# University of Hull Department of Computer Science Playing with Python

Vsn. 1.0 Rob Miles 2014

# Advanced Classes

# Practical Break 1: Creating a well behaved constructor

In this lab we are going to start by improving the construction of our Player objects so that it is harder for other programmers to make instances that contain invalid attributes

## Using your previous program

We are going to build on the Cricket Player program that you built last week.

|  |  |
| --- | --- |
| BD18212_ | Before you go any further; perform the following:   1. Start the Idle environment. 2. Use “**File>New Window**” to find your program file and open it to work with. |

## Modifying the class constructor

The constructor is the method that will create an instance of an object. It is **very sensible** to set the attributes for an object at the time you create it, and you can do this by passing parameters into the constructor for an object. The present version of our constructor just takes the values that are given and sets the attributes in the object using them:

**class player:**

**def \_\_init\_\_(self, name, score):**

**self.name = name**

**self.score = score**

The Python above creates a class called player. The first parameter is copied into the name attribute and the second into the score attribute. We could use the following to create a player instance.

**p = player(10,'Fred')**

|  |  |
| --- | --- |
| BS01890_[1] | This statement is wrong. It is perfectly legal Python, according to the meanings we are ascribing to items in our programs, it will cause problems when the code runs. Why? |

This is wrong because the score is set to ‘Fred’ and the name is set to 0. At some point we might obey a statement like this:

**p = player(10,'Fred')  
totalScore = 0  
totalScore = totalScore + p.score**

|  |  |
| --- | --- |
| BS01890_[1] | This sequence of statements would fail when the program ran. It is trying to add the score active by player Fred to the total scores, but the program would in fact crash. Why? |

This would fail because the score attribute of player p has been set to ‘Fred’, and attempting to add the number stored in totalScore to a string held in p.score would cause Python to fail:

**Traceback (most recent call last):  
 File "C:\Users\Rob\SkyDrive\Wrestling with Python\Season 2\Week 05 Advanced Classes\cricket\_class.py", line 18, in <module>  
 totalScore = totalScore + p.score  
TypeError: unsupported operand type(s) for +: 'int' and 'str'**

To fix this the constructor could test the type of each of the parameters and raise an exception (stop the program) if the type was not what was required. The name must be a string and the score must be an integer.

**if type(name) is str:  
 self.name = name  
else:  
 raise Exception('Invalid player name type')**

The above snippet of Python tests the type of a parameter called name. If this is a string it stores the value in an attribute called name. If it is not a string it raises an exception with an appropriate message. Your program could use code like this in the constructor for a player to make sure that the name was always given as a string when an instance was created.

|  |  |
| --- | --- |
| BD18212_ | Before you go any further; perform the following:   1. Modify the constructor for the player object so that the name must be given as a name and the score must be given as an integer. 2. Test your program by trying to make some invalid player values and make sure that the exceptions are raised correctly. |

This is a very useful technique. It makes sure that a program cannot ever create a player which does not have a value name and score. At the moment the constructor only raises exceptions if the types of the inputs are incorrect, but you could also add conditions to make sure that the players that are created are valid in other ways too. You might like to add code to make sure that the player always has a name which contains text (i.e. is not an empty string) and that the score is never less than 0 or greater than 500.

# Practical Break 2: Counting players

We can add static variables to a class. These are created once for the whole class, i.e. they are not created each time a new class instance is created. We can use them to keep track of class values that are to be stored for the entire class. One such value might be the total number of instances of the class that have been created.

**class player:  
  
 count = 0**

The class variable count is set to 0 when the class is first loaded. A program can access this variable without need to have created an instance of the class:

**player.count = player.count + 1**

This would increase the value of the class variable count in the player class.

|  |  |
| --- | --- |
| BD18212_ | Before you go any further; perform the following:   1. Add a count member to the player class that will count how many player instances have been created. Remember to increment the counter only after any code that might raise exceptions, otherwise you might have counted players that were not subsequently created. |

You can create as many static variables as you like. A fellow programmer has suggested that you create one called totalScore in the player class. Each time a player was constructed their score would be added to the totalScore, so that the total score is always available. This would work, and is completely legal Python.

|  |  |
| --- | --- |
| BS01890_[1] | Why is it a stupid idea to create a static variable to hold the total score of all the players? |

This is stupid because if one of the score values of a player was changed for any reason the total would be instantly out of date. Also, from a design point of view the total score is not a property of the players, but should be a property of the team which the player is part of. If the program is used to create players for lots of teams a total score value held as part of the player class and managed in this way would be meaningless as it would hold the total score of all the players created

You need to think carefully about where data is held and the context in which you are going to us it.

# Practical Break 3: Adding addresses

You have been asked to store the address of each player. You could put the address information in the player class directly, but it is actually more sensible to create an address class. That way if we ever need to manage other items that have addresses, for example the cricket ground, you can just add an address attribute to that item.

**class address:  
 def \_\_init\_\_(self, firstLine, postcode):  
 self.firstLine = firstLine  
 self.postcode = postcode**

The code above describes an address item which contains a first line and a postcode. We can create an instance of this as follows:

**pAddress = address('Hull', 'HU6 7RX')**

Note that there is no error checking on the elements that are passed into the constructor for the address item.

|  |  |
| --- | --- |
| BD18212_ | Before you go any further; perform the following:   1. Add an address class to your program. 2. Improve the validation to make sure that an address always has a string for the firstline and postcode items. If something else is supplied the constructor for the address class should raise an exception. |

Once you have created an address class you can use it in your program.

|  |  |
| --- | --- |
| BD18212_ | Before you go any further; perform the following:   1. Update the constructor for the player so that it stores the address of the player. 2. Write some code that creates an instance of a player that has an address. 3. It would be useful if the constructor make sure that a valid address was always supplied. How do you think you might do this? Perhaps that you can use the same pattern that you used to make sure that the score is an integer and the name a string? |

# Practical Break 4: Creating a system

You now know enough to be able to design a data storage system for an application. Pick one of the tutorial systems and create a class to manage one of the data items in that application. It is best to start with one of the simpler classes first. For the class do the following:

* Decide what data attributes are to be stored in the class
* Identify the type of each of the attributes
* Write a constructor that is used to set the initial values from parameters that were supplied
* Write some test code to create instances of the object and print their contents

Rob Miles March. 2014