Laboratory Activity No. 7	
Polymorphism	
Course Code: CPE103	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed: 02/22/2025**
Section: 1A	**Date Submitted: 02/22/2025**
Name: CATAHAN, JOSHUA A.	**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

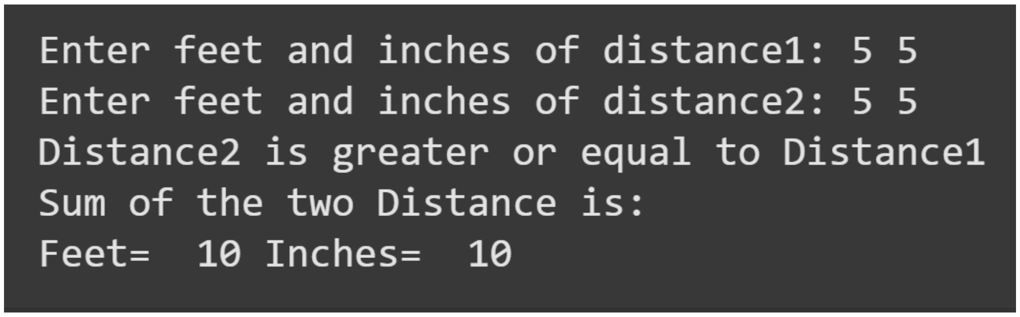
5. Procedure:

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:

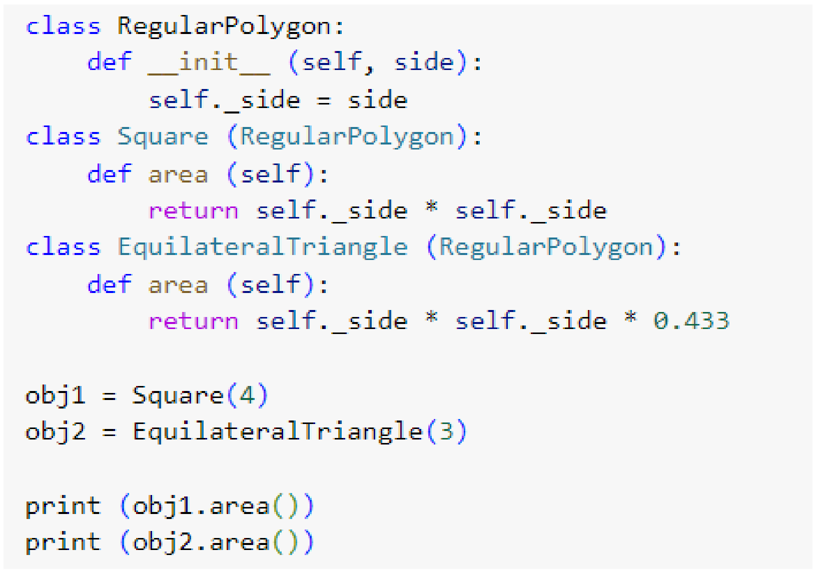


```
Enter feet and inches of distance1: 5 5
Enter feet and inches of distance2: 5 5
Distance2 is greater or equal to Distance1
Sum of the two Distance is:
Feet=  10 Inches=  10
```

https://colab.research.google.com/drive/15HpLXDVCZH2Tv1Gijgo9DA7jGzbf_pbz#scrollTo=rqFNTJ5Ibijy&line=39&uniqifier=1

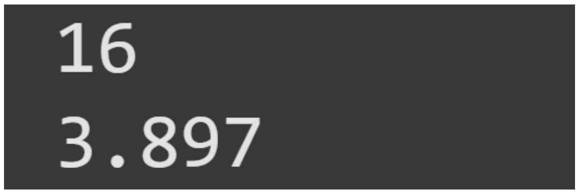
Testing and Observing Polymorphism

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:



```
16
3.897
```

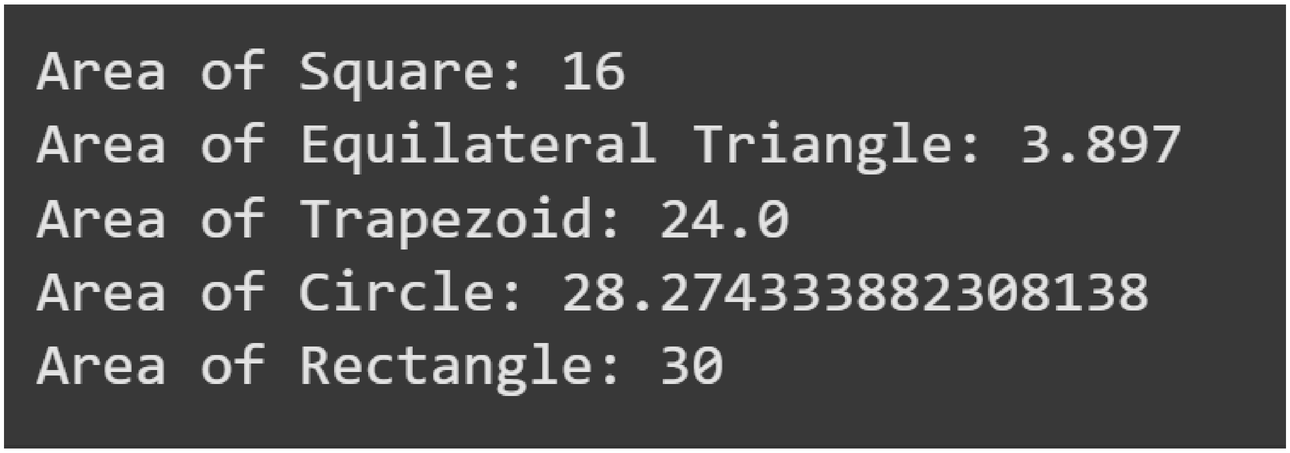
https://colab.research.google.com/drive/15HpLXDVCZH2Tv1Gijgo9DA7jGzbf_pbz#scrollTo=mxWVx6hjboKi&line=11&uniqifier=1

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

This code defines a simple class structure where RegularPolygon is the parent class, and both Square and EquilateralTriangle are child classes. The RegularPolygon class stores the side length of a shape. The Square class uses that side length to calculate the area of a square (side × side), while the EquilateralTriangle calculates the area of an equilateral triangle using the formula side^2 * 0.433. When the code runs, it creates two objects: a square with side length 4 and an equilateral triangle with side length 3. The area of the square is calculated as 16, and the area of the triangle is approximately 4.497. The result is printed for both shapes.

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 Trapezoid: 24.0
Area of Circle: 28.274333882308138
Area of Rectangle: 30
```

https://colab.research.google.com/drive/15HpLXDVCZH2Tv1Gijgo9DA7jGzbf_pbz#scrollTo=G91nHyO1bv6P&line=36&uniqifier=1

1. **Square: The area of a square is calculated by multiplying the side length by itself, using the formula side × side. With a side length of 4 units, the area is 4 × 4 = 16 square units. The Square class inherits from RegularPolygon and uses the side length (_side) passed during initialization (e.g., obj1 = Square(4)). The area() method calculates the area of the square by multiplying the side length by itself.**
2. **Equilateral Triangle: The area of an equilateral triangle is given by the formula side**2 × 0.433. For a side length of 3 units, the area is approximately 3 × 3 × 0.433 = 4.497 square units. The EquilateralTriangle class also inherits from RegularPolygon. It uses the side length (_side) passed during initialization (e.g., obj2 = EquilateralTriangle(3)). The area() method computes the area of the triangle based on the side length, assuming it's equilateral.**
3. **Trapezoid: The area of a trapezoid is calculated using the formula 0.5 × ($\mathrm{b1}$ + $\mathrm{b2}$) × h, where $\mathrm{b1}$ and $\mathrm{b2}$ are the lengths of the parallel sides, and h is the height. For parallel sides of 5 and 7 units, and a height of 4 units, the area is 24 square units. The Trapezoid class inherits from RegularPolygon, but it has additional attributes: b1, b2 (the two bases), and height (the perpendicular height). These values are passed during initialization (e.g., obj3 = Trapezoid(0, 5, 7, 4)), and the area() method calculates the area using these values, though the side (_side) from the base class is not directly used in the trapezoid's area calculation.**
4. **Circle: The area of a circle is found using the formula π × r**2, where r is the radius. For a radius of 3 units, the area is approximately 28.274 square units. The Circle class inherits from RegularPolygon and uses the**

radius (_side) passed during initialization (e.g., obj4 = Circle(3)). The area() method calculates the area of the circle based on the radius, using the formula for the area of a circle.

5. **Rectangle: The area of a rectangle is calculated by multiplying its two side lengths using the formula side1 × side2 . For side lengths of 5 and 6 units, the area is 30 square units. The Rectangle class does not directly inherit from RegularPolygon in a conventional way, as it uses two sides (_side1 and _side2) passed during initialization (e.g., obj5 = Rectangle(5, 6)). The area() method calculates the area by multiplying these two side lengths.**

Questions

1. Why is Polymorphism important?

Polymorphism is important because it allows different classes to use the same interface while implementing their specific behavior. This makes the code more flexible, reusable, and easier to maintain.

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

Advantages: Polymorphism makes the code more flexible and extensible, allowing for easier maintenance and modification. It enables the use of a common interface for different object types.
Disadvantages: It can make the code harder to understand, especially for beginners. It might also introduce performance issues if not used wisely.

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

Advantages: The program simplifies reading and writing data to files, making data storage and retrieval easier. It's also adaptable for different file formats (CSV and JSON).
Disadvantages: It may not handle errors or invalid data gracefully, which could lead to crashes or incorrect data processing.

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

When implementing polymorphism, consider how different classes will share the same method names but have different implementations. It's important to ensure that the classes make sense to be grouped together and that the behavior is logical and consistent.

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

Polymorphism is used in programs like media players, where the same "play" button can work for both music and video files. It's also used in graphical user interfaces (GUIs), where buttons and sliders can have different actions but are controlled with the same methods.

7. Conclusion:

In conclusion, polymorphism is a key concept in object-oriented programming that enhances flexibility, reusability, and maintainability by allowing different classes to implement the same methods in their own ways. While it can make code more complex and harder to understand, its benefits often outweigh the drawbacks, especially in large projects. In the context of our code, polymorphism allowed us to define different shapes with a common interface for calculating areas, streamlining the process. Programs that handle file reading and writing, like our CSV and JSON example, benefit from polymorphism by making it easier to work with various data formats, though they may need better error handling. Ultimately, understanding and applying polymorphism helps create more adaptable and efficient software.

8. Assessment Rubric: