Objects And Classes - I - Class Definition

This lecture will discuss the most important construct in object oriented programming - `classes`. A class can be thought of as a blueprint for creating objects. Each object is described by its attributes and has behavior associated with it.
The behavior is captured through the `methods` of an object while the `attributes` describe different properties of the object. While all objects of a class will have the same set of attributes and methods, the values of the attributes will likely be different for each object.

An object of a class is also called an `instance` of a class and hence the attributes asscociated with an object are also called `instance variables`.

For example:
Class: **Dog**
Attributes: **name, age, breed, color**
Methods: **bark, eat, sleep**

Object: **dog object**
attributes: **name='Kaazi', age=3, breed='labrador', color = 'black'**

In this notebook we will study how

1. class definitions are written
2. attributes (also called instance variables) are associated with a class
3. to write methods for a class
4. to create an object of a class

In this notebook, we will write the class definition for a class called Rectangle. We begin by specifying the requirements for the Rectangle class are as described below.

1. We will create the definition for a simple Rectangle class.
2. This class has two instance variables (or attributes) - length and width
3. We will ensure that the objects are encapsulated by "hiding" the instance variables to the extent that is possible in Python.
4. Since the instance variables are "hidden", this class definition will also require the necessary accessor (get) and mutator (set) methods.
5. We also desire to print out the details (length, width, area, and perimeter) of every rectangle object and need an appropriate method for this.
6. Finally, the class has two additional methods - `computer_area()` and `compute_perimeter()` that will compute and return the appropriate values

Requirement 1. **We will write the first statement for defining the Rectangle class in this cell. Ignoring any import statements your definition may require, a class definition always begins with the word 'class' followed by the name of the class and then a :. If the definition requires any import statements, they will, as usual, go before the class statement.**

In [1]:
```
'''
By convention, the first letter of a class name will be upper case.
The names of objects will always start with lower case letters like we have done for all other var
iables so far.
'''
class Rectangle:
    pass
```

Python provides a bulitin method called __init()__ which is automatically called when a new object is created. In this course, we will ALWAYS redefine the __init()__ method and initialize all instance variables.

Note that every method in a class definition will have at least one parameter. This is by convention called `self`.
In addition, a method in a class may also have other parameters as needed. The other parameters will always be listed after the `self` parameter. The `self` parameter denotes the specific instance of the class that the method should apply to.

Requirement 2. **We will continue with our class definition by redefining the __init()__ method to initialize the instance variables.**

In [2]:

```
 '''
The __init__ method for our Rectangle class will therefore have three parameters: 1) self, 2) leng
th, and 3) width.
While you can use any suitable variable name for length and width, note that in this course,
you will always use the name 'self' for the first parameter.
'''
class Rectangle:

    #Method heading.  The first parameter is self, then we will include the length and width param
eters
    def __init__(self, length, w):

        '''
         Each instance variable name must be prefixed by the word 'self'. This tells Python
exactly which object the
         variable applies to.
         The object's length variable is initialized by assigning it the value provided in the met
hod argument.
         This is also where you will validate your input as needed.
        '''
        self.length = length

        #Similarly for the width
        self.width = w
```

The next two cells discuss how data hiding is achieved in Python. Python does not strictly prohibit users from directly accessing instance variables. However, there are mechanisms in place to restrict such direct access. In this course, we will only use the mechanism described in the second cell.

The convention in Python is to use a leading underscore to name instance variables not meant for general access by clients. Note that the single _ character does not prevent direct access. It is just a convention to let other programmers know that this variable is not intended for direct access.

Requirement 3 - I. "Hide" the instance variables. Note the instance variables are the variables associated with the self parameter. These are the ones that need to be made "hidden" by prefixing with _ .

In [3]:

```
'''
Requirement 3 - I. "Hide" the instance variables.  Note the instance variables are the variables
associated with the self parameter.  These are the ones that need to be made "hidden" by prefixing
with _.
'''
class Rectangle:
    def __init__(self, l, w):

        '''
        Note the single '_' character will signal to other programmers that direct access of the
        variable is not recommended.
        '''
        self._length = l
        self._width = w
```

Python offers another mechanism for making direct access of variables difficult. This is done by prefixing each variable name by two _ (underscore) characters. This is referred to as name mangling. In this course, we will use this mechanism for hiding our variables.

Requirement 3 - II. "Hide" the instance variables. Note the instance variables are the variables associated with the self parameter. These are the ones that need to be made "hidden" by prefixing with __.

In [4]:

```
'''
Requirement 3 - II. "Hide" the instance variables.  Note the instance variables are the variables
associated with the self parameter.  These are the ones that need to be made "hidden" by prefixing
with __.
'''
class Rectangle:
    def __init__(self, l, w):
        self.__length = l  #Note the two underscore characters before the instance variable name
        self.__width = w
```

Since we do not want our instance variables to be directly accessed by outside methods, we will need to create methods that will enable us to access or make changes to the instance variables. To retrieve the value of an instance variable, we will need an `accessor` method. These are also call `getters` or `get methods`. To change the value of an instance variable, we will need a `mutator` method. These are also call `setters` or `set methods`.

Requirement 4. Redefine the class to include the accessor and mutator methods.

In [5]:

```
'''
Requirement 4.  Redefine the class to include the accessor and mutator methods.
'''

class Rectangle:
    def __init__(self, l, w):
        self.__length = l  #Note the two underscore characters before the instance variable name
        self.__width = w

    '''
    All accessor methods are defined with just the self parameter and will contain a single return
statement to return the value
    of the instance variable.  By convention, we will name the accessor methods starting with the
word 'get' followed by
    underscore and the name of the variable.
    '''

    #This class has two accessor method
    def get_length(self):
        return self.__length

    def get_width(self):
        return self.__width

    '''
    All mutator methods are defined with the self parameter and an additional parameter containing
the new value for
    the instance variable.  The method body will contain an assignment statement to reset the old
value of the instance variable
    with this new value. By convention, we will name the mutator methods starting with the word 's
et' followed by underscore
    and the name of the variables
    '''
    #This class has two mutator methods
    def set_length(self, l):
        self.__length = l

    def set_width(self, w):
        self.__width = w
```

Another common requirement is to print out the values of all instance variables of an object. To do this, we redefine a builtin function called __str__(). This function returns a string value containing the instance variable values as desired. Other methods can then use the print function to automatically print out the data.

Requirement 5. We will define the __str__() function. This function just requires the `self` parameter. This is a fruitful function and will return a string containing all contents of the object that you are interested in. You must therefore ensure that your redefined code correctly returns a string.

In [6]:

```
'''
Requirement 5.  We will define the __str__() function.  This function just requires the self param
eter.
This is a fruitful function and will return a string containing all contents of the object that yo
u are interested in.
You must therefore ensure that your redefined code correctly returns a string.
'''
class Rectangle:
    def __init__(self, l, w):
        self.__length = l
        self.__width = w

    def get_length(self):
        return self.__length

    def get_width(self):
        return self.__width

    def set_length(self, l):
        self.__length = l

    def set_width(self, w):
        self.__width = w

    def __str__(self):
        return 'length = ' + str(self.__length) + '; width = ' + str(self.__width)
```

Requirement 6: Write two additional methods: `compute_area()` and `compute_perimeter()`.

Inputs for the two methods: As described earlier, every method needs to have the self parameter. These methods will also, therfore, have the self parameter. We only require the instance variables of the object to compute the area as well as the parameter and we already have access to these variables via the self parameter. Hence, we do not need any other external values to compute either the area or the perimeter.

Return value of the two methods: Like in other functions or methods we have seen earlier, a method of a class may also include a return statement as appropriate. In our Rectangle class definition, we will require both the area and the perimeter methods to return the value that is computed

So in this cell, we will continue with our class definition by defining both these methods.

In [7]:

```
'''
Requirement 6:  Write two additional methods: compute_area() and compute_perimeter().
'''
class Rectangle:

    def __init__(self, l, w):
        self.__length = l
        self.__width = w

    def get_length(self):
        return self.__length

    def get_width(self):
        return self.__width

    def set_length(self, l):
        self.__length = l

    def set_width(self, w):
        self.__width = w

    def __str__(self):
        return 'length = ' + str(self.__length) + '; width = ' + str(self.__width)

    def compute_area(self):
        return self.__width * self.__length

    def compute_perimeter(self):
        return 2*(self.__width + self.__length)
```

Now that we have defined the class, we can use it in other scripts. Let us suppose we wish to create two rectangles. The first will have a length of 5 and a width of 8 while the corresponding numbers for the second rectangle are 6 and 7 respectively. Your code should create the two rectangles. Then print the length and width of each rectangle. Finally, compute the areas and primeters of both and then print appropriate messages stating which perimeter is larger and which area is larger.

In [8]:

```
'''
Now that we have defined the class, we can use it in other scripts.
'''

#Create the first rectangle.  Note that the length argument is first followed by the width argumen
t.  This is in the same
#order as listed in the __int__() method
r1 = Rectangle(5, 8)

#Create the second rectangle
r2 = Rectangle(6, 7)

#print the rectangle data.  Note this returns the length and width of the the rectangle in the arg
ument.  This is because
# we redefined the __str__().
print(r1)
print(r2)

#Get and store the two areas
area1 = r1.compute_area()
area2 = r2.compute_area()

#Get and store the two perimeters
peri1 = r1.compute_perimeter()
peri2 = r2.compute_perimeter()

#Compare areas and perimeters and print appropriate messages
if area1 > area2:
    print('The first rectangle has a larger area')
elif area2 > area1:
    print('The second rectangle has a larger area')
else:
    print('The areas of the two rectangles are the same')

if peri1 > peri2:
    print('The first rectangle has a larger perimeter')
elif peri2 > peri1:
    print('The second rectangle has a larger perimeter')
else:
    print('The perimeters of the two rectangles are the same')
```

```
length = 5; width = 8
length = 6; width = 7
The second rectangle has a larger area
The perimeters of the two rectangles are the same
```

We revisit name mangling in this cell. Recall that we prefixed two _ characters before each variable name to restrict direct access. In this cell, we will see how this works.

We first create the one rectangle object as before and then try to access its length (one of its instance variables) directly.

In [9]:

```
'''
We revisit 'name mangling' in this cell.
'''

#Create a rectangle.
r1 = Rectangle(5, 8)
'''
The statement below will return all the instance variables and methods availabe to the r1 Rectangl
e object.
You will notice that the instance variables __length and __width do not show up.  This is due to t
he name mangling
that occurs.
'''
print(dir(r1))
'''
Hence if you execute the statement below, it will return an error since there is no variable named
__length.
Uncomment the statement below and try it yourself.
'''
print(r1.__length) # This will return an error as shown below.
```

```
['_Rectangle__length', '_Rectangle__width', '__class__', '__delattr__', '__dict__', '__dir__', '__
doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '_
_init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'compute_area', 'compute_perimeter', 'get_length', 'get_width', 'set_length',
'set_width']
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-9-f7eea7567f6c> in <module>
     15 Uncoment the statement below and try it yourself.
     16 '''
---> 17 print(r1.__length) # This will return an error as shown below.
     18 

AttributeError: 'Rectangle' object has no attribute '__length'
```

One can circumvent this by prefixing the variable name with the name of the class which in its turn is prefixed by a single underscore.

In [10]:

```
'''
One can circumvent this by prefixing the variable name with the name of the class which in its tur
n is prefixed
by a single underscore.
'''
#The following two statements will execute correctly.
print(r1._Rectangle__length)
print(r1._Rectangle__width)
```

```
5
8
```