COMP2152 LAB MANUAL

OPEN SOURCE DEVELOPMENT

This booklet will help the reader understand the concepts, principles, and implementation of the Python programming language. By the end of the booklet, the reader will be able to code comfortably in Python.

Prepared by Ben Blanc

TABLE OF CONTENTS

CHAPTER 5

OBJECT CLASSES & METHODS

CHAPTER 5 AT A GLANCE

In this chapter you will learn how to create your custom class as well as how to create methods of any class object. You will learn how to set properties of classes and how to access and change the values of the properties.

CREATING A CLASS

To begin the process of creating a class in python, create a new python file. Name the file **firstclass**. Type the following code

```
class Class1 :
    pass
```

The structure for creating a class is

class CLASS_NAME:
statements

Create another python file named **runner**. Type the following code

```
from firstclass import Class1
```

You now have imported your Class1 class into your **runner** python file.

To instantiate the new class, type the following code

```
ob = Class1()
```

Notice that we must adhere to the structure of declaring an instance of anything by the following structure of:

name_of_var = CLASS_NAME()

CREATE A BOX CLASS

Let us create a Box class.

Follow the steps above to create a new python file. Name the file **boxclass**. In this new file, create a Box class.

```
class Box :

    pass
```

CONSTRUCTORS

Next, we need the ability to add build the class object and declare its properties. Ideally, we would want to execute the following code (in our **runner** file)

```
box = Box("My Box", 12, 34, 56)
```

And have a new Box object constructed with a name of “My Box”, a width of 12, a height of 34 and a depth of 56.

To allow for this, we need to create a constructor. If we navigate to our Box class, we can create a constructor by inserting the following code:

```
class Box :

    def __init__(self, Name, Width, Height, Depth):

        pass
```

The constructor allows us to pass values to our class object while instantiating the Box class object.

The format for a constructor is

def __init__(self [, optional parms to pass to object])
statements

The **self** keyword refers to the current class. It is the equivalent to the **this** keyword in other programming languages. The **self** keyword must be included as the first parameter when declaring a class method.

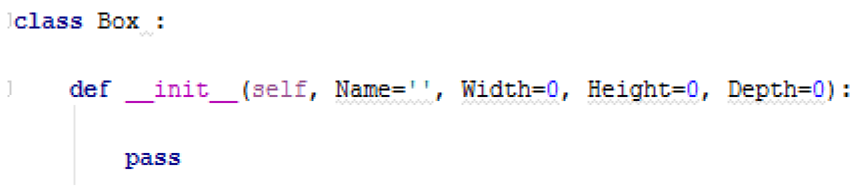
Multiple constructors are NOT permitted. To make the constructor parameters optional, give them default values

```
class Box :

    def __init__(self, Name='', Width=0, Height=0, Depth=0):

        pass
```

To create a constructor with no parameters passed to it, write the following code

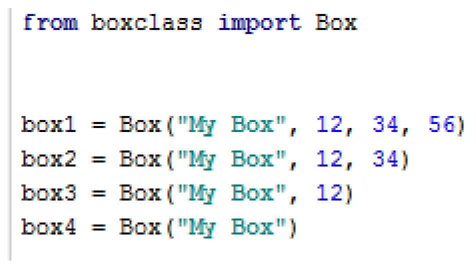
```
class Box :
    def __init__(self):

        pass
```

The constructor we will choose for our Box class will be the second constructor:



```
class Box :

    def __init__(self, Name='', Width=0, Height=0, Depth=0):

        pass
```

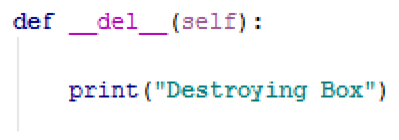
The above constructor would allow us to construct the Box object in the following ways (in our **runner** file):



```
from boxclass import Box


box1 = Box("My Box", 12, 34, 56)
box2 = Box("My Box", 12, 34)
box3 = Box("My Box", 12)
box4 = Box("My Box")
```

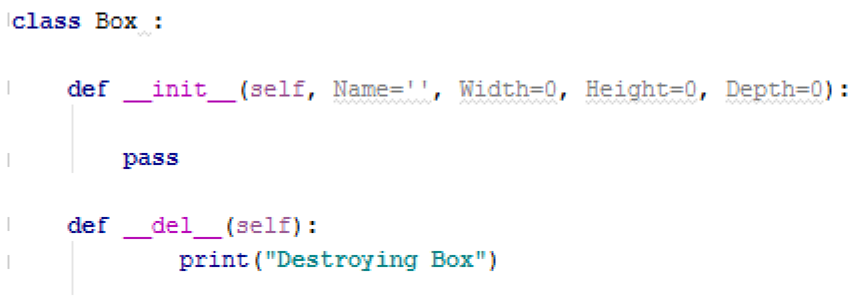
DESTRUCTORS

In case you want to incorporate clean-up statements when an object is destroyed, you can code them in the destructor statement



```
def __del__(self):

    print("Destroying Box")
```

Together with the constructor would be:



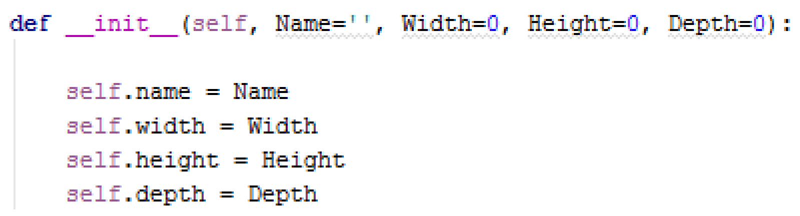
```
class Box :

    def __init__(self, Name='', Width=0, Height=0, Depth=0):

        pass

    def __del__(self):
            print("Destroying Box")
```

Destructors take the following structure:

def __del__(self) :
statements

CLASS PROPERTIES

Next we will add some properties to this class. Properties are attributes related to the class. For the Box class, we will add a name, height, width, depth properties.



```python
def __init__(self, Name='', Width=0, Height=0, Depth=0):

    self.name = Name
    self.width = Width
    self.height = Height
    self.depth = Depth
```

The lines

self.name = Name

self.width = Width

self.height = Height

self.depth = Depth

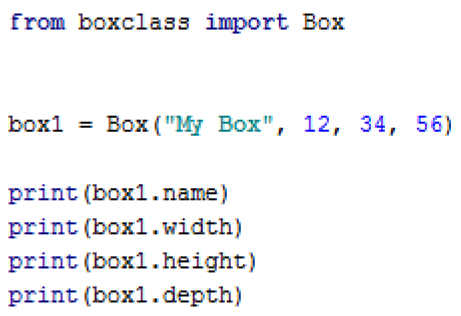
Sets four(4) properties of the Box class to the values passed from the constructor.

The **self** keyword must be included before declaring any properties.

The format for a declaring a property is:

self.PROPERTY_NAME = VALUE

We can now go to our **runner** file, instantiate the Box class, and output the Box properties.However, when we try to view and update its property values, we run into an issue.



```python
from boxclass import Box


box1 = Box("My Box", 12, 34, 56)

print(box1.name)
print(box1.width)
print(box1.height)
print(box1.depth)
```

PROTECTION LEVELS

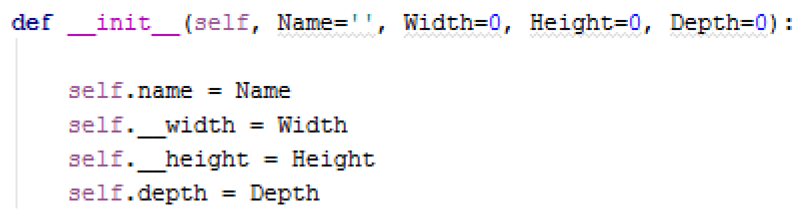
There are two protection levels in python:

public: Access is not restricted.

private: Access is limited to the containing type.

By default, properties have the public protection level.

To create a private property, prepend two underscores to the property name.



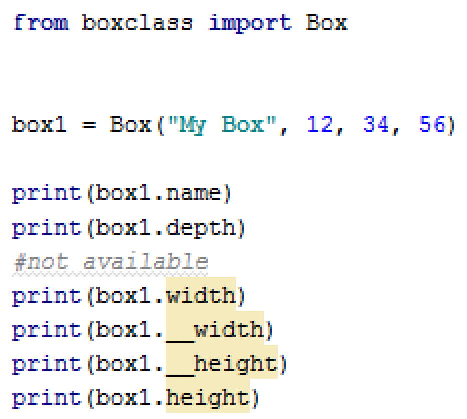
```
def __init__(self, Name='', Width=0, Height=0, Depth=0):

    self.name = Name
    self.__width = Width
    self.__height = Height
    self.depth = Depth
```

In the above code, width and height are private properties.

The format for a declaring a private property is:

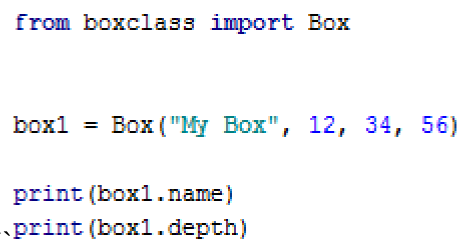
self.__PROPERTY_NAME = VALUE

Now can access our property name and depth from our **runner** file, however, the other properties are not available.



```
from boxclass import Box


box1 = Box("My Box", 12, 34, 56)

print(box1.name)
print(box1.depth)
#not available
print(box1.width)
print(box1.__width)
print(box1.__height)
print(box1.height)
```

Removal of the unavailable properties allows the **runner** file to run error-free again.



```
from boxclass import Box


box1 = Box("My Box", 12, 34, 56)

print(box1.name)
print(box1.depth)
```

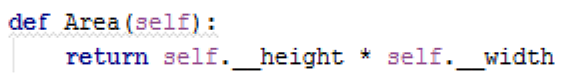
X`

BOX CLASS METHODS

Continuing with our Box class, we will be creating methods for the class.

The first method we will create is an area method

Navigate to the Box class and type in the following code:



```
def Area(self):
    return self.__height * self.__width
```

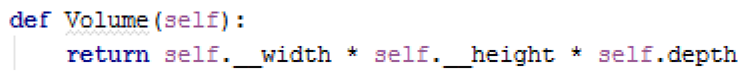
To refer to a property value within the class, the code structure is **self.Property_Name**

From the code above, we can note that the structure of create a method is

def NAME_OF_METHOD (self [,parameters]) :
statements

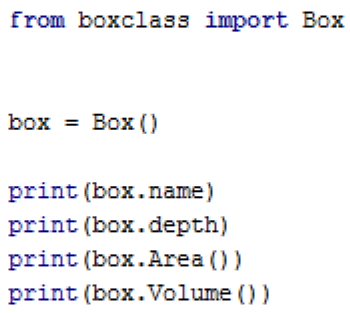
The second method we will create is a volume method

Add the following code to the Box class



```
def Volume(self):
    return self.__width * self.__height * self.depth
```

If we navigate to our **runner** file, we can add to our code to view our methods in action.



```
from boxclass import Box


box = Box()

print(box.name)
print(box.depth)
print(box.Area())
print(box.Volume())
```

The structure for accessing a public property of a class is

Instantiated_object_variable.PROPERTY_NAME

The structure for accessing a method of a class is

Instantiated_object_variable.METHOD_NAME()

Notice that the method outputs are zero because some property values are set to zero. We cannot directly set the values of these properties because of their protection level. We will need to code accessors and mutators also known as getters and setters.

To summarize, here is the structure of a class

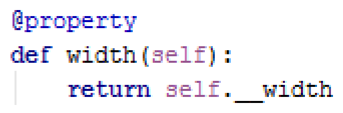
```
class CLASS_NAME :

        //constructor

        def __init__(self [, optional parms to pass to object] )
                statements

        //methods

        def NAME_OF_METHOD ( self [,parameters] ) :
                statements
```

ACCESSORS/MUTATORS AKA GETTERS/SETTERS

The term Accessor or Getter refers to giving the ability for a program to display or return a value of a property whose protection level would otherwise prohibit you from completing this task.

The term Mutator or Setter refers to giving the ability for a program to update a value of a property whose protection level would otherwise prohibit you from completing this task.

Below is how to code an Accessor



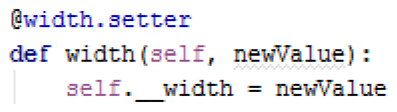
```
@property
def width(self):
    return self.__width
```

The structure for coding an Accessor/Getter is

```
@property
def accessor_property_name(self) :
    (statements)
```

The annotation of **“@property”** tells python that the method we have just declared is an accessor for a property

Below is how to code a Mutator/Setter



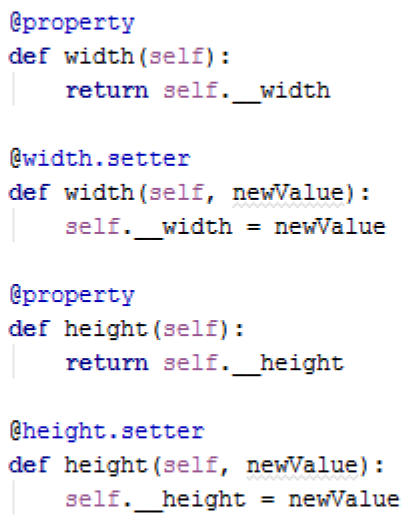
```
@width.setter
def width(self, newValue):
    self.__width = newValue
```

The structure for coding a Getter is

```
@accessor_property_name.setter
def accessor_property_name(self, newValueParam) :
    (statements)
```

The annotation of **“@accessor_property_name.setter”** tells python that the method we have just declared is an getter for a property

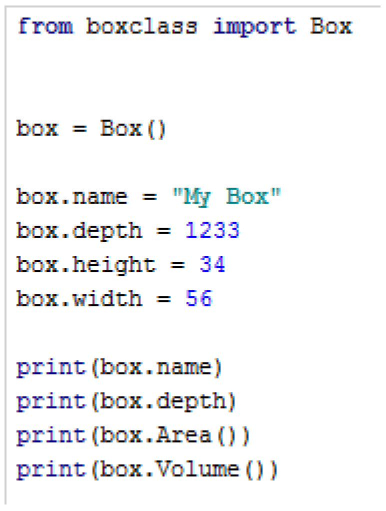
Let’s add the accessors and mutators for height and width properties to our Box class.



```
@property
def width(self):
    return self.__width

@width.setter
def width(self, newValue):
    self.__width = newValue

@property
def height(self):
    return self.__height

@height.setter
def height(self, newValue):
    self.__height = newValue
```

.

Now we can go to our **runner** file and use these accessor and mutators.

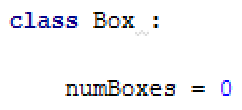


```
from boxclass import Box


box = Box()

box.name = "My Box"
box.depth = 1233
box.height = 34
box.width = 56

print(box.name)
print(box.depth)
print(box.Area())
print(box.Volume())
```

CLASS METHODS AND VARIABLES

STATIC CLASS VARIABLES

You can declare a static class variable in python by declaring a variable outside of a constructor or method of a class



```
class Box :

    numBoxes = 0
```

The structure for creating a static class variable

class CLASS_NAME:

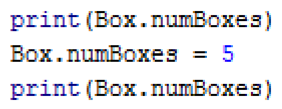
        variable_declaration = value

        statements

To access your static variable by

CLASS_NAME.declared_variable

You can output and alter the value of the static class variable if it is included in your scope or in the class itself.

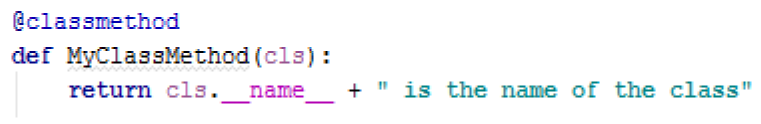


```
print(Box.numBoxes)
Box.numBoxes = 5
print(Box.numBoxes)
```

CLASS METHOD

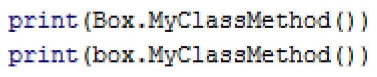
A class method is a method that passed a pointer to the class and not the object itself.
The structure for declaring a class method is

@classmethod
def NAME_OF_METHOD (cls [,parameters]) :
        statements



```
@classmethod
def MyClassMethod(cls):
    return cls.__name__ + " is the name of the class"
```

The keyword **cls** points to the class. A class method can be called by the class itself or by an instantiated object.



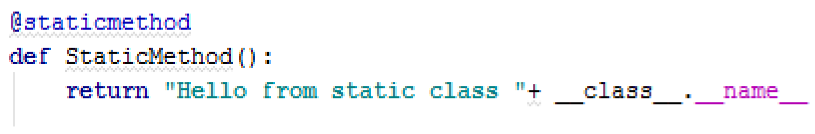
```
print(Box.MyClassMethod())
print(box.MyClassMethod())
```

STATIC CLASS METHOD

A static class method is a method that can be called without instantiating an object. It does not take self or class as a pointer.

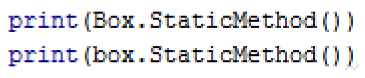
The structure for declaring a static method is

@staticmethod
def NAME_OF_METHOD ([parameters]) :
statements



```
@staticmethod
def StaticMethod():
    return "Hello from static class "+ __class__.__name__
```

A static class method can be called by the class itself or by an instantiated object.



```
print(Box.StaticMethod())
print(box.StaticMethod())
```