OOPS - Class & Object

July 28, 2022

Object-Oriented Programming in Python

Python is an object-oriented programming language where programming stresses more on objects.

Almost everything in Python is objects.

What is OOP ?

Object-oriented programming (OOP) is a programming concept based on the "Objects" that interact with each other to perform the functions

Which may contain data, in the form of fields, often known as attributes and code, in the form of procedures, often known as methods.

Example : A person is an object which has certain properties such as height, gender, age, etc.It also has certain methods such as move, talk..

Why Object-Oriented Programming Matters ?

They reduce the redundancy of the code by writing clear and re-usable codes

They are easier to visualize because they completely relate to real-world scenarios.

Every object in oops represent a different part of the code and have their own logic and data to communicate with each other

It is more secure

Group together data and behavior in one place

Isolates different parts of the program from each other.

What is Class ?

Class in Python is a collection of objects, we can think of a class as a blueprint or sketch or prototype. It contains all the details of an object.

A class is like a container that holds all of our functions (methods) and we can use this class whenever we need to make a call to a function

A more formal definition would be, that a class is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods)

Some points on Python class :

Classes are created by keyword class.

There is no memory allocation until we create its object.

Attributes are the variables that belong to a class.

Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute

[1]:
```
#To define a class in Python, you use the class keyword followed by the class␣
↪name and a colon
class Person:
   pass
```

What is Object ?

An object is usually an instance of a class. It is used to access everything present inside the class.

An object is container that contains state and behavior.

There can be multiple instances of a class in a program

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.

[2]:
```
#This will create an object named obj of the class Person defined above.
obj = Person()
print(obj)
```

```
<__main__.Person object at 0x000001DE90043070>
```

Constructor in Python

Constructor in Python is a special method which is used to initialize the members of a class during run-time when an object is created.

In Python, we have some special built-in class methods which start with a double underscore (___) and they have a special meaning in Python.

Every class must have a constructor, even if you don't create a constructor explicitly it will create a default constructor by itself.

Properties of Constructor :

If we create three objects, the constructor is called three times and initialize each object.

The **init**( ) function can contain any number of parameters, but the first argument must always be the self variable.

The name of the constructor will always be **init**(self).

[3]:
```
# Adding details to the Person class
class Person:
```

```
    '''Doc Strings : This Class is to store person Details'''
    def __init__(self, name, age, gender):
        #Instance Variables
        self.name = name
        self.age = age
        self.gender = gender
```

[5]:
```
#Our class Person now has three attributes, name, age, and company.
#When we create a new person, we can pass these parameters in to make our␣
 ↪person a bit more interesting!

# Creating our first Person instance
Nit = Person('Nitheesh', 25, 'Male')
print(Nit)
```

```
<__main__.Person object at 0x000001DE90043310>
```

If we observe in the above example, we are not calling the **init**( ) method, because it will be called automatically when we create an object to that class and initialize the data members if any.

Always remember that a constructor will never return any values, hence it does not contain any return statements.

Self in Python

The self in python represents or points the instance which it was called.

self let's Python know that the attribute or methods should be applied to that object (and only that object)

In essence, the self parameter binds the attributes with the given arguments.

When you created the first object above, the variables that were passed in were assigned the self bindings.

So, for example : self.name was assigned the argument of the name parameter

[6]:
```
#The class does not know which variable values belong to which object.
obj1 = Person('Nitheesh', 25, 'Male')
obj2 = Person('Bharat', 27, 'Male')
obj3 = Person('Manikanta', 28, 'Male')
```

[7]:
```
print(obj1.name)
print(obj2.name)
print(obj3.name)
```

```
Nitheesh
Bharat
Manikanta
```

In object orientated programming, you can create many objects from a class.Each object can have unique values for variables.

The class does not know which variable values belong to which object.

By using the self variable, it can set or get the objects variables.Python then knows it should access the variables for that objects.

Python Class and Instance Attributes

There are two main types of attributes contained in Python classes

Class Attributes

Instance attributes

[8]:
```
# Let's load some code again and take a look at the difference between class␣
 ↪and instance attributes

class Person:
    # Class attribute
    species = 'Human'
    def __init__(self, name, age, gender):
        #Instance Variables
        self.name = name
        self.age = age
        self.gender = gender

Nit = Person('Nitheesh', 25, 'Male')
```

So, when should you use one over the other ?

If you want an attribute to be the same for every instance of your class, such as the species attribute in the Person class, then use a class attribute.

This prevents you from needing to pass it in as a value each time you create a class.

If you want an attribute to specific to an object, then use an instance attribute.

This lets you customize the object to meet your needs.

[9]:
```
# x and y are instances of class Person
x = Person('Manikanta', 28, 'Male')
y = Person('Bharat', 27, 'Male')

print(x.species)  # species are class attributes, hence will have same value␣
 ↪for all instances
print(y.species)

# name, gender and age will have different values per instance, because they␣
 ↪are instance attributes
print(f"Hi! My name is {x.name}. I am a {x.gender}, and I am {x.age} years old")
print(f"Hi! My name is {y.name}. I am a {y.gender}, and I am {y.age} years old")
```

```
Human
Human
```

```
Hi! My name is Manikanta. I am a Male, and I am 28 years old
Hi! My name is Bharat. I am a Male, and I am 27 years old
```

Python Functions and Methods

In object-oriented programming, functions also exist. However, methods refer to functions contained in an object.

While a function can be called from anywhere, a class method can only be called from an instance of that class

Because of this, every method is a function but not every function is a method.

[10]:
```
# Writing your first object method
class Person:
    # Class attribute
    species = 'Human'
    def __init__(self, name, age, gender):
        #Instance Variables
        self.name = name
        self.age = age
        self.gender = gender
    #instance method
    def greet(self):
        print('Hi there! My name is ', self.name)

Nit = Person('Nitheesh', 25, 'Male')
Nit.greet()
```

```
Hi there! My name is  Nitheesh
```

Defining an object method is nearly the same as creating a regular function. There are a number of key differences:

The function is defined inside the object

The self argument is required

The first argument is required to be self

[11]:
```
#Let's now create a method that modifies the object itself.

class Person:
    # Class attribute
    species = 'Human'
    def __init__(self, name, age, gender):
        #Instance Variables
        self.name = name
        self.age = age
        self.gender = gender
    #instance method
    def greet(self):
```

```
        print('Hi there! My name is ', self.name)

    def have_birthday(self):
        self.age += 1

Nit = Person('Nitheesh', 25, 'Male')
Nit.greet()
print(Nit.age)          # Returns: 25
Nit.have_birthday()
print(Nit.age)   # Returns: 26
```

```
Hi there! My name is  Nitheesh
25
26
```