## ****Classes and instances****

* Classes define the behavior of all instances of a specific class. In Python, the code defining a class is, itself, an object; classes can be used without instantiating a single object, such as when using static methods
* Remember, each variable of a specific class is an instance or object.
* In Python, "getters and setters" are methods used for controlling access to an object's attributes. The getter method retrieves the value of an attribute, while the setter method sets or changes the attribute's value, often including some sort of validation or modification to the data before setting the value.
* You can access an instance's attribute, like **name**, by calling self.name within the class methods, or **<instance>.name** outside the class, where **<instance>** is the specific instance of the class you're working with.Objects can have attributes, which store information about the object.
* You can make objects do work by calling their methods.
* The first parameter of the methods, **(self)**, represents the current instance.
* Methods are just like functions, but they can only be used through a class.
* You can use class methods in conjunction with a class variable to track the number of instances of a class, incrementing the class variable each time an instance is created in the class's **\_\_init\_\_** method.

## ****Special methods****

* Special methods start and end with **\_\_**.
* Special methods have specific names, like **\_\_init\_\_** for the constructor or **\_\_str\_\_** for the conversion to string.
* The methods **\_\_str\_\_** and **\_\_repr\_\_** allow you to define human-readable and unambiguous string representations of your objects, respectively.
* By defining methods like **\_\_eq\_\_**, **\_\_ne\_\_**, **\_\_lt\_\_**, **\_\_gt\_\_**, **\_\_le\_\_**, and **\_\_ge\_\_**, you can control how **objects** of your **class** are compared.