Python – week1

Contents

Intro:

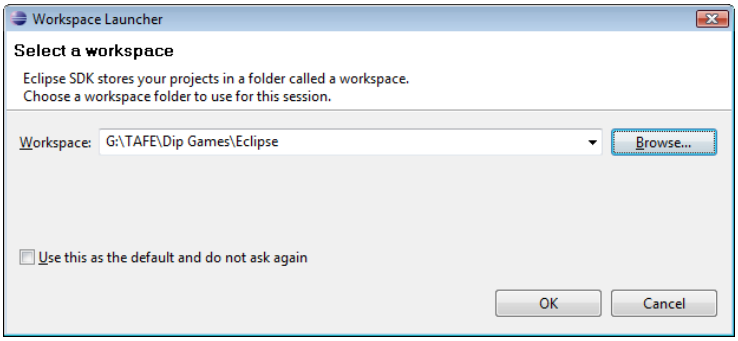
Pygame is a python wrapper for SDL, written by Pete Shinners. What this means is that, using pygame, you can write games or other multimedia applications in Python that will run unaltered on any of SDL's supported platforms (Windows, Unix, Mac, beOS and others).

If we get some time towards the end of term4 we will look at wxPython and py2exe.

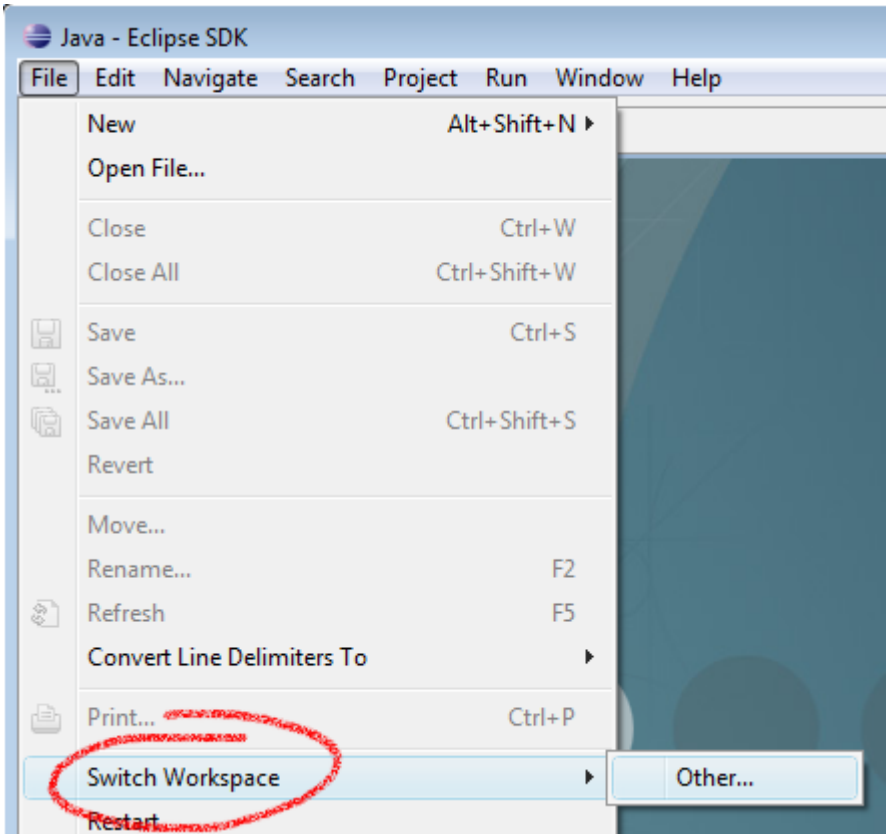
Getting Started with Eclipse

Eclipse unpacks to a folder ready to use. Find the exe and run it. We need to add PyDev so follow these instructions.

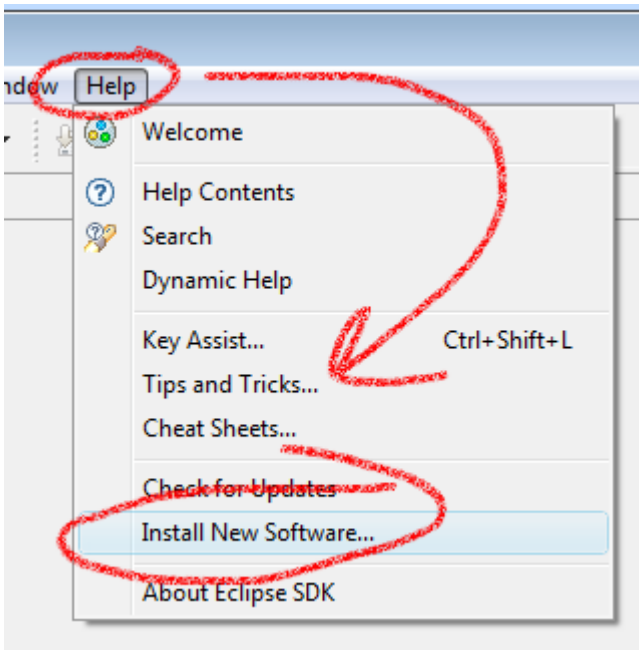
1. When Python first starts it will ask you for a default workspace. As TAFE computers' desktops get nuked every time there is a problem set this to your Hard Drive instead.



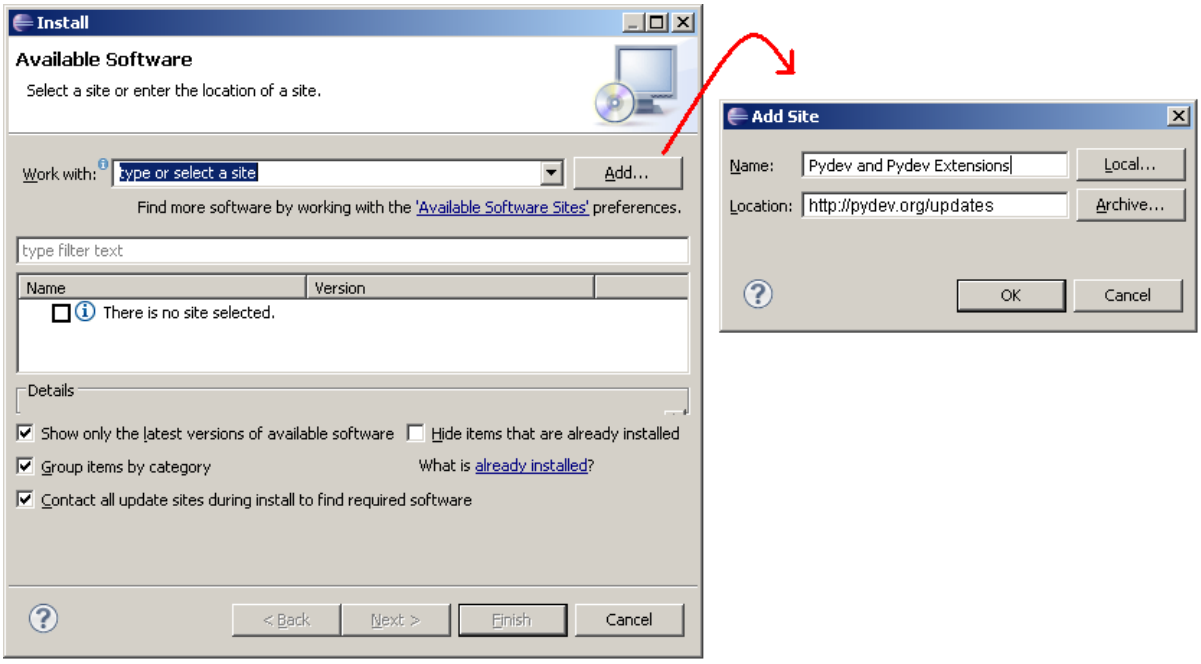
2. If you run into problems with workspaces, once the navigator has started just switch using this menu...



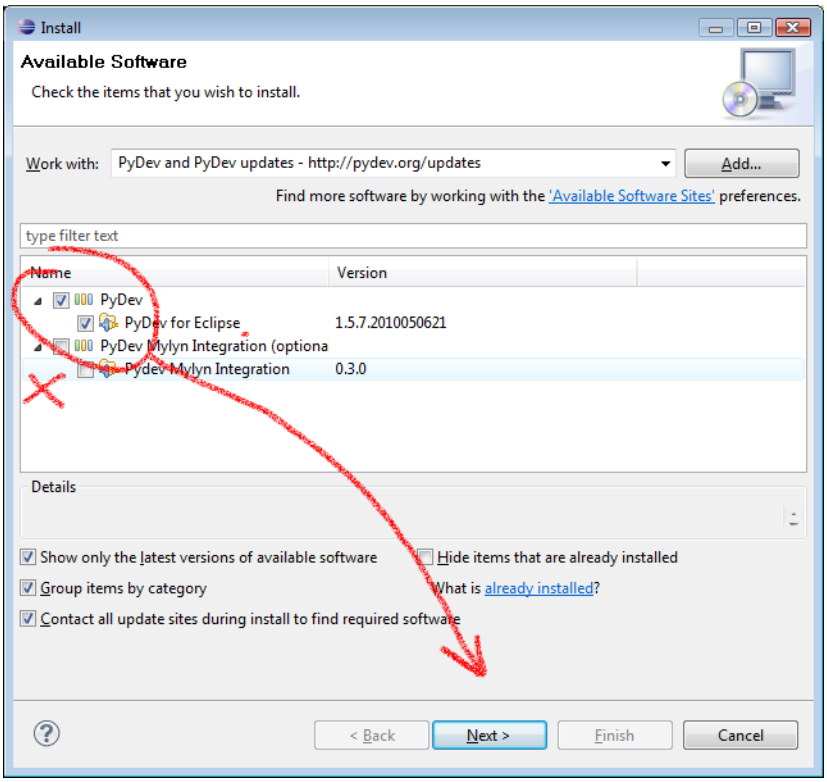
3. To install Pydev and Pydev Extensions using the Eclipse Update Manager, you need to use the **Help > Install New Software...** menu (note that in older versions, this would be the 'Find and Install' menu).



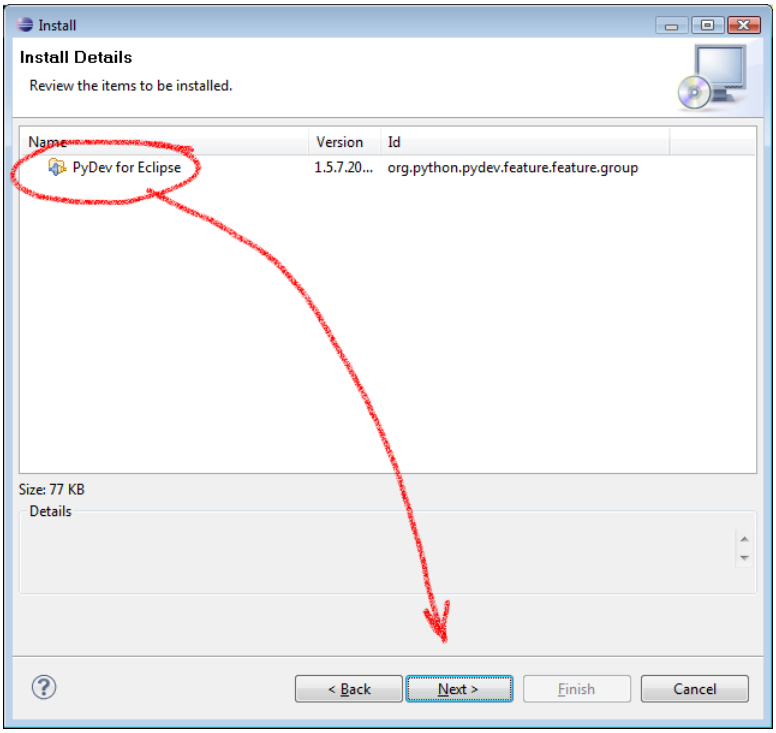
4. Type in a custom name for the plugin addition and add the link provided (http://pydev.org/updates)



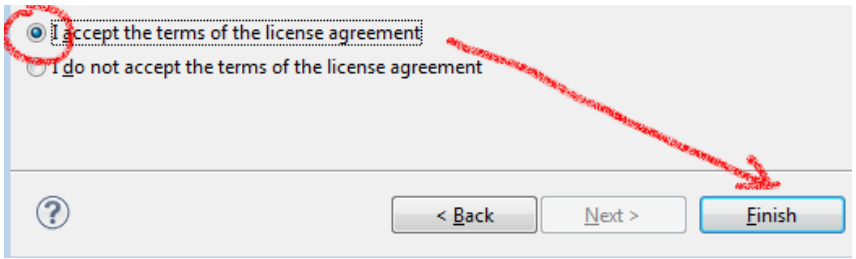
5. Now you get this screen. Only tick PyDev and it's child then click next.



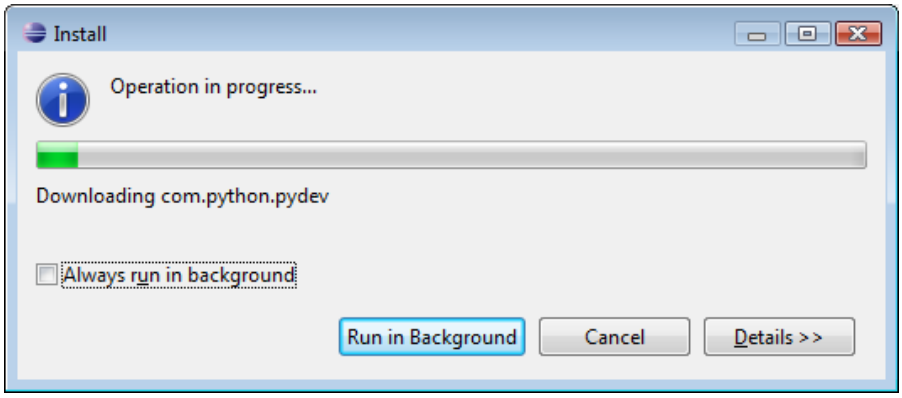
6. You will see this window next once it's located the resources online. Click Next to continue.



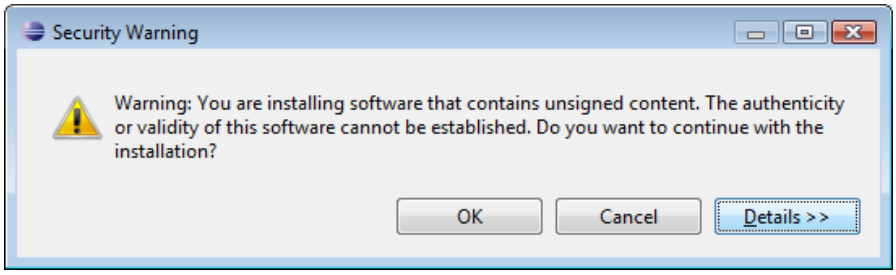
7. Accept the terms and conditions and click on Next.



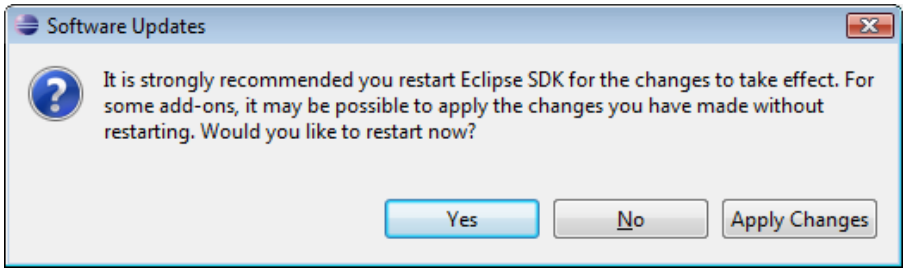
8. Next you will see the updating in progress – just let it run.



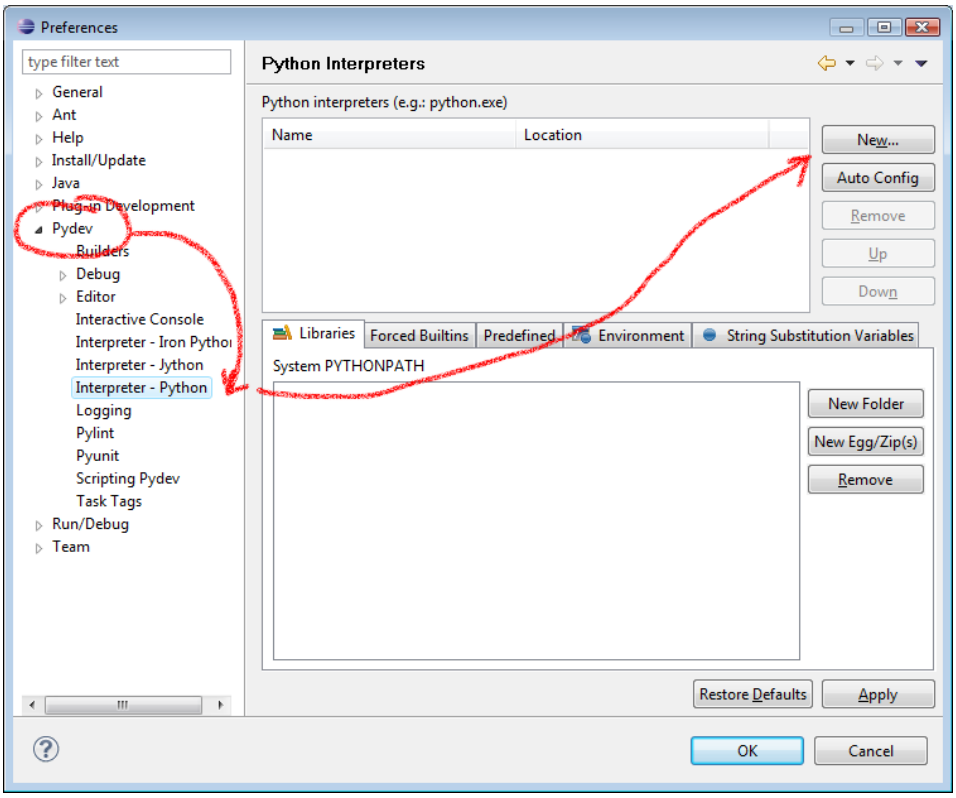
9. If you see this warning (probably only Vista) just click Yes



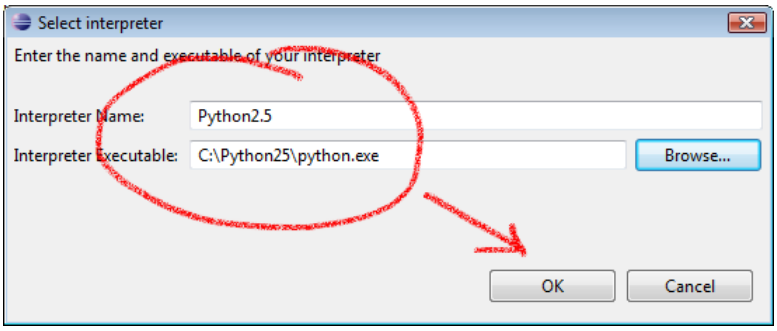
10. Once completed you will see this dialog. Restart Eclipse by clicking on Yes.



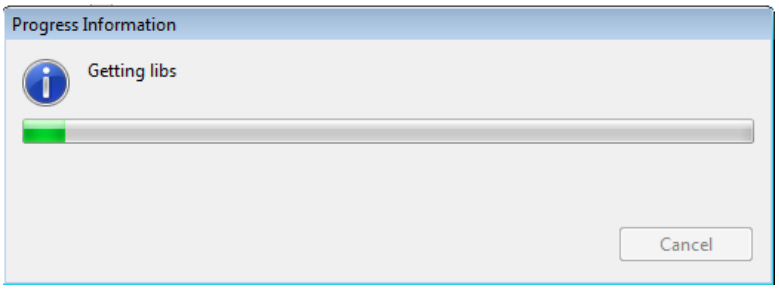
11. Configuration of Eclipse. You have to maintain in Eclipse the location of your Python installation. Open in the menu Window -> Preference and select Pydev-> Interpreter Python.



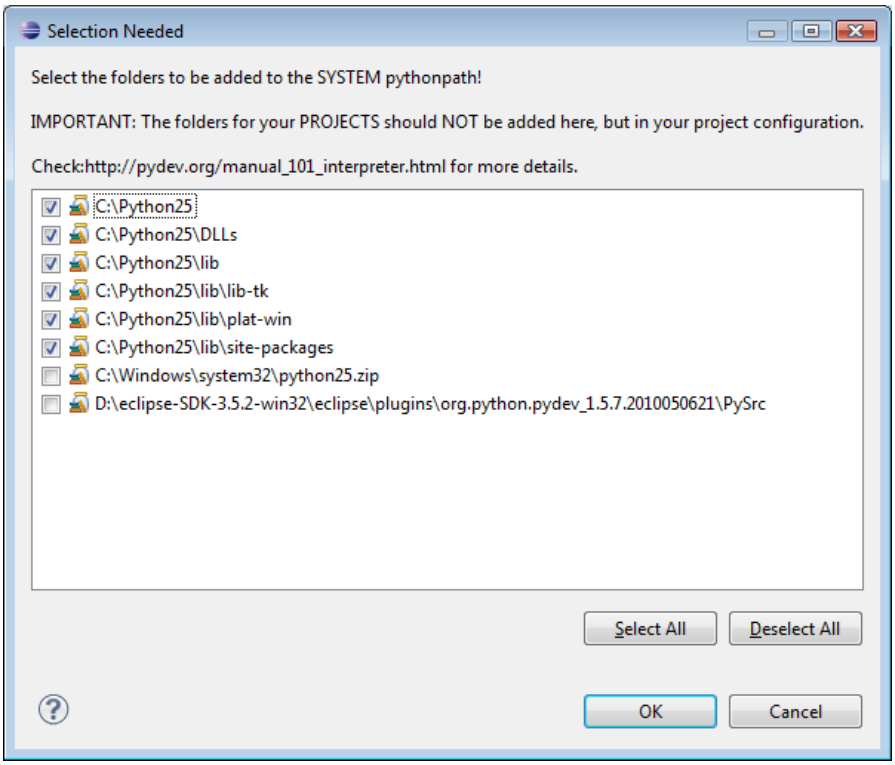
12. Press New then point the Python25 folder – it should be off the root of C:\. Now press okay



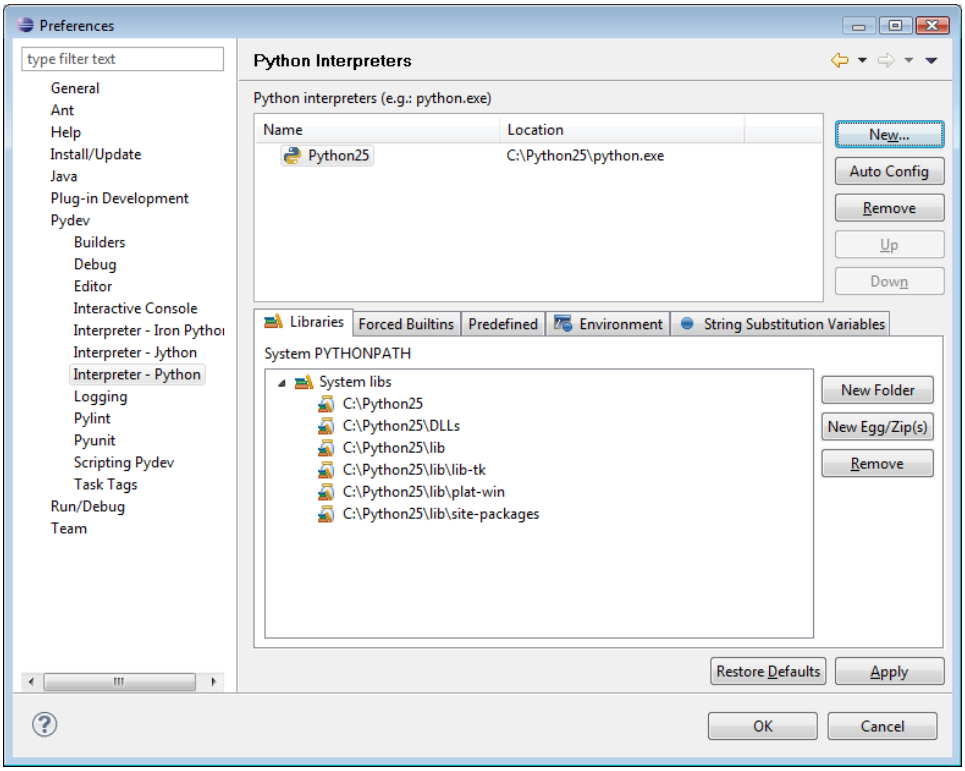
13. You will see this dialog open as it processes the files.



14. Finally you will see this screen.



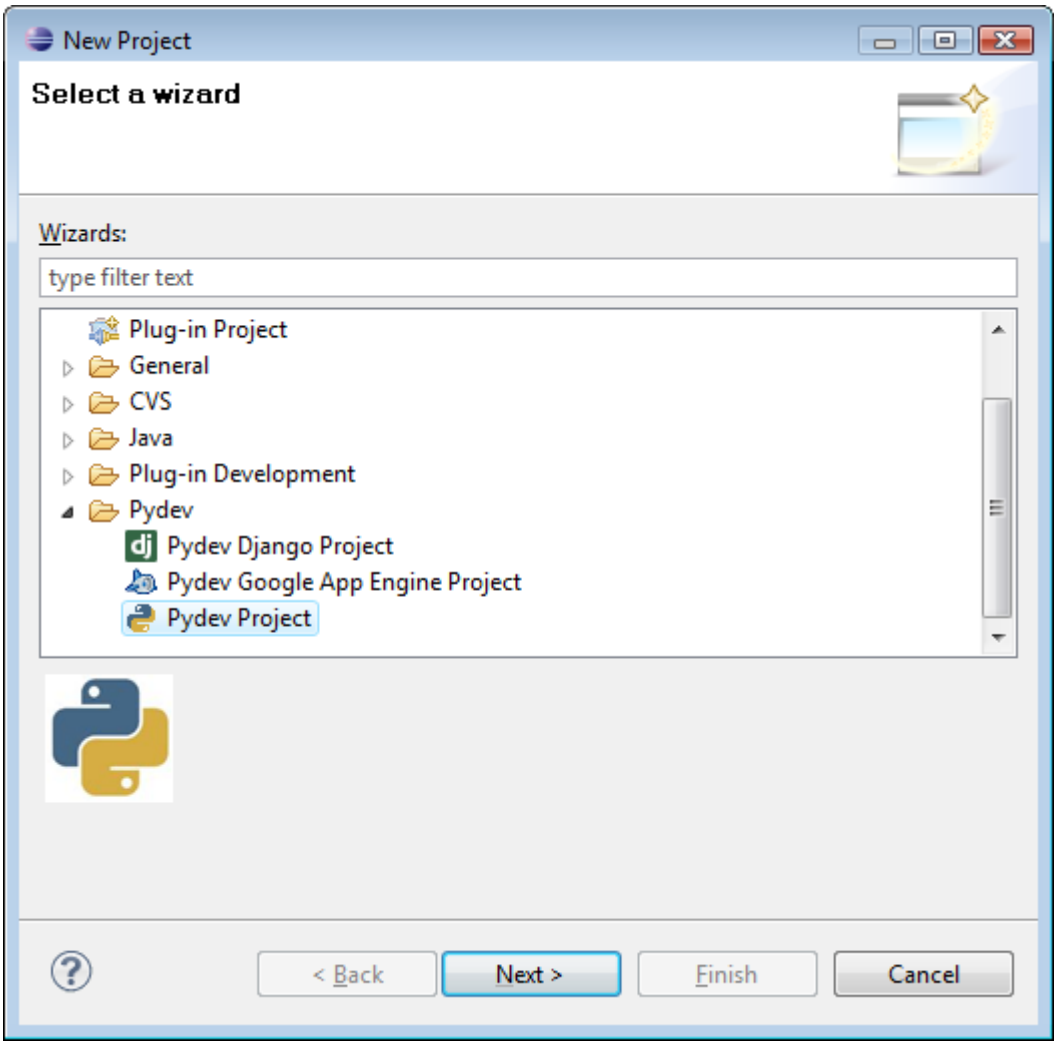
15. Don't change anything – just click okay.
16. Now you will see the original screen look like this. You are ready to start programming in Python!



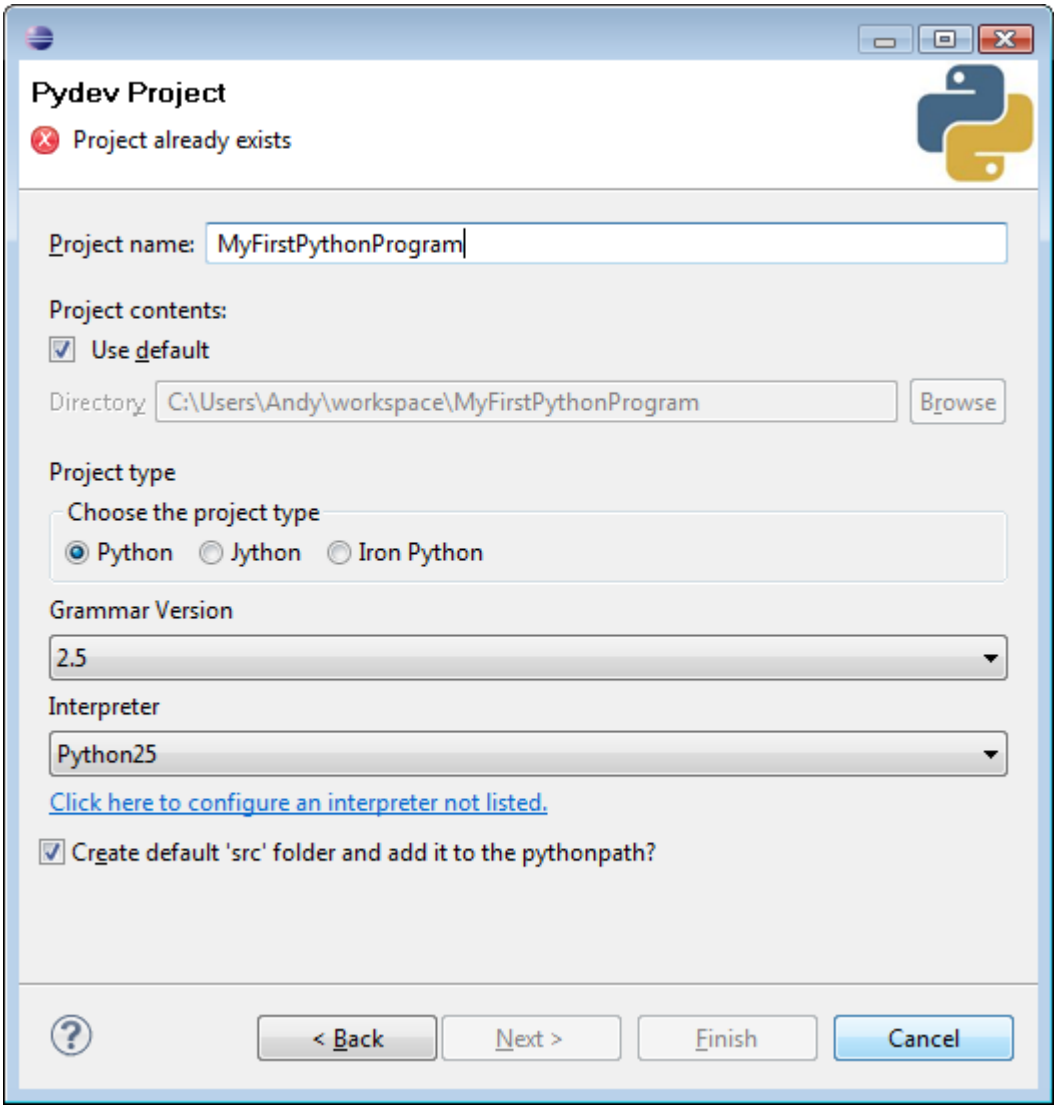
Lessons:

Your first Python program.

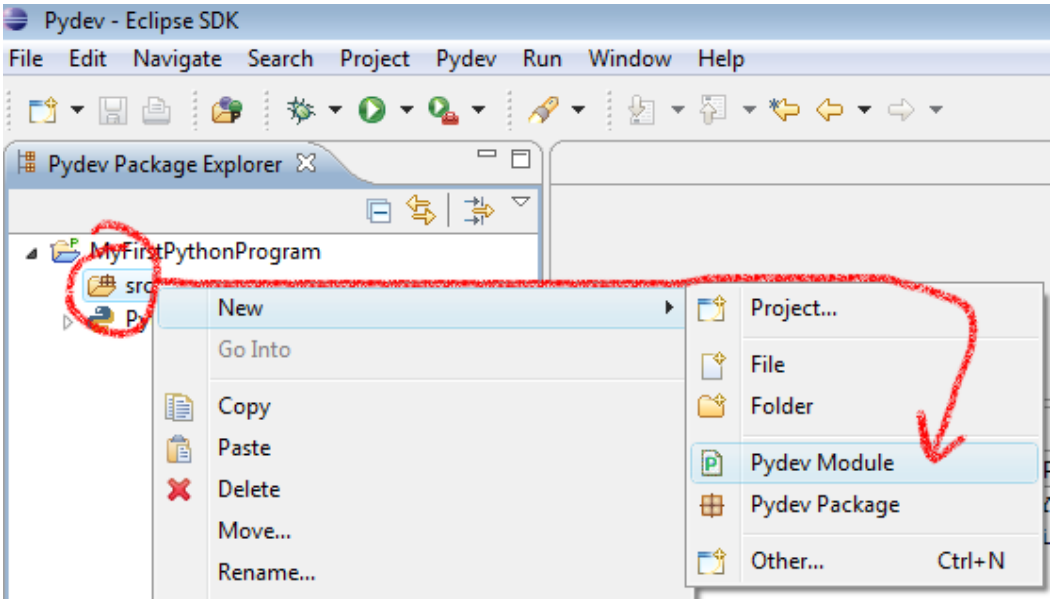
1. Open Eclipse
2. File->New Project -> PyDev->PyDev Project



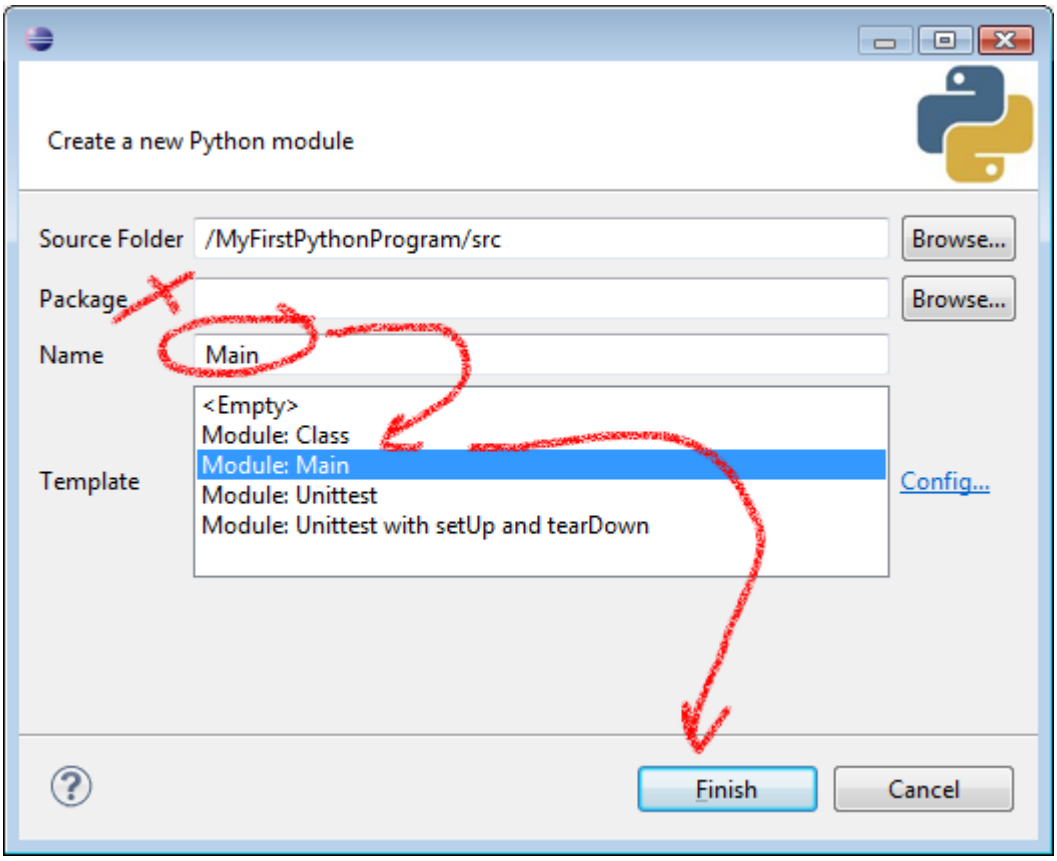
3. Click Next then change interpreter to Python25 and Grammer to 2.5, and give it a name at the top. Click Finish



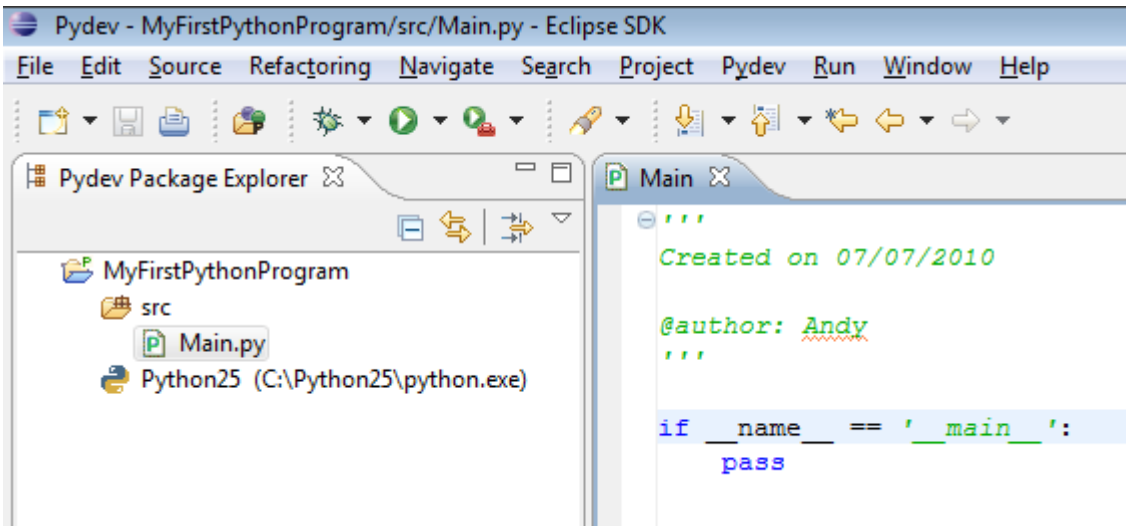
4. Right click on the “src” folder and click on New->Python Module



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



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



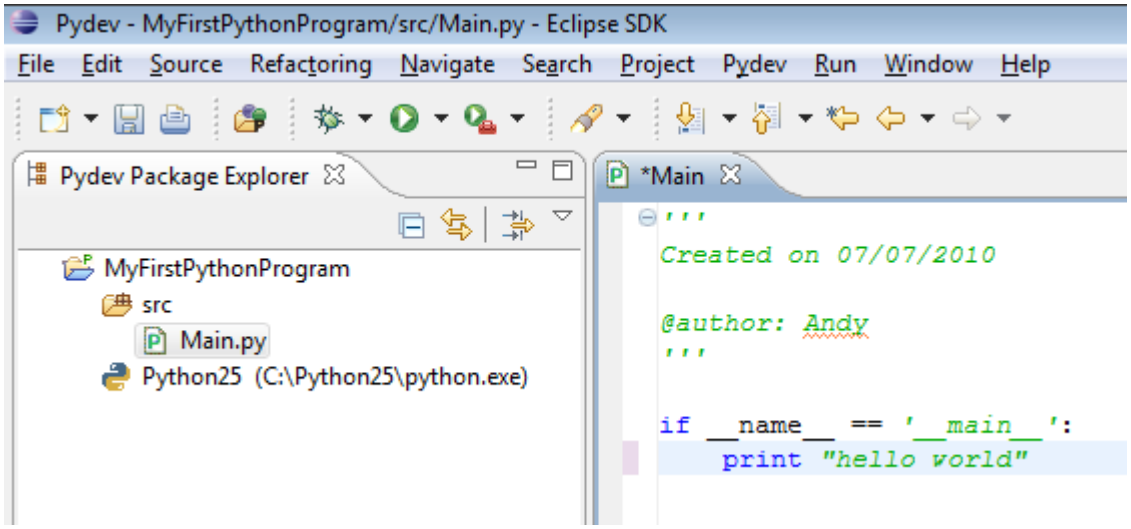
7. What does it mean? Let's take a look.

8. The `'''` characters at the top is the standard DOCSTRING notation and can be considered as commenting. Comment as much as you can. You always end a comment block with a terminating ''' as seen after `@author: Andy`

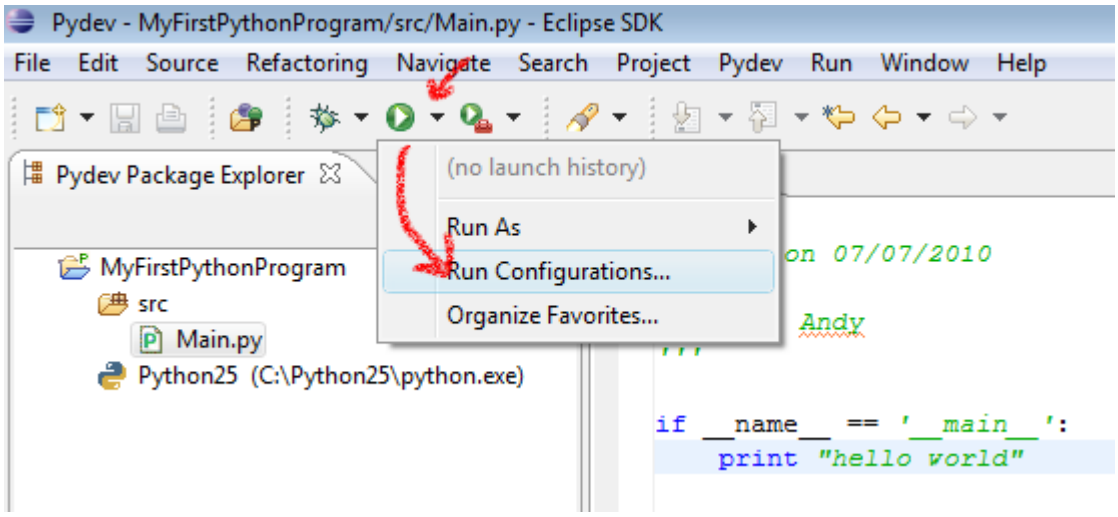
9. The next line is curious `if __name__ == '__main__': then pass`. This is a trick for Python which allows you to have this .py file exist as a module or a standalone file. See appendix for more information.

10. The `pass` just means that – do nothing and move on. But we want it to do something.

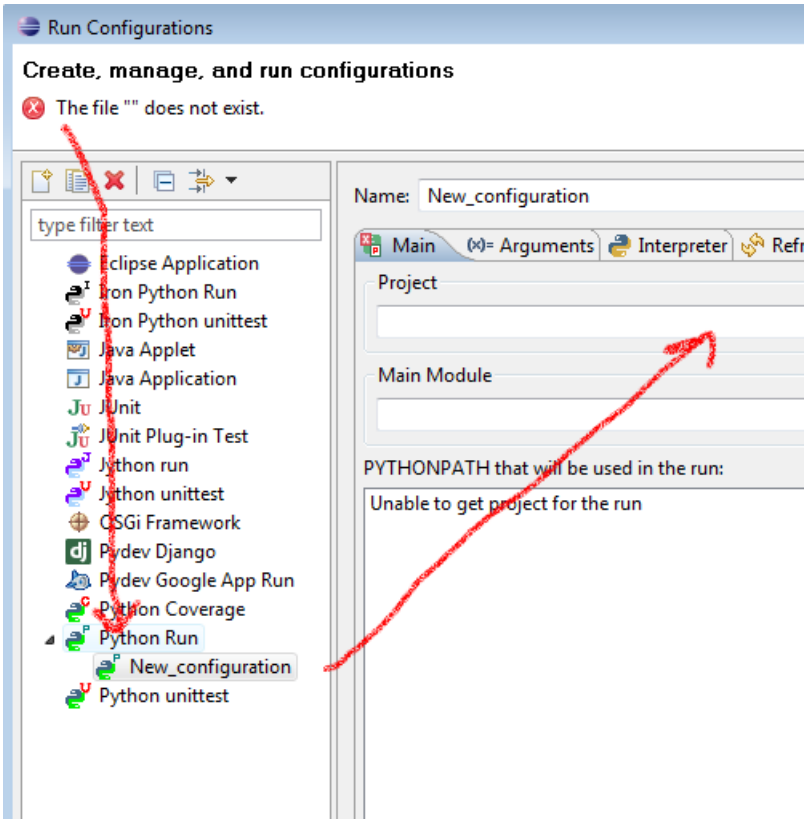
11. Introducing the *print* statement! Print mainly prints characters to the screen. So type in the follow. Erase *pass* with the words `print "hello world".`



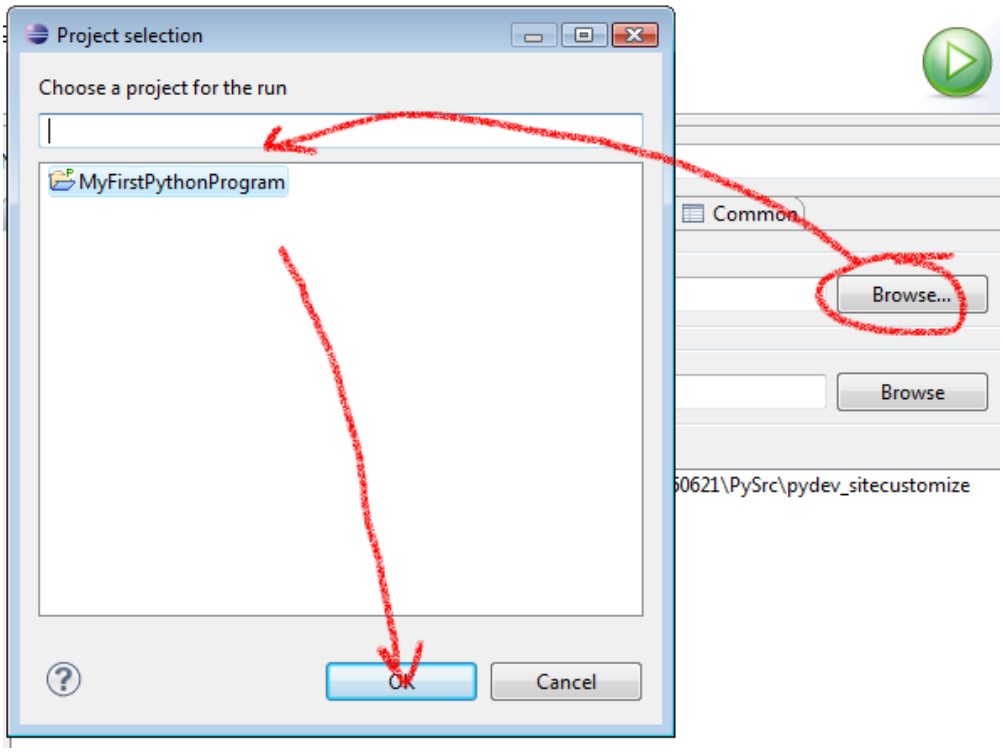
12. Now you need to run it. But before that you need to create a run configuration. Up the top you'll see a green play button – a big one – there's a down arrow next to it – click on that to bring up the menu below. Click on that then click on Run Configurations.



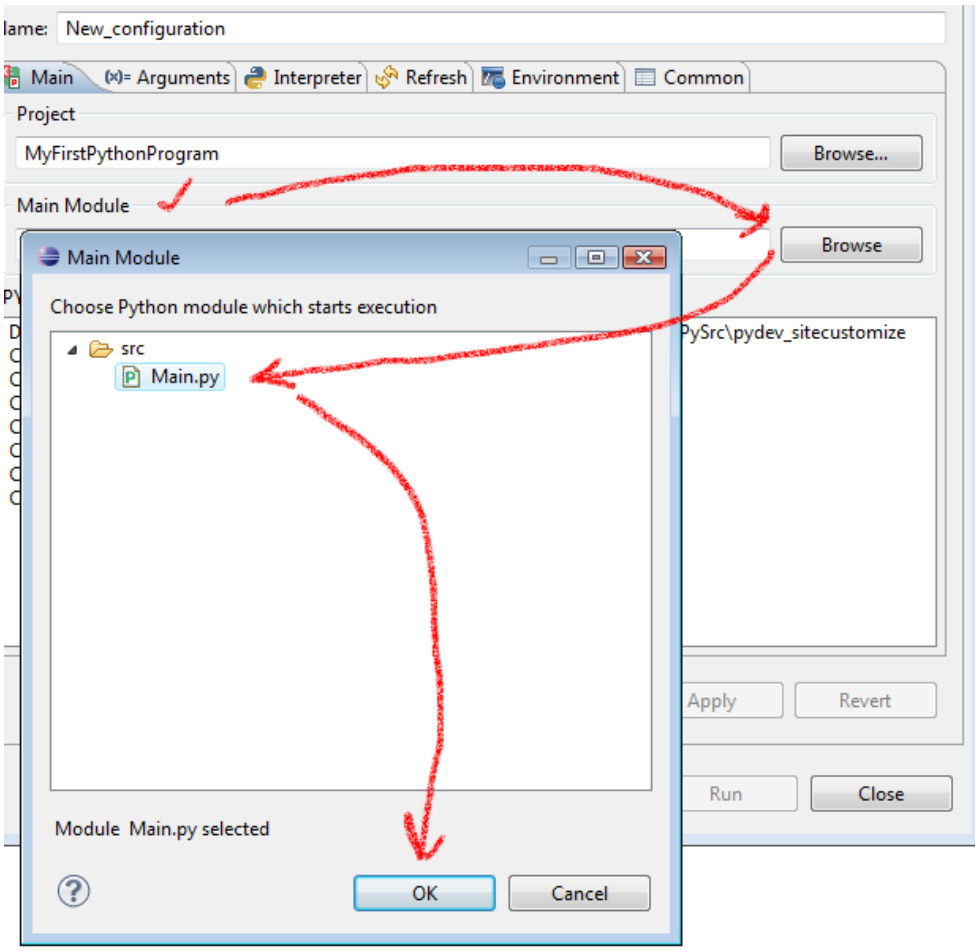
13. You'll see this screen next, double click on Python Run to create a New_configuration.



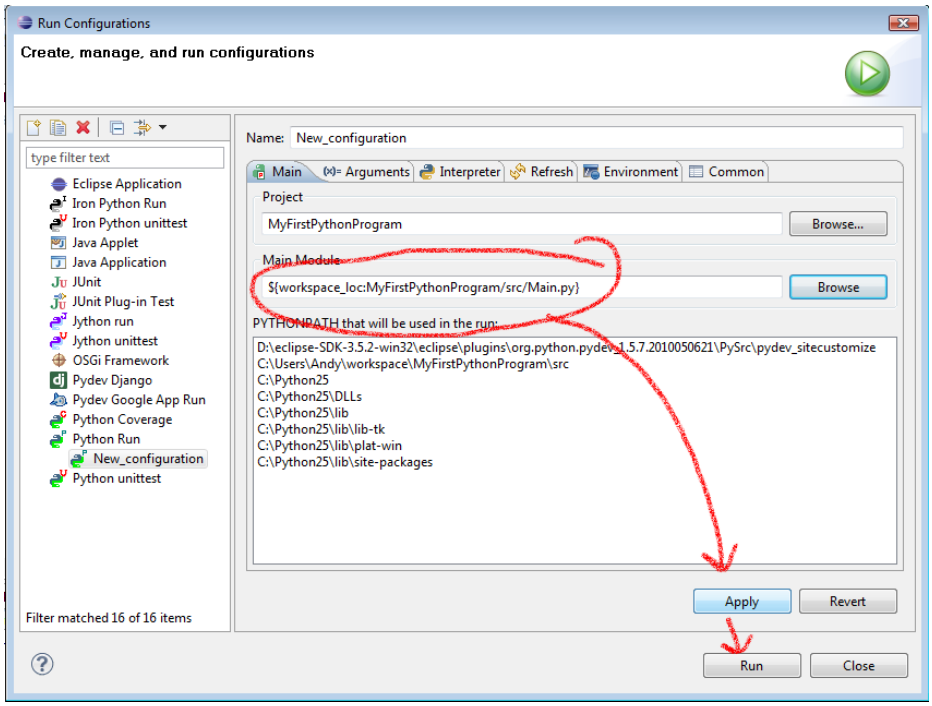
14. Next you need to browse to the main source file – which we already have.



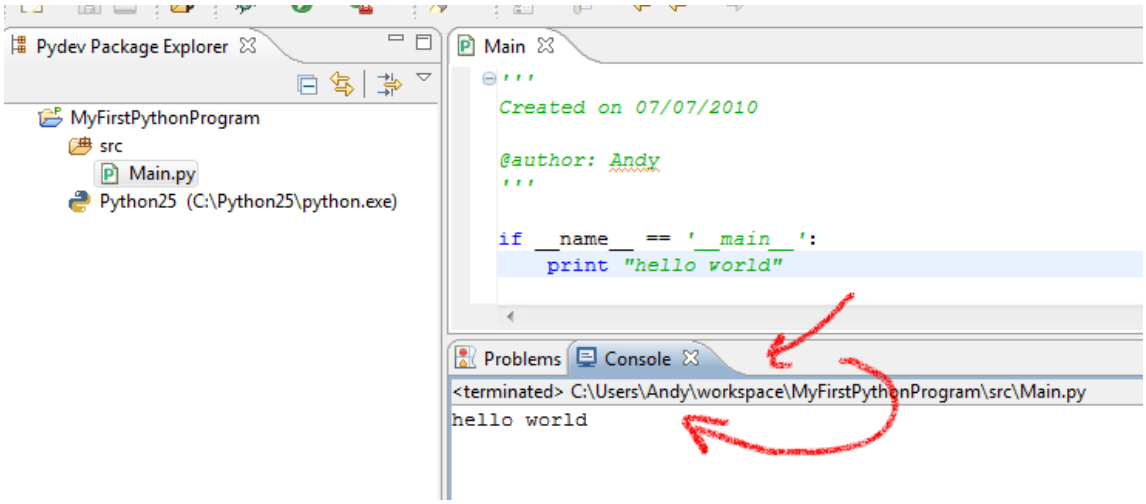
15. Now click on the Main Module browse button then expand src and click on Main.py. As this is your main module. Then click on Okay.



16. Next click on Apply, then Run.



17. In the console you will the follow “hello world” – well done! You have created your own hello world program in Python.



18. So what’s this hello world stuff anyway? Hello World is a standard practice for programmers. Basically it’s the icing on the cake when you start a new project, using new software and want to make sure you know how to compile your program and get the results on the screen. It could be elaborated to “Finally I got through all that technical guff and got something on screen, so HELLO WORLD! – Phew”.

Application # 1.

So we are doing this unit called Automate Processes. What does this mean exactly? Well computers are great at it. You usually apply it to something that would take a human a long time to do, so make a computer do it. You are probably thinking I'm going to get you to write some ugly processing thing like a staff list and mailing list – let's look at something more practical and relevant.

You may have an iPod / iPhone and it's got this great feature called Shuffle. But how does it work exactly? Well let's write our own using Python.

Design.

Let's identify the core functionality of the Shuffle process.

1) Takes a list of songs
2) Randomises the play order.
3) Keeps track of that play list so you can skip backward and forward
4) Never plays the same song twice in a shuffled playlist.

So how do we write this in Python? Let's do one step at time and learn a bit about programming along the way.

Python has a great system of storage called a `list`. A `list` is an array of items, and in this case we want to store the name of each song.

Coding.

In your Main.py function go to the line that says `print "hello world"` and replace it with

```
print "My iPod Shuffle Simulator"...
```

and after than type the following...

```
shuffleList()

showList()
```

So your code should look like this ... (Don't worry about the red X's – the IDE is just saying these function don't exist.)

```
if __name__ == '__main__':
    print "My iPod Shuffle simulator"
    shuffleList()
    showList()
```

Okay so let's write the function headers... remember we can use *pass* so that the function doesn't do anything? So that's what we'll do.

A function is defined first before it can be called. We use the keyword `def`, the name of the function, an empty parameter list and then ending with a *colon* `(:)`. Be sure to TAB in on the next line when adding the `pass` keyword. Should look like this...

```
*Main
'''
Created on 07/07/2010

@author: Andy
'''

def shuffleList():
    pass

def showList():
    pass

if __name__ == '__main__':
    print "My iPod Shuffle simulator"
    shuffleList()
    showList()
```

Outline
shuffleList
showList

You will also notice on the right hand side that Outline contains the new functions.

We need to create the variable `myItunesLibrary` at the top of the program so all of the code can access it. Let's add an empty list, then fill it with some songs using the append function available to lists – it's on the network so just copy and paste (playlist.txt)

```
Main
'''
Created on 07/07/2010

@author: Andy
'''

myItunesLibrary = []
myItunesLibrary.append("Love The Way You Lie - Eminem - Recovery")
myItunesLibrary.append("Dynamite - Taio Cruz - Rokstarr")
myItunesLibrary.append("California Gurls - Katy Perry - California Gurls(single)")
myItunesLibrary.append("I Like It - Enrique Inglesias - I Like It(single)")
myItunesLibrary.append("Cooler Than Me - Mike Posner - Cooler Than Me(single)")
myItunesLibrary.append("Ridin' Solo - Jason Derulo - Jason Derulo")
myItunesLibrary.append("Airplanes - B.o.B - B.o.B(single)")
myItunesLibrary.append("Jars of Hearts - Christini Perri - Jars of Hearts(single)")
myItunesLibrary.append("Billionaire - Travie McCoy - Billionaire(single)")
myItunesLibrary.append("OMG - Usher - Raymond V Raymond")
```

Above that let's add another list to be the actual playlist called myPlaylist.

```
myPlayList = []

myItunesLibrary = []
myItunesLibrary.append("Love The Way You Lie - Eminem - Recovery")
```

Okay so now for the hard part – the shuffling. Shuffling is done by randomly picking a song from a temporary library that originally is a copy of the iTunes library, which gets smaller and smaller as we remove each randomly selected song. We do this because we don't want the same song repeated in the play list.

So let's break it down into smaller steps or pseudocode.

1) Make a copy of the library.

2) Continue until the temp list is empty

3) Randomly pick a song from the temp list.

4) Add that song to the playlist.

5) Remove the picked song from the temp list

Now let's code it in.

Firstly we need to access a randomise function called choice. This is sitting in a module called random. So firstly we code the part where we import that functionality.

Go to the `shuffleList()` function and type this...

```
def shuffleList():
    '''
        Shuffles the list from a temp buffer
    '''
    # access external random functions
    from random import choice
```

Also notice the use of ''' comment ''' and the # hash symbol. These are two ways of commenting in Python. The convention is to the use the DOCSTRING type on commented at the beginning of a function to demonstrate it functionality. The second method of using # is inline commenting. Please comment as much as possible because of the abbreviated nature of Python it's often difficult to understand what you wrote weeks after writing some code and trying to work out what it's doing.

Next let's begin with step 1 from our functional design.

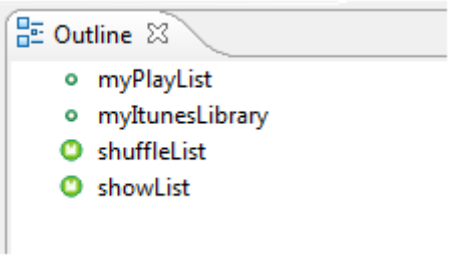
1) Make a copy of the library –

```
    # access external random functions
    from random import choice

    # create a temp list
    __myTempList = []
    __myTempList = myItunesLibrary
```

Notice we use two underscores. This is Python's way of creating a local variable.

If you check the Outline it won't show it. This is to do with scope. Scope determines where a variable is valid and useable.



In this case we only want to use it for the one function and once we pop out of the function the variable goes out of scope and the memory is returned to the system.

Next we need loop through the list.

2) Continue until the temp list is empty

We do this because of the way loops work. We need to set the condition of the loop first.

```
__myTempList = []
__myTempList = myItunesLibrary

# continue until temp list is empty
while (len(__myTempList) > 0):
```

Now that we have added `while (len(__myTempList) > 0):` line, everything after that needs to be indented to make sure it only runs during this condition.

Next step...

3) Randomly pick a song from the temp list.

4) Add that song to the playlist.

Why are we doing two steps in one? Well Python is highly optimised and allows to do stuff like this – this is why Python rocks. (notice we have indented the code so it only runs while the __myTempList still has songs to choose from.

```
while (len(__myTempList) > 0):

    # Randomly pick a song from the temp list and add it to the playlist
    __myChoice = choice(__myTempList)
    myPlayList.append(  myChoice)
```

Finally we want to avoid picking the song again so we remove it from the temp list.

5) Remove the picked song from the temp list.

This is easy because Python lists have a function just for this – remove.

```
# remove choice from temp list
__myTempList.remove(__myChoice)
```

Okay – `shuffleList()` function done.

Last thing is to display this list. So let's add to the `showList()` function.

```
def showList():
    '''
        Display the list on the screen
    '''
    # display the final play list in human readable form
    for __index, __item in enumerate(myPlayList):

        print __index + 1, __item
```

We use the `enumerate` function because we need to have the place of the item in the list as well as the item itself. Also as is true for most lists and array in computer programming languages, they are 0 based, but humans like lists to start with 1, so we add 1 to the `__index` variable.

Also notice we are using local variables because we aren't interested in them after the function pops out.

Okay run it – what do we see? Keep running it and every time you get a new playlist, that never repeats a song.

```
Problems  Console  (x)= Variables
<terminated> C:\Users\Andy\workspace\MyFirstPythonProgram\src\Main.py
My iPod Shuffle simulator
1 OMG - Usher - Raymond V Raymond
2 Dynamite - Taio Cruz - Rokstarr
3 California Gurls - Katy Perry - California Gurls(single)
4 Airplanes - B.o.B - B.o.B(single)
5 Ridin' Solo - Jason Derulo - Jason Derulo
6 I Like It - Enrique Inglesias - I Like It(single)
7 Love The Way You Lie - Eminem - Recovery
8 Cooler Than Me - Mike Posner - Cooler Than Me(single)
9 Billionaire - Travie McCoy - Billionaire(single)
10 Jars of Hearts - Christini Perri - Jars of Hearts(single)
```

Because the play list is now stored we could go back forth using `index` or go back by remember the previous song using a temp variable if we wanted to.

Here's one way you could do it.

```
# skip forward 2 then back one
print "********** NOW PLAYING ***************"
__index = 0
# get next song
__prev = __index
__index += 1
print myPlayList[__index]

# get next song
__prev = __index
__index += 1
print myPlayList[__index]

# get prev song
__index = __prev
print myPlayList[__index]
```

Here's the new output, notice the skip forward 2 then back one at the end.

```
Problems   Console   (x)= Variables
<terminated> C:\Users\Andy\workspace\MyFirstPythonProgram\src\Main.py
My iPod Shuffle simulator
1 I Like It - Enrique Inglesias - I Like It(single)
2 Dynamite - Taio Cruz - Rokstarr
3 Billionaire - Travie McCoy - Billionaire(single)
4 Love The Way You Lie - Eminem - Recovery
5 Airplanes - B.o.B - B.o.B(single)
6 Ridin' Solo - Jason Derulo - Jason Derulo
7 Cooler Than Me - Mike Posner - Cooler Than Me(single)
8 Jars of Hearts - Christini Perri - Jars of Hearts(single)
9 OMG - Usher - Raymond V Raymond
10 California Gurls - Katy Perry - California Gurls(single)
********** NOW PLAYING ***************
Dynamite - Taio Cruz - Rokstarr
Billionaire - Travie McCoy - Billionaire(single)
Dynamite - Taio Cruz - Rokstarr
```

All done? Let's revisit what we did to start with by reading the appendix.

APPENDIX

What is 'if __name__ == "__main__"' for?

The `if __name__ == "__main__": ...` trick exists in Python so that our Python files can act as either reusable modules, or as standalone programs. As a toy example, let's say that we have two files:

```
$ cat mymath.py
def square(x):
    return x * x

if __name__ == '__main__':
    print "test: square(42) ==", square(42)


$ cat mygame.py
import mymath

print "this is mygame."
print mymath.square(17)
```

In this example, we've written mymath.py to be both used as a utility module, as well as a standalone program. We can run mymath standalone by doing this:

```
$ python mymath.py
test: square(42) == 1764
```

But we can also use mymath.py as a module; let's see what happens when we run mygame.py:

```
$ python mygame.py
this is mygame.
289
```

Notice that here we don't see the 'test' line that mymath.py had near the bottom of its code. That's because, in this context, mymath is not the main program. That's what the `if __name__ == "__main__": ...` trick is used for.

Resources:

1) Python 2.5 (http://www.python.org/download/releases/2.5.5/)

2) PyGame for Python 2.5 (http://www.pygame.org/download.shtml)

3) Eclipse (http://www.eclipse.org/downloads/download.php?file=/eclipse/downloads/drops/R-3.5.2-201002111343/eclipse-SDK-3.5.2-win32.zip)

4) PyDev (http://sourceforge.net/projects/pydev/)

(just for my reference - don't download) Tutorials:

http://www.vogella.de/articles/Python/article.html#configuration

GLU-IT (http://www.downloadsofts.com/download/Graphic-Apps/Editors/Glue-Sprites-download-details.html)
AUDACITY (http://audacity.sourceforge.net/download/)

Tortoise SVN (http://downloads.sourceforge.net/tortoisesvn/TortoiseSVN-1.6.3.16613-win32-svn-1.6.3.msi?download)

Python 2.0 Quick Reference