Classes and Objects in Python (OOP)

Objects are a representation of the real world objects like cars, dogs, bikes, etc. The objects share two main characteristics: *data* and *behavior*.

Cars have ***data*** like number of wheels, number of doors, seating capacity and also have ***behavior***: accelerate, stop, show how much fuel is missing and so many other.

We call *data* as ***attributes*** and *behavior* as ***methods*** in object oriented programming. Again:

Data → Attributes & Behavior → Methods

And a **Class** is the blueprint from which individual objects are created. In the real world we often find many objects with all the same type. Like cars. All the same make and model (have an engine, wheels, doors, …). Each car was built from the same set of blueprints and has the same components.

Python as an Object Oriented programming language has these concepts: ***class*** & ***object***.

A class is a blueprint, a model for its objects.

So again, a class it is just a model, a way to define ***attributes*** and ***behavior*** (as we talk in the ***theory section***). As an example, a Vehicle ***class*** has its own ***attributes*** that defines what is a Vehicle **object**. Number of wheels, type of tank, seating capacity and maximum velocity are all attributes of a vehicle. With this in mind, let's understand Python syntax for classes:

```
class Vehicle:
  pass
```

We define classes with ***class statement*** and that's it. Easy! And objects are instances of a class. We create an instance by calling the class.

```
car = Vehicle()
print(car) # <__main__.Vehicle instance at 0x7fb1de6c2638>
```

Here car is an ***object*** (or instance) of the ***class*** Vehicle.

"A car is an instance of the class of objects known as vehicles."

Remember that our Vehicle ***class*** has 4 ***attributes***: Number of wheels, type of tank, seating capacity and maximum velocity. We set all this ***attributes*** when creating a vehicle ***object***. So here we define our ***class*** to receive data when instantiates it.

```
class Vehicle:
    def __init__(self, number_of_wheels, type_of_tank, seating_capacity,
maximum_velocity):
        self.number_of_wheels = number_of_wheels
        self.type_of_tank = type_of_tank
        self.seating_capacity = seating_capacity
        self.maximum_velocity = maximum_velocity
```

We use the init ***method***. We call it a constructor ***method***. So when create the vehicle ***object***, we can define this ***attributes***. Imagine that we love the **Tesla Model S** and we want to create this kind of ***object***. It has 4 wheels, it is an electric car so the tank's type is electric energy, it has space for 5 seats and the maximum velocity is 250km/hour (155 mph). Let's create this ***object***! :)

```
tesla_model_s = Vehicle(4, 'electric', 5, 250)
```

4 wheels + Electric type of tank + 5 seats + 250km/hour maximum speed.

All attributes set. But how can we access this attributes values? We *send a message to the object asking about them*. We call it a ***method***. It's the ***object***'s ***behavior***. Let's implement it!

```
class Vehicle:
    def __init__(self, number_of_wheels, type_of_tank, seating_capacity,
maximum_velocity):
        self.number_of_wheels = number_of_wheels
        self.type_of_tank = type_of_tank
```

```
        self.seating_capacity = seating_capacity
        self.maximum_velocity = maximum_velocity

    def number_of_wheels(self):
        return self.number_of_wheels

    def set_number_of_wheels(self, number):
        self.number_of_wheels = number
```

This is an implementation of two methods: ***number_of_wheels*** and ***set_number_of_wheels***. We call it getter & setter. Because the first get the attribute value, and the second set a new value for the attribute.

In Python, we can do that using *@property* (*decorators*) to define *getter* and *setters*. Let's see it with code!

```
class Vehicle:
    def __init__(self, number_of_wheels, type_of_tank, seating_capacity,
maximum_velocity):
        self.number_of_wheels = number_of_wheels
        self.type_of_tank = type_of_tank
        self.seating_capacity = seating_capacity
        self.maximum_velocity = maximum_velocity

    @property
    def number_of_wheels(self):
        return self.number_of_wheels

    @number_of_wheels.setter
    def number_of_wheels(self, number):
        self.number_of_wheels = number
```

And we can use this methods as attributes:

```
tesla_model_s = Vehicle(4, 'electric', 5, 250)
print(tesla_model_s.number_of_wheels) # 4
```

```
tesla_model_s.number_of_wheels = 2 # setting number of wheels to 2
print(tesla_model_s.number_of_wheels) # 2
```

This is slightly different than defining methods. The methods work as attributes. For example, when we set the new number of wheels, we don't pass 2 as parameter, but set the value 2 to number_of_wheels. This is one way to write pythonic getter and setter code.

But we can also use methods to other things like "***make_noise***" method. Let's see it!

```
class Vehicle:
    def __init__(self, number_of_wheels, type_of_tank, seating_capacity,
maximum_velocity):
        self.number_of_wheels = number_of_wheels
        self.type_of_tank = type_of_tank
        self.seating_capacity = seating_capacity
        self.maximum_velocity = maximum_velocity

    def make_noise(self):
        print('VRUUUUUUUM')
```

When we call this method, it just returns a string ***"VRRRRUUUUM"***.

```
tesla_model_s = Vehicle(4, 'electric', 5, 250)
tesla_model_s.make_noise() # VRUUUUUUUM
```

That's it!

We learnt a lot of things about Python object oriented programming:

- Objects & Classes
- Attributes as objects' data
- Methods as objects' behavior
- Using Python getters and setters & property decorator