University of Hull
Department of Computer Science

Wrestling with Python – Week 01
Playing with Python

Vsn. 1.0 Rob Miles 2013

Introduction

Welcome to our Python sessions.

Please follow the instructions carefully. You need to enter your programs exactly as they are written, otherwise they will do the wrong thing, or nothing at all.

In this session you will learn a bit about Python and how to get started with the language.

This indicates an activity you should perform in at this point in the text. You may be given precise instructions, or you may have to work something out for yourself.

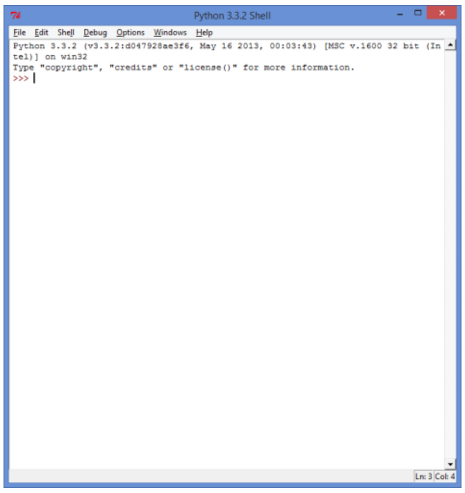
This indicates something that you may want to think about later.

This indicates a warning to be careful about this bit.

Data processing with Python

Data processing is a posh term for fiddling with stuff. We are going to start with some numbers and see how we can get Python to work on them for us. The first thing we need to do is start the Python shell.

Press the Start button and search for IDLE. Start the program running.



```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:03:43) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
```

Once you have got the Shell running you should see a display like the one above. You can type Python statements now and they will run instantly. While this is not how you would write an enormous program, it is a good way to get started and experiment with the language.

Working with Expressions

Type in the following and press enter:

```
2
```

Note that the value is echoed back to you. The Python shell just takes the statement and sends back a result. If you give it the value 2, you get 2 back.

Type in this expression and press enter:

```
2 + 2
```

This time the shell will work out the expression that you gave it, and produce a result.

Now try this:

```
5 * 10
```

This should leave you with the impression that perhaps the * operator does multiplication. Which it does.

How about this one?

```
11 / 3
```

The program will print out 3.6666666, which is close enough to the correct answer

There is something here you need to be aware of here (sorry about this). The two versions of Python (2 and 3) behave differently at this point.

Python 3 – does a floating point division and provides a floating point result (3.66666)

Python 2 – does an integer division and provides an integer result (3)

I feel bad about telling you this, but I'd feel worse about keeping it a secret. You can get a Python 2 program to behave itself by telling it to use the updated division routines:

```
from __future__ import division
```

The differences are not that many, but they are there, and you need to be mindful that if something doesn't seem to work that should do, it might be due to a language version difference. I will highlight these as I go through the text.

Strange integer division foibles aside, Python behaves as you might expect it, when you give it expressions.

Do brackets work?

```
(2+3) * 5
```

This should print 25, because the brackets cause the addition to be performed first.

Storing numeric values in variables

We can use Python as a calculator, but we know that computers are really all about storing and processing data. To do this the computer needs a way of actually doing the storing. Python lets you create variables to hold values.

Enter this code to create a variable:

```
age = 25
```

Notice anything different? If the Python shell sees an expression it will work out the answer and send it straight back. If it sees a Python statement it will perform it and then wait for the next command. The statement above puts the value 25 into a variable called `age`. It doesn't evaluate to a result, and so nothing is printed. You can think of a variable as a named box in memory that holds a particular value. We can change the contents of a box in memory by using an assignment statement.

```
age = 25
```

The statement has an expression on one side (the things we have been working out) and a variable on the other (a box we are going to put the result in). The equals character is the thing that identifies this as an assignment operation.

The very first time you use a variable the Python system will reserve space for that variable and give it the name that was set. Whenever the program uses that name in the future the Python system will go and get the contents of that variable.

Enter this statement:

```
age
```

We are back in expression country again, in that Python will just work out the value of the expression (25) and print it. We are pretty much back at the first thing we did, except that the number is now coming from a variable, rather than the value 2 that we entered at the very start.

Enter this statement:

```
Age * 2
```

This should evaluate twice the age and then display the result (because that is what Python does). But it doesn't work. (If it works you've typed it wrong). If you type in **exactly** what you see above you should get a message along the lines of:

```
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    Age * 2
NameError: name 'Age' is not defined
```

It will probably be in red, to indicate that the error is a *bad thing*.

What does this tell you about how Python manages variables?

To answer my own question, Python will create a variable the first time that it sees it, and will then go and get that value each time the program refers to it. But the names of the variables are **case sensitive**, in other words the variable `age` and the variable `Age` are different things. You could (but it might be confusing) create two variables with those names, what the red text above indicates is that if you use capital letters in the wrong place you will get problems.

We create variables every time we need to store something. We try to give them names that reflect what they are being used for. You can create as many variables as you need, and you can use them in expressions alongside literal values such as 2 or 3.

Working with Text

Now we can try something else:

Enter this statement:

```
Hello world
```

Python will not like this. Which is a bit of a shame if we want to put the phrase "Hello world" into our program. We can address this by putting the text inside *delimiters*:

Enter this statement:

```
'Hello world'
```

This doesn't produce an error, in fact Python just seems to echo the text.

What does this tell us about strings enclosed in quotes?

We saw that when we typed a value, or an expression, Python just worked out the answer and responded with it. When we type a string in quotes it seems that Python treats that as a value too.

We can test this theory. Enter the following:

```
'Hello ' + 'world'
```

If we are right, Python will add the two strings together and print the result. And it does.

This is our first brush with a thing called *overloading*. The + operator will do addition if you put it between two numbers, but if you put it between two strings it will concatenate them.

We can even create variables that hold strings:

Making a string variable is the same as making an integer one:

```
name = 'Rob'
```

We can create a string variable and print it out, just as you would expect. We can then use the string anywhere that it would be appropriate.

I've used single quote characters to mark the start and the end of the string. That's fine, and we have seen it work. Python will also tolerate other delimiters, for example double quote, ", which might be useful if the string you are entering contains single quote characters. I'll let you discover how you can create a string that contains both kinds of delimiter.

Reading text strings

At the moment we can write Python that will let us use the machine as a calculator with a bunch of memories. We can do things with numbers and also store and concatenate strings. Which is a start. Fortunately we can also read from the user:

Run this statement and see if you can work out what it does. Enter a sensible number if you happen to get asked any questions of that kind….

```
ageString = input('Enter your age : ')
```

Our Python system now contains two versions of age. One, that we entered earlier, has the name `age` and is stored as a number. The other, which we have just entered, is called `ageString` and contains a string.

Use Python to view the contents of the `age` and `ageString` values. See if you can spot the difference between the two.

When Python displays the result of a string evaluation it will put single quotes around it.

Converting from a string to a value

We need a way of converting from a string of text (that the user types in) into a value (that we can do sums with). Fortunately Python provides us with this.

The good news is that this works.

The int method will take a string and return the integer value that the string contains:

```
age = int(ageString)
```

This should set the age to the value that you typed in a moment ago. The int method is quite picky. If you try the following you will not get a happy ending:

```
age = int('kaboom')
```

If you enter this statement you will get another error, and quite right too.

Printing a Message

You can get a program to print an output by using the print statement

The print method will take something and print it. We will use this when we start to write programs.

```
print(ageString)
```

Making an Adder in Python (snake humour)

We now know enough to make a program that will read in two numbers, add them together and print out the result.

Working with Python code

At the moment we have been using the Python Shell part of IDLE to enter Python program statements which are obeyed instantly. This is a great way to experiment with the language, but we have discovered that if we want to run the program more than once we have to type the commands again. You might be getting tired of typing the same things time and time again. In this section we are going to find out how you can create and save your program files and also how you can use cut and paste to reuse your "experimental" program code.

What we want to do is make a program that reads in two numbers and then prints out their sum.

Write out on paper a sequence of statements that get the numbers from the user and then displays the result.

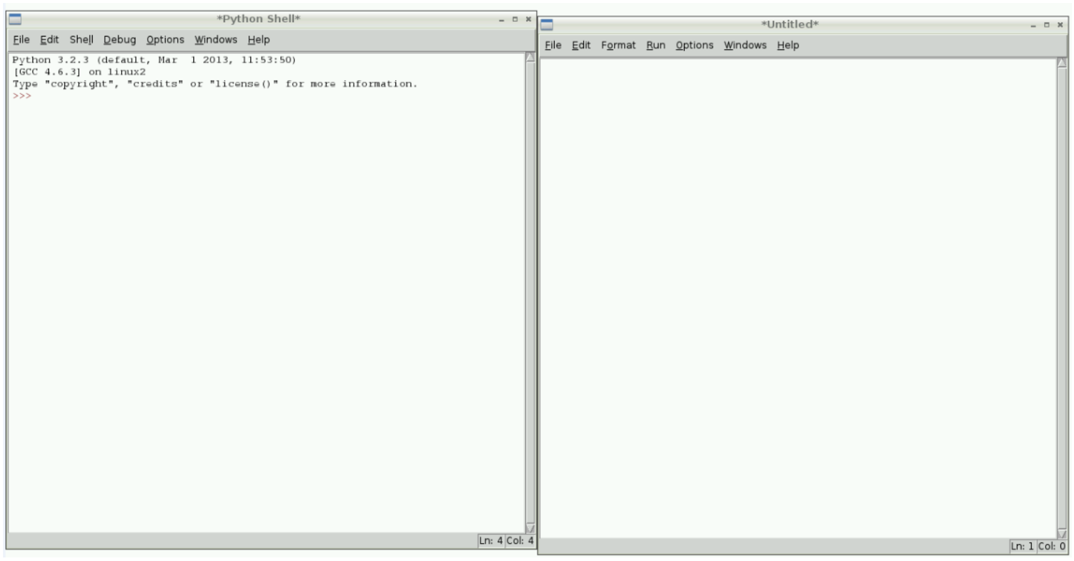
Writing Python Programs

At the moment we have been using the Python Shell part of IDLE to enter Python program commands. This is a great way to experiment with the language, but we have discovered that if we want to run the program more than once we have to type the commands again. We can do clever things with copy and paste (see above) but what we really want to do is store our programs and load them again. It turns out that we can do this in IDLE quite easily.

From within IDLE we can open up a new window where we can work on Python programs. When we think the program might work we can then ask IDLE to run the program so that we can see whether or not the program works properly. This is exactly what professional developers do when they write programs.

To create a Python program the first thing you need to do is open a new Window on the desktop. Click on the `File` tab on the top line of the IDLE window and select `New Window` from the menu that appears. Alternatively, you can press `CTRL+N`.

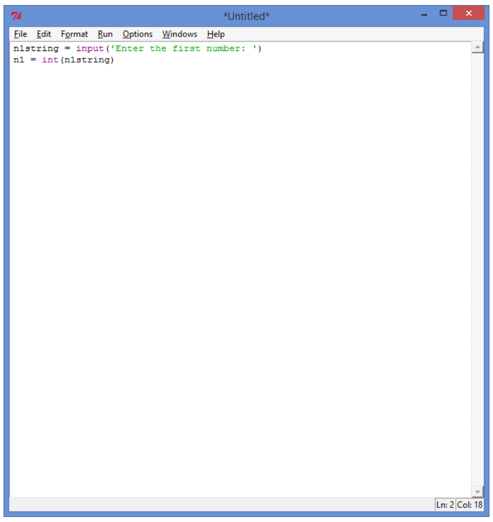
This should cause a second window to appear on the screen, which will have the helpful title `*Untitled*`.


Python Shell
File Edit Shell Debug Options Windows Help
Ln: 4 Col: 4
Untitled
File Edit Format Run Options Windows Help
Ln: 1 Col: 0

```
Python 3.2.3 (default, Mar  1 2013, 11:53:50)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>>
```

Above shows the two windows at the start of a Python development session. Unlike the Python shell window, the program lacks the `>>>` prompt where you type commands. You can type lines of Python into the new window and they are not executed. They are held as a program.

You can treat this window as any other text editor that you have used before. Think of it as a word processor for programs. You can type in the statements you have written and then run them.



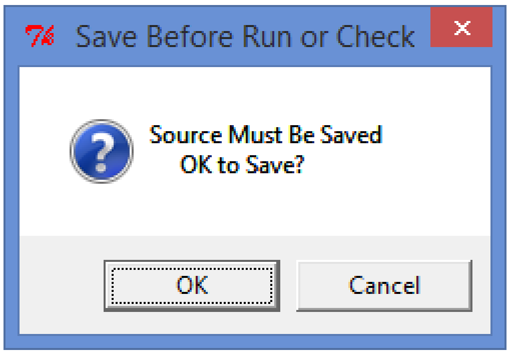
```
n1string = input('Enter the first number: ')
n1 = int(n1string)
```

Above shows the program as it appears after being typed in. Note that the editor “colours in” some of the words that are part of the Python language. I used the `Save` command on the `File` menu to save the program text in a file.

You can now create Python programs and save them to work on next time. You can also email your Python code to other people to have a look at and work with and you can run programs that they send you. If you use the `Open` command from the `File` menu in IDLE3 you can open a file that you have already created and work on it later.

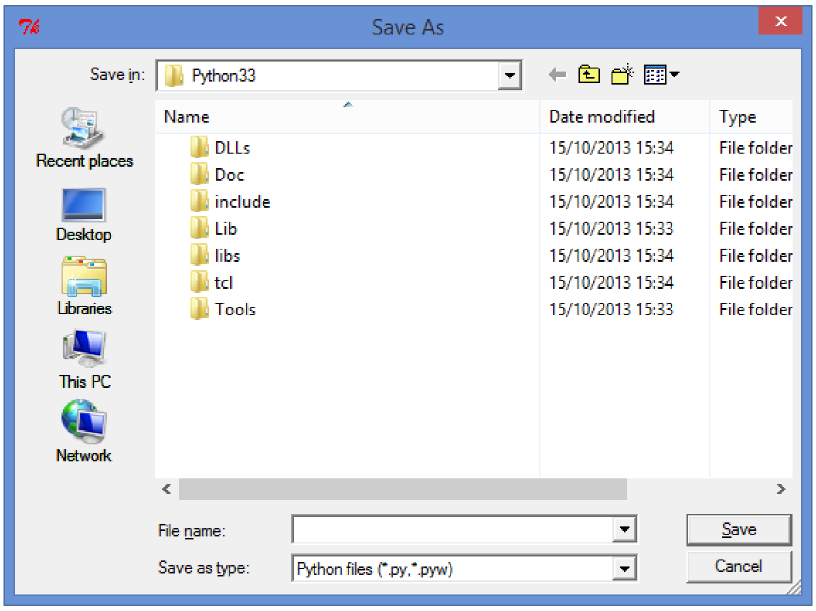
Running Python Programs

Once you have entered the program code you can run it by opening the `Run` menu from the top line and selecting `Run Module` from the menu. Alternatively you can press the `F5` function key on your keyboard. The first time you run a new program the system will insist that you save the program in a file somewhere.


Save Before Run or Check

Source Must Be Saved
OK to Save?

This is the save request. You will see this each time you try to run the program. You **must** save the file in order to be able to run it. Choose a name for the file and then save the program in that file. Make sure that the name you use has the characters `.py` at the end so that the operating system knows the file contains a Python program.


Save As

Save in: Python33

Name	Date modified	Type
DLLs	15/10/2013 15:34	File folder
Doc	15/10/2013 15:34	File folder
include	15/10/2013 15:34	File folder
Lib	15/10/2013 15:33	File folder
libs	15/10/2013 15:34	File folder
tcl	15/10/2013 15:34	File folder
Tools	15/10/2013 15:33	File folder

File name:
Save as type: Python files (*.py,*.pyw)

The save commands are very similar to the ones that you have seen on other computer systems. You can move around the file store and pick somewhere to put the files.

When you save a program as a file, make sure that you add the *language extension* `.py` (for Python) onto the end of the filename. A language extension is how the operating system (Windows or Mac OS) knows what kind of data the file contains. If the system can't tell that the file is a Python program it might not work correctly when you try to run it.

Program syntax errors

When you start the program running the IDLE environment checks your program to make sure it is correct and then runs it in the IDLE Python shell. If you have made any mistakes you will see these described in the shell and you will have to look in your edit window to see where the mistake is.

If the Python runtime system doesn't like your program it will display the error box above. This means that you must look through your program to find out where you went wrong. There are lots of reasons why Python might not like your program.

Rob Miles

October 2013