06 OOPs in Python

March 15, 2023

0.1 What Is Object-Oriented Programming in Python?

Object-oriented programming is a programming paradigm that provides a means of structuring programs so that **properties and behaviors** are bundled into individual objects.

For instance, an object could represent a person with properties like a **name, age, and address** and behaviors such as **walking, talking, breathing, and running**. Or it could represent an email with properties like a recipient list, subject, and body and behaviors like adding attachments and sending.

Another common programming paradigm is **procedural programming**, which structures a program like a recipe in that it provides a set of steps, in the form of functions and code blocks, that flow sequentially in order to complete a task.

The key takeaway is that objects are at the center of object-oriented programming in Python, not only representing the data, as in procedural programming, but in the overall structure of the program as well.

0.2 Define a Class in Python

Primitive data structures—like numbers, strings, and lists—are designed to represent simple pieces of information, such as the cost of an apple, the name of a poem, or your favorite colors, respectively. What if you want to represent something more complex?

[3]:
```
Stud1=["Ashwini",21,"MSc-II","2001-01-01",84,45,57]
Stud2=["Rashmi",22,"MSc-II","2000-01-01",84,76,87]
Stud3=["Chanchal",23,"MSc-I","1999-01-01",88,45,64]
Stud4=["Adip",24,"MSc-I","1998-01-01",84,89,64]
```

[4]:
```
sum(Stud1[4:])/300*100
```

[4]:
```
62.0
```

[6]:
```
class stud:
    Name='xxx'
    Age=0
    div='yyy'
    Marks=[0,0,0]

    def cre(self,nam,a,d,e):
```

```
        self.Name=nam
        self.Age=a
        self.div=d
        self.Marks=e

    def prt(self):
        print("Object Created",self.Name,self.Age,self.div)
        print("Percentages are as follows:", sum(self.Marks)/300*100)

S1=stud()
S1.prt()

S2=stud()
S2.cre("Meena",21,"MSc-2",[48,98,98])
S2.prt()
stud.prt(S2)
```

```
Object Created xxx 0 yyy
Percentages are as follows: 0.0
Object Created Meena 21 MSc-2
Percentages are as follows: 81.33333333333333
Object Created Meena 21 MSc-2
Percentages are as follows: 81.33333333333333
```

[9]:
```
x=[1,2,3]
type(x)
x.append(4)
x
```

[9]:
```
[1, 2, 3, 4]
```

[21]:
```
class cmpx:

    def __init__(self,a,b):
        self.real=a
        self.img=b

    def pr(self):
        if self.img>0:
            print('%1.2f + %1.0f i'%(self.real,self.img))
        else:
            print('%1.2f %1.0f i'%(self.real,self.img))

    def conj(self):
        temp=cmpx(self.real,-self.img)
        return temp

    def add(self,b):
```

```
        temp=cmpx(self.real+b.real,self.img+b.img)
        return temp

    def sub(self,b):
        temp=cmpx(self.real-b.real,self.img-b.img)
        return temp

    def mult(self,b):
        temp=cmpx(self.real*b.real- self.img*b.img ,self.real*b.img + self.
↪img*b.real)
        return temp

a=cmpx(4,3)
a.pr()
b=cmpx(2,5)
b.pr()
c=a.sub(b)
c.pr()
```

```
4.00 + 3 i
2.00 + 5 i
2.00 -2 i
```

[23]:
```
a.conj().pr()
```

```
4.00 -3 i
```

[ ]:

[31]:
```
class cmpx:
    def __init__(self,a,b=2):     # Constructor
        self.real=a
        self.img=b
        # print("Complex number created")

    def pr(self):
        if(self.real!=0 and self.img!=0):
            print("Complex number is ",self.real,"+ i",self.img)
        elif(self.real!=0):
            print("Real number is ",self.real)
        else:
            print("Complex number is i",self.img)

    def conj(self):
        print("Complex Conjugate is ",self.real,"- i",self.img)
        temp=cmpx(self.real,-self.img)
        return(temp)
```

```
    def add(self,c4):
        temp=cmpx(self.real+c4.real,self.img+c4.img)
        return temp

    def mult(self,obj):
        temp=cmpx(self.real*obj.real-self.img* obj.img,self.real*obj.img+self.
↪img* obj.img)
        return temp

#    a+ib  c+id
#    (ac-bd)+i*(ad+bc)

#    (self.a*obj.c-self.b*obj.d)(self.a*obj.d+self.b*obj.c)



c1=cmpx(2,3) # Instance/Object
c2=cmpx(4,2)
c3=c1.conj()
c4=c1.mult(c2)
c5=c1.add(c2)
#c5.pr()
print("C1:")
c1.pr()
print("C2:")
c2.pr()
print("C4:")
c4.pr()
```

```
Complex Conjugate is  2 - i 3
C1:
Complex number is  2 + i 3
C2:
Complex number is  4 + i 2
C4:
Complex number is  2 + i 10
```

0.3 Classes vs Instances?

Classes are used to create user-defined data structures. Classes define functions called methods, which identify the behaviors and actions that an object created from the class can perform with its data. A class is a blueprint for how something should be defined. It doesn't actually contain any data.

0.4 How to Define a Class?

All class definitions start with the class keyword, which is followed by the name of the class and a colon. Any code that is indented below the class definition is considered part of the class's body.

Class attributes are defined directly beneath the first line of the class name and are indented by four spaces. They must always be assigned an initial value. When an instance of the class is created, class attributes are automatically created and assigned to their initial values.

Use class attributes to define properties that should have the same value for every class instance. Use instance attributes for properties that vary from one instance to another.

0.5 Instantiate an Object in Python

Creating a new object from a class is called instantiating an object.

0.6 Instance Methods

Instance methods are functions that are defined inside a class and can only be called from an instance of that class. Just like .___init___(), an instance method's first parameter is always self.

[1]:
```
class stud:
    def __init__(self,name,year,a=16,b=16,c=16):
        self.name=name
        self.brth=year
        self.age=2022-int(self.brth)
        self.a,self.b,self.c=a,b,c

    def pr(self):
        self.result()
        print("Hello %s, your age is %d years and your percentage are %f"%(self.
 ↪name,self.age,self.prct))

    def result(self):
        self.prct=(self.a+self.b+self.c)/300*100

    def updateMarks(self,a,b,c):
        self.a,self.b,self.c=a,b,c

s1=stud("Chetan",2000,56,87,23)

s1.pr()

s1.updateMarks(87,86,85)
s1.pr()
```

```
Hello Chetan, your age is 22 years and your percentage are 55.333333
Hello Chetan, your age is 22 years and your percentage are 86.000000
```

0.7 Inherit From Other Classes in Python

Inheritance is the process by which one class takes on the **attributes** and **methods of parent classes**. Newly formed classes are called child classes, and the classes that child classes are derived from are called parent classes.

Child classes can override or extend the **attributes** and **methods of parent classes**. In other words, child classes inherit all of the parent's attributes and methods but can also specify attributes and methods that are unique to themselves.

[24]:
```
class prnt1:
    def fun1(self):
        print("Parent 1 Function 1 Called")
    def fun2(self):
        print("Parent 1 Function 2 Called")
    def fun9(self):
        print("Parent 1 Function 9 Called")

class chld1(prnt1):# Single Inheritance
    def fun1(self):
        print("Child 1 Function 1 Called")
    def fun3(self):
        print("Child 1 Function 3 Called")

A=chld1()
A.fun1()
A.fun2()
A.fun3()
```

```
Child 1 Function 1 Called
Parent 1 Function 2 Called
Child 1 Function 3 Called
```

1 Multiple Inheritance

[25]:
```
class prnt2:
    def fun5(self):
        print("Parent 2 Function 5 Called")
    def fun6(self):
        print("Parent 2 Function 6 Called")
    def fun9(self):
        print("Parent 2 Function 9 Called")

class chld2(prnt1,prnt2):# Multiple Inheritance
    def fun1(self):
        print("Child 2 Function 1 Called")
    def fun7(self):
        print("Child 2 Function 7 Called")
    def fun8(self):
        print("Child 2 Function 8 Called")

B=chld2()
B.fun2()
```

```
B.fun6()
B.fun8()
B.fun9()
```

```
Parent 1 Function 2 Called
Parent 2 Function 6 Called
Child 2 Function 8 Called
Parent 1 Function 9 Called
```

[26]:
```
class grandchld(chld1):  # Multilevel
    def fun10(self):
        print('Grandchild 1 function 10')

C=grandchld()
C.fun1()
C.fun2()
C.fun3()
C.fun10()
```

```
Child 1 Function 1 Called
Parent 1 Function 2 Called
Child 1 Function 3 Called
Grandchild 1 function 10
```

2 Hybrid Inheritance

[49]:
```
class Student():
    def fun1(self):
        print('Student fun 1')

class LibStudent(Student):
    def fun2(self):
        print('Student fun 2')

class FinStudent(Student):
    def fun3(self):
        print('Student fun 3')

class NoDuesStudent(LibStudent,FinStudent):
    def fun4(self):
        print('Student fun 4')

Rahul=NoDuesStudent()
Rahul.fun1()
Rahul.fun2()
```

```
Student fun 1
Student fun 2
```

2.1 Parent Classes vs Child Classes

2.2 Extend the Functionality of a Parent Class

There are mainly two types of inheritance, a combination of which yields every other type.

1. Single Inheritance: One base class, inherited by one derived class. This is the simplest type of inheritance. Also, the minimal possible one. The derived class automatically invokes the base class constructor.
2. Multiple Inheritance: Multiple base classes inherited by one derived class. The base class constructors are invoked in the order in which classes were derived.

Derived Types of Inheritance

Combining the above two forms of inheritance can lead to the following types of inheritance:

1. Hierarchical Inheritance: One base class inherited by multiple derived classes. Each derived class will work independently, but objects share the class variables across different classes.
2. Multi-level Inheritance: A derived class serving as a base class for another derived class. The base class constructors are invoked recursively in this case.
3. Hybrid Inheritance: A combination of multiple instances of the above-mentioned types of inheritance. This could lead to any imaginable combination of classes.

[ ]: