Laboratory Activity No. 7	
Polymorphism	
Course Code: CPE103	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:** 02/22/25
Section: BSCPE 1-A	**Date Submitted:** 02/22/25
Name: PALMES, LEWIS CLARK L.	**Instructor:** ENGR. MARIA RIZETTE SAYO

1. Objective(s):

This activity aims to familiarize students with the concepts of Polymorphism in Object-Oriented Programming

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Identify the use of Polymorphism in Object-Oriented Programming
2.2 Implement an Object-Oriented Program that applies Polymorphism

3. Discussion:

Polymorphism is a core principle of Object-Oriented that is also called "method overriding". Simply stated the principles says that a method can be redefined to have a different behavior in different derived classees.

For an example, consider a base file reader/writer class then three derived classes Text file reader/writer, CSV file reader/ writer, and JSON file reader/writer. The base file reader/writer class has the methods: read(filepath="") , write(filepath=""). The three derived classes (classes that would inherit from the base class) should have behave differently when their read, write methods are invoked.

Operator Overloading:
Operator overloading is an important concept in object oriented programming. It is a type of polymorphism in which a user defined meaning can be given to an operator in addition to the predefined meaning for the operator.

Operator overloading allow us to redefine the way operator works for user-defined types such as objects. It cannot be used for built-in types such as int, float, char etc., For example, '+' operator can be overloaded to perform addition of two objects of distance class.

Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator on objects, the magic method ___add___() is automatically invoked in which the meaning/operation for + operator is defined for user defined objects.

4. Materials and Equipment:

Windows Operating System
Google Colab

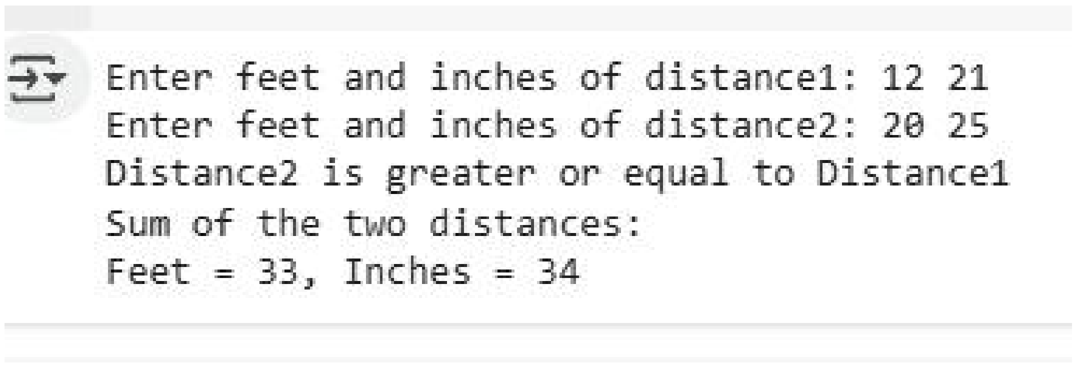
5. Procedure:

Creating the Classes

1. Create a folder named oopfa1<lastname>_lab8
2. Open your IDE in that folder.
3. Create the base polymorphism_a.ipynb file and Class using the code below:

```
Coding:
# distance is a class. Distance is measured in terms of feet and inches class
distance:
 def __init__(self, f,i):
 self.feet=f
 self.inches=i

# overloading of binary operator > to compare two distances def
 ______gt__(self,d):
 if(self.feet>d.feet):
 return(True)
 elif((self.feet==d.feet) and (self.inches>d.inches)): return(True)
 else:
 return(False)

# overloading of binary operator + to add two distances def
 ______add___(self, d):
 i=self.inches + d.inches
 f=self.feet + d.feet if(i>=12):
 i=i-12
 f=f+1
 return distance(f,i)

# displaying the distance def
 show(self):
 print("Feet= ", self.feet, "Inches= ",self.inches)

a,b= (input("Enter feet and inches of distance1: ")).split() a,b
=[int(a),int(b)]
c,d= (input("Enter feet and inches of distance2: ")).split() c,d
=[int(c),int(d)]
d1 = distance(a,b) d2
= distance(c,d)

if(d1>d2):
 print("Distance1 is greater than Distance2")
else:
 print("Distance2 is greater or equal to Distance1")
d3=d1+d2
print("Sum of the two Distance is:") d3.show()
```

4. Screenshot of the program output:



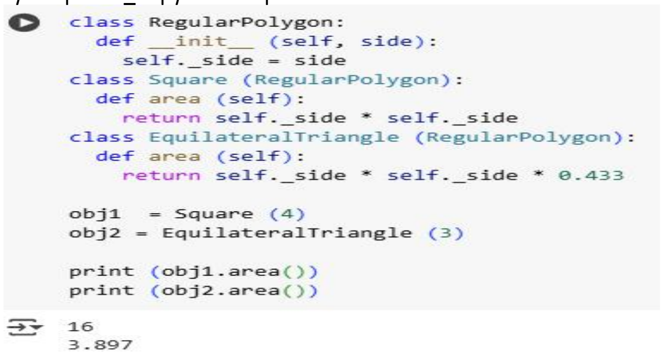
```
Enter feet and inches of distance1: 12 21
Enter feet and inches of distance2: 20 25
Distance2 is greater or equal to Distance1
Sum of the two distances:
Feet = 33, Inches = 34
```

Testing and Observing Polymorphism

1. Create a code that displays the program below:

```
class RegularPolygon:
    def __init__ (self, side):
        self._side = side
class Square (RegularPolygon):
    def area (self):
        return self._side * self._side
class EquilateralTriangle (RegularPolygon):
    def area (self):
        return self._side * self._side * 0.433

obj1 = Square(4)
obj2 = EquilateralTriangle(3)

print (obj1.area())
print (obj2.area())
```

2. Save the program as polymorphism_b.ipynb and paste the screenshot below:



```
class RegularPolygon:
  def __init__ (self, side):
    self._side = side
class Square (RegularPolygon):
  def area (self):
    return self._side * self._side
class EquilateralTriangle (RegularPolygon):
  def area (self):
    return self._side * self._side * 0.433

obj1  = Square (4)
obj2 = EquilateralTriangle (3)

print (obj1.area())
print (obj2.area())
```

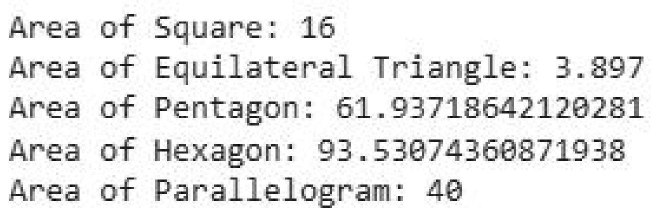
```
16
3.897
```

3. Run the program and observe the output.
4. Observation:

When running the program, I observed that both the Square and EquilateralTriangle classes inherit from the RegularPolygon base class. Despite sharing the same method name area(), each class calculates the area differently based on the shape. The Square class multiplies the side by itself, while the EquilateralTriangle class multiplies the side by itself and a constant (0.433). This demonstrates polymorphism, where the area() method behaves differently depending on the object's class.

6. Supplementary Activity:

In the above program of a Regular polygon, add three more shapes and solve for their area using each proper formula. Take a screenshot of each output and describe each by typing your proper labeling.



```
Area of Square: 16
Area of Equilateral Triangle: 3.897
Area of Pentagon: 61.93718642120281
Area of Hexagon: 93.53074360871938
Area of Parallelogram: 40
```

FOR THE PROGRAM PLEASE REFER TO THIS LINK IN GITHUB:
CPE-103-OOP-1-A/LAB_7.ipynb at main · Lewis-Clark-Palmes/CPE-103-OOP-1-A

1. **RegularPolygon** is the base class that initializes the side length of a regular polygon. It serves as the foundation for other polygon classes.
2. Square is a derived class that represents a square, where all sides are equal. The area is calculated using the formula side multiplied by side.
3. **EquilateralTriangle** is a derived class that represents an equilateral triangle, where all sides are equal. The area is approximated using the formula side squared multiplied by 0.433, which is derived from the square root of three divided by four.
4. **Pentagon** is a derived class that represents a regular pentagon, a five-sided polygon. The area is calculated using a mathematical formula involving the square root of five and its related terms.
5. **Hexagon** is a derived class that represents a regular hexagon, a six-sided polygon. The area is calculated using the formula three times the square root of three divided by two, multiplied by side squared.
6. **Parallelogram** is a derived class that represents a parallelogram, a quadrilateral with opposite sides parallel. It has additional attributes for base and height. The area is calculated using the formula base multiplied by height.

Each of these polygon classes is instantiated as objects, and the area of each shape is printed using the area method.

Questions

1. Why is Polymorphism important?

   Polymorphism allows one interface to handle different data types, providing flexibility and making code easier to manage. This enables developers to write cleaner, more maintainable, and scalable code.

2. Explain the advantages and disadvantages of using applying Polymorphism in an Object-Oriented Program.

   Advantages: Polymorphism makes code more flexible, reusable, and easier to maintain.
   Disadvantages: It can increase complexity and may introduce performance overhead due to dynamic method binding.

3. What maybe the advantage and disadvantage of the program we wrote to read and write csv and json files?

   Advantage: The program can handle different file types with the same code, improving versatility and reducing redundancy.
   Disadvantage: The added complexity might make the code harder to understand and maintain, and polymorphic behavior may slow down performance.

4. What maybe considered if Polymorphism is to be implemented in an Object-Oriented Program?

   Proper design patterns should be followed to keep code organized and maintainable. Performance impact and extensive testing are essential to ensure correct behavior, along with clear documentation for other developers.

5. How do you think Polymorphism is used in an actual programs that we use today?

   Polymorphism is used in GUIs to handle different user inputs and in data processing libraries like Pandas to manage various data formats. It's also utilized in game development for managing different entities like players, enemies, and objects.

7. Conclusion:

Polymorphism is a fundamental concept in object-oriented programming that enhances flexibility, reusability, and maintainability of code. While it may introduce complexity and potential performance overhead, its advantages in promoting a scalable and modular design far outweigh the drawbacks. Understanding and effectively implementing polymorphism is crucial for developing robust and efficient software systems.

8. Assessment Rubric: