EXPERIMENT NO : 3D

Python Programs to implement Abstract class, Abstract method and Interfaces in Python.

NAME : AKASH RAMKRIT YADAV		***ID.NO: VU4F2122016***
BATCH : A	***BRANCH : IT***	***DIV : A***

Aim :- Python Programs to implement Abstract class, Abstract method and Interfaces in Python.

THEORY:

OUTPUT:

Python 3.11.0a4 (main, Mar 1 2023, 10:57:32) [MSC v.1929 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
#AKASH YADAV ID.NO:VU4F2122016 EXP:3D DATE:1/3/2023

Abstract Classes in Python

An abstract class can be considered as a blueprint for other classes.
It allows you to create a set of methods that must be created within any child classes built from the abstract class.
A class which contains one or more abstract methods is called an abstract class.

An abstract method is a method that has a declaration but does not have an implementation.
While we are designing large functional units we use an abstract class.
When we want to provide a common interface for different implementations of a component, we use an abstract class.

By default, Python does not provide abstract classes.
Python comes with a module that provides the base for defining Abstract Base classes(ABC) and that module name is ABC.

ABC works by decorating methods of the base class as abstract and then registering concrete classes as implementations of the abstract base.
A method becomes abstract when decorated with the keyword @abstractmethod.

The partially implemented classes are called an abstract class;every abstract class in python should be a child of ABC class, which is present in the abc module.

EXAMPLE : 1]

```
# Python program showing
# abstract base class work


from abc import ABC, abstractmethod
class Animal(ABC):
    def move(self):
        pass

class Human(Animal):
    def move(self):
        print("i can run & walk")

class Dog(Animal):
    def move(self):
        print("I can bark")

class Snake(Animal):
    def move(self):
        print("I can crawl")

class Tiger(Animal):
    def move(self):
        print("I can roar")
```

 # Driver code

```
A=Human()
A.move()

B=Dog()
B.move()

C=Snake()
C.move()

D=Tiger()
D.move()
```

>>> i can run & walk

I can bark

I can crawl

I can roar

2]

```
from abc import ABC, abstractmethod

class Vehicle(ABC):     ##abstarct class
    @abstractmethod    ##abstract method
    def getNoOfWheels(Self):
        pass


class Bus(Vehicle):    ##implementing parent abstract class by using child
class
    def getNoOfWheels(self):
        return 6


class Auto(Vehicle):    ##implementing parent abstract class by using child
class
    def getNoOfWheels(self):
        return 3


b=Bus()
print(b.getNoOfWheels())



a=Auto()
print(a.getNoOfWheels())
```

>>> 6

3

Abstract Method in python

An Abstract method is a method which is declared but does not have implementation such type of methods are called as abstract methods.

In Python, we can declare an abstract method by using @abstractmethod decorator.

This abstract method is present in the abc module in python, and hence, while declaring the abstract method, we have to import the abc module compulsory.

Example:

```
from abc import abstractmethod
class Vehicle:

 @abstractmethod
    def getNoOfWheels(Self):
        pass
```

The above program does not have any implementation, and we won't get any output.

Here the Child class is responsible for providing an implementation for the parent class abstract method.

Interfaces in Python.

An abstract class can contain both abstract and non-abstract methods if an abstract class contain only abstract method, then the class is called an interface.

A 100% pure abstract class is nothing but an interface. An interface acts as a service requirement specification.

An interface is a collection of method signatures that should be provided by the implementing class.

An interface contains methods that are abstract in nature. The abstract methods will have the only declaration as there is no implementation.
An interface in python is defined using python class and is a subclass of interface.Interface which is the parent interface for all interfaces.
The implementations will be done by the classes which will inherit the interface.

How to declare an interface in Python

Class MyInterface(zope.interface.Interface)

Firstly, we will import zope.interface module.
The zope.interface is a module that is used to implement the object interface in python.
The zope.interface library is the way to come out of when something is not clear.
The interface act as a blueprint for designing classes. Here, @zope.interface.implementer(Lunch) is implemented using implementer decorator in class.
This package export attributes and interfaces directly.
To overcome the uncertainty of the interface zope module is implemented.

Implementation by(class) – This function returns a boolean value. If the class implements the interface it results in True else False.

Example: 1]

```
import zope.interface

class Lunch(zope.interface.Interface):
        northindian = zope.interface.Attribute("chocolate")
        def food(self, northindian):
                pass

def colddrinks(self, beverages):
                pass

@zope.interface.implementer(Lunch)


class Lunch(zope.interface.Interface):

    def food(self, northindian):
        return northindian

    def colddrinks(self,beverages):
            return beverages
```

```
colddrinks = Lunch['colddrinks']
food = Lunch['food']

print(Lunch.implementedBy(Lunch))

print(type(colddrinks))

print(type(food))
```

2]

```
from abc import *

class CollegeAutomation(ABC):    ##requirement apecification

    @abstractmethod
    def method1(self):pass

   @abstractmethod
    def method2(self):pass

   @abstractmethod
    def method3(self):pass


class AaruSoftImpl(CollegeAutomation):
    def method1(self):
        print("Method1 implementation")


   def method2(self):
        print("Method2 implementation")


   def method3(self):
        print("Method3 implementation")


d=AaruSoftImpl()

##creating an object and calling method1, method2 and method3

d.method1()

d.method2()

d.method3()
```

>>> Method1 implementation

Method2 implementation

Method3 implementation