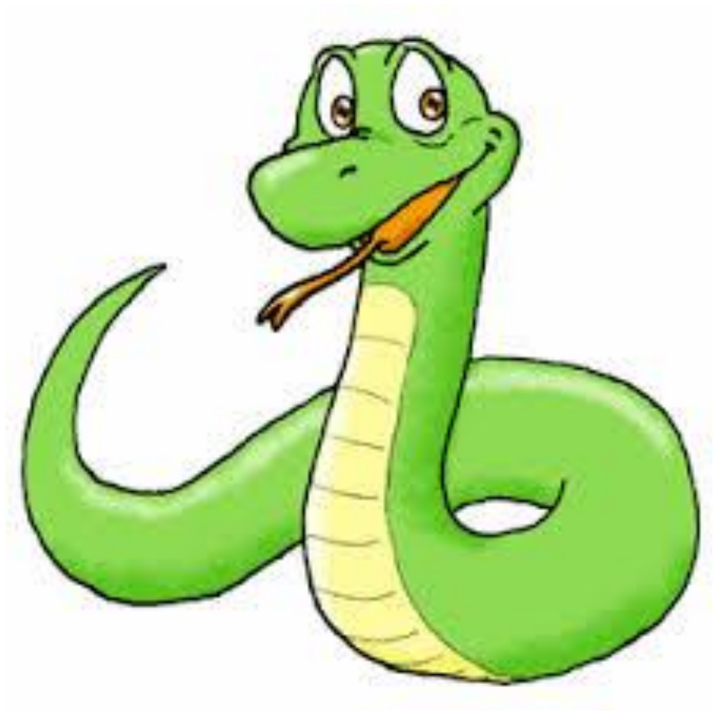
Application of GIS with Python

Chapter 8: Classes and Objects



https://www.python.org/
http://www.tutorialspoint.com/python/

Table of Content

- Object-oriented programming permits us to treat every real world problems as objects or to organize the programs as interactions of objects.
- Every objects can be classified to specific class on the basis of common features of the objects.
- In python, a class provides the definition of the structure and behavior of the objects.
- Each created object is an instance of a class.

```
>>> a=1
>>> b='WRC'
>>> c=[1,'WRC','IOE']
#a, b, c represent instances of respective classes
```

```
>>> type(a)
<class 'int'>
>>> type(b)
<class 'str'>
>>> type(c)
<class 'list'>
```

- A user-defined type is also called a class.
- A class definition creates a new class object.
- Class Object contains information about a user-defined type.
- The class object can be used to create instances(new object) of the type.

The class is simply a template or factory for creating the instance objects.

```
>>> class Point:
         """represents a point in 2-D space"""
```

```
>>> print (Point)
<class '__main__.Point'>
```

```
>>> blank = Point()
>>> print (blank)
<__main__.Point object at 0x02EADD30>
```

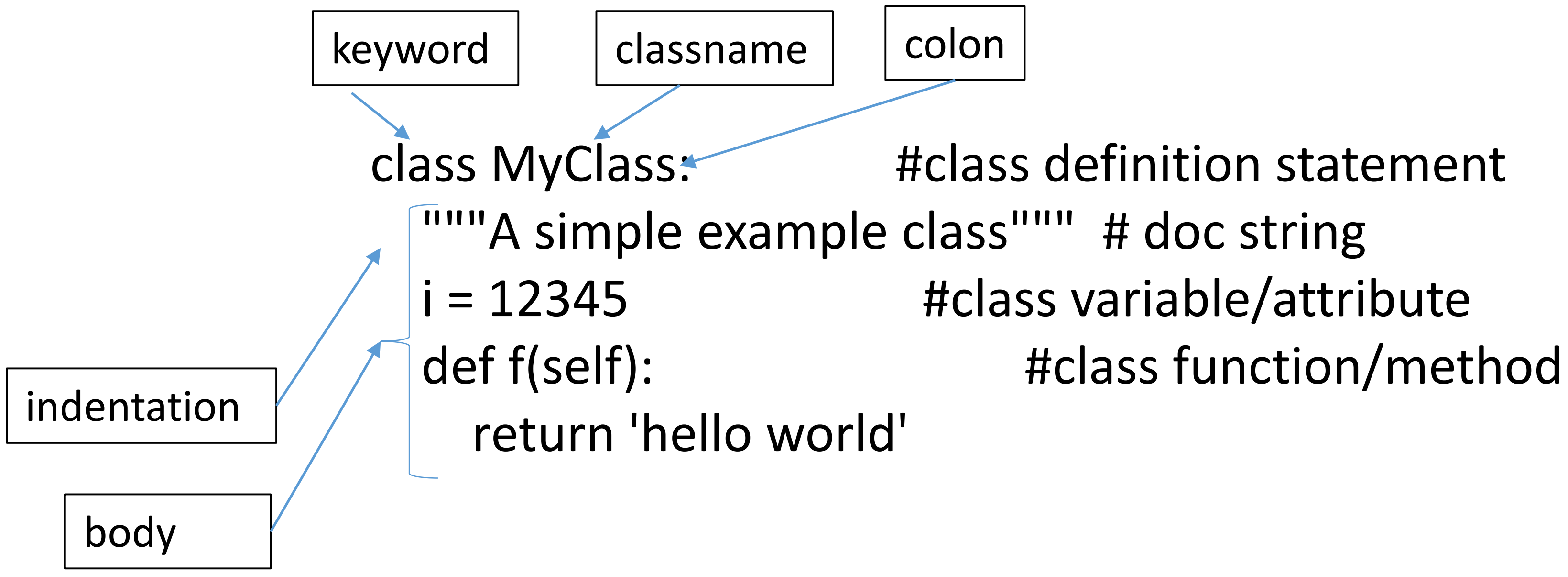
- The simplest form of class definition looks like this

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

- The ClassName is the name of the class, and this name is used to create new objects that are instances of the class.
  Traditionally, class names are capitalized and class elements (variables and methods) are not capitalized.
- Body may contain variables and series of function definitions, which define the class.
  All of these function definitions must have a first positional argument, self, which Python uses to identify each object’s unique attribute values.

➢The simplest form of class definition looks like this

```
class ClassName:
   <statement-1>
   .
   .
   .
   <statement-N>
```



```
class MyClass:              #class definition statement
   """A simple example class"""  # doc string
   i = 12345                    #class variable/attribute
   def f(self):                     #class function/method
      return 'hello world'
```

Attribute References

```
class MyClass:
    """A simple example
class"""
    i = 12345
    def f(self):
        return 'hello world'
```

- can reference using dot notation
- MyClass.i and MyClass.f are valid attribute references, returns an integer and a function object, respectively

#methods are functions defined inside a class, all these function has **self** as the first argument

# **self** as argument in method represents the temporary place for the object refers to

Instantiation and References

- Creating a new object is called instantiation, and the object is an instance of the class.
- Instance may has own variables and methods

```
class MyClass:
    """A simple example
class"""
    i = 12345
    def f(self):
        return 'hello world'
```

```
x = MyClass()                    # x is a instance of MyClass
```

- creates a new instance of the class and assigns this object to the local variable x
- Then attributes can be referenced as object attributes

x.i

Instantiation and References

```
class MyClass:
   """A simple example
class"""
   i = 12345
   def f(self):
      return 'hello world'
```

```
x = MyClass()                              # x is a instance of
MyClass
>>> x.i
12345
>>> x.f()
'hello world‘
>>> x.__doc__
'A simple example class‘    #returns the docstring
belonging to the class
```

Instantiation and References

```
>>> class MyClass:
        """A simple example class"""
        i = 12345
        def f(self,name):
                return 'hello '+name
```

```
>>> x=MyClass()
>>> x.f('ram')
'hello ram'
>>> y=MyClass()
>>> y.f('harry')
'hello harry'
```

Constructors

- Many classes like to create objects with instances customized to a specific initial state.
- A class may define a special method named __init__() to get an initialized instance.
- Class instantiation automatically invokes __init__() for the newly-created class instance

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Constructors

- Class instantiation automatically invokes __init__() for the newly-created class instance. This special method is called constructor.

```
>>> class Company:
        def __init__( self, name, symbol, stockPrice ):
                self.name= name
                self.symbol= symbol
                self.price= stockPrice
        def valueOf( self, shares ):
                return shares * self.price
```

```
>>> c1= Company( "General Electric", "GE", 30.125 )
>>> c1.valueOf(100)
3012.5
```

Copy

- The copy module contains a function called copy that can duplicate any object

```
>>> class Point:
        """represents a point in
2-D space"""“
>>> print (Point)
<class '__main__.Point'>
>>> p1 = Point()
>>> p1.x = 3.0
>>> p1.y = 4.0
>>> import copy
>>> p2 = copy.copy(p1)
```

p1 and p2 contain the same data, but they are not the same Point.

Copy

- The copy module contains a function called copy that can duplicate any object

```
>>> def print_point(p):
                print ((p.x, p.y))
>>> print_point(p1)
(3.0, 4.0)
>>> print_point(p2)
(3.0, 4.0)
>>> p1 is p2
False
```

# can pass an instance as an argument.

p1 and p2 contain the same data, but they are not the same Point(different objects).

Shallow Copy

- Incase of first object embedded in another(second) object when the second object is copied using copy function it copies the second object but not the embedded object.

```
>>> class Rectangle:
            """represent a
rectangle.
attributes: width, height,
corner."""“
>>> box = Rectangle()
>>> box.width = 100.0
>>> box.height = 200.0
```

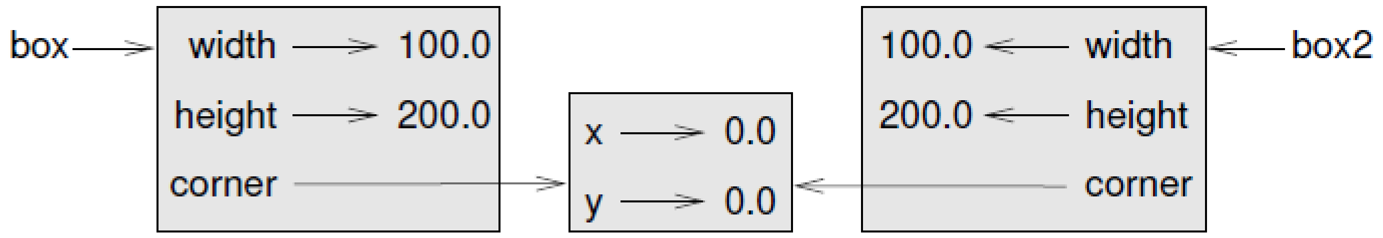
```
>>> box.corner = Point()
>>> box.corner.x = 0.0
>>> box.corner.y = 0.0
>>> box2 = copy.copy(box)
>>> box2 is box
False
>>> box2.corner is box.corner
True
```

It copies the object and any references it contains, but not the embedded objects, this is called shallow copy.

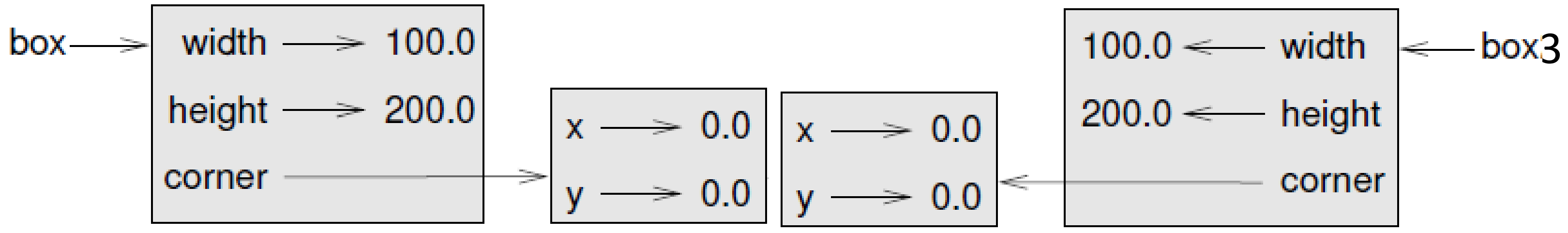
Deep Copy

- ➢The copy module contains a method named deep copy that copies not only the object but also the objects it refers to, and the objects they refer to, and so on. This operation is called deep copy.

```
>>> box3 = copy.deepcopy(box)
>>> box3 is box
False
>>> box3.corner is box.corner
False
```



Shallow Copy



Deep Copy

Inheritance

- Class can be derived from another class
- The syntax for a derived class definition looks like this:

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

Inheritance

- Class can be derived from another class (when the base class is defined in another module)
- The syntax for a derived class definition looks like this:

```
class DerivedClassName(modname.BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

Inheritance

- Class can be derived from other classes (when multiple base classes)
- The syntax for a derived class with multiple base classes definition looks like this:

```
class DerivedClassName(Base1, Base2, Base3):
   <statement-1>
   .
   .
   .
   <statement-N>
```

Assignment 8:

- What do you understand by Class and object in Python Programming? Demonstrate the use of Class and object with Python Programming.
- What do you understand by attribute references? What is the use of attribute references in python programming?
- What is Instantiation? Write a python program to create a new instance of the class and assigns this object to the local variable .
- What do you understand by constructor in Python programming? Differentiate between Constructors and Instantiation with suitable python example.
- What is the use of copy module in python programming? List out and differentiate different types of Copy module of python with python example.
- What is Inheritance ? Write syntax of different forms of inheritance in python programming ? Give python example of any one form of Inheritance. (Assume suitable data).