

Batch: G3**Roll No.: 16010421063****Experiment / assignment / tutorial No.****Grade: AA / AB / BB / BC / CC / CD / DD****Signature of the Staff In-charge with date****TITLE: Polymorphism and Abstract Class****AIM:**

1. Write a program that defines an abstract class called Vehicle containing an abstract method speed (). Derive from it two classes - FourWheeler and TwoWheeler. Create objects of derived classes and call the speed () method using these objects, passing to it the name of vehicle and speed of vehicle. In the speed () method print the vehicle name and the speed of vehicle to which speed () belongs.
2. Write a program to create Box class as abstract class with functionalities like adding, removing and counting items. Create one subclass of collections like List to implement concept of abstraction by performing above functionalities.

Expected OUTCOME of Experiment:

CO1: Use basic data structures in Python

CO2: Use different Decision Making statements and Functions in Python.

CO3: Apply Object oriented programming concepts in Python

Resource Needed: Python IDE

Theory:**Polymorphism:**

Polymorphism is taken from the Greek words Poly (many) and morphism (forms). It means that the same function name can be used for different types. This makes programming more intuitive and easier.

Polymorphism in Python:

A child class inherits all the methods from the parent class. However, in some situations, the method inherited from the parent class doesn't quite fit into the child class. In such cases, you will have to re-implement method in the child class.

There are different methods to use polymorphism in Python. You can use different function, class methods or objects to define polymorphism.

Polymorphism with Function and Objects:

You can create a function that can take any object, allowing for polymorphism.

Example:

Create a function called “func()” which will take an object which we will name “obj”. and let give the function something to do that uses the ‘obj’ object which is passed to it. Now, call the methods type() and color(), each of which is defined in the two classes ‘Tomato’ and ‘Apple’ by creating instances of both the ‘Tomato’ and ‘Apple’ classes if they do not exist:

```
class Tomato():
    def type(self):
        print("Vegetable")
    def color(self):
        print("Red")
class Apple():
    def type(self):
        print("Fruit")
    def color(self):
        print("Red")
```

```
def func(obj):
    obj.type()
    obj.color()
```

```
obj_tomato = Tomato()
obj_apple = Apple()
func(obj_tomato)
func(obj_apple)
```

Output:

```
Vegetable
Red
Fruit
Red
```

Polymorphism with Class Methods:

Python uses two different class types in the same way. Here, you have to create a for loop that iterates through a tuple of objects. Next, you have to call the methods without being concerned about which class type each object is. We assume that these methods actually exist in each class

Example:

```

class India():
    def capital(self):
        print("New Delhi")

    def language(self):
        print("Hindi and English")

class USA():
    def capital(self):
        print("Washington, D.C.")

    def language(self):
        print("English")

obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
    country.capital()
    country.language()

```

Output:

```

New Delhi
Hindi and English
Washington, D.C.
English

```

Polymorphism with Inheritance:

Polymorphism in python defines methods in the child class that have the same name as the methods in the parent class. In inheritance, the child class inherits the methods from the parent class. Also, it is possible to modify a method in a child class that it has inherited from the parent class.

This is mostly used in cases where the method inherited from the parent class doesn't fit the child class. This process of re-implementing a method in the child class is known as Method Overriding.

Example:

```

class Bird:
    def intro(self):
        print("There are different types of birds")

    def flight(self):
        print("Most of the birds can fly but some cannot")

class parrot(Bird):
    def flight(self):
        print("Parrots can fly")

class penguin(Bird):
    def flight(self):
        print("Penguins do not fly")

obj_bird = Bird()
obj_parr = parrot()
obj_peng = penguin()

obj_bird.intro()
obj_bird.flight()

obj_parr.intro()
obj_parr.flight()

obj_peng.intro()
obj_peng.flight()

```

Output:

```

There are different types of birds
Most of the birds can fly but some cannot
There are different types of bird
Parrots can fly
There are many types of birds
Penguins do not fly

```

Abstract Class:

Abstract Class is concept of object-oriented programming based on DRY (Don't Repeat Yourself) principle. In a large project, code duplication is approximately equal to bug reuse and one developer is impossible to remember all classes' details. Therefore, it's very helpful to use an abstract class to define a common interface for different implementations.

An abstract class has some features, as follows:-

- An abstract class doesn't contain all of the method implementations required to work completely, which means it contains one or more abstract methods. An abstract method is a method that just has a declaration but does not have a detail implementation.
- An abstract class cannot be instantiated. It just provides an interface for subclasses to avoid code duplication. It makes no sense to instantiate an abstract class.
- A derived subclass must implement the abstract methods to create a concrete class that fits the interface defined by the abstract class. Therefore it cannot be instantiated unless all of its abstract methods are overridden.

Define Abstract Class in Python:

Python comes with a module called abc which provides methods for abstract class.

Define a class as an abstract class by abc.ABC and define a method as an abstract method by abc.abstractmethod. ABC is the abbreviation of abstract base class.

Example:

```
from abc import ABC, abstractmethod
```

```
class Animal(ABC):
    @abstractmethod
    def move(self):
        pass
a = Animal()          # TypeError: Can't instantiate abstract class Animal with abstract methods
                        move
```

```
class Animal():
    @abstractmethod
    def move(self):
        pass
a = Animal()          # No errors
```

Invoke Methods from Abstract Classes:

An abstract method is not needed to be "totally abstract" in Python. We can define some content in an abstract method and use super() to invoke it in subclasses.

Example:

```
from abc import ABC, abstractmethod
```

```
class Animal(ABC):
    @abstractmethod
    def move(self):
        print('Animal moves')
```

```
class Cat(Animal):
    def move(self):
        super().move()
        print('Cat moves')
```

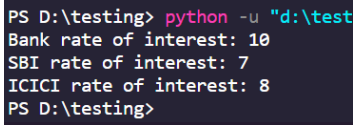
```
c = Cat()
c.move()
```

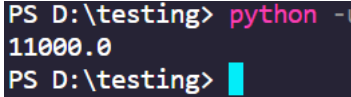
Output:

Animal moves
Cat moves

Problem Definition:

1. For given program find output

Sr.No	Program	Output
1	<pre>class Bank: def getroi(self): return 10 class SBI: def getroi(self): return 7 class ICICI: def getroi(self): return 8 b1=Bank() b2=SBI() b3=ICICI() print("Bank rate of interest:",b1.getroi()) print("SBI rate of interest:",b2.getroi()) print("ICICI rate of interest:",b3.getroi())</pre>	

2.	<pre> from abc import ABC, abstractmethod class Employee(ABC): @abstractmethod def calculate_salary(self,sal): pass class Developer(Employee): def calculate_salary(self,sal): finalsalary=sal*1.10 return finalsalary emp_1=Developer() print(emp_1.calculate_salary(10000)) </pre>	
----	--	--

3. Write a program that defines an abstract class called Vehicle containing an abstract method speed (). Derive from it two classes - FourWheeler and TwoWheeler. Create objects of derived classes and call the speed () method using these objects, passing to it the name of vehicle and speed of vehicle. In the speed () method print the vehicle name and the speed of vehicle to which speed () belongs.

4. Write an “abstract” class, Box, and use it to define some methods which any box object should have: add, for adding any number of items to the box, empty, for taking all the items out of the box and returning them as a list, and count, for counting the items which are currently in the box.

Write a simple Item class which has a name attribute and a value attribute – you can assume that all the items you will use will be Item objects. Now write one subclass of Box which use different underlying collections to store items like ListBox should use a list

Books/ Journals/ Websites referred:

1. Reema Thareja , “Python Programming: Using Problem Solving Approach”, Oxford University Press, First Edition 2017, India
 2. Sheetal Taneja and Naveen Kumar,” Python Programing: A Modular Approach”, Pearson India, Second Edition 2018, India
 3. <https://www.edureka.co/blog/polymorphism-in-python/>
 4. <https://www.geeksforgeeks.org/inheritance-in-python/>
-

Implementation details:

3.

```
'''Write a program that defines an abstract class called Vehicle
containing an abstract method speed (). Derive from it two classes -
```

FourWheeler and TwoWheeler. Create objects of derived classes and call the speed () method using these objects, passing to it the name of vehicle and speed of vehicle. In the speed () method print the vehicle name and the speed of vehicle to which speed () belongs.

```
'''  
  
from abc import ABC, abstractmethod  
  
class Vehicle(ABC):  
    @abstractmethod  
    def speed(self, name, speed):  
        print(f"Name- {name} Speed-{speed}km/hr")  
  
class FourWheeler(Vehicle):  
    def speed(self, name, speed):  
        super().speed(name, speed)  
  
class TwoWheeler(Vehicle):  
    def speed(self, name, speed):  
        super().speed(name, speed)  
a=TwoWheeler()  
a.speed("arya", 50)  
  
b=FourWheeler()  
b.speed("Nair", 100)
```

4.

```
from abc import ABC, abstractmethod  
# Item is a class that has a name and a value.  
class Item:  
    def __init__(self, name, value):  
        self.name = name  
        self.value = value  
  
# The Box class is an abstract class that has three abstract methods:  
add, empty, and count  
class Box(ABC):  
    def __init__(self, items):  
        self.items = items  
  
    @abstractmethod  
    def add(self, new_items):  
        pass  
    @abstractmethod
```

```
def empty(self):
    pass
# It's printing "Invalid choice"
@abstractmethod
def count(self):
    pass

# It's a class that inherits from the Box class, and adds the ability to
add items to the box, empty
# the box, and count the number of items in the box
class ListBox(Box):
    def __init__(self, items=None):
        if items is None:
            items = []
        super(ListBox, self).__init__(items)
    def add(self, new_items):
        print(new_items)
        self.items += new_items
        return self.items
    def empty(self):
        temp_items = self.items.copy()
        self.items.clear()
        return temp_items
    def count(self):
        return len(self.items)

List=ListBox()
# It's a menu that allows the user to add items to the list, empty the
list, count the number of items
# in the list, and exit the program.
while True:
    print("1. Add Item")
    print("2. Empty Box")
    print("3. Count Items")
    print("4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        number = list(map(int,input("Enter items to be added to
List:").split()))
        List.add(number)
    elif choice == 2:
        print(List.empty())
    elif choice == 3:
        print(List.count())
    elif choice == 4:
```



```
        break
    else:
        print("Invalid choice")
```

Output(s):

3.

```
PS D:\testing> python -u "d:\
Name- arya Speed-50km/hr
Name- Nair Speed-100km/hr
PS D:\testing> █
```

4.

```
PROBLEMS  OUTPUT  TERMINAL  JUPYTER  DEBUG CONSOLE

KeyboardInterrupt
PS D:\data\coding project\image-upload> python -u "d:\data\coding project\image-upload\test.py"
1. Add Item
2. Empty Box
3. Count Items
4. Exit
Enter your choice: 1
Enter items to be added to List:5 6 7 8
[5, 6, 7, 8]
1. Add Item
2. Empty Box
3. Count Items
4. Exit
Enter your choice: 3
4
1. Add Item
2. Empty Box
3. Count Items
4. Exit
Enter your choice: 2
[5, 6, 7, 8]
1. Add Item
2. Empty Box
3. Count Items
4. Exit
Enter your choice: 3
0
1. Add Item
2. Empty Box
3. Count Items
4. Exit
Enter your choice: █

Reconnect to Discord  Ln 62, Col 32 (1620 selected)  Tab Size: 4  UTF-8  CRLF  Python  Select Interpreter
```

Conclusion:

Successfully understood the concept of polymorphism and abstract classes.

Post Lab Descriptive Questions**1. Explain difference between inheritance and abstract class with example**

Abstraction is an OOP concept that hides the implementation details and shows only the functionality to the user. In contrast, Inheritance is the methodology of creating a new class using the properties and methods of an existing class. Thus, this reflects the main difference between abstraction and inheritance.

Inheritance-

```
class A:
    def display(self):
        print("A")

class B(A):
    def new_display(self):
        print("B")

test=B()
test.new_display()
test.display()
```

```
PS D:\data\coding project\image-upload> python -u "d:\da
B
A
PS D:\data\coding project\image-upload> █
```

Abstract class

```
from abc import ABC, abstractmethod

class Polygon(ABC):
    @abstractmethod
    def noofsides(self):
        pass

class Triangle(Polygon):
```

```
# overriding abstract method
def noofsides(self):
    print("I have 3 sides")

test=Triangle()
test.noofsides()
```

```
PS D:\data\coding proje
I have 3 sides
PS D:\data\coding proje
```

```
PS D:\data\coding project\  
I have 3 sides  
PS D:\data\coding project\  

```

Date: _____

Signature of faculty in-charge