Classes

Although you don't need to use object-oriented programming in this course, it is good to have a rough understanding of what classes do and how to interact with them in order to understand what some of the modules we will use are doing in the background.

Classes encapsulate functionality. They have attributes and methods. Attributes are what the function "has" or "knows" (variables), methods are what a class can do (functions). Think for example of example of a car. A car has a number of things it "has", like the number of seats, horsepower, ... and it also can do something, like accelerating, decelerating, use the brakes, steer, ...

Let's use a much simpler example here. A Window has a status that indicates whether it is opened or closed, and it can open, or close. As a class, this looks like the following:

In [1]:

```
class Window:               # class names are Capitalized! This is convention.
    def __init__(self):
        self.open = False

    def openUp(self):
        if self.open:
            print("Window already open.")
        else:
            self.open = True

    def closeDown(self):
        if not self.open:
            print("Window already closed.")
        else:
            self.open = False

    def checkWindowStatus(self):
        if self.open:
            print("Window is open.")
        else:
            print("Window is closed.")
```

This is like a blueprint for an *instance* of a class, called an **object**. Objects are instantiated from classes similarly to initializing variables:

In [2]:

```
# here, we instantiate a Window in the variable "window"
window = Window()

print(type(window))

# here, we use the methods of the object
window.openUp()
window.openUp()
window.checkWindowStatus()
window.closeDown()
window.checkWindowStatus()
```

```
<class '__main__.Window'>
Window already open.
Window is open.
Window is closed.
```

Classes can also be instantiated with arguments:

In [2]:

```
class Window:
    def __init__(self, opened):    # __init__ function must be implemented
        self.open = opened

    def openUp(self):
        if self.open:
            print("Window already open.")
        else:
            self.open = True

    def closeDown(self):
        if not self.open:
            print("Window already closed.")
        else:
            self.open = False

    def checkWindowStatus(self):
        if self.open:
            print("Window is open.")
        else:
            print("Window is closed.")


window = Window(False)
window.checkWindowStatus()
print(window.open)
```

```
Window is closed.
False
```

This uses the inbuilt special function `__init__` , which is called when an instance of a class is created. This function *must always* be implemented, even if you don't initialize your instance with arguments.

You probably noticed the extensive use of `self`. This *must* be the first argument of every method you implement in a class. The reason is that a class is only a blueprint. When you later create an instance called, say, `w1`, and use a method like `w1.closeDown()` in your code, the `self` will contain your object name. This way, Python knows that when you call `w1.closeDown()`, it need to perform all the actions on attributes produced by that method on those attributes belonging to `w1`. This does take some getting used to, so a little bit of practicing with your own ideas for classes goes a long way.

This is hopefully all you need to know to follow along with the lecture and the exercises. Of course, this is in no way a comprehensive Python course and we missed out many of the features and comfortable properties Python offers. Since people proficient in the language are currently highly sought for, it is probably wise to accustom oneself with it further. We will learn everything else we need as we move on.