**Batch: A3 Roll No.: 16010121051**

**Experiment / assignment / tutorial No. 8**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE: Polymorphism and Abstract Class** |

**AIM:**

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.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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.

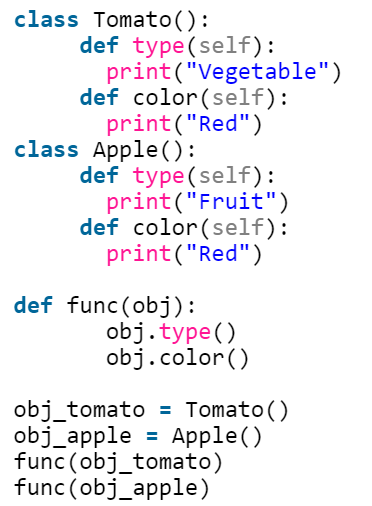
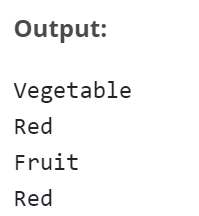
There are different methods to use polymorphism in Python. You can use different function, class methods or objects to define polymorphism.

**Polymorphism with Function and Objects:**

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

**Example:**

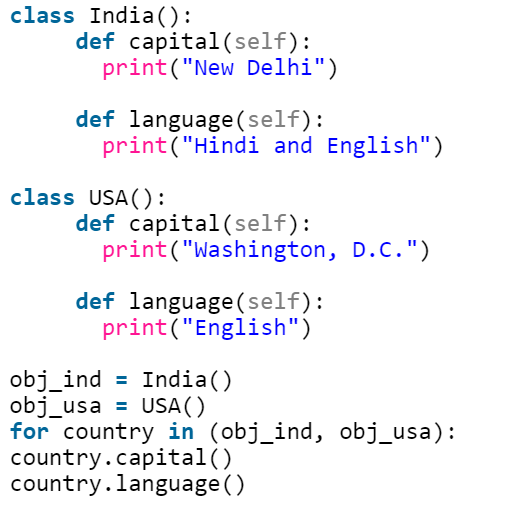
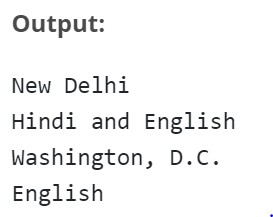
Create a function called “func()” which will take an object which we will name “obj”. and let give the function something to do that uses the ‘obj’ object which is passed to it. Now, call the methods type() and color(), each of which is defined in the two classes ‘Tomato’ and ‘Apple’ by creating instances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

**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:**

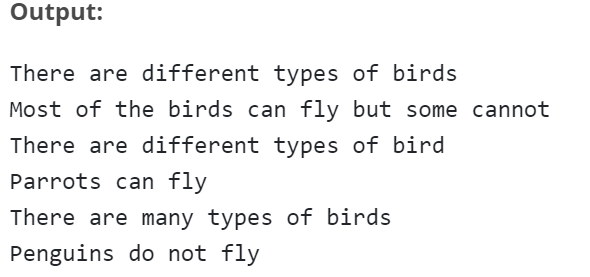
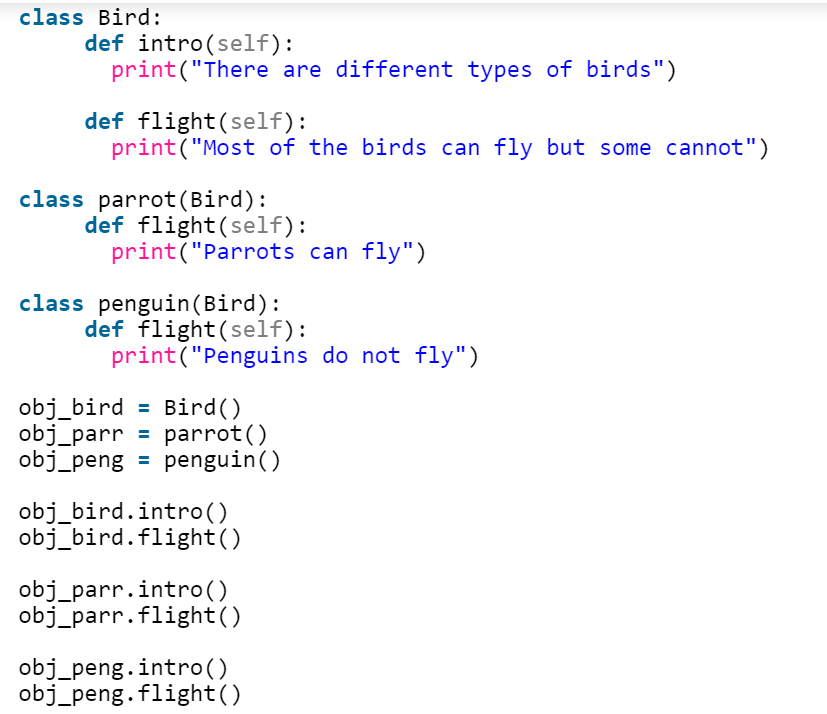
 

**Polymorphism with Inheritance:**

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:**



**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 | 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()) | Bank rate of interest: 10  SBI rate of interest: 7  ICICI rate of interest: 8 |
| 2. | 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)) | 11000.0 |

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.

**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:**

# Ritvik jindal a3 16010121073

from abc import ABC,abstractmethod

class vehicle(ABC):   #defining classes

    @ abstractmethod

    def speed(self):

        print("Information about the vehicle is: ")

class fourwheeler(vehicle):

    def \_\_init\_\_(self,name,speed1):

        self.name=name

        self.speed1=speed1

    def speed(self):

        super().speed()

        print("The name of the vehicle is: ",self.name)

        print("The speed of the vehicle is: ",self.speed1)

class twowheeler(vehicle):

    def \_\_init\_\_(self,name,speed1):

        self.name=name

        self.speed1=speed1

    def speed(self):

        super().speed()

        print("The name of the vehicle is: ",self.name)

        print("The speed of the vehicle is: ",self.speed1)

vehicle1=fourwheeler("Maruti",20)

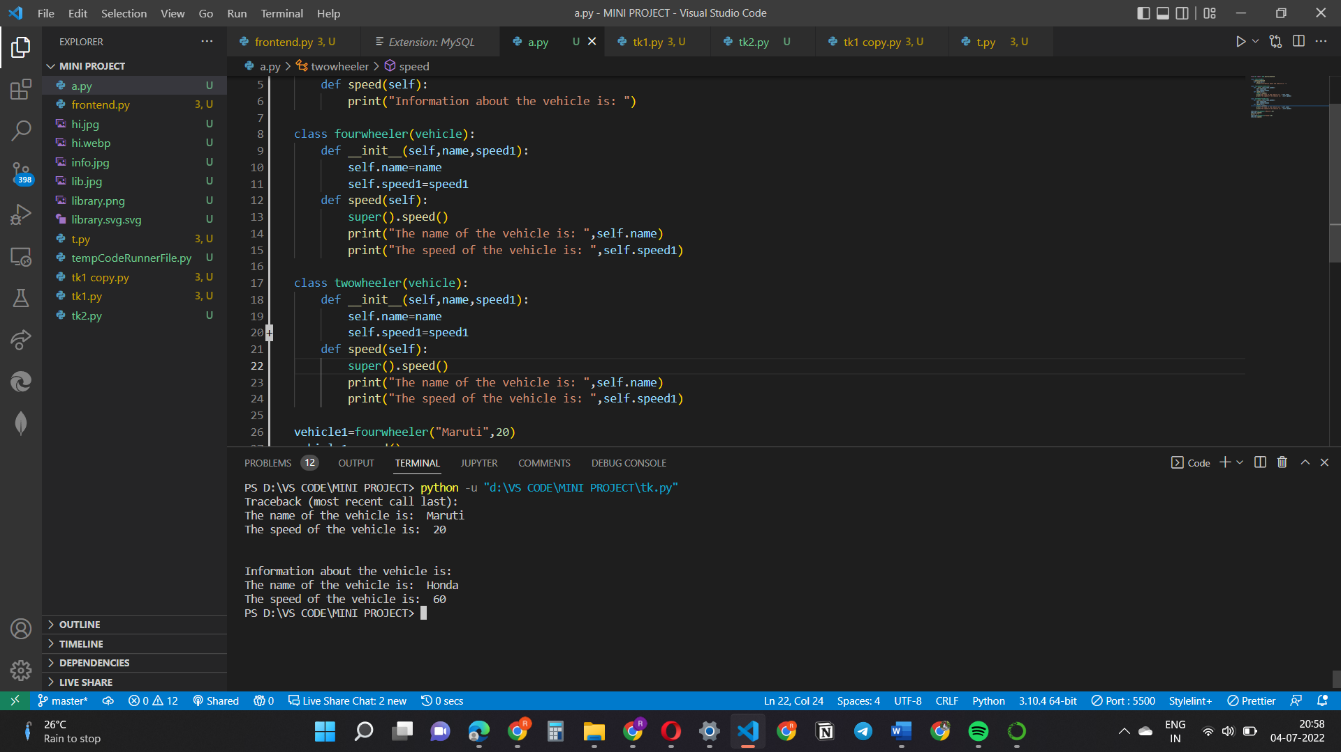
vehicle1.speed()

print("\n")

vehicle2=twowheeler("Honda",60)

vehicle2.speed()

**Output(s):**



**Conclusion:**

**Implementation of polymorphism and abstraction has been understood and implemented successfully**

**Post Lab Questions**

**1. What will be the output of the following Python code?**

class A:

def \_\_init\_\_(self):

self.multiply(15)

print(self.i)

def multiply(self, i):

self.i = 4 \* i;

class B(A):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def multiply(self, i):

self.i = 2 \* i;

obj = B()

**Options**

a) 15

b) 60

c) An exception is thrown

d) 30

**D)** output is 30 since the multiply method of the parent class is overridden by the multiply method of the child class

1. A class in which one or more methods are only implemented to raise an exception is called an abstract class.

a) True

b) False

**A) True**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**