OOPS

October 3, 2022

1 Object oriented programming in python

Object oriented programming (OOPs) can be broken into object and oriented

Object means - "an entity that exists in the real world" and oriented means "interested in a particular kind of thing of entity"

it is a paradigm that uses objects and classes in programming.

It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

why we need OOPs?

suppose we have to track the employees in an organization which have different attributes like name, age, gender, department. if we used list to do that it is going to confusing for which employee what we have used. and if we want to add more then we have to spent more time to understand that.

Instead of that we can create an class based upon which we can build objects for every employee and can store attribute related to any employee in easy manner.

Main Concepts of OOPs

1) class
2) objects
3) polymorphism
4) encapsulation
5) inheritance
6) data abstraction

2 class:

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

How class are created ?

[2]:
```
class Person:
    pass
```

3 Objects:

The object is an entity that has a state and behavior associated with it.

An object consists of :

State: It is represented by the attributes of an object. It also reflects the properties of an object. Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects. Identity: It gives a unique name to an object and enables one object to interact with other objects.

[3]:
```
#example of object creation

obj=Person()
```

by using **init** method we can make constructor of the class

** we need to give self as first parameter in class

[5]:
```
class Person:

    #class attribute
    class_attribute = "Employee"

    #instance attribute
    def __init__(self,name,age,dept):
        self.name=name
        self.age=age
        self.dept=dept

#object initiation
rohan=Person('rohan',22,'IT')

print("rohan is an {}".format(rohan.__class__.class_attribute))
print("my name is {}, i am {} year old, i work in {} department".format(rohan.
  ↪name,rohan.age,rohan.dept))
```

```
rohan is an Employee
my name is rohan, i am 22 year old, i work in IT department
```

[7]:
```
class Person:

    #class attribute
    class_attribute = "Employee"

    #instance attribute
    def __init__(self,name,age,dept):
        self.name=name
        self.age=age
        self.dept=dept
```

```
    def details(self):
        print("my name is {}, i am {} year old, i work in {} department".
 ↪format(self.name,self.age,self.dept))

#object initiation
rohan=Person('rohan',22,'IT')

#accessing class method
rohan.details()
```

```
my name is rohan, i am 22 year old, i work in IT department
```

4 Inheritance

it is property of class by which it inherits the properties of other class the class which inherits called child class and from which it inherits called as parent class

Types of inheritance———

5 Single inheritance:

Single-level inheritance enables a derived class to inherit characteristics from a single-parent class.

[18]:
```
class Person:

    #class attribute
    class_attribute = "Employee"

    #instance attribute
    def __init__(self,name,age,dept):
        self.name=name
        self.age=age
        self.dept=dept

    def details(self):
        print("my name is {}, i am {} year old, i work in {} department".
 ↪format(self.name,self.age,self.dept))

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

        #invoking parent class constructor
        super().__init__(name,age,dept)

    def details(self):
```

```
        print("my name is {}, i am {} year old, i work in {} department".
 ↪format(self.name,self.age,self.dept))
        print("I work at {} post".format(self.post))

#object initiation
rohan=employee('rohan',22,'IT','Manager')

rohan.details()
```

```
my name is rohan, i am 22 year old, i work in IT department
I work at Manager post
```

6 Multilevel inheritance:

In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class.

[23]:
```
#base class

class first:
    def __init__(self,name):
        self.name=name
    def Print_name(self):
        return self.name

#intermediate class
class second(first):
    def __init__(self,name,age):
        super().__init__(name)
        self.age=age
    def Print_age(self):
        return self.age

#derived class
class third(second):
    def __init__(self,name,age,study):
        super().__init__(name,age)
        self.study=study
    def Print_study(self):
        return self.study

#object creation
obj=third('rohan',22,'Graduate')
print(obj.Print_name(),obj.Print_age(),obj.Print_study())
```

```
rohan 22 Graduate
```

7 Hierarchical Inheritance:

When more than one derived class are created from a single base this type of inheritance is called hierarchical inheritance.

[24]:
```
# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class1
class Child1(Parent):
    def func2(self):
        print("This function is in child 1.")

# Derivied class2
class Child2(Parent):
    def func3(self):
        print("This function is in child 2.")

object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

```
This function is in parent class.
This function is in child 1.
This function is in parent class.
This function is in child 2.
```

8 Multiple Inheritance:

When a class can be derived from more than one base class this type of inheritance is called multiple inheritances.

class Mother: mothername = " "

```
def mother(self):
    print(self.mothername)
```

#Base class2 class Father: fathername = " "

```
def father(self):
    print(self.fathername)
```

#Derived class class Son(Mother, Father): def parents(self): print("Father :", self.fathername) print("Mother :", self.mothername)

#Driver's code s1 = Son() s1.fathername = "Rohan" s1.mothername = "Geeta" s1.parents()

9 Polymorphism

[ ]:
```
Polymorphism means having many forms.
```

[31]:
```
class bird:
    def intro(self):
        print("this is a bird class")
    def can_fly(self):
        print('some of them can fly some cannot')
class parrot(bird):
    def can_fly(self):
        print("Parrot can fly")
class ostrich(bird):
    def can_fly(self):
        print("ostrich can not fly")
obj_bird=bird()
obj_parr=parrot()
obj_ost=ostrich()

obj_bird.intro()
obj_bird.can_fly()

obj_parr.can_fly()
obj_ost.can_fly()
```

```
this is a bird class
some of them can fly some cannot
Parrot can fly
ostrich can not fly
```

10 Encapsulation

It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as private variables.

[33]:
```
# Creating a Base class
class Base:
    def __init__(self):
        self.a = "GeeksforGeeks"
        self.__c = "GeeksforGeeks"

# Creating a derived class
class Derived(Base):
    def __init__(self):
```

```
        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__c)


# Driver code
obj1 = Base()
print(obj1.a)

# if we print(obj1.c) then it will give attribute error
```

```
GeeksforGeeks
```

11 Abstraction

Abstraction is used to hide the internal functionality of the function from the users. The users only interact with the basic implementation of the function, but inner working is hidden. User is familiar with that "what function does" but they don't know "how it does."