OOPS

Like other general-purpose programming languages, Python is also an object-oriented language since its beginning. It allows us to develop applications using an Object-Oriented approach. In Python, we can easily create and use classes and objects.

An object-oriented paradigm is to design the program using classes and objects. The object is related to real-word entities such as book, house, pencil, etc. The oops concept focuses on writing the reusable code. It is a widespread technique to solve the problem by creating objects.

- Class
- Object
- Method
- Inheritance
- Polymorphism
- Data Abstraction
- Encapsulation
- Class

1. The class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods. For example: if you have an employee class, then it should contain an attribute and method, i.e. an email id, name, age, salary, etc.
2. Method

- The method is a function that is associated with an object. In Python, a method is not unique to class instances. Any object type can have methods.

3. Inheritance

- Inheritance is the most important aspect of object-oriented programming, which simulates the real-world concept of inheritance. It specifies that the child object acquires all the properties and behaviors of the parent object.

It is an excellent representation of relationships in the real world. It allows code reuse. It doesn't require us to create the same code repeatedly and again. It also allows us to add options to an existing class without having to modify the existing code. It is a transitive nature, meaning that if B is inherited from another class A, all the subclasses belonging to B will inherit directly from class A.

Single Inheritance

- Single inheritance allows a derivate class to inherit properties of one parent class,and this allows code reuse and the introduction of additional features in existing code.

Multiple Inheritance

- If a class is able to be created from multiple base classes, this kind of Inheritance is known as multiple Inheritance. When there is multiple Inheritance, each of the attributes that are present in the classes of the base has been passed on to the class that is derived from it.

Multilevel inheritance,

- the features that are part of the original class, as well as the class that is derived from it,

In [ ]:

In [5]:
```
#class  A  ------------>  class B
```

In [8]:
```
class A():
    def rev(self,x):
        self.x = x
        self.y = x[-1::-1]

    def cap(self,z):
        self.z = z
        self.m = z.upper()
```

In [11]:
```
c = A()
c.rev("akansha")
c.cap("nitin")
```

In [10]:
```
print("Revese the Text :",c.y)
```

```
Revese the Text : ahsnaka
```

In [12]:
```
print("Capital letter text :  ",c.m)
```

```
Capital letter text :   NITIN
```

In [ ]:

In [ ]:

In [15]:
```
class A():
    def table(self,x):
        self.x = x
        for i in range(1,11):
            print(f"{i} X {x} = {i*x}")

class B(A):
    def even(self,y):
        self.y = y
        for i in range(y):
            if i%2==0:
                print("Even Number :",i)
```

In [19]:
```
c = B()
```

In [20]:
```
c.table(5)
```

```
1 X 5 = 5
2 X 5 = 10
3 X 5 = 15
4 X 5 = 20
5 X 5 = 25
6 X 5 = 30
7 X 5 = 35
8 X 5 = 40
9 X 5 = 45
10 X 5 = 50
```

In [21]:
```
c.even(20)
```

```
Even Number : 0
Even Number : 2
Even Number : 4
Even Number : 6
Even Number : 8
Even Number : 10
Even Number : 12
Even Number : 14
Even Number : 16
Even Number : 18
```

In [ ]:
```

```

In [22]:
```
#  __init__ function :- its call the function without function name
```

In [26]:
```
class A():
    def __init__(self,x,y):
        self.x = x
        self.y = y
        self.z = x % y
```

In [28]:
```
c = A(89,15)

print("Modules :- ",c.z)
```

```
Modules :-  14
```

In [ ]:
```

```

In [31]:
```
class A():
    def __init__(self,name):
        self.name = name

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

        print("Hii My name is :",name)
        print('I am',age,"year old")
```

In [32]:
```
c = B("prince sharma",24)
```

```
Hii My name is : prince sharma
I am 24 year old
```

In [ ]:
```

```

Multipule Inheritance

In [40]:
```
class A():
    def first(self,x):
        self.x = x
        print("Name : ",x)

class B(A):
    def second(self,y):
        self.y = y
        print("Age  : ",y)

class C(B):
    def third(self,z):
        self.z = z
        print("City : ",z)
```

In [41]:
```
c = C()
c.first("Viart kohli")
c.second(35)
c.third("Delhi")
```

```
Name :  Viart kohli
Age  :  35
City :  Delhi
```

In [ ]:
```

```

Hybrid Inheritance

In [44]:
```
class A():
    def first(self,x):
        self.x = x
        print("Name : ",x)

class B():
    def second(self,y):
        self.y = y
        print("Age  : ",y)

class C(A,B):
    def third(self,z):
        self.z = z
        print("City : ",z)
```

In [46]:
```
c = C()
c.first("Viart kohli")
c.second(35)
c.third("Delhi")
```

```
Name :  Viart kohli
Age  :  35
City :  Delhi
```

In [ ]:
```

```

In [ ]:
```

```

In [50]:
```
class First():
    def __init__(self,a):
        self.a = a

class Second():
    def __init__(self,b):
        self.b = b

class third(First,Second):
    def __init__(self,a,b,c):
        First.__init__(self,a)
        Second.__init__(self,b)
        self.c = c


        print("name :",a)
        print("age :",b)
        print("City :",c)
```

In [52]:
```
k = third("Rohit sharma",38,"Mumbai")
```

```
name : Rohit sharma
age : 38
City : Mumbai
```

In [ ]: