Python Classes and Objects

G A Vignaux

July 2008

Python is an object-oriented language and its constructs are usually classes and objects. This introduces Python Classes and their objects. An object can have its own data and methods for manipulating them.

Revision : 1.23

Outline

What is a Class?

A Very Simple Class: Things
- The __init__() method
- The __str__() method
- The docstring

The Ship Class

Summary on Methods and Fields

Importing a Class from a module

What is a Class?

- "A `class` is a user-defined type which you can `instantiate` to obtain `instances`, meaning objects of that type."
- For example you could have a class of `StudentRecords` in a program. You can instantiate many instances of student records to keep data on the students in the University.

A `Class` is..

- a blueprint or plan that
- describes *objects* of the class.
- Each object can have its own *data*.
- Each object can have its own *methods*.
- But, defining a Class does *not* itself define any objects

Defining a Class

Use the `Class` statement:

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

A Class definition

```
class Ship:
   """A class of ships"""
   def __init__(self,nm=''):
       self.name = nm
       self.load = 0
       self.location = (0.0,0.0)

   def setload(self,ld):
       self.load = ld

   def setlocation(self,loc=(0.0, 0.0)):
       self.location = loc
```

Inside the Class statement

- the Class name starts with a capital letter
- The statements define data and methods
- `methods` are like functions

Creating a new Ship object

- Once we have defined a class we can construct (i.e, `instantiate`) any number of objects (`instances`) of that class
- The class name is used like a function and returns an object:

```
arahura = Ship('Arahura')
ship2 = Ship()
ship3 = Ship()
```

A class with no data and no methods

- A very simple class. The objects hold no data and have no methods.

  ```
  class Thing:
      pass
  ```

- The definition must have some sort of statement so for this simplest class I just use the Python statement that does nothing: `pass`.

Creating an object

Create an object of class `Thing` by using the Class name as a function:

```
thing1 = Thing()
```

Create two `Things`

```
thing1 = Thing()
thing2 = Thing()
```

and put them into a box:

```
box = [thing1,thing2]
```

Adding data attributes to an object

- An object can hold its own data in its *fields* or *attributes*.
- The fields are referred to by a dot notation (*object.attribute*).
- We can provide existing objects with attributes:

  ```
  thing1.name = 'Thing1'
  thing1.colour = 'red'

  thing2.name = 'Thing2'
  thing2.colour = 'blue'
  ```

Things with actions: Methods

- Objects of a class can have `methods` (behaviour attributes).
- These look very much like functions.
- In the definition the first argument of a method must be `self`.
- Here the `Thing` class is extended by a method called `setColour`.
- For clarity, I have left out the docstrings.

  ```
  class Thing:
      def setColour(self,col):
          self.colour = col
  ```

Using setColour

- The `setColour` method shows that it is defined just like a function.
- BUT the first argument is `self` which refers to the particular object that is using the method.
- There may be many `Things` running about and we may wish to call the method on each one separately.
- To use the `setColour()` method for a particular object we execute it using the dot notation *but without the self argument*:

```
thing1.setColour('red')
thing2.setColour('blue')
```

The __init__() Method

- We often need to initialise data when an object is created.
- The `__init__` method does this.
- If one has been defined for a class it is called automatically whenever a new object is created.
- Arguments of the Class can be passed to the object.

Thing with __init__()

- Here we give our `Thing` class such a method.
- We initialise the name and colour attributes.
- Since we are assigning them inside the object we must prefix them with `self`.

```
class Thing:
    def __init__(self,nm,col):
        self.name = nm
        self.colour = col

thing1=Thing('Thing1','Red')
thing2=Thing('Thing2','Blue')
```

The __str__() method

- It is good practice also to define a special method called __str__.
- This should return a string that displays data for the object in a clear format.
- __str__ is recognised by Python. When you `print` the object you get the data printed out nicely.

Add a __str__()

Add a __str__ method to the definition.

```
class Thing:
   def __init__(self,nm,col):
      self.name = nm
      self.colour = col

   def __str__(self):
      return self.name+' is '+self.colour

thing1=Thing('Thing1','Red')
print thing1
```

This gives

```
Thing1 is Red
```

A docstring

- It is good practice to give every class a *documentation string*.
- Called a `docstring`
- This is placed first in the class definition.
- There really should be a docstring for every method as well (left out here for space reasons)

```
class Thing:
''' Objects of this class do not do much
'''
   def __init__(self,nm,col):
      self.name = nm
      self.colour = col

   def __str__(self):
      return self.name+' is '+self.colour
```

The Ship class

```
class Ship:
   """A class of ships"""
   def __init__(self,nm=''):
       self.name = nm
       self.load = 0
       self.location = (0.0,0.0)

   def setload(self,ld):
       self.load = ld

   def setlocation(self,loc=(0.0, 0.0)):
       self.location = loc

   def __str__(self):
       return self.name+' is at '+str(self.location)+
            ' with '+str(self.load)+ ' tonnes'
```

Creating a Ship object

```
arahura = Ship('Arahura')
arahura.setload(1000)
arahura.setlocation((120.0, 99.0))
print arahura
```

This gives

```
Arahura is at (120.0, 99.0) with 1000 tonnes
```

Summary of Methods and Fields

- A method is defined using the `def`.
- All methods have `self` as their first argument. This is **required** for methods. But the `self` is *not* used when the methods are called (see below).
- The `__init__` method will be executed when the classname `Ship` is used to construct a new ship. It lets you initialise the fields of the new ship instance.
- Within the object, these data fields are referred to using the `self` as a prefix. So the ship's `load` will be referred to as `self.load` in any of the methods of the ship.
- Outside the object the fields are referred to with the object as a prefix. For example, outside the ship `hermes` its load is `hermes.load`

Importing a Class from a module

I made a file `shipmodule.py` to hold the Ship definition above (a module) and then imported the class definition from that.

```
>>> from shipmodule import Ship
>>> hermes = Ship('Hermes')
>>> pinafore = Ship('Pinafore')
>>> hermes.setload(1000.0)
>>> hermes.setlocation((100, 130))
>>> print hermes.location
```