07_Classes

April 15, 2020

1 Classes

A class is declared as follows

class class_name:

`method(s)`

[1]:
```
class FirstClass:
    pass
```

pass in python means do nothing.

Above, a class object named "FirstClass" is declared, now consider a "egclass" which has all the characteristics of "FirstClass". So all you have to do is, equate the "egclass" to "FirstClass". In python jargon this is called as creating an instance. "egclass" is the instance of "FirstClass"

[2]:
```
egclass = FirstClass() # creating object
```

[3]:
```
type(egclass)
```

[3]:
```
__main__.FirstClass
```

Now let us add some "functionality" to the class. So that our "FirstClass" is defined in a better way. A function inside a class is called as a "Method" of that class

Most of the classes will have a function named "___init___". These are called as magic methods. In this method, you basically initialize the variables of that class or any other initial program logic which is applicable to all methods is specified. A variable inside a class is called an attribute.

These helps simplify the process of initializing a instance. For example,

Without the use of magic method or ___init___ which is otherwise called as constructors. One had to define a **init( )** method and call the **init( )** function.

[ ]:
```
eg0 = FirstClass()
eg0.__init__()
```

We will make our "FirstClass" to accept two variables - name and age.

[4]:
```
class FirstClass:
    def __init__(self,name,age):
```

```
        self.name = name
        self.age = age
```

Now that we have defined a function and added the *__init__* method. We can create an instance of FirstClass which now accepts 2 arguments.

[5]:
```
eg1 = FirstClass('ram',24)
eg2 = FirstClass('ravi',25)
```

[6]:
```
print (eg1.name, eg1.age)
print (eg2.name, eg2.age)
```

```
ram 24
ravi 25
```

dir( ) function comes very handy in looking into what the class contains and what all method it offers

dir( ) of an instance also shows it's defined attributes.

[7]:
```
dir(eg1)
```

[7]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
```

```
 'age',
 'name']
```

Let us add some more methods to FirstClass.

[8]:
```
class FirstClass:
    def __init__(self,name,symbol):
        self.name = name
        self.symbol = symbol
    def square(self):
        return self.symbol * self.symbol
    def cube(self):
        return self.symbol * self.symbol * self.symbol
    def multiply(self, x):
        return self.symbol * x
```

[9]:
```
eg4 = FirstClass('Five',5)
```

[10]:
```
print("name:",eg4.name, "Symbol:", eg4.symbol)
```

```
name: Five Symbol: 5
```

[11]:
```
print (eg4.square()) # fn calling
print (eg4.cube())
```

```
25
125
```

[12]:
```
eg4.multiply(3)
```

[12]: 15

The above can also be written as,

[13]:
```
FirstClass.multiply(eg4,2)
```

[13]: 10

1.1 Example: Simple Caculator

[14]:
```
class cal:
    def __init__(self,a,b):
        self.a=a
        self.b=b
    def add(self):
        return self.a+self.b
    def mul(self):
        return self.a*self.b
```

```
    def div(self):
        return self.a/self.b
    def sub(self):
        return self.a-self.b

a=int(input("Enter first number: "))
b=int(input("Enter second number: "))

obj=cal(a,b)
choice=1
while choice!=0:
    print("0. Exit")
    print("1. Add")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    choice=int(input("Enter choice: "))
    if choice==1:
        print("Result: ",obj.add())
    elif choice==2:
        print("Result: ",obj.sub())
    elif choice==3:
        print("Result: ",obj.mul())
    elif choice==4:
        print("Result: ",round(obj.div(),2))
    elif choice==0:
        print("Exiting!")
    else:
        print("Invalid choice!!")
print()
```

```
Enter first number: 2
Enter second number: 3
0. Exit
1. Add
2. Subtraction
3. Multiplication
4. Division
Enter choice: 1
Result:  5
0. Exit
1. Add
2. Subtraction
3. Multiplication
4. Division
Enter choice: 3
Result:  6
```

```
0. Exit
1. Add
2. Subtraction
3. Multiplication
4. Division
Enter choice: 4
Result:  0.67
0. Exit
1. Add
2. Subtraction
3. Multiplication
4. Division
Enter choice: 0
Exiting!
```

[15]:
```python
class Emp:
    def __init__(self,eid,name,age,salary):
        self.eid=eid
        self.name= name
        self.age=age
        self.salary=salary
    def display(self):
        print("ID=",self.eid)
        print("Name=", self.name)
        print("Age=",self.age)
        print("Salary=",self.salary)

eid = int(input("Enter employee id:"))
name = input("Enter name:")
age = int(input("Enter age:"))
salary=int(input("Enter salary:"))

ob=Emp(eid,name,age,salary)
print()
ob.display()
```

```
Enter employee id:101
Enter name:kumar
Enter age:25
Enter salary:23456

ID= 101
Name= kumar
Age= 25
Salary= 23456
```

[16]:
```
class Emp1:
    def getdata(self,eid,name,age,salary):
        self.eid=eid
        self.name= name
        self.age=age
        self.salary=salary
    def display(self):
        print("ID=",self.eid)
        print("Name=", self.name)
        print("Age=",self.age)
        print("Salary=",self.salary)

eid = int(input("Enter employee id:"))
name = input("Enter name:")
age = int(input("Enter age:"))
salary=int(input("Enter salary:"))

ob = Emp1()
ob.getdata(eid,name,age,salary)
print()
ob.display()
```

```
Enter employee id:101
Enter name:kumar
Enter age:25
Enter salary:23456

ID= 101
Name= kumar
Age= 25
Salary= 23456
```

[17]:
```
class Emp2:
    def getdata(self):
        self.eid=int(input("Enter employee id:"))
        self.name= input("Enter name:")
        self.age= int(input("Enter age:"))
        self.salary=int(input("Enter salary:"))
    def display(self):
        print("ID=",self.eid)
        print("Name=", self.name)
        print("Age=",self.age)
        print("Salary=",self.salary)

ob = Emp2()
ob.getdata()
print()
```

```
ob.display()
```

```
Enter employee id:102
Enter name:xyz
Enter age:23
Enter salary:1234

ID= 102
Name= xyz
Age= 23
Salary= 1234
```

1.2 Inheritance

There might be cases where a new class would have all the previous characteristics of an already defined class. So the new class can "inherit" the previous class and add it's own methods to it. This is called as inheritance.

1.2.1 Example 1:

Consider class SoftwareEngineer which has a method salary.

[18]:
```
class SoftwareEngineer:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def salary(self, value):
        self.money = value
        print (self.name,"earns",self.money)
```

[19]:
```
a = SoftwareEngineer('Kumar',26)
```

[20]:
```
a.salary(40000)
```

```
Kumar earns 40000
```

[21]:
```
dir(a)
```

[21]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
```

```
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'age',
 'money',
 'name',
 'salary']
```

Now consider another class Artist which tells us about the amount of money an artist earns and his artform.

[22]:
```
class Artist:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def money(self,value):
        self.money = value
        print (self.name,"earns",self.money)
    def artform(self, job):
        self.job = job
        print (self.name,"is a", self.job)
```

[23]:
```
b = Artist('Nitin',20)
```

[24]:
```
b.money(50000)
b.artform('Politician')
```

```
Nitin earns 50000
Nitin is a Politician
```

[25]:
```
dir(b)
```

[25]:
```
['__class__',
 '__delattr__',
 '__dict__',
```

```
'__dir__',
'__doc__',
'__eq__',
'__format__',
'__ge__',
'__getattribute__',
'__gt__',
'__hash__',
'__init__',
'__init_subclass__',
'__le__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__',
'age',
'artform',
'job',
'money',
'name']
```

money method and salary method are the same. So we can generalize the method to salary and inherit the SoftwareEngineer class to Artist class. Now the artist class becomes,

[26]:
```
class Artist(SoftwareEngineer):
    def artform(self, job):
        self.job = job
        print (self.name,"is a", self.job)
```

[27]:
```
c = Artist('Nishanth',21)
```

[28]:
```
c.salary(60000)
c.artform('Dancer')
```

```
Nishanth earns 60000
Nishanth is a Dancer
```

[29]:
```
dir(c)
```

[29]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'age',
 'artform',
 'job',
 'money',
 'name',
 'salary']
```

Suppose say while inheriting a particular method is not suitable for the new class. One can override this method by defining again that method with the same name inside the new class.

[30]:
```
class Artist(SoftwareEngineer):
    def artform(self, job):
        self.job = job
        print (self.name,"is a", self.job)
    def salary(self, value):
        self.money = value
        print (self.name,"earns",self.money)
        print ("I am overriding the SoftwareEngineer class's salary method")
```

[31]:
```
c = Artist('Nishanth',21)
```

[32]:
```
c.salary(60000)
c.artform('Dancer')
```

```
Nishanth earns 60000
I am overriding the SoftwareEngineer class's salary method
Nishanth is a Dancer
```

1.2.2 Example 2:

[36]:
```
class A:
    def disp(self):
        print("disp fn of A")
```

[37]:
```
class B(A):
    def disp(self):
        super().disp() # calls disp() fn class A
        print("disp fn of B")
```

[38]:
```
b = B()
b.disp()
```

```
disp fn of A
disp fn of B
```

[39]:
```
class C(B):
    def disp(self):
        super().disp()
        print("disp fn of C")
```

[40]:
```
c = C()
c.disp()
```

```
disp fn of A
disp fn of B
disp fn of C
```

1.3 Inheritance Types

1. Single Inheritance: A->B
2. Multiple Inheritance: A,B->C
3. Multilevel Inheritance: A->B->C
4. Hirarchical Inheritance: A->B,C
5. Hybrid Inheritance: Combination of above

2 Where to go from here?

Practice alone can help you get the hang of python. Give your self problem statements and solve them. You can also sign up to any competitive coding platform for problem statements. The more

you code the more you discover and the more you start appreciating the language.

Now that you have been introduced to python, You can try out the different python libraries in the field of your interest. I highly recommend you to check out this curated list of Python frameworks, libraries and software http://awesome-python.com

The official python documentation : https://docs.python.org/3/