Practical Lesson: Object-Oriented Programming in Python

Learning Objectives:

By the end of this practical lesson, students will:

1. Understand the concepts of public, protected, and private members in Python.
2. Write Python programs demonstrating the use of public, protected, and private access modifiers.
3. Implement properties and property decorators in classes.

Program 1: Public Members

Task: Create a `Student` class with public attributes and demonstrate access.

Code:

```
class Student:
    schoolName = 'XYZ School'  # public class attribute

    def __init__(self, name, age):
        self.name = name  # public instance attribute
        self.age = age    # public instance attribute

# Create an instance of Student
std = Student("Steve", 25)

# Access and modify public attributes
print(std.schoolName)  # Output: XYZ School
print(std.name)        # Output: Steve
std.age = 20
print(std.age)         # Output: 20
```

Output:

```
XYZ School
Steve
20
```

Program 2: Private Members

Task: Use private attributes and access them through name mangling.

Code:

```
class Student:
    __schoolName = 'XYZ School'  # private class attribute

    def __init__(self, name, age):
        self.__name = name  # private instance attribute
        self.__age = age    # private instance attribute

# Create an instance of Student
std = Student("Bill", 25)

# Access private attributes using name mangling
print(std._Student__name)  # Output: Bill
print(std._Student__age)   # Output: 25
```

Output:

```
Bill
25
```

Program 3: Using Property Decorators

Task: Use `@property` and `@setter` to create getters and setters for protected attributes.

Code:

```
class Student:
    def __init__(self, name):
        self._name = name  # protected instance attribute

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, newname):
        self._name = newname

# Create an instance of Student
std = Student("Swati")
```

```
# Use property to get and set values
print(std.name)  # Output: Swati
std.name = "Dipa"
print(std.name)  # Output: Dipa
```

Output:

```
Swati
Dipa
```

Program 4: Practical Use Case - Employee Management System

Task: Create an `Employee` class that uses public, protected, and private members.

Code:

```
class Employee:
    company = "ABC Corp"  # public class attribute

    def __init__(self, name, salary):
        self._name = name        # protected instance attribute
        self.__salary = salary  # private instance attribute

    @property
    def salary(self):
        return self.__salary

    @salary.setter
    def salary(self, newsalary):
        if newsalary > 0:
            self.__salary = newsalary
        else:
            print("Invalid salary!")

# Create an instance of Employee
emp = Employee("John", 5000)

# Access protected and private members
print(emp._name)           # Output: John
print(emp.salary)          # Output: 5000
emp.salary = 6000          # Update salary
print(emp.salary)          # Output: 6000
emp.salary = -1000         # Invalid salary update
```

Output:

```
John
5000
6000
Invalid salary!
```

Lab Activities

1. **Activity 1:** Modify `Program 5` to include a method that calculates the annual bonus based on the employee's salary.
2. **Activity 2:** Create a `Car` class with attributes for brand, model, and price. Use public, protected, and private members to manage access.
3. **Activity 3:** Extend `Program 3` by adding a private method that prints a message containing the student’s name and age.