EXPERIMENT NO : 3A

Python Programs to Implement Classes, Object, Static method, Constructors , Inner Class.

NAME : AKASH RAMKRIT YADAV ID.NO: VU4F2122016

BATCH : A BRANCH : IT DIV : A

Aim :- Python Programs to Implement Classes, Object, Static method,

. Constructors and Inner Class.

THEORY:

OUTPUT:

Python 3.11.0a4 (main, Jan 17 2022, 12:57:32) [MSC v.1929 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
#AKASH YADAV ID.NO:VU4F2122016 EXP:3A DATE:13/2/2023

#Python Classes/Objects

Python is an object oriented programming language.Almost everything in Python is an object, with its properties and methods.A Class is like an object constructor, or a "blueprint" for creating objects."""

#Create a Class

To create a class, use the keyword class:

```
#Create a class named AKASH, with a property named x:
class AKASH:
   x=5

print(AKASH)
```

>><class '__main__.AKASH'>

#CREATING OBJECT

Now we can use the class named AKASH to create objects:
Create an object named a1, and print the value of x:

```
a1=AKASH()
print(a1.x)
```

#>>5

#The __init__() Function

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:"""

#Create a class named Akash, use the __init__() function to assign values for name and age:

EXAMPLE:

```
class Akash:
    def __init__(self,friends,age):
        self.friends=friends
        self.age=age

# creating object
a1=Akash("viram",22)
a2=Akash("suraj",21)
a3=Akash("rama",22)
a4=Akash("suray",23)

#for printing ,call by bojectname.varable
print(a1.friends)
print(a1.age)

print(a2.friends)
print(a2.age)

print(a3.friends)
print(a3.age)

print(a4.friends)
print(a4.age)
```

```
>>viram
  22
  suraj
  21
  rama
  22
  suray
  23
```

#The __str__() Function

The __str__() function controls what should be returned when the class object is represented as a string.

If the __str__() function is not set, the string representation of the object is returned:

 # 1) The string representation of an object WITHOUT the __str__() function:

```
class akash:

    # The init method or constructor
    def __init__(self,friends,age):

        # Instance Variable
        self.friends=friends
        self.age=age

# Objects of class
a1=akash("viram",22)

print(a1)
```

```
>><__main__.akash object at 0x00000169B1750990>
```

#2) The string representation of an object with the __str__() function:

```
class akash:
    def __init__(self,friends,age):
        self.friends=friends
        self.age=age

    def __str__(self):
        return f"{self.friends}({self.age})"

a1=akash("viram",22)

print(a1)
```

```
>>viram(22)
```

#Python object

An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values.
Python is object-oriented programming language that stresses on objects i.e.

it mainly emphasizes functions. Objects are basically an encapsulation of data variables and methods acting on that data into a single entity.

Instance defining represent memory allocation necessary for storing the actual data of variables. Each time when you create an object of class the copy of each data variables defined in that class is created.
In simple language we can state that each object of a class has its own copy of data members defined in that class. """

EXAMPLE:

```
class AKASH:
   x=5
a1=AKASH()
print(a1.x)
```

```
>>5
```

#Self Variable:

SELF is a default variable that contains the memory address of the current object.
Instance variables and methods can be referred to by the self variable.
When the object of a class is created, the memory location of the object is contained by its object name.
This memory location is passed to the SELF internally, as SELF knows the memory address of the object, so the variable and method of an object is accessible.
The first argument to any object method is SELF because the first argument is always object reference.
This process takes place automatically whether you call it or not

EXAMPLE:

```
class Akash:
  def __init__(a1, a, b):
       a1.country = a
       a1.capital = b

  def myfunc(y1):
    print("Capital of  " + y1.country +" is:" + y1.capital)

x = Akash("India", "Delhi")
x.myfunc()
```

```
>>Capital of  India is:Delhi
```

#Class Method in Python

The @classmethod decorator is a built-in function decorator that is an expression that gets evaluated after your function is defined.
The result of that evaluation shadows your function definition. A class method receives the class as an implicit first argument,

just like an instance method receives the instance

#Syntax Python Class Method:

```
class C(object):
    @classmethod
    def fun(cls, arg1, arg2, ...):
        ....
```

fun: function that needs to be converted into a class method
returns: a class method for function.

EXAMPLE:

```
class AKASH:
    def __init__(self, value):
        self.value = value

    def get_value(self):
        return self.value

# Create an instance of AKASH
obj = AKASH("VU4F2122016")

# Call the get_value method on the instance
print(obj.get_value())  \

# Output: VU4F2122016
```

 # the Static Method in Python

A static method does not receive an implicit first argument. A static method is also a method that is bound to the class and not the object of the class. This method can't access or modify the class state. It is present in a class because it makes sense for the method to be present in class.

Syntax Python Static Method:

```
class C(object):
    @staticmethod
    def fun(arg1, arg2, ...):
        ...
returns: a static method for function fun."""
```

EXAMPLE:

```
class akash:
    def __init__(self, value):
        self.value = value

    @staticmethod
```

```
    def get_max_value(x, y):
        return max(x, y)

# Create an instance of akash
obj = akash(10)

print(akash.get_max_value(43, 316))

print(obj.get_max_value(1044, 544))

>>316
    1044
```

#The difference between the Class method and the static method is:

A class method takes cls as the first parameter while a static method needs no specific parameters.
A class method can access or modify the class state while a static method can't access or modify it.
In general, static methods know nothing about the class state. They are utility-type methods that take some parameters and work upon those parameters.
On the other hand class methods must have class as a parameter.
We use @classmethod decorator in python to create a class method and we use @staticmethod decorator to create a static method in python.

 # Python program to demonstrate
 # use of class method and static method.

```
from datetime import date


class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # a class method to create a Person object by birth year.
    @classmethod
    def fromBirthYear(cls, name, year):
        return cls(name, date.today().year - year)

    # a static method to check if a Person is adult or not.
    @staticmethod
    def isAdult(age):
        return age > 18


person1 = Person('mayank', 21)
person2 = Person.fromBirthYear('mayank', 1996)
```

```
print(person1.age)
print(person2.age)

# print the result
print(Person.isAdult(22))

>>21
   27
```

#Constructors in Python

Constructors are generally used for instantiating an object.
The task of constructors is to initialize(assign values) to the data members of the class when an object of the class is created.
In Python the __init__() method is called the constructor and is always called when an object is created.
Syntax of constructor declaration :

```
def __init__(self):
    # body of the constructor
```

#Types of constructors :

#1 default constructor: The default constructor is a simple constructor which doesn't accept any arguments.
*Its **definition** has only one argument which is a reference to the instance being constructed.*

EXAMPLE:

```
class akash:

# default constructor
    def __init__(a1):
        a1.name="akash"

 # a method for printing data members
    def print_akash(a1):
        print(a1.name)

# creating object of the class
obj=akash()

# calling the instance method using the object obj
obj.print_akash()

>>True
   akash
```

#parameterized constructor:

constructor with parameters is known as parameterized constructor.
The parameterized constructor takes its first argument as a reference to the instance being constructed
known as self and the rest of the arguments are provided by the programmer.

EXAMPLE:

```
class Addition:
    num1=0
    num2=0
    sum=0

 # parameterized constructor
    def __init__(a1,n1,n2):
        a1.num1=n1
        a1.num2=n2

    def display(a1):
        print("FIRST NUMBER = " + str(a1.num1) )
        print(" SECOND NUMBER = " + str(a1.num2))
        print("A ADITTION OF TWO NUMBER IS = " + str(a1.sum))

    def calculation(a1):
        a1.sum = a1.num1 + a1.num2

# creating object of the class
# this will invoke parameterized constructor
obj1=Addition(3,22)
obj1.calculation()
obj1.display()

>>FIRST NUMBER = 3
    SECOND NUMBER = 22
   A ADITTION OF TWO NUMBER IS = 25
```

#Inner Class in Python

A class defined in another class is known as an inner class or nested class.
If an object is created using child class means inner class then the object can also be used by parent class or root class.
A parent class can have one or more inner classes but generally inner classes are avoided.
We can make our code even more object-oriented by using an inner class. A single object of the class can hold multiple sub-objects.
We can use multiple sub-objects to give a good structure to our program.

#snyap

```
# create NameOfOuterClass class
class NameOfOuterClass:

  # Constructor method of outer class
  def __init__(self):

    self.NameOfVariable = Value


   # create Inner class object
    self.NameOfInnerClassObject = self.NameOfInnerClass()

  # create a NameOfInnerClass class
  class NameOfInnerClass:
     # Constructor method of inner class
    def __init__(self):
      self.NameOfVariable = Value

# create object of outer class
outer = NameOfOuterClass()
```

#Types of inner classes are as follows:

#1 Multiple inner class
#2 Multilevel inner class

#1Multiple inner class

The class contains one or more inner classes known as multiple inner classes. We can have multiple inner class in a class, it is easy to implement multiple inner classes.

 # create outer class

```
# create outer class
class college:
    def __init__(self):
        self.name = 'college'
        self.gov = self.goverment()
        self.pri = self.private()

    def show(self):
        print('In outer class')
        print('Name:', self.name)

    # create a 1st Inner class
```

```
    class goverment:
        def __init__(self):
            self.name = 'V.J.T.I'
            self.package = '50 LACK'

        def display(self):
            print(" College Name:", self.name)
            print("package:", self.package)

    # create a 2nd Inner class
    class private:
        def __init__(self):
            self.name = 'PVPPCOE'
            self.package = '12 LACK'

        def display(self):
            print("\nCollege Name:", self.name)
            print("package:", self.package)


# create a object
# of outer class
outer = college()
outer.show()

# create a object
# of 1st inner class
d1 = outer.gov

# create a object
# of 2nd inner class
d2 = outer.pri
print()
d1.display()
print()
d2.display()


>>In outer class
    Name: college

   College Name: V.J.T.I
   package: 50 LACK


   College Name: PVPPCOE
   package: 12 LACK
```

#Multilevel inner class

The class contains an inner class and that inner class again contains another inner class, this hierarchy is known as the multilevel inner class.

```
# create an outer class
class AKASH_YADAV:

    def __init__(self):
        # create an inner class object
        self.inner = self.Inner()

    def show(self):
        print('\nThis is an outer class')

    # create a 1st inner class

    class Inner:
        def __init__(self):
            # create an inner class of inner class object
            self.innerclassofinner = self.Innerclassofinner()

        def show(self):
            print('This is the inner class')

        # create an inner class of inner

         class Innerclassofinner:
            def show(self):
                print('This is an inner class of inner class')


# create an outer class object
# i.e.AKASH_YADAV class object
outer = AKASH_YADAV()
outer.show()
print()

# create an inner class object
A1 = outer.inner
A1.show()
print()

# create an inner class of inner class object
A2 = outer.inner.innerclassofinner
A2.show()
```

>> This is an outer class

This is the inner class

This is an inner class of inner class