Oops in python

→ Dinesh kumar

Difference Between Oops and Pop

Oops	Pop
1) Programs are divided into parts know as Objects	1) In this programs are divided into functions
2) The main focus of Oop is on the data	2) The main focus of Pop is on the procedure
3) It follows bottom-up approach	3) It follows top-down approach
4) It has access Specifiers such as public , private etc.	4) Pop does not have any access Specifiers
5) It provides data hiding, data associated with the Program.so security is provide	5) Pop does not provide any data security 6) In this most functions use Global data
6) Ease of Modification	7) Modification is difficult

Data Functions ←--→ Data Functions

Functions Data

Global Data

Function(1) Function(2) Function(3)

Local Data Local Data Local Data

Oops Features

1) Class
2) Object
3) Inheritance
4) Polymorphism
5) Encapsulation
6) Data abstraction

1) Class

Class is a blueprint (or) template of Objects

Example

Car is a class and Steering, Wheels, Seats, Brakes, etc. are Objects

2) Objects

Object is a Physical entity

Example

Like we said above steering, wheels, seats, brakes, etc are Objects of Class(car).

Code:

```
class Dinesh:               #Class keyword
    a =20                   #data member
    def Output(self):       #method(self)
        print(self.a)
obj = Dinesh()              #declaring of object
obj.Output()                #using object we can call the methods
```

Output:

20

Using __init__ in Class

__init__ is also known as Constructor (or) Method

- Constructor are generally used for instantiating.
- Python doesn't support Multiple Constructor
- __init__ is always called when an Object is created

Code:

```
class Rishi:
    def __init__(self,a,b,c):    #__init__ (constructor)
        self.x=a                 #(self) we can access the attributes and
methods of the class
        self.y=b
        self.z=c

    def thor(self):
        print(self.z)            #upto to this class is created,and to add
objects for memory creation

objj = Rishi(10,2,1)             #using object we assign values to parameters
objj.thor()
```

Output:

1

3) Inheritance

Acquiring properties of one class to the other class.

Example

Grand father(class) -→ Father(class) -→ Child(class)

- Grand father(class) is known as Super class, Parent class, Base class
- Father class and Child class is know as Child class, Sub class, Derived class

Types of Inheritance

1) Single level inheritance

2) Multilevel inheritance

3) Multiple inheritance

4) Hierarchical inheritance

3.1. Single level inheritance

Single level inheritance consists of one Parent class and one Child class

Example

Father(class) → Son(class)

Father is a Parent class and Son is a Child class

Code:

```
class Parent:                          #here i took class(Parent)
    def outer(self):                   #(self) we can access the attributes and
methos of Class
        print("This is parent class")
class Child(Parent):                   #i took another class(Child)
    def outerchild(self):
        print("This is child class")
singleinh=Child()                      #here i took only Child(class) because
Parent properties derived to child class
singleinh.outer()                      #so now i can call parent(class) using
Child
singleinh.outerchild()
```

Output:

This is parent class

This is child class

3.2 Multilevel inheritance

Features of the base class and the derived class are further inherited into the new derived class

Example

Grand Father(class) → Father(class) → Child(class)

Grand Father is a Parent class, Father and Child is Child class

Code:

```
class Grandfather:                         #here i took Granfather as
Class(Parent)
    def outergf(self):
```

```
        print("This is g.f class")
class Father(Grandfather):                  #here Father will be Child(child)
    def outerf(self):
        print("This is father class ")
class Child(Father):                        #here Child will Child(class)
    def outerc(self):
        print("This is child class ")
mulihn1=Child()                             #properties of Grandfather
inherited to father and to child
mulihn1.outergf()                           #so now i can call Grandfather and
Father classes just using Child
mulihn1.outerf()                            #properties of Grandfather
inherited to father and to child.
mulihn1.outerc()
```

Output:

This is g.f class

This is father class

This is child class

3.3 Multiple inheritance

When a class can be derived from more than one base class

Example

Mother(class) ← → Father(class) → Child(class)

Mother and Father are two Parent classes with one Child class.

Code:

```
class Mother:                           #here mother is Parent class
    def outer(self):
        print("This is mother class")
class Father:                           #here father is also a Parent class
    def outerf(self):
        print("This is father class")
class Child(Mother,Father):             #So here mother and father properties
inherited to Child class
    def outerc(self):
        print("This is child class")
mulinh=Child()                          #here i took only child class
mulinh.outer()                          #but i can call mother class
mulinh.outerf()                         #father class
mulinh.outerc()                         #child class
```

Output:

This is mother class

This is father class

This is child class

3.4 Hierarchical inheritance

Father(class) → Child(class)←→ Child(class)

Father is a Parent class with similar Child classes.

Code:

```
class Father:                              # here Father is a Parent class
    def outer(self):
        print("This is father class")
class Child1(Father):                      # Child1 is sub class
    def outerc1(self):
        print("This is child1 class")
class Child2(Father):                      # child2 is sub class
    def outerc2(self):
        print("This is child2 class")
hierar1=Child1()                           #here father(Parent) class
properties inherited to both child1 and child2
hierar2=Child2()
hierar1.outer()
hierar2.outerc2()
hierar1.outerc1()
```

Output:

This is father class

This is child1 class

This is chid2 class

4) Polymorphism

Poly = Many, morphism = Forms

Example

A person named Dinesh

Dinesh = Son = Employee = influencer = Blogger = Student

Dinesh is a class has many forms

Two topics in Polymorphism

1) Method Overloading

2) Method Overriding

4.1 Method Overloading

Method Overloading defined when,

- Same Class
- Same Function
- Different Parameters

Code :

```
class Methodoverload:                          #here i took Methooverload as
class
    def outer(self,a=None,b=None,c=None):    #here i took different parameters
        print(a,b,c)
obj=Methodoverload()                           #again i took same class as
Methodoverload
obj.outer(1,2,3)                               #so when ever i call the function
if values not assign to parameters it returns as None
obj.outer(1,2)
obj.outer(1)
```

Output:

1,2,3

1,2,None

1,None,None

4.2 Method Overriding

Method Overriding defined when,

- Different Class
- Same Function
- Different Parameters

Code:

```
class Methodoverri:                          #here i took Methodoverri as classd
    def index(self):
        print("This is parent class")
class Child(Methodoverri):                  #here i took different class as Child
    def index(self):
        print("This is child class")
        super().index()
obj=Child()                                 #so in aove i took different class so
now code is override
obj.index()
```

Output:

This is child class

This is parent class

5) Encapsulation

Wrapping of Data into single unit

Example

Class = variables→← methods

^

|

Encapsulation

Access Specifiers

* Public

*Private - (__)

*Protected - (_)

*Protected

Code:

```
class Grandfather:
    def __init__(self,a):
        self._y=a
class Father(Grandfather):
    def outer1(self):
        print(self._y)
class Child(Father):
    def outer2(self):
        print(self._y)
obj=Child(12)
obj.outer2()
```

Output:

12

*Private

Code:

```
class Grandfather:
    def __init__(self,a):
        self.__y=a
class Father(Grandfather):
    def outer1(self):
        print(self.__y)
class Child(Father):
```

```
    def outer2(self):
        print("child2",self.__y)
obj = Child(10)
obj.outer2()
```

Output:

Error: because as (__Y) is private access specifier so only Grandfather(class) can only access.

6) Data abstraction

Hiding the data

*There is no Body

* Cannot create Object

* Can contain one (or) more abstract methods