Object-Oriented Programming

Object-Oriented Programming(OOP), is all about creating "objects". An object is a group of interrelated variables and functions. These variables are often referred to as properties of the object and functions are referred to as the behavior of the objects. These objects provide a better and clear structure for the program.

For example, a car can be an object. If we consider the car as an object then its properties would be – its color, its model, its price, its brand, etc. And its behavior/function would be acceleration, slowing down, gear change.

Another example- If we consider a dog as an object then its properties would be- his color, his breed, his name, his weight, etc. And his behavior/function would be walking, barking, playing, etc.

What is a Class?

A class is a collection of objects.

```
class class_name:
    class body
```

Example

Consider the case of a car showroom. You want to store the details of each car. Let's start by defining a class first-

```
class car:
    pass
```

Note: I've used the pass statement in place of its body because the main aim was to show how you can define a class and not what it should contain.

Object-Oriented Programming

Objects and object instantiation

When we define a class only the description or a blueprint of the object is created. There is no memory allocation until we create its **object**.
The **objector instance** contains real data or information.

Instantiation is nothing but creating a new object/instance of a class. Let's create the object of the above class we defined-

```
class car:
    pass

c_object=car()
print(c_object)
```

<__main__.car object at 0x0000020616D31F70>

Since our class was empty, it returns the address where the object is stored

Class constructor

Until now we have an empty class Car, time to fill up our class with the properties of the car. The job of the class constructor is to assign the values to the data members of the class when an object of the class is created.

There can be various properties of a car such as its name, color, model, brand name, engine power, weight, price, etc

```
class car:

    def __init__(self,name,color):
        self.name=name
        self.color=color
```

Object-Oriented Programming

So, the properties of the car or any other object must be inside a method that we call **__init__( )**. This __init__() method is also known as **the constructor method**. We call a constructor method whenever an object of the class is constructed.

the parameter of the __init__() method. So, the first parameter of this method has to be self. Then only will the rest of the parameters come.

The two statements inside the constructor method are –

1. **self.name = name**
2. **self.color = color:**

This will create new attributes namely **name** and **color** and then assign the value of the respective parameters to them. The "self" keyword represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class. It is useful in method definitions and in variable initialization. The "self" is explicitly used every time we define a method.

Note: You can create attributes outside of this __init__() method also. But those attributes will be universal to the whole class and you will have to assign the value to them.

Suppose all the cars in your showroom are Sedan and instead of specifying it again and again you can fix the value of car_type as Sedan by creating an attribute outside the __init__().

```
class Car:
    car_type = "Sedan"                  #class attribute
    def __init__(self, name, color):
        self.name = name                #instance attribute
        self.color = color              #instance attribute
```

Object-Oriented Programming

Class methods

We have added the properties of the car. Now it's time to add some behavior. Methods are the functions that we use to describe the behavior of the objects. They are also defined inside a class. Look at the following code-

```
class Car:
    car_type = "Sedan"

    def __init__(self, name, mileage):
        self.name = name
        self.mileage = mileage

    def description(self):
        return f"The {self.name} car gives the mileage of {self.mileage}km/l"

    def max_speed(self, speed):
        return f"The {self.name} runs at the maximum speed of {speed}km/hr"

obj2 = Car("Honda City",24.1)
print(obj2.description())
print(obj2.max_speed(150))
```

The methods defined inside a class other than the constructor method are known as the **instance** methods. Furthermore, we have two instance methods here- **description**() and **max_speed**(). Let's talk about them individually-

- **description**()- This method is returning a string with the description of the car such as the name and its mileage. This method has no additional parameter. This method is using the instance attributes.
- **max_speed**()- This method has one additional parameter and returning a string displaying the car name and its speed.

Object-Oriented Programming

Notice that the additional parameter speed is not using the "self" keyword. Since speed is not an instance variable, we don't use the self keyword as its prefix. Let's create an object for the class described above

```
Skoda = Car("Skoda Octavia",13)
print(Skoda.max_speed(210))
```