# **Python Course Syllabus**

This course includes exercises, quizzes, and minor projects for practice outside class hours.

## **Day 1: Python Introduction**

**Theory (1 hour):**

* What is a programming language? Overview of high-level vs. low-level languages.
* Introduction to Python: History, features, and applications (web, data science, automation).
* Python as an interpreted language: How Python executes code.
* Setting up Python: Installing Python and an IDE (e.g., VS Code, PyCharm).
* Virtual environments: Purpose and basic setup using venv.

**Practical (1 hour):**

* Install Python and an IDE on student machines.
* Create and activate a virtual environment.
* Write and run the first Python program (print("Hello, World!")).
* Explore basic Python syntax and execute simple scripts.

**Exercises:**

* Write a program to print your name and a greeting.
* Experiment with basic arithmetic operations in Python.

## **Day 2: Python Basics - Data Types and Variables**

**Theory (1 hour):**

* Python data types: int, float, str, bool, None.
* Variables: Declaration, naming conventions, and dynamic typing.
* Operator precedence: Arithmetic, comparison, logical operators.

**Practical (1 hour):**

* Create variables of different data types and perform operations.
* Write programs using basic arithmetic and comparison operators.
* Practice variable assignments and printing results.

**Exercises:**

* Write a program to swap two variables without using a temporary variable.
* Create a simple calculator for addition and multiplication.

**Quiz:**

* What is the difference between int and float?
* Explain the precedence of \* vs. + in Python.

## **Day 3: Python Basics - Strings and Type Conversion**

**Theory (1 hour):**

* Strings: Creation, concatenation, and methods.
* Escape sequences and formatted strings (f-strings, .format()).
* String indexing, slicing, and immutability.
* Type conversion: int(), str(), float().

**Practical (1 hour):**

* Write programs to manipulate strings (e.g., uppercase, slicing).
* Use f-strings to format output with variables.
* Practice type conversion with user input.

**Exercises:**

* Create a program to reverse a string.
* Write a program to format a user’s name and age into a sentence.

**Quiz:**

* What is an escape sequence? Give two examples.
* Why are strings immutable in Python?

## **Day 4: Python Basics - Data Structures**

**Theory (1 hour):**

* Lists: Creation, indexing, slicing, and methods (e.g., append, remove).
* Dictionaries: Key-value pairs, accessing, and modifying.
* Sets: Unique elements, set operations (union, intersection).
* Tuples: Immutable sequences, use cases.

**Practical (1 hour):**

* Create and manipulate lists, dictionaries, sets, and tuples.
* Write a program to store and retrieve student grades using a dictionary.
* Practice set operations (e.g., finding common elements).

**Exercises:**

* Write a program to sort a list of numbers.
* Create a dictionary to count word frequencies in a sentence.

**Quiz:**

* What is the difference between a list and a tuple?
* Name two methods of a dictionary.

**Minor Project 1:**

* Build a contact book using a dictionary to store names and phone numbers, with options to add, remove, and display contacts.

## **Day 5: Python Basics 2 - Conditionals and Loops**

**Theory (1 hour):**

* Conditionals: if, elif, else, and indentation.
* Logical operators: and, or, not.
* is vs. ==: Identity vs. equality.
* Introduction to loops: for and while.

**Practical (1 hour):**

* Write programs using if statements to check conditions (e.g., age eligibility).
* Create a for loop to iterate over a list.
* Use a while loop to repeatedly prompt for input until valid.

**Exercises:**

* Write a program to check if a number is even or odd.
* Create a loop to print the first 10 Fibonacci numbers.

**Quiz:**

* What is the difference between is and ==?
* Explain the role of indentation in Python.

## **Day 6: Python Basics 2 - Advanced Loops and Functions**

**Theory (1 hour):**

* Loop control: break, continue, pass.
* range() and enumerate() for iteration.
* Functions: Definition, parameters, return statements.
* Default parameters, keyword arguments, and docstrings.

**Practical (1 hour):**

* Write functions with default and keyword arguments.
* Use range() to generate sequences and enumerate() for indexing.
* Practice loop control with break and continue.

**Exercises:**

* Write a function to calculate the factorial of a number.
* Create a loop using enumerate() to print index-value pairs of a list.

**Quiz:**

* What is the purpose of break in a loop?
* What is a docstring, and why is it useful?

## **Day 7: Python Basics 2 - Advanced Function Concepts**

**Theory (1 hour):**

* \*args and \*\*kwargs: Variable-length arguments.
* Scope: Local, global, and nonlocal variables.
* Methods vs. functions: Object-oriented vs. standalone.

**Practical (1 hour):**

* Write a function using \*args to sum multiple numbers.
* Create a function with \*\*kwargs to process key-value pairs.
* Experiment with variable scope in nested functions.

**Exercises:**

* Write a function that accepts any number of arguments and returns their product.
* Create a program to demonstrate global vs. local variable scope.

**Quiz:**

* What does \*args do in a function definition?
* Explain the difference between local and global scope.

**Minor Project 2:**

* Develop a simple calculator program using functions to perform basic operations (add, subtract, multiply, divide) with user input.

## **Day 8: Advanced Python - Object-Oriented Programming (OOP) Basics**

**Theory (1 hour):**

* What is OOP? Classes and objects.
* \_\_init\_\_: Constructor method.
* Instance methods and attributes.

**Practical (1 hour):**

* Create a simple class (e.g., Student) with attributes and methods.
* Instantiate objects and call methods.
* Practice modifying object attributes.

**Exercises:**

* Write a class to represent a Car with attributes like model and speed.
* Create objects of the Car class and call a method to display details.

**Quiz:**

* What is the purpose of \_\_init\_\_ in a class?
* What is the difference between a class and an object?

## **Day 9: Advanced Python - OOP Concepts**

**Theory (1 hour):**

* Four pillars of OOP: Encapsulation, abstraction, inheritance, polymorphism.
* @classmethod and @staticmethod: Use cases.
* super(): Accessing parent class methods.

**Practical (1 hour):**

* Create a parent class and a child class to demonstrate inheritance.
* Use @classmethod to define a class-level method.
* Implement polymorphism with method overriding.

**Exercises:**

* Write a program with a parent class Animal and child classes Dog and Cat.
* Create a @staticmethod to perform a utility task (e.g., unit conversion).

**Quiz:**

* What is encapsulation in OOP?
* How does super() work in inheritance?

## **Day 10: Advanced Python - OOP Advanced**

**Theory (1 hour):**

* Dunder methods: \_\_str\_\_, \_\_repr\_\_, \_\_add\_\_, etc.
* Multiple inheritance: Combining multiple parent classes.

**Practical (1 hour):**

* Implement dunder methods for a class (e.g., \_\_str\_\_ for readable output).
* Create a class with multiple inheritance and test method resolution.
* Practice customizing object behavior with dunder methods.

**Exercises:**

* Write a class with a \_\_str\_\_ method to format output.
* Create a class inheriting from two parent classes.

**Quiz:**

* What is a dunder method? Give an example.
* What is the method resolution order in multiple inheritance?

**Minor Project 3:**

* Build a bank account management system using classes to handle deposits, withdrawals, and balance inquiries.

## **Day 11: Advanced Python - Useful Functions**

**Theory (1 hour):**

* Pure functions: Characteristics and benefits.
* map(), filter(), zip(), and reduce(): Functional programming tools.
* Lambda expressions: Anonymous functions.

**Practical (1 hour):**

* Use map() to transform a list (e.g., square numbers).
* Apply filter() to select elements based on a condition.
* Practice zip() and lambda expressions for concise operations.

**Exercises:**

* Write a program using map() to double all numbers in a list.
* Use filter() to extract even numbers from a list.

**Quiz:**

* What makes a function pure?
* What is the syntax for a lambda expression?

## **Day 12: Decorators and Generators**

**Theory (1 hour):**

* Decorators: Enhancing function behavior.
* Generators: yield keyword and lazy evaluation.

**Practical (1 hour):**

* Write a simple decorator to log function calls.
* Create a generator function to produce a sequence (e.g., Fibonacci).
* Practice using generators with for loops.

**Exercises:**

* Write a decorator to measure the execution time of a function.
* Create a generator for the first n prime numbers.

**Quiz:**

* What is the purpose of a decorator?
* How does yield differ from return?

## **Day 13: Error Handling and Debugging**

**Theory (1 hour):**

* Error handling: try, except, else, finally.
* Debugging: Using print statements and IDE debuggers.

**Practical (1 hour):**

* Write a program with try-except to handle division by zero.
* Practice debugging a program with intentional errors using an IDE.
* Use raise to create custom exceptions.

**Exercises:**

* Write a program to handle invalid user input with try-except.
* Debug a program to find and fix a logical error.

**Quiz:**

* What is the purpose of the finally block?
* Name two debugging techniques in Python.

## **Day 14: Modules, File I/O, and Logging**

**Theory (1 hour):**

* Modules: Importing custom, standard, and third-party modules.
* File I/O: Reading/writing .txt, .json, .csv files.
* Logging: Setting up basic logging for debugging.

**Practical (1 hour):**

* Create and import a custom module.
* Read and write data to a .txt and .csv file.
* Set up logging to record program events.

**Exercises:**

* Write a program to read a .csv file and print its contents.
* Create a module with a function and import it into another program.

**Quiz:**

* How do you import a specific function from a module?
* What is the difference between open() modes r and w?

**Minor Project 4:**

* Develop a program to read student grades from a .csv file, calculate averages, and log results to a file.

## **Day 15: Regular Expressions, Testing, and Career in Python**

**Theory (1 hour):**

* Regular expressions: Basics of pattern matching with re module.
* Testing: Introduction to unittest for automated testing.
* Career in Python: Roles (e.g., web developer, data scientist), learning resources.

**Practical (1 hour):**

* Write a regex to validate email addresses.
* Create a simple unittest test case for a function.
* Discuss and explore Python-related job opportunities online.

**Exercises:**

* Write a regex to extract phone numbers from text.
* Create a test case for a function that checks if a number is prime.

**Quiz:**

* What does the re module do in Python?
* What is the purpose of unit testing?

**Bonus Topics (Integrated Throughout or as Homework):**

* **Introduction to Data Science (Days 13-14):**
  + Use numpy for array operations, pandas for data manipulation, and matplotlib for plotting.
  + Exercise: Create a simple line plot of student grades using matplotlib.
* **Introduction to Machine Learning (Day 15):**
  + Overview of scikit-learn and a simple linear regression model.
  + Exercise: Build a basic ML model to predict house prices (dataset provided).
* **Streamlit and FastAPI (Day 15):**
  + Create a simple Streamlit app to display data.
  + Build a basic FastAPI endpoint to return JSON data.
  + Exercise: Develop a Streamlit app to visualize a dataset.

**Final Project:**

* Develop a student management system with a Streamlit interface, file I/O for data storage, and basic regex validation for input. Include unit tests and logging.