and display the contents of a text file.

2. Write a program to copy the contents of one file to another.

3. Implement a program that counts the occurrences of a word in a text file.

4. Use exception handling to handle a division by zero error.

5. Create a program that reads a CSV file and processes its data.

6. Write a program that reads and parses a JSON file.

7. Implement a program that converts markdown files to HTML using regular expressions.

8. Handle different types of exceptions using multiple `except` blocks.

9. Build a program that searches for a specific string in all text files within a directory.

10. Create a log file and write logs using the `logging` module.

**Chapter 9: Advanced Python Concepts**

Tasks:

1. Use decorators to measure the execution time of a function.

2. Implement a generator that generates prime numbers.

3. Create a context manager to measure the time taken by a code block.

4. Use the `functools` module to create and use partial functions.

5. Build a custom iterator and iterable for a range of numbers.

6. Implement multithreading to perform parallel tasks.

7. Use the `asyncio` library to create asynchronous tasks.

8. Build a web scraper using the `requests` and `BeautifulSoup` libraries.

9. Use regular expressions to validate email addresses.

10. Create a GUI application using a library like `Tkinter` or `PyQt`.

**Chapter 10: Working with Databases**

Tasks:

1. Create a SQLite database and perform basic CRUD operations.

2. Build a program to retrieve and display data from a database.

3. Implement a simple ORM using classes to interact with a database.

4. Write a program that imports data from a CSV file into a database.

5. Implement transactions to ensure data consistency.

6. Build a program that queries and filters data using SQL statements.

7. Use an external database library (e.g., SQLAlchemy) to work with databases.

8. Create a program that generates reports from database data.

9. Implement data migration for a database schema update.

10. Build a RESTful API using a web framework like Flask or Django.

**Chapter 11: Web Development with Python**

Tasks:

1. Set up a basic web server using a framework like Flask or Django.

2. Create a simple web page using HTML and CSS.

3. Implement dynamic content rendering using template engines.

4. Build a RESTful API to perform CRUD operations on a resource.

5. Create a registration and login system with user authentication.

6. Implement form handling and validation for user inputs.

7. Build a simple blogging platform with user-generated posts.

8. Use AJAX to create asynchronous interactions on a web page.

9. Integrate third-party APIs into your web application.

10. Deploy your web application to a hosting platform.

**Chapter 12: Data Visualization with Python**

Tasks:

1. Create line and bar charts using a data visualization library.

2. Build a scatter plot to visualize the relationship between two variables.

3. Implement a histogram to show the distribution of data.

4. Create a pie chart to represent data proportions.

5. Build a time series plot to visualize trends over time.

6. Use a library to create interactive data visualizations.

7. Implement a geographical map to visualize location-based data.

8. Create a dashboard with multiple visualizations.

9. Use data visualization to explore and analyze a dataset.

10. Customize the appearance of your visualizations.

**Chapter 13: Machine Learning and Data Analysis with Python**

Tasks:

1. Perform data preprocessing and cleaning on a dataset.

2. Implement linear regression to predict a continuous variable.

3. Build a classification model using logistic regression.

4. Create a decision tree for a classification problem.

5. Implement k-means clustering for unsupervised learning.

6. Use a library to visualize the decision boundaries of a classification model.

7. Perform feature selection and dimensionality reduction.

8. Build a recommendation system using collaborative filtering.

9. Implement a natural language processing task, like sentiment analysis.

10. Use a pre-trained neural network for image classification.

**Chapter 14: Working with APIs and Data Formats**

Tasks:

1. Make a GET request to a public API and process the response.

2. Implement token-based authentication for API requests.

3. Use a library to parse JSON data from an API response.

4. Send POST requests to submit data to an API.

5. Implement pagination when dealing with APIs that return large datasets.

6. Work with XML data by parsing and extracting information.

7. Implement OAuth2 authentication to access user data.

8. Use a library to handle CSV data and perform analysis.

9. Parse and extract data from HTML using a library.

10. Create a script that automates data retrieval and storage from APIs.

**Chapter 15: Advanced Topics and Projects**

Tasks:

1. Implement data encryption and decryption using cryptography libraries.

2. Create a command-line tool with argument parsing.

3. Implement a basic neural network from scratch.

4. Build a web scraping bot to gather data from multiple websites.

5. Develop a chatbot using natural language processing.

6. Use Python to control and interact with hardware (e.g., Raspberry Pi).

7. Build a recommendation system with collaborative and content-based filtering.

8. Create a project that integrates multiple APIs to provide a unique service.

9. Develop a game using a library like Pygame.

10. Work on a self-directed project that combines multiple concepts learned.