Python Programming Course: From Basics to Advanced

Module 1: Introduction to Python and Setup

Objective:

Understand what Python is, how to install it, and how to write your first Python program.

1. What is Python?

- **History and Features**: Python is a high-level, interpreted programming language developed by Guido van Rossum and first released in 1991. It is known for its simplicity and readability. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

- **Python Use Cases**: Python is used in web development, data analysis, artificial intelligence, machine learning, automation, scientific computing, and more.

- **Python vs Other Languages**:

  - Python is known for its readability and simplicity compared to languages like Java, C++, or JavaScript.
  - It has an extensive standard library, making it versatile for various tasks.

2. Setting Up Python

- **Installing Python**: Install Python from the official Python website. The installation is straightforward for Windows, macOS, and Linux.

- **Installing an IDE**:

  - **VSCode**: Install from Visual Studio Code.
  - **PyCharm**: Install from PyCharm.
  - **Jupyter Notebook**: Ideal for data science, install via `pip install notebook`.

- **Python Interpreter**: Python comes with an IDLE (Python's default interactive shell), or you can use the REPL (Read-Eval-Print Loop) in the terminal.

3. Your First Python Program

- **Writing Hello, World! Program**:

  ```
  print("Hello, World!")
  ```

  Running Python Programs: Run programs through the command line or in an IDE by pressing Ctrl+Enter (in VSCode) or using the Run button in PyCharm.

4. Basic Syntax

Python Keywords: Reserved words in Python (e.g., if, else, for, import).

Indentation and Syntax Rules: Python uses indentation (spaces or tabs) to define blocks of code.

Comments:

Single-line comments: # This is a comment. Multi-line comments:

```
"""
This is a
multi-line comment
"""
```

Exercise: Write a Python program that prints your name and age:

```
print("My name is John Doe")
print("I am 25 years old")
```

Module 2: Variables, Data Types, and Operators

Objective:

Learn about variables, data types, and how to manipulate data using operators.

1. Variables

Naming Conventions: Variable names should start with a letter or underscore, followed by letters, numbers, or underscores.

Assigning Values:

```
name = "Alice"
age = 30 2.
```

2. Data Types

Integers, Floats, Strings:

```
int_var = 10
float_var = 3.14
str_var = "Hello"
```

Lists, Tuples, Dictionaries, Sets:

- List: Ordered, mutable collection.

```
my_list = [1, 2, 3]
```

- Tuple: Ordered, immutable collection.

```
my_tuple = (1, 2, 3)
```

- Dictionary: Key-value pair collection.

```
my_dict = {"name": "Alice", "age": 30}
```

- Set: Unordered collection without duplicates.

```
my_set = {1, 2, 3}
```

Boolean:

```
is_active = True
```

Type Casting:

```
x = 5
y = str(x) # Converts integer to string 3.
```

3. Operators

- Arithmetic Operators:

```
a = 10
b = 5
print(a + b) # 15
print(a - b) # 5
```

- Comparison Operators:

```
print(a == b) # False
```

- Logical Operators:

```
print(True and False) # False
```

- Assignment Operators:

```
a += 5 # a = a + 5
```

- Membership Operators:

```
my_list = [1, 2, 3]
print(2 in my_list) # True
```

Exercise: Create a simple calculator using arithmetic operators and check if a number is prime.

Module 3: Control Flow (Conditionals and Loops)

Objective:

Understand conditional statements and how to work with loops in Python.

1. If-Else Statements

- If, Else, Elif:

```
if age > 18:
print("Adult")
else:
print("Not an adult")
```

- Nested Conditionals:

```
if age > 18:
if age > 21:
print("Adult over 21")
else:
print("Adult under 21")
```

2. Loops

- For Loop:

```
for i in range(5):
print(i)
```

- While Loop:

```
while x < 5:
x += 1
```

- Break, Continue, Else with Loops:

```
for i in range(5):
if i == 3:
break
print(i)
```

- Comprehensions:

```
squares = [x**2 for x in range(5)]
```

Exercise: Write a program to print Fibonacci sequence and check if a number is even or odd.

Module 4: Functions

Objective: Understand how to define and use functions in Python.

1. Defining Functions Syntax:

```
def greet(name):
return "Hello " + name
```

- Parameters and Return Values:

```
def add(a, b):
return a + b
```

- Function Scope: Variables inside a function are local to that function.

2. Lambda Functions

- Anonymous Functions:

```
square = lambda x: x\*\*2
```

3. Recursion

- Recursive Functions:

```
def factorial(n):
if n == 0:
return 1
return n \* factorial(n - 1)
```

Exercise: Create a function that returns the factorial of a number.

Module 5: Data Structures

Objective:

Learn about Python's built-in data structures and their operations.

1. Lists Creating and modifying lists:

```
my_list = [1, 2, 3]
my_list.append(4)
```

2. Tuples Tuple immutability:

```
my_tuple = (1, 2, 3)
```

3. Dictionaries Key-value pairs:

```
my_dict = {"name": "Alice", "age": 30}
```

4. Sets Set operations:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 & set2) # Intersection
```

Exercise: Create a dictionary for contact information and find intersection of two lists.

Module 6: File Handling

Objective: Learn how to work with files (reading, writing, and manipulating).

1. Opening Files

- Using open():

```
  file = open("file.txt", "r")
  content = file.read()
```

2. Reading and Writing Files

- Reading and writing:

```
with open("file.txt", "w") as file:
file.write("Hello World")
```

Exercise: Write a program to read a file and count word occurrences.

Module 7: Error Handling and Exceptions

Objective:

Learn how to handle errors and exceptions in Python.

1. Types of Errors

- Syntax, Runtime, and Logical Errors

2. Exception Handling

- Try-Except Block:

```
try:
x = 1 / 0
except ZeroDivisionError:
print("Cannot divide by zero")
```

Exercise: Handle divide-by-zero errors and prompt the user.

Module 8: Object-Oriented Programming (OOP)

Objective:

Understand core OOP concepts.

1. Classes and Objects

```
class Car:
def **init**(self, model, color):
self.model = model
self.color = color
```

2. Inheritance

```
class ElectricCar(Car):
def **init**(self, model, color, battery_size):
super().**init**(model, color)
self.battery_size = battery_size
```

Exercise: Create a Car class and an ElectricCar subclass.

Module 9: Libraries and Modules

Objective:

Understand how to use and create Python libraries.

1. Importing Libraries

```
import math
```

2. Creating Modules

- Create my_module.py with functions, then import them in your main file:

```
# my_module.py
def add(a, b):
return a + b
```

Module 10: Advanced Topics

Objective: Explore advanced Python topics.

1. Generators

```
def my_generator():
yield 1
yield 2
```

2. Decorators

```
def my_decorator(func):
def wrapper():
print("Before function")
func()
print("After function")
return wrapper
```

3. Concurrency

- Threading: Use threads for parallelism.
- Asyncio: For asynchronous programming.

Exercise: Create a decorator to log function execution time.

Module 11: Final Project

Objective:

Apply the learned concepts in real-world projects.

Project Ideas:

- Build a text-based adventure game.
- Develop a to-do list application.
- Create a weather application using an API.