LAB 1:

Introduction to Classes and Objects

Objective:

- Understand the concept of classes and objects in Python.
- Create and use class objects.

Theory:

In object-oriented programming (OOP), a class is a blueprint for creating objects. An object is an instance of a class. The class defines the properties (attributes) and behaviors (methods) that the objects of that class will have.

For example, consider a class called **Rectangle**. The **Rectangle** class can have attributes like width and height to represent the dimensions of a rectangle. The class can also have methods like **calculate_area** to perform calculations related to rectangles.

Tasks:

1. Create a class called **Rectangle** with attributes **length** and **width**. Define a constructor to initialize the values of these attributes.
    - Explanation: In this task, you need to create a class called **Rectangle** that represents a rectangle object. It should have two attributes, **length** and **width**, which will hold the dimensions of the rectangle. You should define a constructor to initialize these attributes when a new object is created.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
```

2. Implement methods in the **Rectangle** class to calculate the area and perimeter of a rectangle.
    - Explanation: In this task, you need to implement two methods in the **Rectangle** class:

- **calculate_area()**: This method should calculate and return the area of the rectangle using the formula **area = length * width**.
- **calculate_perimeter()**: This method should calculate and return the perimeter of the rectangle using the formula **perimeter = 2 * (length + width)**.

```
def calculate_area(self):
        return self.length * self.width


    def calculate_perimeter(self):
        return 2 * (self.length + self.width)
```

3. Create an object of the **Rectangle** class and test the area and perimeter calculation methods.
    - Explanation: In this task, you should create an object of the **Rectangle** class and test the implemented methods:
        - Create a new **Rectangle** object with specific values for **length** and **width**.
        - Use the **calculate_area()** method to calculate the area of the rectangle and print the result.
        - Use the **calculate_perimeter()** method to calculate the perimeter of the rectangle and print the result.

```
rect = Rectangle(5, 3)
print("Area:", rect.calculate_area())
print("Perimeter:", rect.calculate_perimeter())
```

LAB 2:

Inheritance and Polymorphism

Objective:

- Learn about inheritance and polymorphism in Python.
- Understand the concepts of base classes, derived classes, and method overriding.

Theory:

Inheritance is a fundamental concept in OOP that allows you to create a new class (derived class) from an existing class (base class). The derived class inherits the attributes and methods of the base class and can add its own attributes and methods.

Tasks:

1. Create a base class called **Animal** with attributes **name** and **age**. Implement a constructor to initialize these attributes.
    - Explanation: In this task, you need to create a base class called **Animal** that will serve as the parent class for specific types of animals. The **Animal** class should have two attributes, **name** and **age**, to store the name and age of the animal. Implement a constructor to initialize these attributes when an **Animal** object is created.

```
class Animal:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

2. Create a derived class called **Dog** that inherits from the **Animal** class. Add an additional attribute **breed** to the **Dog** class and implement a constructor to initialize it.

- Explanation: In this task, you should create a derived class called **Dog** that inherits from the **Animal** class. The **Dog** class should have an additional attribute called **breed** to store the breed of the dog. Implement a constructor in the **Dog** class to initialize the **name**, **age**, and **breed** attributes.

```
class Dog(Animal):
    def __init__(self, name, age, breed):
        super().__init__(name, age)
        self.breed = breed
```

3. Override the **__str__** method in both the **Animal** and **Dog** classes to display relevant information.
    - Explanation: In this task, you need to override the **__str__** method in both the **Animal** and **Dog** classes to provide a string representation of the objects. The overridden **__str__** method should return a formatted string that includes relevant information about the animal or dog, such as its name, age, and breed.

```
def __str__(self):
    return f"Dog: {self.name}, Age: {self.age}, Breed: {self.breed}"
```

4. Create objects of both the **Animal** and **Dog** classes and test the overridden **__str__** method.
    - Explanation: In this task, you should create objects of both the **Animal** and **Dog** classes and test the overridden **__str__** method:
        - Create an **Animal** object with a name and age.

- Create a **Dog** object with a name, age, and breed.
- Print the objects, and the overridden **__str__** method should display the relevant information.

```
animal = Animal("Max", 5)
print(animal)


dog = Dog("Buddy", 3, "Labrador")
print(dog)
```

Lab 3: Encapsulation and Abstraction

Objective:

- Understand encapsulation and abstraction in Python.
- Implement encapsulation using access modifiers.
- Demonstrate abstraction using abstract classes and interfaces.

Theory:

Encapsulation is an OOP principle that bundles the data (attributes) and methods (behaviors) together within a class and restricts direct access to the data from outside the class. Access modifiers define the level of access to class members (attributes and methods).

Access modifiers are used to restrict access to the variables and methods of the class.

Tasks:

1. Create a class called **Employee** with private attributes **name** and **salary**. Implement getter and setter methods for these attributes.
    - Explanation: In this task, you need to create a class called **Employee** that represents an employee. The **Employee** class should have private attributes, **name** and **salary**, to store the name and salary of the employee. Implement getter and setter methods to access and modify

```
class Employee:
    def __init__(self):
        self.__name = None
        self.__salary = None
    def get_name(self):
        return self.__name
    def set_name(self, name):
        self.__name = name
    def get_salary(self):
        return self.__salary
    def set_salary(self, salary):
        self.__salary = salary
```

2. Create an abstract class called **Shape** with an abstract method **calculate_area()**. Implement two derived classes, **Rectangle** and **Circle**, that inherit from the **Shape** class and implement the **calculate_area()** method.

    - Explanation: In this task, you should create an abstract class called **Shape** that defines the concept of a shape and has an abstract method called **calculate_area()**. Implement two derived classes, **Rectangle** and **Circle**, that inherit from the **Shape** class. Each derived class should implement the **calculate_area()** method according to its specific shape formula.

```
from abc import ABC, abstractmethod


class Shape(ABC):
    @abstractmethod
    def calculate_area(self):
        pass
```

3. Create objects of the **Rectangle** and **Circle** classes and test the **calculate_area()** method.

    - Explanation: In this task, you should create objects of the **Rectangle** and **Circle** classes and test the **calculate_area()** method:
        - Create a **Rectangle** object with specific dimensions and call the **calculate_area()** method to calculate and print the area.
        - Create a **Circle** object with a specific radius and call the **calculate_area()** method to calculate and print the area.

```
class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return 3.14 * self.radius ** 2
```

Lab 4: Polymorphism and Method Overloading

Objective:

- Explore polymorphism and method overloading in Python.
- Understand how to implement polymorphism using inheritance and method overriding.
- Demonstrate method overloading using default arguments.

Theory:

Polymorphism allows objects of different classes to be treated as objects of a common base class. Method overloading is a form of polymorphism where multiple methods in a class have the same name but different parameters.

Tasks:

1. Create a base class called **Shape** with an abstract method **calculate_area()**. Implement two derived classes, **Rectangle** and **Circle**, that inherit from the **Shape** class and implement the **calculate_area()** method.
    - Explanation: In this task, you should create a base class called **Shape** that has an abstract method **calculate_area()**. Implement two derived classes, **Rectangle** and **Circle**, that inherit from the **Shape** class. Each derived class should implement the **calculate_area()** method according to its specific shape formula.
2. Create a class called **Calculator** with a method called **add()** that can add two numbers or concatenate two strings based on the arguments passed.
    - Explanation: In this task, you need to create a class called **Calculator** that has a method called **add()**. The **add()** method should be capable of performing addition if given two numbers as arguments or concatenation if given two strings as arguments. Implement the logic to check the types of the arguments and perform the appropriate operation.
3. Create objects of the **Rectangle**, **Circle**, and **Calculator** classes and test the polymorphic behavior and method overloading.
    - Explanation: In this task, you should create objects of the **Rectangle**, **Circle**, and **Calculator** classes and test the polymorphic behavior and method overloading:
        - Create a **Rectangle** object and a **Circle** object.

- Call the **calculate_area()** method on both objects and print the results, demonstrating polymorphism.
- Create a **Calculator** object and call the **add()** method with different types of arguments (numbers and strings), testing the method overloading behavior.

```
# Lab 4: Polymorphism and Method Overloading
class Shape:
    def calculate_area(self):
        pass

class Rectangle(Shape):
    def calculate_area(self, width, height):
        return width * height

class Circle(Shape):
    def calculate_area(self, radius):
        return 3.14 * radius * radius

# Test the Shape, Rectangle, and Circle classes
rectangle = Rectangle()
rectangle_area = rectangle.calculate_area(5, 3)
print("Rectangle Area:", rectangle_area)

circle = Circle()
circle_area = circle.calculate_area(2)
print("Circle Area:", circle_area)
```