

Student Name: _____ Roll No: _____

Section: _____

Lab Series No. 08

Lab 8 – Polymorphism in Object Oriented

Lab Objectives:

1. Introduction with Polymorphism
2. Polymorphism with Class Method
3. Polymorphism with a Function

1. Introduction with Polymorphism

Polymorphism is the ability to leverage the same interface for different underlying forms such as data types or classes. This permits functions to use entities of different types at different times. For object-oriented programming in Python, this means that a particular object belonging to a particular class can be used in the same way as if it were a different object belonging to a different class. Polymorphism allows for flexibility and loose coupling so that code can be extended and easily maintained overtime.

1.1 What Is Polymorphism?

Polymorphism is an important feature of class definition in Python that is utilized when you have commonly named methods across classes or subclasses. This allows functions to use objects of any of these polymorphic classes without needing to be aware of distinctions across the classes. Polymorphism can be carried out through inheritance, with subclasses making use of base class methods or overriding them.

Python's duck typing, a special case of dynamic typing, uses techniques characteristic of polymorphism, including late binding and dynamic dispatch. The term "duck typing" is derived from a quote of writer James Whitcomb Riley: "When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.", the use of duck typing is concerned with establishing the suitability of an object for a specific purpose.

When using normal typing this suitability is determined by the type of an object alone, but with duck typing the presence of methods and properties are used to determine suitability rather than the actual type of the

Student Name: _____

Roll No: _____

Section: _____

object in question. That is to say, you check whether the object quacks like a duck and walks like a duck rather than asking whether the object is a duck.

When several classes or subclasses have the same method names, but different implementations for these same methods, the classes are polymorphic because they are using a single interface to use with entities of different types. A function will be able to evaluate these polymorphic methods without knowing which classes are invoked.

2. Creating Polymorphic Classes

To make use of polymorphism, we're going to create two distinct classes to use with two distinct objects. Each of these distinct classes need to have an interface that is in common so that they can be used polymorphically, so we will give them methods that are distinct but that have the same name.

We'll create a Shark class and a Clownfish class, each of which will define methods for swim(), swim_backwards(), and skeleton().

Program 1: Create a python file name it as fish_polymorphic.py .

Code:

```
class Shark():
    def swim(self):
        print("The shark is swimming.")

    def swim_backwards(self):
        print("The shark cannot swim backwards, but can sink backwards.")

    def skeleton(self):
        print("The shark's skeleton is made of cartilage.")

class Clownfish():
    def swim(self):
        print("The clownfish is swimming.")
```

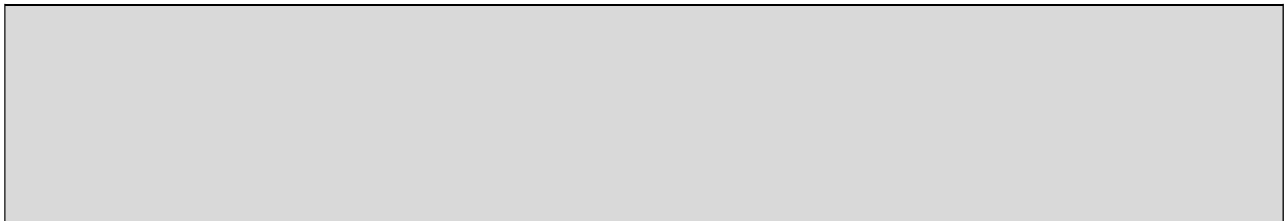
Student Name: _____ Roll No: _____ Section: _____

```
def swim_backwards(self):
    print("The clownfish can swim backwards.")

def skeleton(self):
    print("The clownfish's skeleton is made of bone.")

# initiate the classes into two objects:
...
Obj_shark = Shark()
Obj_shark.skeleton()

Obj_clownfish = Clownfish()
Obj_clownfish.skeleton()
```

Output:

3. Polymorphism with Class Methods

To show how Python can use each of these different class types in the same way, we can first create a for loop that iterates through a tuple of objects. Then we can call the methods without being concerned about which class type each object is. We will only assume that these methods actually exist in each class.

Program 2: Write a Python program to show the polymorphism with class methods.

```
# initiate the classes into two objects:
...
Obj_shark = Shark()
Obj_shark.skeleton()

Obj_clownfish = Clownfish()
Obj_clownfish.skeleton()

for fish in (obj_shark, obj_clownfish):
    fish.swim()
    fish.swim_backwards()
    fish.skeleton()
```

Student Name: _____

Roll No: _____

Section: _____

Output:

Program 3: Write a Python program to show the polymorphism with class methods in real world problem.

```
class AudioFile:
    def __init__(self, filename):
        if not filename.endswith(self.ext):
            raise Exception("Invalid file format")
        self.filename = filename
class MP3File(AudioFile):
    ext = "mp3"
    def play(self):
        print("playing {} as mp3".format(self.filename))
class WavFile(AudioFile):
    ext = "wav"
    def play(self):
        print("playing {} as wav".format(self.filename))
class OggFile(AudioFile):
    ext = "ogg"
    def play(self):
        print("playing {} as ogg".format(self.filename))

class FlacFile:
    def __init__(self, filename):
        if not filename.endswith(".flac"):
            raise Exception("Invalid file format")
        self.filename = filename
    def play(self):
        print("playing {} as flac".format(self.filename))
```

Output:

Student Name: _____

Roll No: _____

Section: _____

Program 4: Write a Python program to show the polymorphism with parrot class and penguin class use fly() methods to show polymorphism.

```
class Parrot:

    def fly(self):
        print("Parrot can fly")

    def swim(self):
        print("Parrot can't swim")

class Penguin:

    def fly(self):
        print("Penguin can't fly")

    def swim(self):
        print("Penguin can swim")

# common interface
def flying_test(bird):
    bird.fly()

#instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
flying_test(blu)
flying_test(peggy)
```

Output:

Student Name: _____

Roll No: _____

Section: _____

Program 5: Write a Python program to show the how we can remove the duplicate code.

```
class Rectangle:
    def __init__(self, color, filled, width, length):
        self._color = color
        self._filled = filled
        self._width = width
        self._length = length

    def get_color(self):
        return self._color

    def set_color(self, color):
        return self._color = color

    def is_filled(self):
        return self.__filled

    def set_filled(self, filled):
        return self.__filled

    def get_area():
        return self._width * self._length

class Circle:
    def __init__(self, color, filled, radius):
        self._color = color
        self._filled = filled
        self._radius = radius

    def get_color(self):
        return self._color

    def set_color(self, color):
        return self._color = color

    def is_filled(self):
        return self.__filled

    def set_filled(self, filled):
        return self.__filled

    def get_area(self):
```

Student Name: _____ Roll No: _____
return math.pi * self._radius ** 2

Section: _____

Output:

Program 6: Write a Python program to correct Program 5.

```
import math

class Shape:

    def __init__(self, color='red', filled=False):
        self._color = color
        self._filled = filled

    def get_color(self):
        return self.__color

    def set_color(self, color):
        self._color = color

    def get_filled(self):
        return self.__filled

    def set_filled(self, filled):
        self._filled = filled

class Rectangle(Shape):

    def __init__(self, length, breadth):
        super().__init__()
        self._length = length
        self._breadth = breadth

    def get_length(self):
        return self.__length

    def set_length(self, length):
```

Student Name: _____ Roll No: _____

Section: _____

```
        self._length = length

    def get_breadth(self):
        return self.__breadth

    def set_breadth(self, breadth):
        self._breadth = breadth

    def get_area(self):
        return self._length * self.__breadth

    def get_perimeter(self):
        return 2 * (self._length + self.__breadth)

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

    def get_radius(self):
        return self.__radius

    def set_radius(self, radius):
        self._radius = radius

    def get_area(self):
        return math.pi * self._radius ** 2

    def get_perimeter(self):
        return 2 * math.pi * self.__radius

r1 = Rectangle(13.5, 4.5)

print("Area of rectangle r1:", r1.get_area())
print("Perimeter of rectangle r1:", r1.get_perimeter())
print("Color of rectangle r1:", r1.get_color())
print("Is rectangle r1 filled ? ", r1.get_filled())
r1.set_filled(True)
print("Is rectangle r1 filled ? ", r1.get_filled())
r1.set_color("orange")
print("Color of rectangle r1:", r1.get_color())
```


Student Name: _____

Roll No: _____

Section: _____

```
c1 = Circle(12)
```

```
print("\nArea of circle c1:", format(c1.get_area(), "0.2f"))
print("Perimeter of circle c1:", format(c1.get_perimeter(), "0.2f"))
print("Color of circle c1:", c1.get_color())
print("Is circle c1 filled ? ", c1.get_filled())
c1.set_filled(True)
print("Is circle c1 filled ? ", c1.get_filled())
c1.set_color("blue")
print("Color of circle c1:", c1.get_color())
```

Output:

4. Polymorphism with a Function

We create two classes: Bear and Dog, both can make a distinct sound. We then make two instances and call their action using the same method. It is same as the previous one but as a student you know for your interview that both as same.

Program 7: Write a Python program to show polymorphism using function.

```
class Bear(object):
    def sound(self):
        print("Groarr")

class Dog(object):
    def sound(self):
        print("Woof woof!")

def makeSound(animalType):
    animalType.sound()

bearObj = Bear()
dogObj = Dog()

makeSound(bearObj)
```

StudentName: _____
makeSound (dogObj)

Roll No: _____

Section: _____

Output:

StudentName: _____

Roll No: _____

Section: _____

Programming Exercise

1. Write a program to show classes of umbrella and provide the polymorphism mechanism in umbrella open(), close(), waterproof().
2. Design class for classical, and latest mobiles. Use polymorphism to show the functions of these mobiles such as button() or touch(), size(), split_screen(), operating_system(). Etc.



3. Design class for Butterfly. Such that butterfly has multiple colors, patters, nature of living, origin, living_in_pupa, living_as_caterpillar. Etc.

