Day 1: Python Basics

Topics:

1. **Introduction to Python:**
   - Key features of Python: Easy syntax, interpreted language, platform-independent, dynamically typed, etc.
   - Applications of Python: Web development, Data Science, AI, Automation, etc.
2. **Python Syntax and Indentation:**
   - Importance of indentation.
   - Writing the first Python program (`print("Hello, World!")`).
3. **Data Types:**
   - Numeric (`int`, `float`, `complex`), String (`str`), Boolean (`bool`).
   - Examples of valid and invalid data types.
4. **Variables:**
   - Declaring and assigning values.
   - Dynamic typing.
   - Rules for naming variables.

Day 2: Input, Output, and Boolean

Topics:

1. **Getting Runtime Values:**
   - Using `input()` for user input.
   - Converting input types (`int(input())`, `float(input())`).
2. **Boolean Data Type:**
   - Understanding `True` and `False`.
   - Logical operators: `and`, `or`, `not`.
   - Boolean comparisons: `<`, `>`, `==`, `!=`.
3. **Practical Examples:**
   - Check if a number is positive or negative.
   - Use boolean conditions to validate input.

Day 3: Operators

Topics:

1. **Arithmetic Operators:**
   - `+`, `-`, `*`, `/`, `%`, `//` (floor division), `**` (exponentiation).
2. **Comparison Operators:**
   - `==`, `!=`, `>`, `<`, `>=`, `<=`.
3. **Logical Operators:**
   - `and`, `or`, `not`.

4. **Bitwise Operators:**
    - `&`, `|`, `^`, `~`, `<<`, `>>`.
5. **Assignment Operators:**
    - `=`, `+=`, `-=`, `*=`, etc.
6. **Special Topics:**
    - Increment and decrement operators (Python's way using `+=` and `-=`).

Day 4: Conditional Statements

Topics:

1. **`if` Statement:**
    - Syntax and examples.
2. **`if-else` Statement:**
    - Syntax and examples.
3. **Nested `if-else`:**
    - Handling multiple levels of conditions.
4. **`elif` Statement:**
    - Simplifying complex conditions.
5. **Examples:**
    - Check if a number is odd/even.
    - Determine if a year is a leap year.
6. **Advanced Concepts:**
    - Ternary operator (conditional expressions): `x = a if condition else b`.

Day 5: Looping Statements

Topics:

1. **`while` Loop:**
    - Syntax and examples.
    - Example: Print numbers from 1 to 10.
2. **`for` Loop:**
    - Iterating through sequences (lists, tuples, strings).
    - Example: Iterate through a string or list.
3. **Loop Control Statements:**
    - `break`, `continue`, `pass`.
4. **Nested Loops:**
    - Examples: Multiplication tables, patterns.
5. **Simulating `do-while`:**
    - Achieving `do-while` behavior using `while`.
6. **Examples:**
    - Factorial of a number.

- Sum of digits of a number.

Day 6: Lists, Tuples, and Dictionaries

Topics:

1. **Lists:**
    - Characteristics of lists (mutable, ordered).
    - List operations: `append()`, `remove()`, `pop()`, `sort()`, `reverse()`.
2. **Tuples:**
    - Characteristics of tuples (immutable, ordered).
    - Tuple operations: `count()`, `index()`.
3. **Dictionaries:**
    - Characteristics of dictionaries (key-value pairs, unordered).
    - Dictionary operations: `keys()`, `values()`, `items()`, `get()`, `update()`.
4. **Examples:**
    - Create a shopping cart (list).
    - Use tuples to store immutable configurations.
    - Store student data in a dictionary.

Day 7: Built-In Functions

Topics:

1. **String Functions:**
    - `lower()`, `upper()`, `capitalize()`, `strip()`, `replace()`, `split()`, `join()`, `find()`, `startswith()`, `endswith()`, `count()`.
2. **List Functions:**
    - `append()`, `extend()`, `pop()`, `remove()`, `index()`, `sort()`, `reverse()`.
3. **Dictionary Functions:**
    - `keys()`, `values()`, `items()`, `get()`, `update()`.
4. **Iterative Functions:**
    - `enumerate()`, `zip()`, `map()`, `filter()`, `reduce()`.
5. **Mathematical Functions:**
    - `sum()`, `max()`, `min()`, `abs()`, `round()`, `divmod()`, `pow()`.
6. **Utility Functions:**
    - `type()`, `id()`, `isinstance()`, `dir()`, `help()`, `eval()`, `exec()`.
7. **Examples:**
    - String manipulations.
    - Create a merged dictionary using `zip()`.
    - Use `map()` to square numbers in a list.

Day 8: Functions (User-defined and Advanced)

- **Basic Concepts**
  - Introduction to functions: definition, purpose, and syntax.
  - Parameters and return values.
  - Local and global variables.
  - Example: A simple function to add two numbers.
- **Intermediate Concepts**
  - Default arguments.
  - Keyword arguments.
  - Variable-length arguments
  - Recursion.
- **Advanced Concepts**
  - Lambda functions (anonymous functions).
  - Higher-order functions (functions that take other functions as arguments).
  - Nested functions and closures.
  - Decorators (creating and using them).

Day 9: File Handling

- **Basic Concepts**
  - Opening, reading, writing, and closing files.
  - File modes: `r`, `w`, `a`, `rb`, `wb`.
  - Example: Reading and writing to a text file.
- **Intermediate Concepts**
  - Reading lines from a file and processing them.
  - Handling binary files.
  - Working with file paths using `os` and `pathlib`.

- **Advanced Concepts**
  - Context managers (`with` statement).
  - File handling exceptions and error handling.
  - Reading and writing CSV, JSON, and XML files.
  - Working with large files efficiently using generators.

Day 10: Error Handling (Exception Handling)

- **Basic Concepts**
  - Introduction to errors and exceptions in Python.
  - `try`, `except`, `else`, and `finally` blocks.
  - Common exceptions (`IndexError`, `ValueError`, etc.).
- **Intermediate Concepts**
  - Raising exceptions using `raise`.
  - Custom exceptions (creating your own exception classes).
  - Handling multiple exceptions in one block.
- **Advanced Concepts**
  - Nested try-except blocks.
  - Using `assert` for debugging.
  - Logging errors using the `logging` module.
  - Best practices for error handling and debugging.

Day 11: Object-Oriented Programming (OOP)

- **Basic Concepts**
  - Introduction to OOP concepts: Classes and objects.
  - Defining classes and creating instances.
  - Instance variables and methods.
  - Example: A simple `Car` class with attributes and methods.
- **Intermediate Concepts**

  - `__init__` (constructor) and `__del__` (destructor).
  - Inheritance and method overriding.
  - Encapsulation (private and public attributes).
  - Polymorphism: method overloading and overriding.

- **Advanced Concepts**
  - Abstract classes and methods.
  - Multiple inheritance.
  - Static and class methods.
  - Composition and aggregation in OOP.

Day 12: Modules and Libraries

- **Basic Concepts**
  - Introduction to Python modules and libraries.
  - Importing modules and using standard libraries (`math`, `datetime`, `os`, etc.).
  - Using `import` and `from ... import ...`.

- **Intermediate Concepts**
  - Creating your own Python modules.
  - Exploring popular third-party libraries: `requests`, `numpy`, `pandas`.
  - Installing and using libraries with `pip`.

- **Advanced Concepts**
  - Python package structure and `__init__.py`.
  - Working with `virtualenv` for environment management.
  - Understanding `__main__` in modules.
  - Creating and distributing your own Python packages.

Day 13: Working with APIs and Data (JSON, REST APIs)

- **Basic Concepts**

- Introduction to APIs and how they work (RESTful APIs).
- Using the `requests` module for HTTP requests.
- Working with JSON data: parsing and converting JSON in Python.

- **Intermediate Concepts**
  - Authentication: API keys, OAuth, and Basic Authentication.
  - Making `GET`, `POST`, `PUT`, and `DELETE` requests.
  - Handling HTTP response status codes.
  - Query parameters and handling paginated data.
- **Advanced Concepts**
  - Error handling in API requests.
  - Working with rate limits and timeouts.
  - Sending and receiving data with APIs in different formats (e.g., XML, CSV).
  - Webhooks and handling real-time data.

Day 14: Working with Databases (SQL, SQLite)

- **Basic Concepts**
  - Introduction to databases: SQL basics.
  - Setting up and connecting to a SQLite database using `sqlite3`.
  - Basic SQL queries: `SELECT`, `INSERT`, `UPDATE`, `DELETE`.
- **Intermediate Concepts**
  - Creating tables, primary keys, and relationships.
  - Using `WHERE`, `JOIN`, and other SQL clauses.
  - Using `commit` and `rollback` for transactions.
- **Advanced Concepts**
  - Handling complex queries with subqueries and aggregations.
  - SQL injection and best practices for preventing it.

  - Using ORM (Object-Relational Mapping) with libraries like SQLAlchemy.
  - Integrating databases with Python applications.

Day 15: Final Project and Wrap-Up

- **Project**
  - A hands-on final project combining various concepts learned (APIs, databases, file handling, OOP, etc.).
  - Examples of project ideas:
    - Build a task management application with a database backend.
    - Create a weather dashboard that pulls data from an external API.
