Python Attributes and Python Methods

A Deeper Look

There is a LOT to cover when it comes to classes in Python, so let us get into it.

Reminder: A class allows you to create a new type of data with methods to go along with it. Classes have attributes and classes have methods (functions). There are two types of attributes, class attributes and instance attributes. Class attributes are attributes which are owned by the class itself. These attributes will be shared by all the instances of the class. Therefore, they have the same value for every instance. We define class attributes outside all the methods – usually they are placed at the top of the class, right below the header.

Every object that is created has its own copy of the instance attribute.

Here is an example class for us to review:

```
class car():

    # init method or constructor
    def __init__(self, model, color):
        self.model = model
        self.color = color


    def show(self):
        print("Model is ", self.model )
        print("Color is", self.color )


# both objects have different self which
# contain their attributes
audi = car("audi a4", "blue")
ferrari = car("ferrari 488", "green")


audi.show()    # same output as car.show(audi)
```

```
ferrari.show()  # same output as car.show(ferrari)
```

Did you notice something different?

The number of arguments in calling a class method is one less from the number of parameters in the definition of the method. The __init__ method has 3 parameters, but the main method is sending only 2 arguments when creating a new car object.

Why?

Because the variable "self" is always included in the method declaration. This variable sends the car object from the main method to be manipulated. The value of "self" will always be the object name and is a given when sending arguments to a class method or class function.

Okay, how do you create a new object?

As you can see from the example, a new object is created by doing

```
audi = car("audi a4", "blue")
```

When this is passed to the __init__ method, the "self" parameter will be "audi", the "model" parameter will be "audi a4", and the "color" parameter will be "blue".

If you do not have an __init__ method, it is fine – you can assign attributes directly to an object inside the main program. Just be sure that you assign ALL the attributes of an object within the main program before you try to do any type of manipulation within class methods. If you do not do this, then you will receive an error.

Let us assume for a moment that there was NOT an __init__ method, and all the variables had to be assigned in the main program.

```
class car():
        def show(self):
                print("Model is ", self.model )
                print("Color is ", self.color )
                print("Engine is ", self.engine)

# both objects have different self which
# contain their attributes
```

```
audi = car()     # creates a car object
audi.model = "audi a4"
audi.color = "blue"

audi.show()     # same output as car.show(audi)
```

Oh no!!

This will cause an error message because the main program did not define the “engine” attribute.

To fix this, we will have to add another attribute

```
audi.engine = “v8”
```

Now the program will execute correctly.

A quick note on the type() function: The type() function returns the type of the specified object.

If we want to find out the type of the object we created, we must type in the following command:

```
type(audi)
```

This would provide us with the following output:

```
__main__.car
```

What does this mean?

It means that audi belongs to the “car” class from the __main__ module. It is a user defined data type.