Python 101

Object Oriented Programming In Python (Part - 1)

Contents

What is Object Oriented Programming?

As we know an object in English means anything that can be observed.

In programming Objects mean the things that are entities involved in our whole program.

When we try to develop our programs focused on the idea of dealing with objects instead of the “logic” then such Object-centric approach is called Object Oriented Programming.

What are Objects and Classes?

Let’s think about a programming problem where we need to create a program for 2D-Geometry.

Ok so here we know we will be dealing with Circles, Triangles, Rectangles, Squares etc. now there can be circles of 1 unit radius, 2 unit radius, n-unit radius similarly Triangles, Rectangles and Squares all can have uncountable types based on their dimensions. Can we possibly make a program that contains all of the possibilities?

Absolutely No. So We need some way by which we can generalize all of these shapes.

What are Objects and Classes? contd..

What about we give a definition of What a **Geometric Shape** is and then create a specific type of Geometric Shape based on the dimensions and type we are required to based on user input in our program?

This way we will not have to define all of the possible Geometric Shapes but still we can **create a new shape** based on user input.

So basically we need **a generalized definition for all of our Geometric Shapes.**

This Generalized Definition of objects is called Class. Based on this class we can create whatever objects we might require.

What are Objects and Classes? contd..

More Formally,

Classes are a blueprint or generalized template whose instances are called Objects.

This process of identifying the objects the relation between them the properties of the objects etc is called Data Modelling.

We can also understand from this that our primary focus should be defining a good class that is generalized enough for Object Oriented Programming.

What are Objects and Classes? contd..

Every object has some attributes, those attributes can be some values or some activity that the object performs.

Like an object of Dog has some values associated with it like color, breed, weight, name etc. whereas it also has some activities associated with it like barking, eating, running etc.

These values that are associated with objects are called **Data Members** and the associated activities that objects can perform are called **Methods.**

What are Objects and Classes? contd..

Speaking more practically, data members are variables in the object and methods are the functions in the object that perform some set of instructions based on or using those variables of data members.

eg. For a Geometrical Shape the data members can be type, dimension etc. and the methods can be area, perimeter etc.

Classes in Python

In order to define a class with name **ClassName** attributes **attr1, attr2**.. methods **method1, method2**.. We use following syntax-

```
class ClassName:

    attr1 = value1

    attr2 = value2

    def method1(self):

        METHOD_1_CODE_BLOCK...
```

Try out the following code on the interpreter

```
>>> class ClassName:
. . . .     pass
>>> object1_for_ClassName = ClassName()
>>> object2_for_ClassName = ClassName()

# Are both of the objects different?

>>> id(object1_for_ClassName) != id(object2_for_ClassName)
True ← as id are not equal means they indeed are different.

>>> class ClassName:
. . . .    def method1(self):
. . . .         print(“Method 1 was called”)

>>> object_for_ClassName = ClassName()
>>> object_for_ClassName.method1()
Method1 was called
```

To check if a method is of a class we use **isinstance method**

```
>>> isinstance(object_for_ClassName, ClassName) ← Boolean value.
```

How to access an object’s own data members inside the object methods?

We know that methods are functions inside an object that perform some set of instructions based on an object’s data members.

Now the problem here is that how can we **consistently access** the data members of an object.

We can do so if we had the reference of our object inside our object itself.

Try out the following code on the interpreter

```
>>> class ClassName:
. . . .    attr = 1
. . . .    def method():
. . . .        pass
>>> object_of_ClassName = ClassName()
>>> object_of_ClassName().method() ← we didn't pass any argument
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: method() takes 0 positional arguments but 1 was given
```

We didn't pass any argument but its still showing that 1 positional argument was given. It means Python is somehow passing something to the methods itself. Let's try seeing what is it.

```
>>> class ClassName:
. . . .     attr = 1
. . . .     def method(pythons_positional_parameter):
. . . .         return pythons_positional_parameter
>>> object_of_ClassName = ClassName()
>>> object_of_ClassName.method()
<__main__.ClassName object at 0x7fef94c8bb38>
>>> id(object_of_ClassName.method()) == id(object_of_ClassName)
True
```

How to access an object's own data members inside the object methods?

contd..

So as we have the first argument to each object method as the object's reference we can access object's own data members using that argument.

Try out the following code on the interpreter

```
>>> class ClassName:
. . . .    attr = 1
. . . .    def method(self):
. . . .        print(self.attr)
>>> object_of_ClassName = ClassName()
>>> object_of_ClassName.method()
1
```

How to access an object's own data members inside the object methods?

Checking equality of two objects

Calculating the id and then comparing the id for object's equivalence is a redundant thing to do.

To resolve this python has a keyword **is**.

If we need to check if two objects are equal we can do so by using **is.**

```
Object1 is Object2
```

It returns Boolean.

How is an object created by class definition?

When we type **ClassName()** Python tries to initialize the class to create an object using the parameters that we pass inside the paranthesis, Although till now we haven't passed any parameter to **ClassName** but we can surely call it as **ClassName(arg1, arg2, arg3..., argn)**. Now the question is where do the passed parameters go?

We can surely say that parameters are passed to functions not to classes. Now the classes have the equivalent of functions called methods. So Python must have been passing the parameters to some method of class. But to which method? Python passes it to a method called **__init__** which **exists** in every class **implicitly**.

Try out the following code on the interpreter

```
>>> class ClassName:
. . . .     def __init__(self, arg1):
. . . .          print('Inside the __init__ method of ClassName')
. . . .          printl('The arg1 passed =',arg1)
. . . .
>>> ClassName(1)
Inside the __init__ method of ClassName
The arg1 passed = 1
```

Instance and Static attributes in Python and classmethods

Instance attributes in Python are those which can only be accessed by the Objects. Thus called **Instance attributes.**

Static attributes in Python are those which can accessed directly using the Class definition and DO NOT require class to be instantiated.

We typically know that static variables do not change their value. It is not true in Python. Their values can be changed.

Classmethods are those methods to which instead of the instance the first attribute is passed as the Class of the object. Like staticmethods these also do not require an instance.

Try out the following code on the interpreter

```
>>> class ClassName:
. . . .     static_variable = 1
. . . .     def __init__(self):
. . . .         self.instance_variable1 = 1
. . . .         self.instance_variable2 = 2

>>> obj = ClassName()
>>> obj.static_variable
1
>>> obj.instance_variable1
1
>>> obj.instance_variable2
2
>>> ClassName.static_variable
1
>>> ClassName.instance_variable1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: type object 'X' has no attribute 'x'
```

Try out the following code on the interpreter

```
>>> class ClassName:
. . . .     static_variable = 1
. . . .    @staticmethod
. . . .    def static_method(): ← Because static methods are not specific to the Objects so they do not get the self
. . . .        ClassName.static_variable += 1                    and due to that they can not access instance
. . . .        print(ClassName.static_variable)                   variables
. . . .    def __init__(self):
. . . .        self.instance_variable = 2
>>> obj = ClassName()
>>> obj.static_variable
1
>>> obj.instance_variable
2
>>> ClassName.static_variable
1
>>> ClassName.instance_variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: type object 'X' has no attribute 'x'
>>> ClassName.static_method() ← Mind that we did not need to create an instance to call this method.
1
```

Try out the following code on the interpreter

```
>>> class ClassName:
. . . .     @classmethod
. . . .     def class_method(cls):
. . . .         print(cls)
. . . .
>>> obj = ClassName()
>>> obj.class_method()
<class '__main__.ClassName’>

>>> ClassName.class_method()
<class '__main__.ClassName’>
```

Encapsulation in a Python Class using Hidden attributes

Encapsulation means wrapping data in a single unit. In languages like java it is done by using **private** variables. But in python there is nothing like **private** variables. Why? Simply because the language isn't designed in such a manner. *PEP20 (Explicit is better than implicit).*

Instead python uses something called Hidden variables in order to make some variable hidden in python the variable must be **instance variable and its name must start with __**

Try out the following code on the interpreter

```
>>> class ClassName:
. . . .       def __init__(self):
. . . .            self.__hidden_variable = 1
. . . .
>>> obj = ClassName()
>>> obj.__hidden_variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'ClassName' object has no attribute '__hidden_variable'
```

Mind that it is called hidden not private for a reason.

On dir(obj) we can see that there exists a variable named **_ClassName__hidden_variable**
But we didn't define any such data member inside our ClassName class?
This variable is actually the **__hidden_variable .**

```
>>> obj._ClassName__hidden_variable
1
```

So as we can see that python didn't abandon the access to variable but just hid the variable by giving it a different name that's why it should be called Hidden variable instead of private variable.

Encapsulation in a Python Class using Hidden attributes contd..

It should be noted that Python highly discourages using this __variable to hide the variable.

Instead PEP8 recommends using single underscore variables and methods to declare that these methods should not be used by the end-user of our class. Instead they are meant to be used internally by our class.

Encapsulation in a Python Class using Hidden attributes

Getter and Setter for Data Members

It is not a good practice to directly modify the data members in an object. Because direct assignment may lead to inconsistency in objects.

Like assignment of a String value to a variable that's supposed to be an integer would create unpredictable runtime errors.

So we create Getters and Setters corresponding to Data Members to provide validations before updating a data member in object.

We use **@property** decorator for this purpose.

Try out the following code on the interpreter

```
>>> class ClassName:
. . . .     def __init__(self):
. . . .          self._some_property = None
. . . .     @property                    # Getter for property name some_property
. . . .     def some_property(self):
. . . .          print('Getting some property value')
. . . .          return self._some_property
. . . .     @some_property.setter       # Setter for proerty name some_property
. . . .     def some_property(self, some_property_value):
. . . .          print('Setting some_property value=', some_property_value)
. . . .          self._some_property = some_property_value

>>> obj = ClassName()
>>> obj.some_property = 1
Setting some property value= 1
>>> obj.some_property
Getting some property value
1
```

A **Complex** Example of a Class

Implement a class for Complex Numbers.

Data Modelling Step-

1. Decide the data members - There can only be two data members for Complex Numbers ie. real and imaginary
2. Decide the Methods - Let’s implement only three methods for now argument, modulus and conjugate.

Try out the following code on the interpreter

```
>>> import math← arctan (tan inverse) for calculating argument
>>> class Complex:
. . . .    def __init__(self, real, img):
. . . .        self._real = real
. . . .        self._img = img
. . . .
. . . .    def argument(self):
. . . .        return math.atan(self._img/self._real)
. . . .
. . . .     def modulus(self):
. . . .        return (self._real**2 + self._img**2)**0.5
. . . .
. . . .     def conjugate(self):
. . . .        return Complex(-self._real, -self._img)
. . . .
. . . .     @classmethod
. . . .     def create_new_complex(cls, real, img):
. . . .            return cls(real, img)
```

Try out the following code on the interpreter

```
>>> complex = Complex(1,2)
>>> complex.argument()
1.1071487177940904

>>> complex.modulus()
2.23606797749979

>>> complex.conjugate()
<__main__.Complex object at 0x7f759d2e4630>

>>> complex.create_new_complex(10,20)
<__main__.Complex object at 0x7f759d2e4748>
```

Interview Questions

Why is @property decorator used?

Its used to create getter and setters for data members in a python class.

How can we hide a variable in python class ?

Using double underscore before it’s identifier. But its highly discouraged.

What is the name of hidden variable by which it is saved in class?

A hidden variable is saved as _<class_name>__<hidden_var_name>

What is difference between instancemethod, classmethod and staticmethod?

Instance method gets the instance on which it’s called as first argument by default.

Class method gets the class of the instance as it’s first argument when its called by default.

Staticmethod gets nothing in arguments by default.

What is the usage of classmethods?

Classmethods are typically used for creating new objects of the same class as of instance with which they are called on runtime.

What is the difference between == and is ?

== checks for equivalence of values of two objects. Where as **is** checks whether the two objects have the same id.

```
A = [1,2,3]

B = [1,2,3]

A == B ← True (values are same)

A is B ← False (ids are not same, both have different objects in memory)
```

What is use of **isinstance**?

isinstance checks if a object is of a class or not

isinstance(object, class) ← returns Boolean value.