

Python Games. Lecture 1b.

Contents

Design.....	1
Variables	3
Lists.	3
Dictionaries.	9
Coding into Eclipse.....	11



This term we are going to look at the sub-systems of PyGame and Python to see what is required to make games. *If you are running this from home, run the Python docs found here first to set up your environment... otherwise skip down the page to DESIGN.*

http://www.drewfx.com/TAFE/python/Lecture_Week1a.pdf

If you need to install Eclipse and Python at home follow these steps...

You will need Python2.5 installed first...

<http://www.drewfx.com/TAFE/python/python-2.5.4.msi>

Then install PyGame ...

<http://www.drewfx.com/TAFE/python/pygame-1.9.1.win32-py2.5.msi>

Then download Eclipse and unpack it to your Hard Drive (not the desktop) and run Eclipse.exe from the Eclipse folder created.

<http://www.drewfx.com/TAFE/python/eclipse-SDK-3.5.2-win32.zip>

Design.

We need the following.

- 1) Input
- 2) Output
- 3) Encapsulation

Input can be thought of anything we do, or the game does to change what's happening in the game system.

Output can be thought of anything the game system shows us as a result of the input.

Encapsulation is information to enclose the entire game system.

Let's have a look at the individual steps...

1. Input

So what are we going to use for input?

- a) Mouse
- b) Keyboard

2. Output

What is coming out of the program?

- a) Sound
- b) Graphics
- c) Joy (in a bubble!)

3. Encapsulation

What kind of enclosure(s) do we need?

- a) Instructions on how to play the game
- b) Feedback on score, health, time etc.
- c) True closure – game won, game over etc.



Okay so let's plan a game. What do we need?

Variables

We are going to make a fighting game. The player is a knight and is beset by the evils of a tyrannical regime created by the evil Lord Necron – obviously able to raise the dead etc.

Cool, so in true RPG style we need a system to store stats, abilities and inventory. What does the player start with?

1. Inventory
 - a) Sword
 - b) Food
 - c) Armour

What are their abilities?

- a) Strength
- b) Dexterity
- c) Constitution
- d) Intelligence

What are their stats?

- a) Level
- b) Hit/Miss ratio
- c) Spell to hit
- d) Health
- e) Mana

Lists.

How do we store this? We'll use what's called a list.

A list is a type of array in Python. Wait what? You have no idea what an array is?

Arrays are a storage method in most programming languages with an index to point to individual elements within that array.

For example. Let's start with an array called myBackpack. We'll make it 3 items big.

We could define our array like this.

```
myBackpack[0]="Sword"
```

```
myBackpack[1] = "Food"
```

```
myBackpack[2]="Armour"
```

Indeed this is how arrays work in many languages. If you know a bit about Python you can see this is kinda looking more like a list. When we want to access a particular part of a array we do it using an iterator. So if we want to know the n^{th} element we would access it like this.

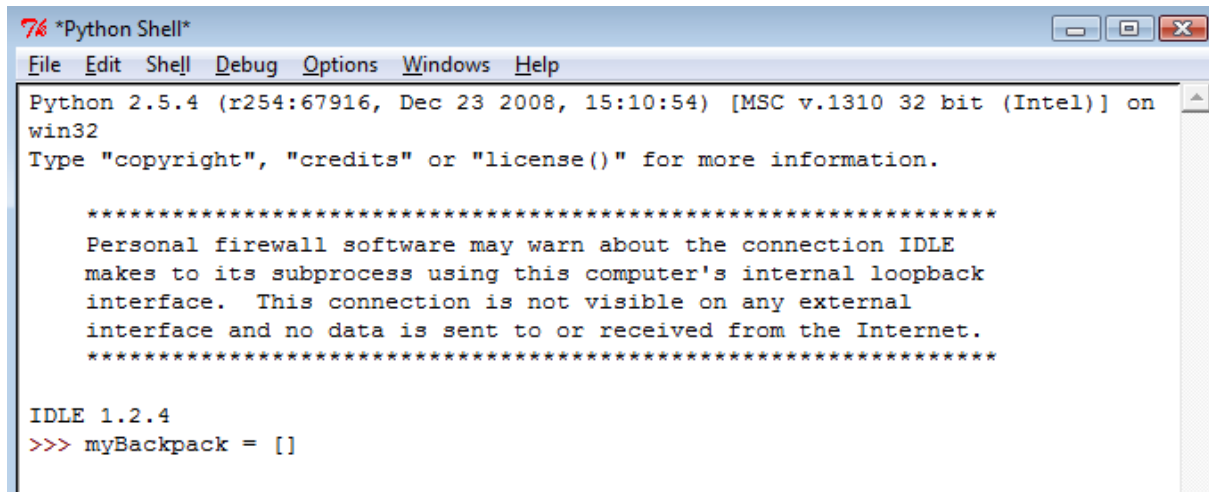
```
myBackpack[ n ]
```

Before we get on to dictionaries let's look at how lists work.

We can assign each of our items to it and see how it works in IDLE, so go to Start->Programs->Python 2.5->IDLE – (this is so much faster to try Python commands)

In Python arrays with no association or key are called lists.

Let's make one. At the command prompt (>>>) type the following...



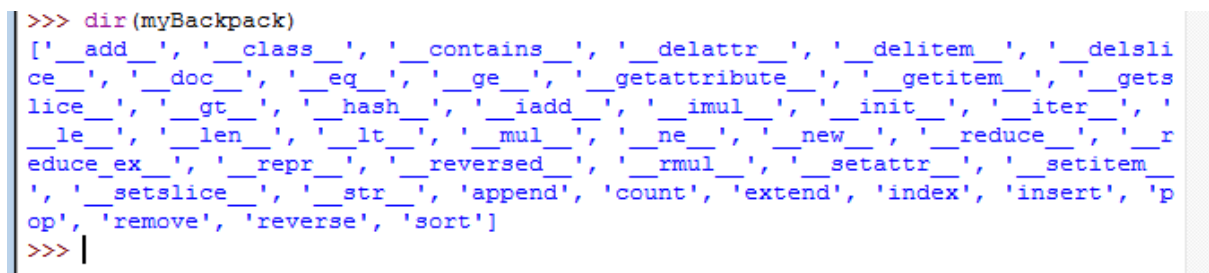
```
*Python Shell*
File Edit Shell Debug Options Windows Help
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.2.4
>>> myBackpack = []
```

This creates an empty list.

Now we can look at the functions of that object (myBackpack) by typing `dir(myBackpack)` and press ENTER...



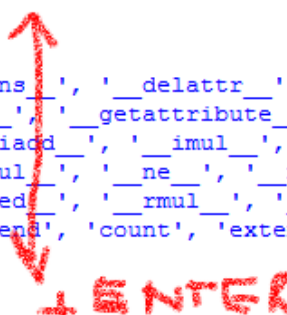
```
>>> dir(myBackpack)
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__', '__delsli
ce__', '__doc__', '__eq__', '__ge__', '__getattr__', '__getitem__', '__gets
lice__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__',
 '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__r
educe_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem_
', '__setslice__', '__str__', 'append', 'count', 'extend', 'index', 'insert', 'p
op', 'remove', 'reverse', 'sort']
>>> |
```

This shows us all the functions we can perform on myBackpack. So let's try one.

Type the following...

But wait, when you type the dot, press TAB key and it will show you a list of possible functions. Arrow up and down to select one then press ENTER. Note when the popup starts, then currently select one is not valid, and if it's the one you want you need to press down then up once to select it.

```
IDLE 1.2.4
>>> myBackpack
>>> dir(myBackpack)
['_add_', '_doc', '_ice_', '_gt_', '_le_', '_len_', '_duce_ex_', '_setslice_', '_op_', '_remove_', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort', 'ins_', '_delattr_', '_delitem_', '_delsli_', '_getattr_', '_getitem_', '_gets_', '_iadd_', '_imul_', '_init_', '_iter_', '_mul_', '_ne_', '_new_', '_reduce_', '_r_', '_sed_', '_rmul_', '_setattr_', '_setitem_', 'pend', 'count', 'extend', 'index', 'insert', 'p']
```



So we end up with the following.

```
>>> myBackpack.count
<built-in method count of list object at 0x0219E8A0>
>>>
```

Let's add something to the list. We can do this two ways because Python rocks.

We can append a value, or can we just tell Python the nth element is something. In many languages trying to add an element a previously unreserved location would cause an error (or exception) because the memory has been illegally accessed.

Let's see what happens if we try ...

```
>>> myBackpack[2] = "Armour"
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    myBackpack[2] = "Armour"
IndexError: list assignment index out of range
>>>
```

Yep – totally fail. Let's try it properly. Let's append a value. Oh and try this too, type just "my" (without the quotes) and then hit TAB – it auto-completes– great hey?

```
>>> myBackpack.append("Armour")
```

So what did we get? Let's try another function on myBackpack.

```
>>> len(myBackpack)
len(object) -> integer
```

See len wants the object – so type in myBackpack...and hit ENTER.

```
>>> len(myBackpack)
1
```

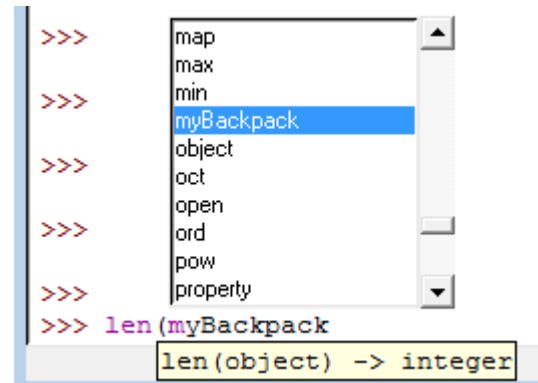
So it's got one element in it.

We want Armour at the end so let's insert two more objects, Food then Sword.

```
>>> myBackpack.insert(0, "Food")
```

```
>>> myBackpack.insert(0, "Sword")
```

Cool now let's look at the list length... but type "len(m" then press TAB – look! Python remembers your object – select it.

A screenshot of a Python interpreter window. The prompt is '>>> len(myBackpack'. A dropdown menu is open, showing a list of suggestions: 'map', 'max', 'min', 'myBackpack' (which is highlighted in blue), 'object', 'oct', 'open', 'ord', 'pow', and 'property'. Below the menu, the text 'len(object) -> integer' is visible, indicating the type of the object being accessed.

```
>>> len(myBackpack
```

Add the closing bracket and then press ENTER.

```
>>> len(myBackpack)
3
```

Now we have 3 elements in the array. Let's see all of them. Just print the whole object...

```
>>> print myBackpack
['Sword', 'Food', 'Armour']
```

How do we address individual elements? What element is Armour for example? You'd think it was element 3 but lists (like most arrays) are zero based, ie 0,1,2,3 ... n

So armour is at index 2...

```
>>> myBackpack[2]
'Armour'
```

If you type index 3 you get an "out of range error" – remember that when you are making your game.

```
>>> myBackpack[3]
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    myBackpack[3]
IndexError: list index out of range
```

What else can we do with lists? We can sort them, so let's see what happens if we do.

```
>>> print myBackpack
['Sword', 'Food', 'Armour']
>>> myBackpack.sort()
>>> myBackpack
['Armour', 'Food', 'Sword']
```

The basic sort did so alphabetically because each “token” or element has an ascii value, a is smaller than z numerically.

We can also reverse the list order...

```
>>> myBackpack.reverse()
>>> myBackpack
['Sword', 'Food', 'Armour']
```

You can also pluck stuff out of it.

```
>>> myBackpack.remove("Food")
>>> myBackpack
['Sword', 'Armour']
```

You might want to find something in lists as well. I’m going to add back the Food first.

```
>>> myBackpack.insert(1, "Food")
>>> myBackpack
['Sword', 'Food', 'Armour']
>>> for item in myBackpack:
    print item

Sword
Food
Armour
```

If you need both the index and the item, use the **enumerate** function:

```
>>> for index, item in enumerate(myBackpack):
    print index, item

0 Sword
1 Food
2 Armour
```

Let’s have a look a few things before we go on. Notice the for... statements ended in colons (:) – this is a standard way to end function definitions and for statements in Python.

Also notice the print was indented. This MUST be done in Python. It's because Python expects code to be written in this way to make it's brevity work. For example if you tried to do the same thing in C++ you might have to do this.

```
for (int i = 0; i < 100; i++ )
{
    print myArray[ i ];
    myArray[i] = "somethingNew";
}
```

In Python it's this...

```
for index, item in enumerate(myArray):
    print item
    myArray[index]="somethingNew"
```

See that you don't need { } braces or semi-colons to end each line. Python basically saves you a lot of typing but if indents are in the place or are not the correct width it causes problems.

If you need only the index, use **range** and **len**:

```
>>> for index in range(len(myBackpack)):
    print index

0
1
2
```

The list object supports the iterator protocol. To explicitly create an iterator, use the built-in **iter** function:

```
>>> myBackpack = ["Food", "Armour", "Sword"]
>>> i = iter(myBackpack)
>>> item = i.next()
>>> item
'Food'
>>> item = i.next()
>>> item
'Armour'
>>> item = i.next()
>>> item
'Sword'
>>> item = i.next()

Traceback (most recent call last):
  File "<pyshell#94>", line 1, in <module>
    item = i.next()
StopIteration
```

See how it errors out when you reach the end?

Also notice how I refilled the array on the first line – another way to assign a list. Remember in the last loop we ended up replacing everything with “somethingNew”

Okay so this is a good way to store stuff, and perhaps a good way to keep all our items held in our backpack, but what if we want particular things to look for and lock in place? This is a typical requirement of the paperdoll you see in many character GUIs for an MMO. So you want to assign a Sword from your Backpack to the Left Hand slot on your paper doll? An easy way to do this is by using Dictionaries.

Dictionaries.

Dictionaries can also be thought of as associative arrays. What this means is that each value is associated with a key. Dictionaries are lists with key, value pairs.

Let’s define a dictionary in IDLE. It’s similar to defining a list but you use the {} braces instead of the [] braces.

```
IDLE 1.2.4
>>> myPaperDoll = {}
>>> dir(myPaperDoll)
['_class_', '__cmp__', '__contains__', '__delattr__', '__delitem__', '__doc__',
 '__eq__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__',
 '__init__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__re
duce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__str__',
'clear', 'copy', 'fromkeys', 'get', 'has_key', 'items', 'iteritems', 'iterkeys',
'itervalues', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
>>>
```

After I defined it, I did a dir() to see what functions I could use.

To add a key, value pair to the myPaperDoll object do this...

```
>>> myPaperDoll["Left Hand"]="Fist"
>>> myPaperDoll
{'Left Hand': 'Fist'}
>>>
```

The key is “Left Hand” and the value is “Fist”. You cannot add another key of “Left Hand”, it will only replace the current value of the key instead. So this is how we assign Sword to Left Hand...

However, so that we don’t try to assign a value to a key that might not exist, let’s check that the key exists first. The reason we do this, is we might have a wandering monster that simply doesn’t have a Left Hand – it might just be a Gigantic Snake with only a Fangs to bite with.

```
>>> if (myPaperDoll.has_key("Left Hand")):
    myPaperDoll["Left Hand"] = "Sword"

>>> myPaperDoll
{'Left Hand': 'Sword'}
>>> |
```

NOTE!!!! The way I typed this in was as follows.

At >>> I typed...

```
if (myPaperDoll.has_key("Left Hand")):
```

... and I pressed ENTER once... which auto-indented for me

Then I type the next line...

```
    myPaperDoll["Left Hand"] = "Sword"
```

Then I pressed ENTER again and once more to tell IDLE I had finished typing the condition and statement. Finally I type on a new line ...

```
myPaperDoll
```

Keys can be of different type within the same dictionary and the same goes for values. Dictionaries aren't just for strings. Dictionary values can be any data type, including strings, integers, objects, or even other dictionaries. And within a single dictionary, the values don't all have to be the same type; you can mix and match as needed.

Dictionary keys are more restricted, but they can be strings, integers, and a few other types. You can also mix and match key data types within a dictionary.

So let's add some modifiers for the Left Hand because they are also going to be wearing a Ring with +2 to Dexterity...

```
>>> myPaperDoll["Left Hand Ring"] = "Ring Of Dexterity"
>>> if (myPaperDoll.has_key("Left Hand Ring")):
    if(myPaperDoll["Left Hand Ring"] == "Ring Of Dexterity"):
        if (myPaperDoll.has_key("Dex Bonus") == False):
            myPaperDoll["Dex Bonus"] = 0
            myPaperDoll["Dex Bonus"] += 2
        ↻
    ← BACKSPACE

>>> myPaperDoll
{'Left Hand Ring': 'Ring Of Dexterity', 'Dex Bonus': 2, 'Left Hand': 'Sword'}
>>>
```

NOTE: Make sure you pay attention to the brackets used; there are square ones for assigning values and round ones for functions such as has_key

What I've done here is similar to the previous example, in the way I have type in the commands, except for the line after...

```
myPaperDoll["Dex Bonus"] = 0
```

It auto-indented for me, but I wanted to step back 1 indent so the next line always got executed, so I pressed the backspace key on the keyboard ONCE then typed...

```
myPaperDoll["Dex Bonus"] += 2
```

I pressed ENTER a few more times to tell IDLE I had finished.

Finally I typed...

```
myPaperDoll
```

... to print out the contents of the dictionary.

So what's going on here?

- 1) Give the player the Ring Of Dexterity (should check the player has this key in future)
- 2) Make sure we have a Left Hand Ring
- 3) Make sure the ring is a Ring Of Dexterity
- 4) Make sure we have a Dex Bonus key because we want to mod it
- 5) Finally if all this goes through, add +2 to the Dex Bonus.

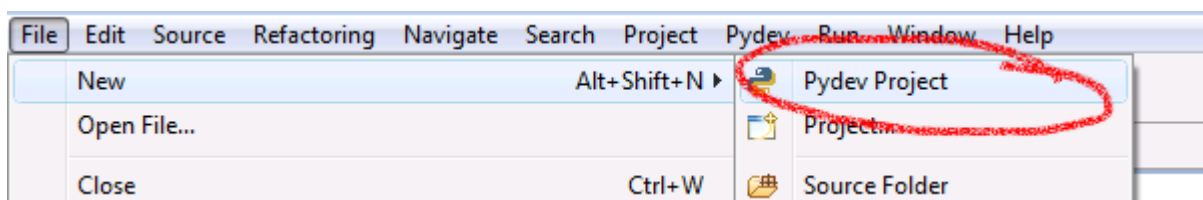
When I printed out myPaperDoll, notice that I have indeed mixed strings with numbers. The Dex Bonus value is a number while the others are strings...

```
>>> myPaperDoll
{'Left Hand Ring': 'Ring Of Dexterity', 'Dex Bonus': 2, 'Left Hand': 'Sword'}
>>> |
```

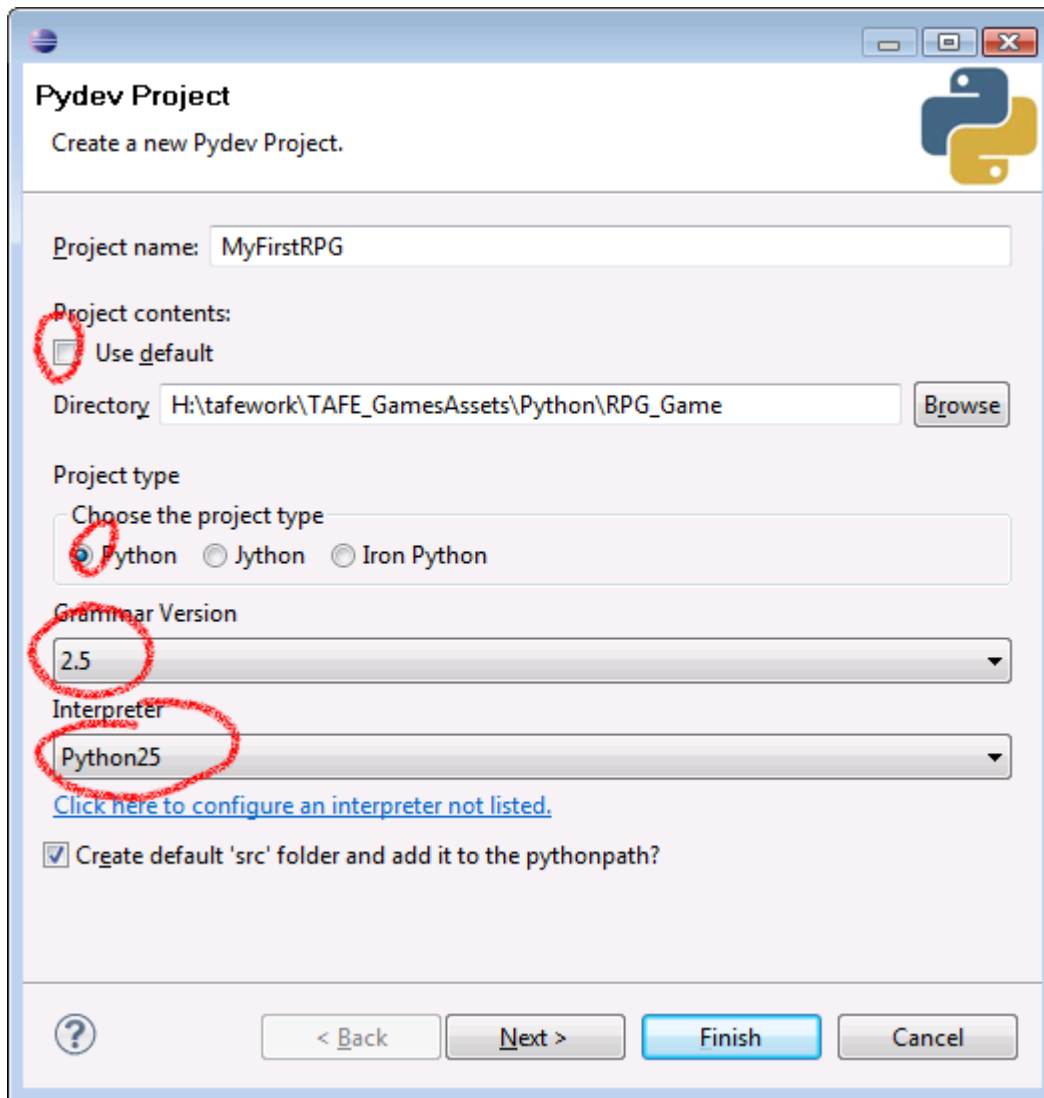
Cool! Now we have this knowledge let's actually write a module to store all this.

Application # 2.

We will make a Main module to run the game from, but we will also write a stand-alone python module so we can import it and use it over and over again for different entities (people/beasts) in the game. So open Eclipse and make a new PyDev Project – store it on H: or on your removable drive...



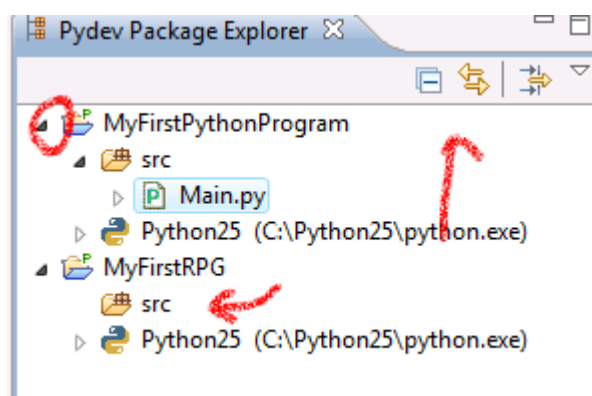
Now you get this screen, so give the project a name, and take care to change the circled settings. To browse for your folder uncheck the USE DEFAULT checkbox...



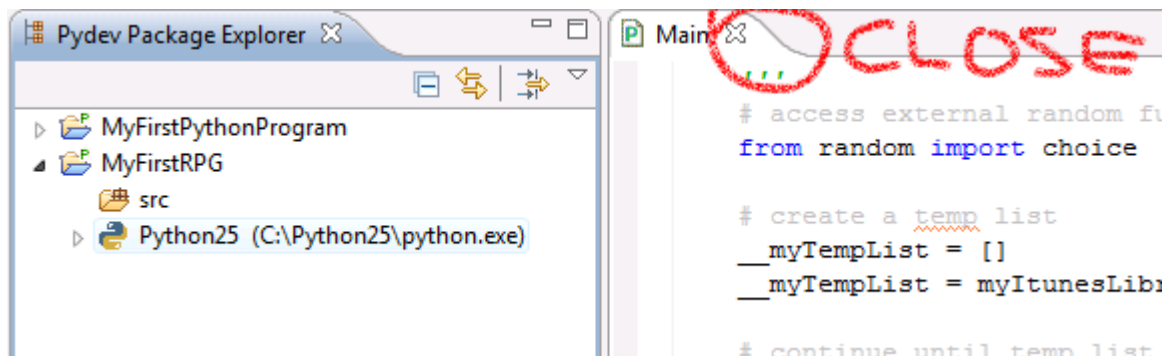
If you have no Interpreter defined check the Lecture1a document on page 5. You can find that doc here...

http://www.drewfx.com/TAFE/python/Lecture_Week1a.pdf

So after pressing Finish you will probably see this because we were previously working on another project. Just hit the little triangle circled in this pic and it will collapse. We are interested in the second project.

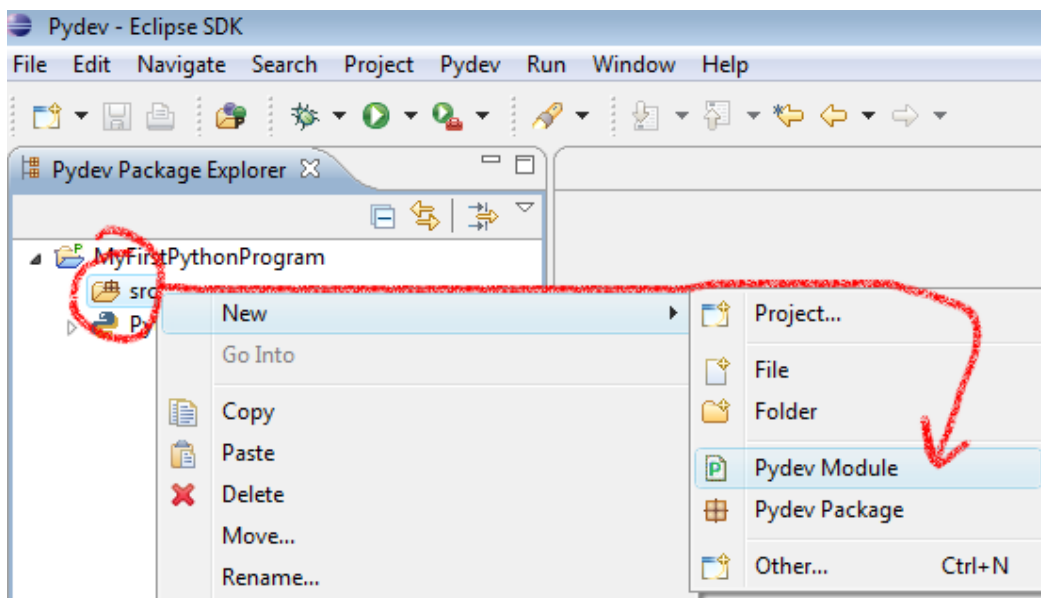


So your view should look like this now... close the Main.py from the other project if it's still open.

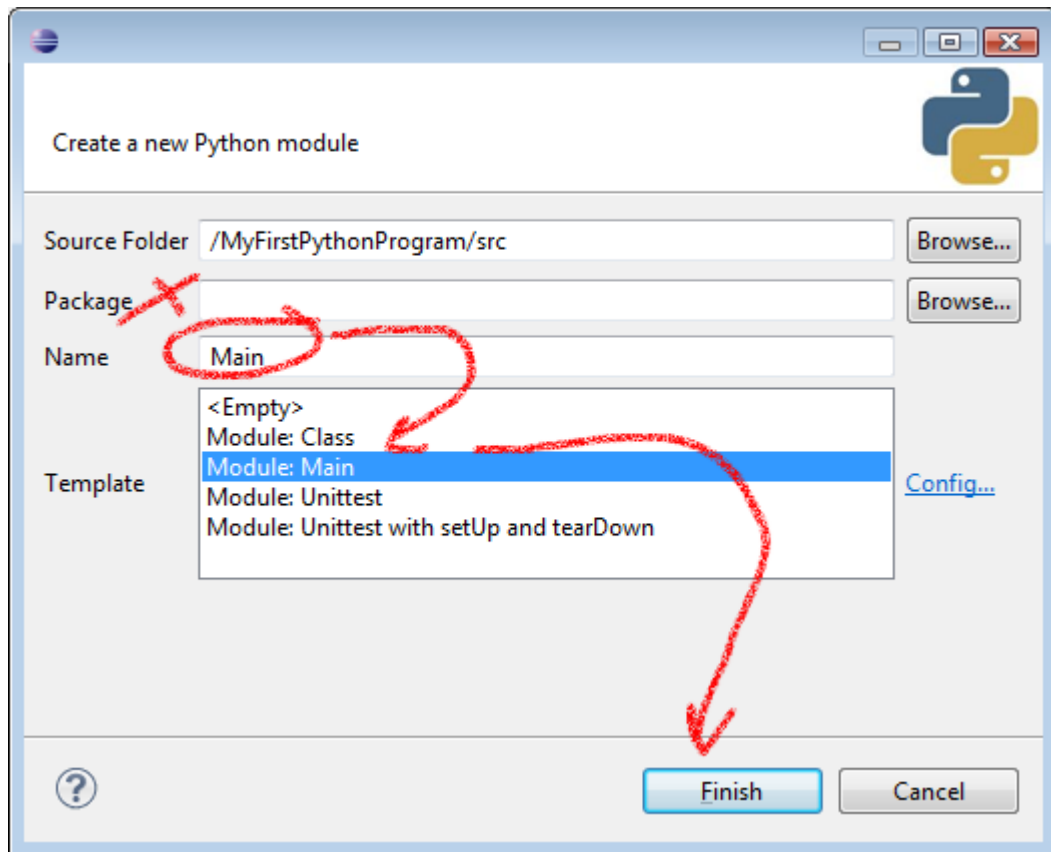


Now add a new Main.py module...

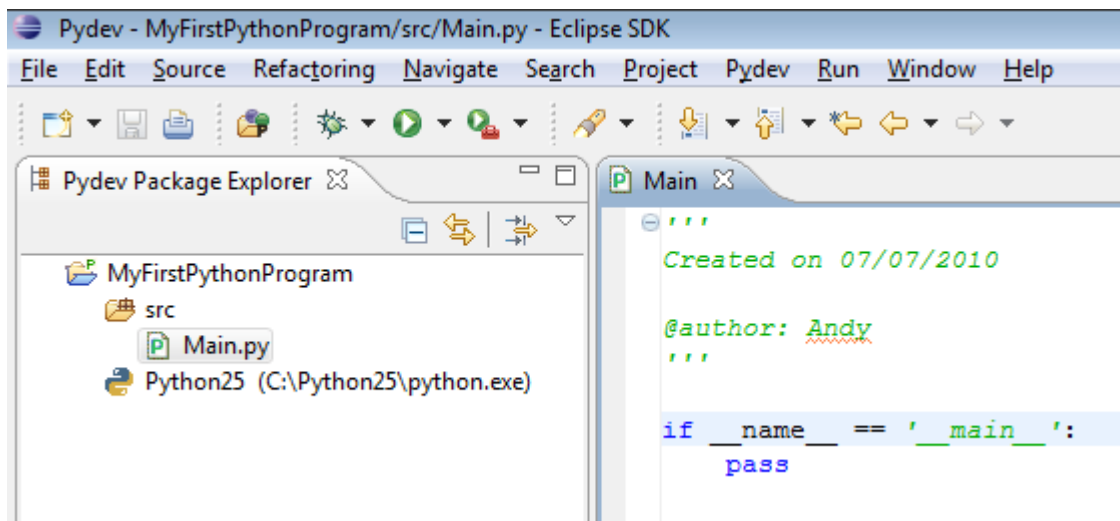
1. Right click on the “src” folder and click on New->Python Module NOTE if you cant see PyDev Module as in the pic below, you will find it under Other...



1. Call the file Main (no Package at this stage), select Template as Module:Main and click Finish

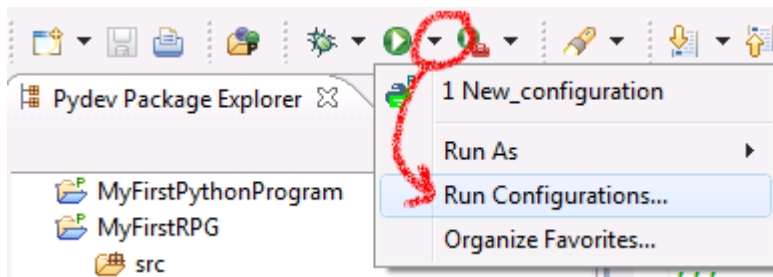


2. Now you'll see this. This is the template code to run as a module.

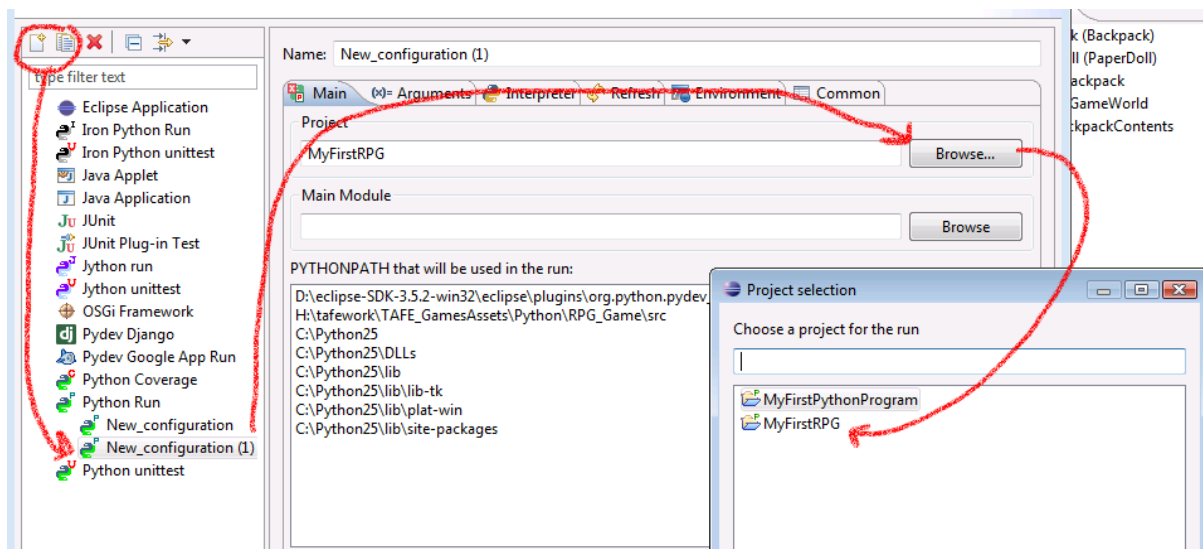


Yes this is the same as we did last time.

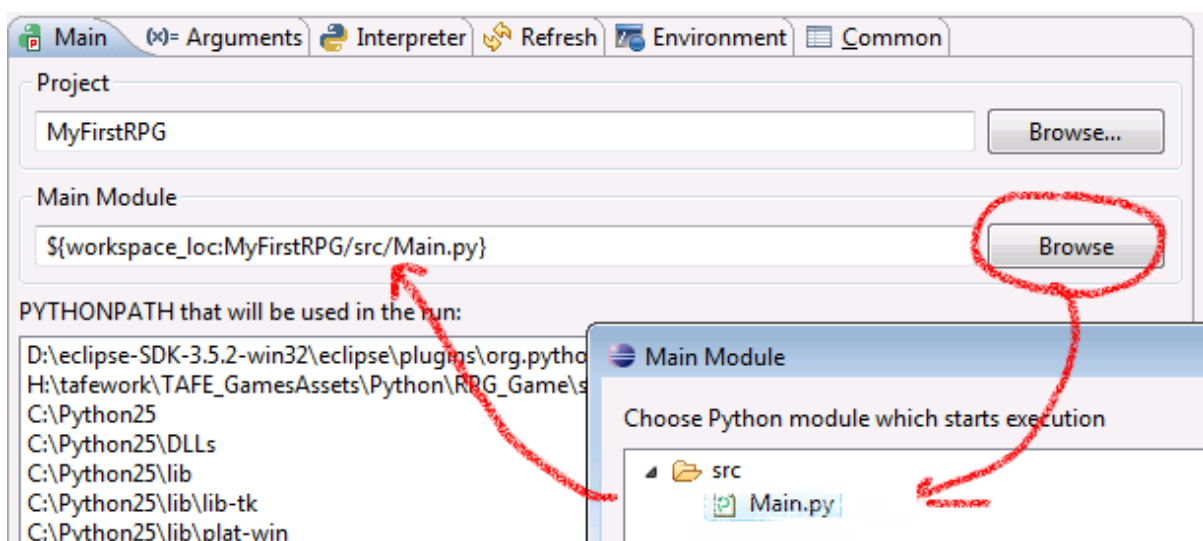
You also want to set up a Run Configuration again.



But this time click on the New button circled to create a new run configuration then fill out the fields as before...



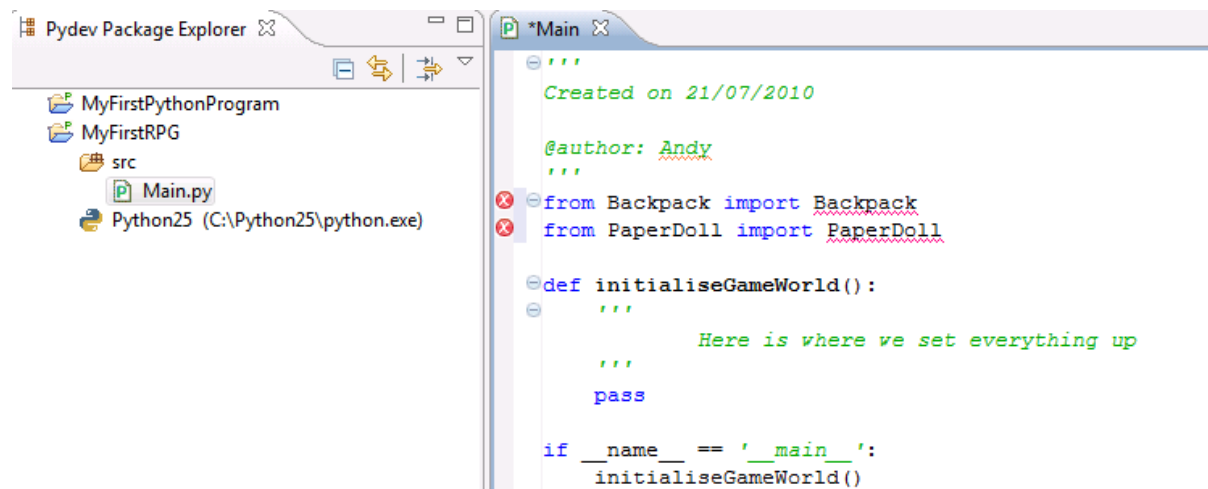
Then click on the Main Module browse and browse to src\Main.py...



Now we want to have some sort of initialisation so scrub out pass, replace it with

```
initialiseGameWorld()
```

... and type in the following. Notice that 2 imports have been added to the top. We will write these next, so ignore the red X's for now.



Add the playerBackPack and the next two functions below the import PaperDoll line...

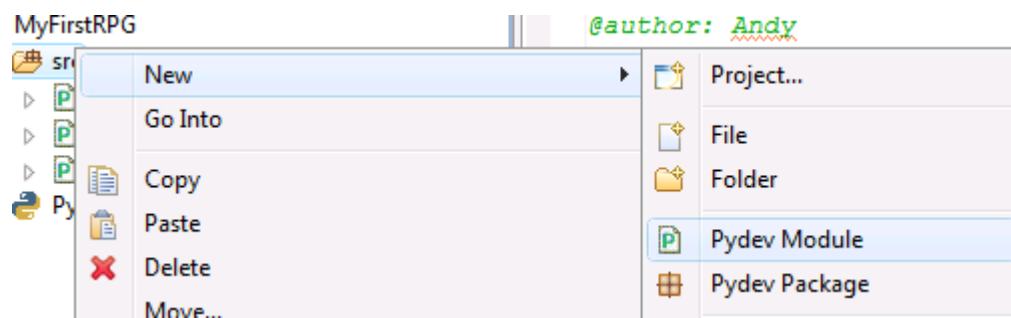
```
from PaperDoll import PaperDoll  
  
playersBackpack = Backpack()  
  
def initialiseGameWorld():  
    '''  
        Here is where we set everything up  
    '''  
    playersBackpack.addToBackpack("Sword")  
  
def showBackpackContents():  
    '''  
        Typically run when you press the C or B key  
    '''  
    playersBackpack.show()
```

And add showBackpackContents after initialiseGameWorld() at the bottom...

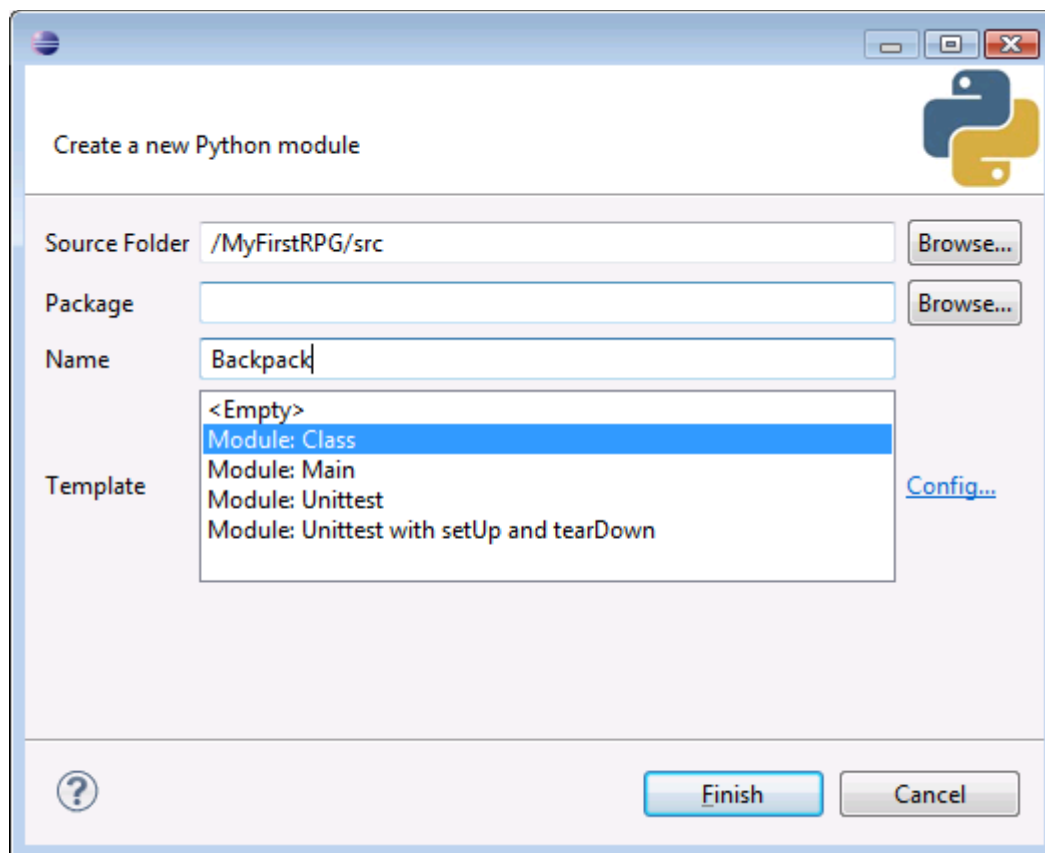


Okay so now we need to add 2 classes, Backpack and PaperDoll.

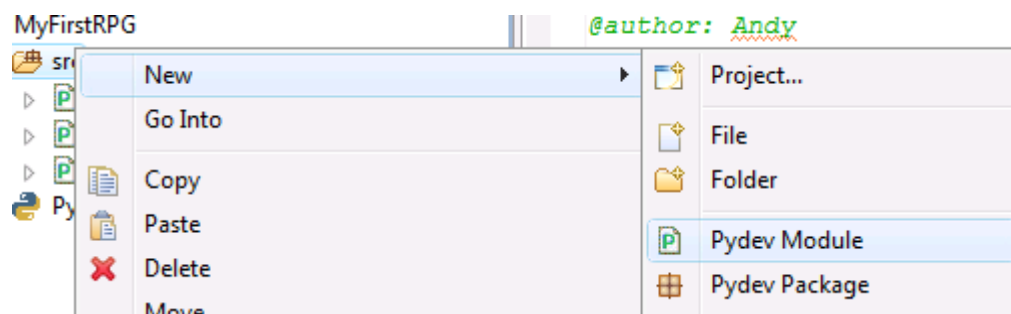
Right click on src and select New->Python Module (or New->Other->Python Module,

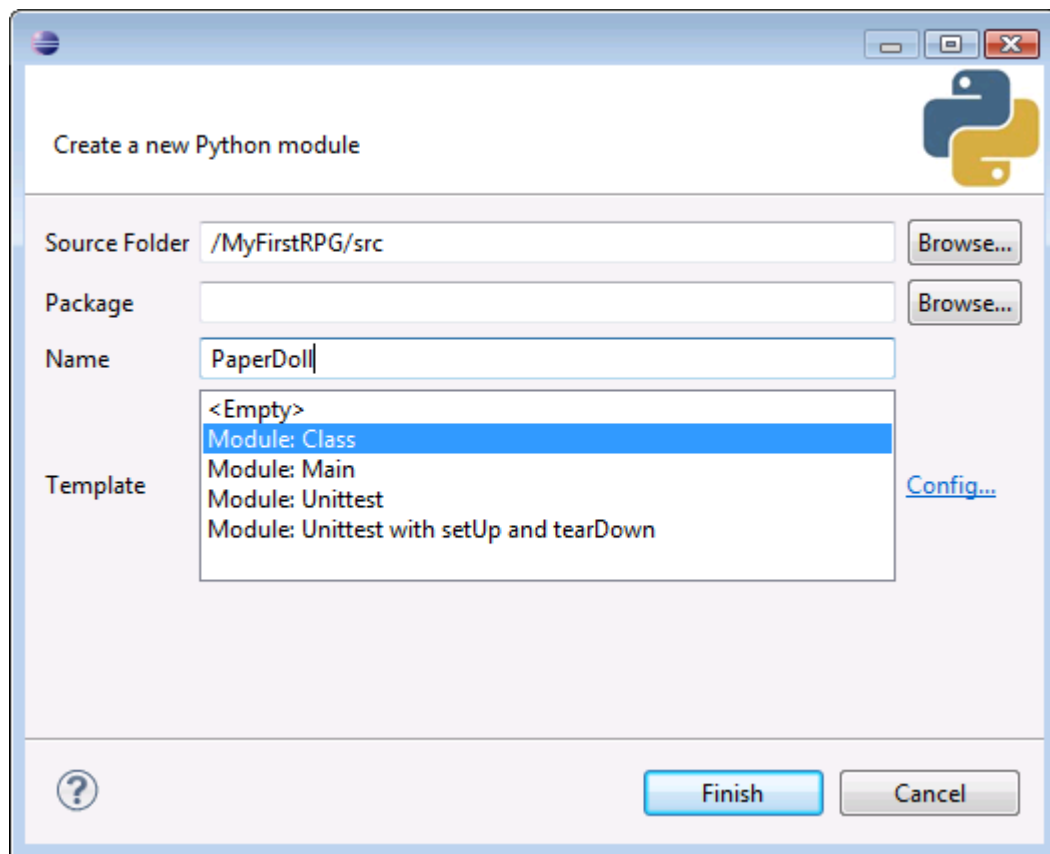


then change the Template to Module:Class and call it Backpack, and hit Finish

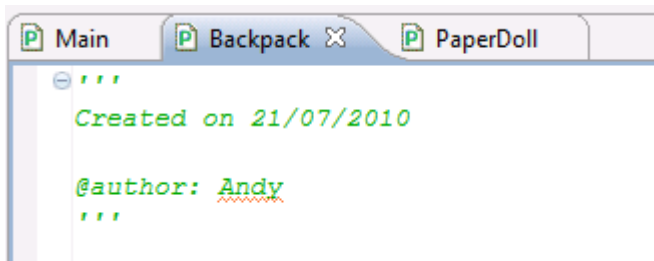


Do the same again for PaperDoll...





Then select Backpack.py from the top tabs...



And overwrite the template code to look like this... make sure you change the parameter lists () to match...

```

'''
Created on 21/07/2010

@author: Andy
'''

class Backpack():
    '''
    classdocs
    '''

    def __init__(self):
        '''
        Constructor
        '''
        self.myBackpack = []

    def addToBackpack(self, item):
        '''
        Allows calling code to add item to private attribute
        '''
        self.myBackpack.append(item)

    def show(self):
        '''
        Show the contents of the backpack
        '''
        print self.myBackpack

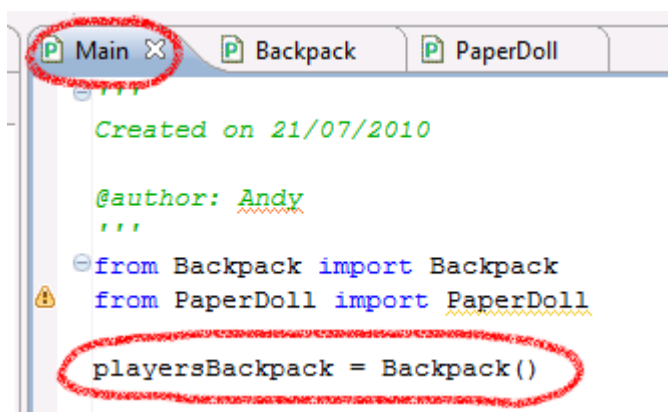
```

Notice we have added two new functions, addToBackpack and show.

Also notice every function uses self as the first parameter in the brackets, this must be done for classes.

Def __init__ is the constructor. This allows us to make a new backpack in the main.py as required.

Click on the Main.py tab and add the following code as circled in red...



```

'''
Created on 21/07/2010

@author: Andy
'''

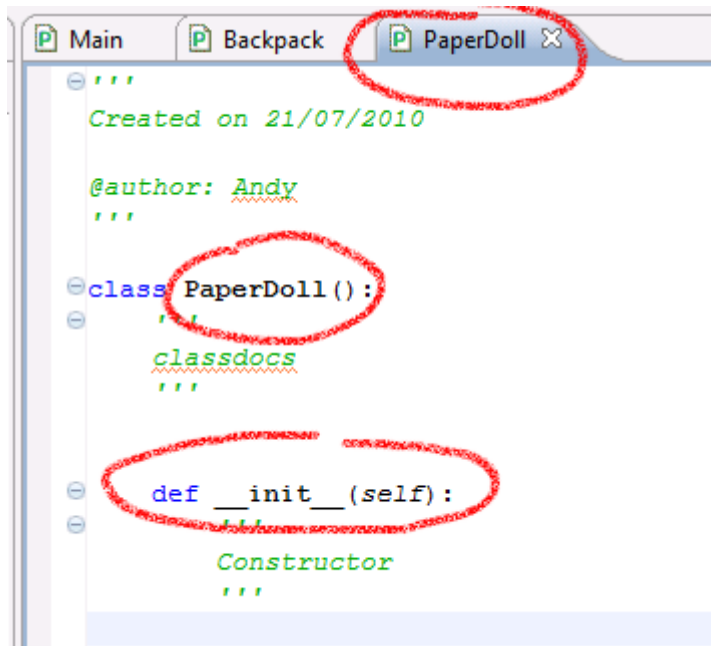
from Backpack import Backpack
from PaperDoll import PaperDoll

playersBackpack = Backpack()

```

This creates a new Backpack object.

Click on the PaperDoll.py in the top tabs and make sure the code is modified to look like this...



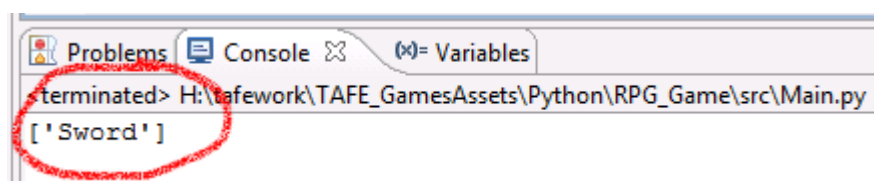
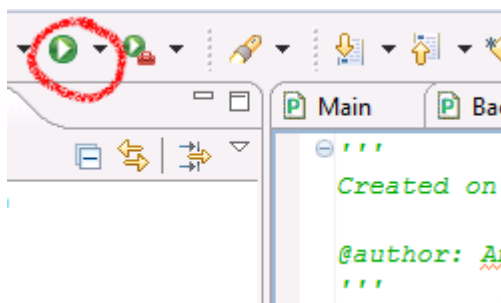
```
'''
Created on 21/07/2010

@author: Andy
'''

class PaperDoll():
    '''
    classdocs
    '''

    def __init__(self):
        '''
        Constructor
        '''
```

Click on the Main.py tab at the top (this is the one that is tied to the Run Configuration) If you run the program now it will show ["Sword"] in the console...



So deconstructing it we can now have a look at the initialiseGameWorld() function and see that once the playerBack has been created above it (making it global) we can call the Backpack's function addToBackpack and give it an item, and then Backpack's show function to display the contents of the backpack.

```

from Backpack import Backpack
from PaperDoll import PaperDoll

playersBackpack = Backpack()

def initialiseGameWorld():
    """
        Here is where we set everything up
    """
    playersBackpack.addToBackpack("Sword")

def showBackpackContents():
    """
        Typically run when you press the C or B key
    """
    playersBackpack.show()

if __name__ == '__main__':
    initialiseGameWorld()
    showBackpackContents()

```

Finally let's instantiate (or create new) the PaperDoll for this player, so add the code just above `playerBackpack = Backpack()`...

```

@author: Andy
"""
from Backpack import Backpack
from PaperDoll import PaperDoll

playersPaperDoll = PaperDoll()
playersBackpack = Backpack()

```

And we need to add some more functionality to PaperDoll to use it, so click on the top tab to change to PaperDoll.py and add the following code...

```

class PaperDoll():
    """
    classdocs
    """

    def __init__(self):
        """
        Constructor
        """
        # create an empty dictionary
        self.myPaperDoll = {}

    def addSlotToPaperDoll(self, slotName):
        """
        Adds a slot as a key
        """
        self.myPaperDoll[slotName] = None

    def updateSlot(self, slotName, value):
        """
        Put an item in the slot
        """
        # make sure the slot exists
        if self.myPaperDoll.has_key(slotName):
            self.myPaperDoll[slotName] = value

    def show(self):
        """ Display the contents of the paperdoll """
        print self.myPaperDoll

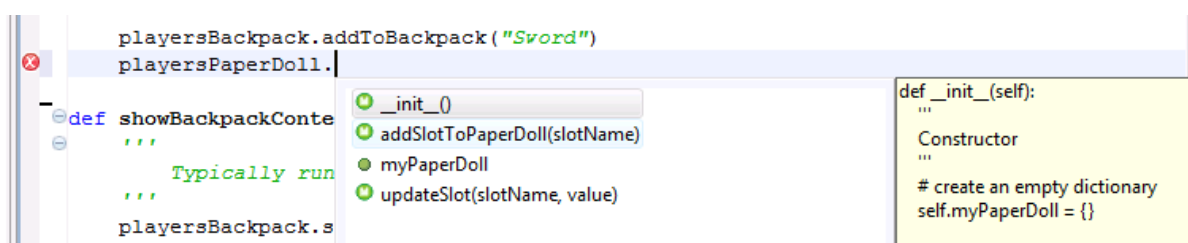
```

Notice in addSlotToPaperDoll, the use of the None as a value. This will allow us to add whatever we want later.

Now click back on to Main.py and add the final changes.

Okay so type in this code...

Notice when you start typing after playersPaperDoll. (dot) it shows you the current functions allowable...



```

from Backpack import Backpack
from PaperDoll import PaperDoll

playersPaperDoll = PaperDoll()
playersBackpack = Backpack()

def initialiseGameWorld():
    """
        Here is where we set everything up
    """
    playersBackpack.addToBackpack("Sword")
    playersPaperDoll.addSlotToPaperDoll("Left Hand")
    playersPaperDoll.updateSlot("Left Hand", "Sword")

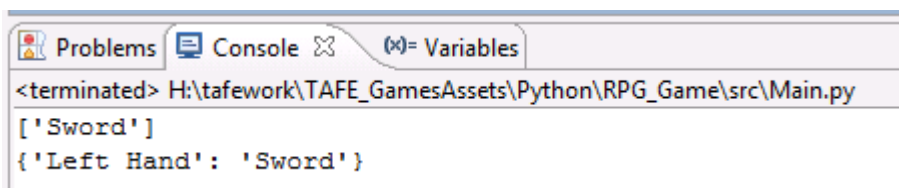
def showBackpackContents():
    """
        Typically run when you press the C or B key
    """
    playersBackpack.show()

def showPaperDoll():
    """
        Show the paperdoll
    """
    playersPaperDoll.show()

if __name__ == '__main__':
    initialiseGameWorld()
    showBackpackContents()
    showPaperDoll()

```

Great! Now run it (make sure Main.py is selected) and you will see the following in the console.



```

<terminated> H:\tafe\work\TAFE_GamesAssets\Python\RPG_Game\src\Main.py
['Sword']
{'Left Hand': 'Sword'}

```

Well done. Well on your way to making an RPG!

You will be able to make one backpack per entity in the game (player, beast) and also give every one of them, their own paperdoll too. This is classes in action. We'll recap next week.