1: Introducing Python

Gareth McCaughan

Revision 239, July 23, 2010

Credits

First steps

You should have a window on your screen with "Python" somewhere in its title, displaying a line that looks something like this:

```
>>>
```

If you don't then check Sheet R (*Running Python*) for how to get one.

Whenever you see stuff in `typewriter type`*, as above, it represents either something you should say to the computer, or something the computer might say to you, or both. Where both are present, things for you to type are underlined. Anything not in typewriter type is just for you to read; you shouldn't type it in, or expect the computer to print it out.*

That >>> is called a "prompt", which means it's something the computer displays to tell you it's ready for your instructions. You can type things into that window, and the computer will obey them straight away. Later on we'll want to collect up a bunch of stuff for the computer to do, but for now we're using the prompt to try things out.

Unfortunately, the computer doesn't understand English. If you type in

```
>>> Tell me the sum of twelve and thirteen.
```

it won't understand at all. Instead, because the computer is very stupid, you have to talk to it in a special language, designed to be easy for the computer to understand. In fact, there are lots of languages designed for computers to understand; the one we're going to look at is called "Python". One of the good things about Python is that it's pretty easy for humans to understand, too.

Here's how you ask the computer to tell you the sum of twelve and thirteen. Try it yourself. (You don't need to type in the >>>, but you do need to hit the key marked `Enter` after typing the line.)

```
>>> 12+13
```

Here are some more examples, with the computer's answers shown too.

```
>>> 1+2+3+4
10
>>> 1+2*3-4                                       Use * for multiplication, not x.
3                                                 If you expected 5, think again!
>>> 200*300
60000
>>> 12/4                                          Use / for division.
3
```

Now, here's a bit of a surprise.

```
>>> 7/3
2
```

You might have expected it to say `2.3333333` or $2\frac{1}{3}$, but in fact the remainder just gets thrown away. There are ways of getting a more accurate answer; we'll find out about them later.

Try experimenting some more with using Python as a calculator.

You can use parentheses to group operations in Python just as you do in mathematics:

```
>>> (1+2)*(3+4)
21
```

Here, Python has calculated `(1+2)` and `(3+4)` (getting 3 and 7), and then multiplied the results together.

Don't be afraid to experiment. Whenever you learn something new that you can do with Python, try making slight changes (or bigger changes!) and play around until you're confident that you understand just what's going on. Don't limit yourself to what's actually printed on the sheets!

Incidentally, if you're still confused about the fact that `1+2*3-4` gives 3 and not 5, the reason is that "multiplication happens before addition". Your maths teacher at school probably calls this BODMAS or something similar. If you're still confused, don't worry. It isn't likely to matter.

Different types of object

So far, all the things you've worked with have been numbers. But Python can handle plenty of things besides numbers. For instance, try the following:

```
>>> 'hello, ' + 'world'
'hello, world'
```

Things between quotation marks are called "strings". As you might guess from the lines above, you can apply + to strings as well as to numbers. It "concatenates" strings; that is, puts one immediately after the other. A little more surprising:

```
>>> 3 * 'hello'
```

You'll notice that this time I haven't told you what the machine says. That's because you're supposed to try it for yourself. You won't learn anything if you don't try the examples. Write down what the machine said when you asked it for `3*'hello'` in the space below, so that you remember.

You can surround strings in either single quotes or double quotes; Python doesn't mind.

```
>>> 'ham' + "mock"
'hammock'
```

Why would you care about that? Well, suppose you wanted a string containing the text `I'm sorry` ?

Python also has “lists”:

```
>>> [1,2,3]
[1, 2, 3]
>>> [1,2,3] + [7,8]
```

Again, I haven’t told you what Python says to that last thing. Write it down in the space below.

Giving names to things

Suppose you know that you’re going to need to do a lot of calculations involving the number 123456. (Maybe it’s your annual salary in pounds, or something.) You could just type the number in every time:

```
>>> 123456*3
370368
>>> 123456/6
20576
>>> 123456-1000
122456
```

This might get very boring after a while. And if anyone else wanted to read what you were doing, they might be confused by the mysterious number 123456 and wonder why it appeared so often.

We can solve either of these problems by giving the number a name. To save typing, give it a short name, like `n` (short for “number”, maybe). To make it more obvious what it means, give it a longer name, like `salary`. Here’s how we do that.

```
>>> salary=123456
>>> salary*4
493824
>>> salary/12
10288
>>> salary
123456
```

The idea is that, after you’ve said `salary=123456`, you can always type `salary` instead of `123456`.

What we’ve called “names”, most people call “variables”. You’ll find out later why they’re called that. For now, “names” is fine.

You can name things other than numbers, too. For instance:

```
>>> MyName = 'Gareth'
>>> 'Hello, ' + MyName + '!'
'Hello, Gareth!'
```

Notice that when you give something a name like this, Python doesn’t print anything in response. We’ll see why in a moment.

Doing something over and over again

So far, we've done very little that your pocket calculator couldn't do equally well. Here's something your calculator probably isn't so good at. The extra spaces on the second line are important, by the way! (This is explained in more detail in Sheet 2.)

```
>>> for x in 1,2,3,4,5:
...   print x,x*x
...
```

The prompt changes, to tell you Python is expecting more. Just press `Enter`.

Can you guess what this will do? ... If you guessed that it prints out the numbers from 1 to 5 along with their squares, well done. Notice that Python conveniently puts a space between the two things you've asked it to print.

The `print` command is used when you want to make the computer display things. The reason we haven't needed it before is that when you type in something at the prompt, and that thing has an answer Python can work out, it automatically displays that answer. Things with answers are called "expressions", for some reason. But Python doesn't print out every *value it computes; only the values of expressions you type in at the >>> prompt. When you write bunches of stuff to do together, like you did just now and will be doing a lot of later, you need to tell it to print stuff out using the `print` command.*

Graphics

You can use Python for drawing pictures. Try this. The first two lines will probably seem rather weird; we'll explain them later.

```
>>> from livewires import *
>>> begin_graphics()
>>> set_colour(Colour.red)
>>> move(100,100)
>>> draw(200,100)
>>> set_colour(Colour.blue)
>>> draw(100,200)
>>> end_graphics()
```

After you've typed that last command, you may find that Python sits doing nothing until you close its graphics window. If so, close its graphics window! Normally the last line will make Python do so itself.

It's all gone horribly wrong

At some point when you're using Python, something like *this* is going to happen.

```
>>> 3+'aardvark'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

This looks pretty scary. Don't be scared. It's just Python's way of saying you did something it didn't understand. You can often ignore everything except the last line (though when it happens to your program the other lines can help find the root cause). To understand the last line, have a look at Sheet E (*Errors*), which has a list of common complaints Python might make at you and explanations of what they might mean.

What next?

We have two kinds of worksheets.

1. *Activity sheets*, each of which takes you through writing a program to do something that might be interesting. These sheets are numbered: Sheet 2, Sheet 3 and so on. (The sheet you've almost finished reading now is Sheet 1.)
2. *Information sheets*, each of which tells you something useful about Python. These sheets are lettered: Sheet P, Sheet L, or whatever. Usually the letters have something to do with what the sheets are about, but that hasn't always been possible.

You should probably read Sheet R (*Running Python*) now, if you haven't read it already.

After that, we recommend that you work through the activity sheets in order. Each one will point you to a few information sheets that you'll want to have handy when working through it.