University of Hull
Department of Computer Science

Playing with Python

Vsn. 1.0 Rob Miles 2014

Working with Classes

Practical Break 1: Creating a player class

In this lab we are going to create a class which will hold cricket score values. If you are not a big fan of cricket you can think of this exercise as storing any collection of values, for example you could store the name of a golf player and the number of shots it took them to go round the course, or perhaps the name of a drinking buddy and the number of shots it took them to go under the table.

Creating a New Program

We are going to make a new version of the cricket scores program. If you can find your old one you can use that as the basis of this work, otherwise you can start from scratch.

Before you go any further; perform the following:

1. Start the Idle environment.
2. Use "**File>New Window**" to create a new program file called `cricket_class`. Alternatively you can open your old cricket program instead and start from that.

Creating a class

Before we can add data to the class we need to create it. We are going to put a method in the class which we can fill in later to print the player details:

```
class player:
    def __init__(self, name, score):
        self.name = name
        self.score = score
```

The Python above creates a class called `player`.

Before you go any further; perform the following:

3. Add a `player` class to your program.
4. You can run the program if you wish, but it won't do anything. At the moment all it does is create the class. Next we have to make an object which is an instance of the class

It is very important that you understand what you have just done here. We have told Python that we want our program to contain objects of type `player`. At the moment a `player` object just holds a single method (that doesn't do much). Later we will use the object as a container for the data that we want to store about our players, for example their name and their cricket score.

Making a player instance

In Python you can create a variable just by using it. The same is true with classes. A program can make a `player` instance just by referring to the name of the class:

```
p = player('Fred', 10)
print(p.name)
```

These two statements create a `player` instance and then prints the name held by that instance.

Before you go any further; perform the following:

5. Add the two statements above to your program.
6. Save and run this program. It should print "Fred" on your screen.

This only works because we have created a `player` class. What do you think would happen if we tried the following statement?

```
w = wallaby()
```

The Python system doesn't know anything about the wallaby class because it has not been declared. The result of this statement would be a complaint from Python that wallaby does not exist:

```
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    w = wallaby()
NameError: name 'wallaby' is not defined
```

As long as we create classes before we try to use them we will be fine.

Practical Break 2: Using attributes in class methods

Storing data in objects is very useful, but we can do a lot more than this. We can add methods to the object so that the object can "look after itself" and provide useful behaviours. We are going to start by completing the `print_details` method that we added earlier.

Using self in print_details

We would like the `print_details` method to actually print something useful about the score and player name in the player instance. To do this the method needs to get hold of the name and score information which is held in the attributes of the player object. When the method is called the self reference is passed as the first parameter.

```
def print_details(self):
      print(self.name, " scored ", self.score)
```

The `print_details` method above uses the `self` reference parameter to get hold of the name and score attributes and then prints them out.

Before you go any further; perform the following:

1. Change the `print_details` method in your `player` class to print out the name and the score as above.

Now that we have the method we can use it to display the details

```
p = player('Fred', 10)
p.print_details()
```

The above sequence creates a new player, creates score and name attributes and then prints them.

You can add more methods to the class, and the methods can be given more parameters. But you need to remember that the first parameter that is passed into the method when it runs is always a reference to the object running the method.

Before you go any further; perform the following:

2. Add a new method to the player class. The method will be called `print_details_congrat`. The method can be instructed to print an optional congratulatory message if the player has scored 100 runs or more. The method should work as follows:

```
a = player('Fred', 100)
a.print_details_congrat(True)
Fred scored 100 *** Congratulations ***
a.print_details_congrat(False)
Fred scored 100
b = player('Jim', 99)
b.print_details_congrat(True)
Jim scored 99
```

3. A good starting point for this would be the full_details version of the print_details method described in the slides.

Understanding Classes and References

We should by now have this idea that simple variables and classes behave in different ways. Now we are going to do some experiments to test these behaviours. You can do all this at the shell prompt in Python.

Before you go any further; perform the following:

1. Run the program so that the player class has been created and you can work with it.

One of the things that can be confusing about Python, particularly if you have used other languages before, is the way that the elements in classes seem to be "made up as the program runs". In other words, we never told the Python system that the player class contains a name attribute, we just created the attribute and it appeared as though by magic.

Before you go any further; perform the following:

2. At the Python shell, (i.e. the >>> prompt) issue the following command:

```
q = player('Ethel', 99)
```

This will create a player instance, which is referred to by the variable (tag) q.

Next we might like to investigate how references work in a bit more detail. We will do this at the shell prompt again.

Before you go any further; perform the following:

3. At the Python shell, (i.e. the >>> prompt) issue the following command:

```
r = q
r.name = 'Nigel'
```

What has this done? What will we see if we call `print_details` on q?

```
q.print_details()
```

We see that the name that is printed is ‘Nigel’. This is because both q and r refer to the same instance in memory.

Practical Break 2: A list of players

One create way of using classes is by making lists of objects. A can hold references to objects in a program.

```
players = []
p = player('Fred',10)
players.append(p)
print(players[0].name)
```

The statements above create an empty players list and then add a player on the end of the list. Then the program prints the name of the player that has been added to the list.

You can convert your cricket score and sorting code from last week into one which works on players. For real style, add a feature that will print out the scores in ascending order.

Before you go any further; perform the following:

1. Create an object based version of the cricket scores program which stores the scores and the player names in a player class.

Rob Miles Feb. 2014