**Lesson 12:OOP**

**Research Work**

**1.** **What is the difference between a class and an object in Python?**

In Python, a **class** is a blueprint used to define the structure and behavior of objects, specifying their attributes and methods. An **object** is an instance of a class that holds actual data and can perform actions defined by the class. You can create many objects from a single class, each with its own unique values. In short, classes define behavior, while objects represent real instances with data.

**2.** **Explain encapsulation and data hiding with code example.**

Encapsulation in Python means wrapping data and methods into a single class, keeping them organized and secure. Data hiding is a part of encapsulation where sensitive data is made private by using name mangling (like \_\_attribute). This prevents direct access from outside the class, enforcing control through methods. Together, they protect an object's internal state and promote cleaner, safer code.

**3.** **What is inheritance and how is it implemented in Python?**

Inheritance in Python allows one class (child) to inherit attributes and methods from another class (parent), enabling code reuse and organization. It is implemented by passing the parent class name in parentheses when defining the child class. The child class can access or override the parent class's methods and properties. This helps create a clear and efficient class hierarchy in programs.

**4.** **What is method overriding in subclasses?**

Method overriding in Python occurs when a subclass defines a method with the same name as one in its parent class. This allows the subclass to provide a specific implementation that replaces the parent's version. When the method is called on a child object, the overridden method runs instead of the parent's. It helps customize behavior and supports polymorphism in object-oriented programming.

**5.** **Why is self required in instance methods?**

In Python, self is required in instance methods to **refer to the specific object** calling the method. It allows access to the object's attributes and other methods within the class. Without self, Python wouldn't know which instance's data to use or modify. It ensures each object maintains its own state and behaves independently.