**Experiment No. 1: Write a Python Program for Classes and Objects**

**🧪 Aim:**

To understand the concept of object-oriented programming (OOP) in Python and implement a simple Python program using **classes and objects**.

**🎯 Objective:**

* Learn how to define and use classes and objects in Python.
* Understand the use of constructors (\_\_init\_\_() method).
* Implement data encapsulation using instance variables.
* Demonstrate behavior through methods inside the class.

**📚 Theory / Concepts:**

**🔹 What is Object-Oriented Programming (OOP)?**

Object-Oriented Programming is a programming paradigm that uses **objects and classes** to structure software. Each object represents a real-world entity with **data (attributes)** and **behavior (methods)**. OOP focuses on code **reusability**, **modularity**, and **extensibility**.

Python is a fully object-oriented language, meaning everything in Python is an object (even functions, classes, etc.).

**🔹 Key OOP Concepts in Python:**

1. **Class:**  
   A class is a **blueprint** for creating objects. It defines attributes and behaviors.

class Student:

# attributes and methods go here

1. **Object:**  
   An object is an **instance** of a class. You can create multiple objects from the same class.

s1 = Student()

1. **Constructor (\_\_init\_\_() method):**  
   A special method in Python used to initialize object attributes when the object is created.

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

1. **Instance Variables:**  
   These are variables that are specific to each object.

self.name = name

1. **Methods:**  
   Functions defined inside a class that operate on object data.

def display(self):

print("Name:", self.name)

1. **Encapsulation:**  
   The practice of keeping data (attributes) and code (methods) that manipulates the data bundled together.

**🔹 Benefits of OOP in Python:**

* **Modularity:** Code is organized into separate classes.
* **Reusability:** Classes can be reused across projects.
* **Scalability:** Easy to manage and extend codebase.
* **Real-world mapping:** Classes and objects represent real-world entities.

**📋 Algorithm / Logic:**

1. Define a class named Student.
2. Use the \_\_init\_\_() constructor to initialize name, roll number, and marks.
3. Create a method display() to print the student details.
4. Create objects of the class.
5. Call the method using the object to display data.

**✅ Result:**

The concept of classes and objects was successfully implemented using a Student class. The attributes were initialized using a constructor, and the method was used to display the information of different student objects.

**❓ Viva Questions:**

1. What is a class in Python?
2. How is an object created from a class?
3. What is the purpose of the \_\_init\_\_() method?
4. What is the difference between a class and an object?
5. What do you mean by encapsulation?
6. How many objects can be created from a class?
7. What is self in Python classes?
8. Can we have multiple constructors in Python?
9. What happens if you do not define an \_\_init\_\_() method?
10. How do you call a method of a class?