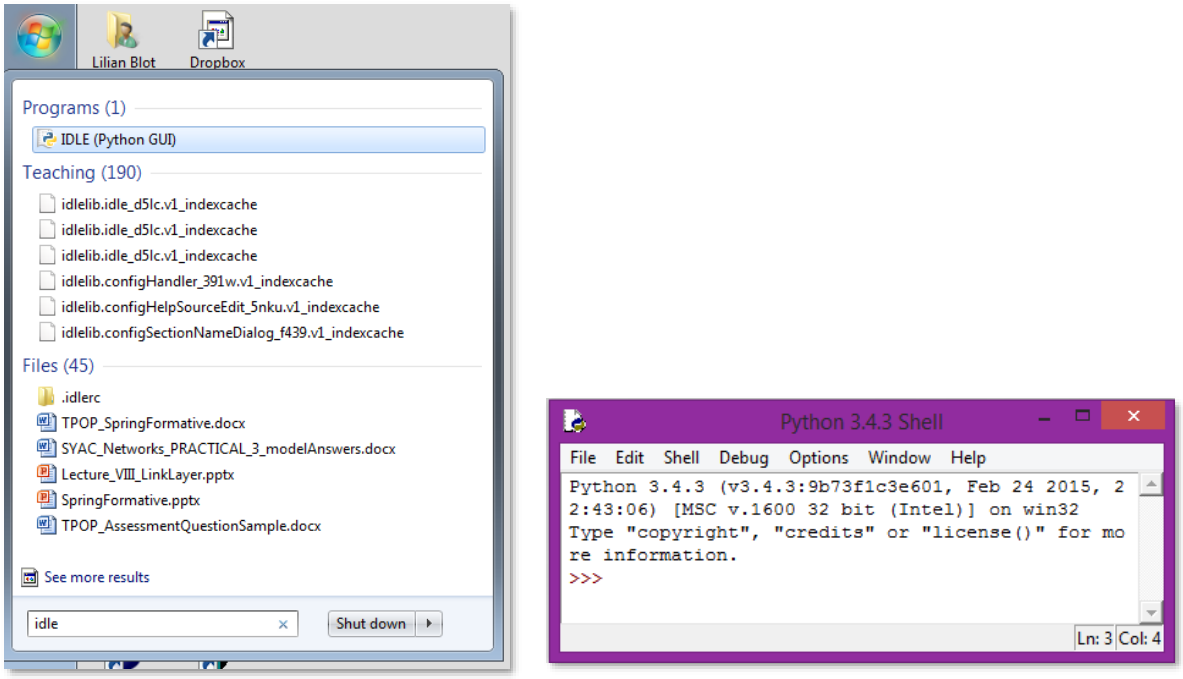
SOFTWARE 1 PRACTICAL

ELEMENTARY PROGRAMMING

Week 2 – Practical 01a

Getting started

The first thing we need to do is opening a Python shell (make sure it is python 3.4 or later, not 2.7). Go to the Windows start button and type idle (see Figure 1a). A new window (shell) will open as shown in Figure 1b (Note, program version may vary). We will use this shell to type our code for the time being. Toward the end of the lecture we will be saving our code in a file.



(a) **(b)**

Figure 1

First lines of code

A Python shell could be considered as a big calculator where we can do calculation using some mathematical expression. The basic element of an expression is a value. In Figure 2, the first line of code represents a value, which is a number and has value 1. We can also use known operators such as addition, multiplication, etc. Such combination of operators and operand is known as an expression. Once you have typed an expression and pressed the return key, the expression is evaluated and the result is printed in blue on the line below.

```
>>> 1
1
>>> 1 + 2
3
>>> 3 * 9
27
>>> 2 * 3 + 4
10
>>> 2 * (3 + 4)
14
>>> -1 - 3
-4
>>> 1 + (-2)
-1
>>> 1 + -2
-1
>>> -2 * -4
8
```

Figure 2

As you know mathematical operators have rules of precedence, it is the same here. The use of parenthesis can be used to counteract the rules of precedence between operators as shown in Figure 2, lines of code 4 and 5.

```
>>> 4/2
2.0
>>>
>>> 4//2
2
>>> 3//2
1
>>>
```

Figure 3

When considering the division operator `//` something strange happen. `3//2` gives `1`. This is called the "integer division" operator. What are the results of `-3/2`, `-3/-2` and `-3.0/-2`?

Even old calculators (e.g. when I was still at school) had a way to memorised values that can be reused later. We have the same principle in Python. Such objects are called variables. In Figure 4 we declare a variable `x` and assign the value `2` to it. The = sign is called the assignment operator, `x` is the name of the variable, and `2` is the value assigned to `x`. Every time we use `x` it is the same (almost) as using the value `2`. We can use `x` in any expressions, we can have more than one variable (here `x` and `y`); we can also use several variables in the same expression as shown in Figure 4. The sixth line of code shows how we can change the value in a variable. From now on, `x` has the value `5` in memory and do not remember having the value `2`.

```
>>> x = 2
>>> x
2
>>> x + 3
5
>>> 2 * x
4
>>> x + x
4
>>> x = 5
>>> x
5
>>> x + 3
8
>>> y = 4
>>> x + y
9
```

Figure 4

Using x and y as name for variables is allowed, however it is not very meaningful. In Figure 5 the variables name are very explicit and therefore it is not difficult to understand what the aim of the program is. You are strongly encouraged to do the same thing. You MUST read the Style Guide for Python Code available at: http://www.python.org/dev/peps/pep-0008/

```
>>> number_cakes = 4
>>> cake_price = 2.50
>>> bill = cake_price * number_cakes
>>> bill
10.0
>>>
```

Figure 5

My first error (and probably not the last one)

What happen if I misspell the name of a variable? I will get an error as we are asking the interpreter to use something it does not know. The interpreter will promptly show you its discontentment in red as shown in Figure 6 .

```
>>> number_cakes = 4
>>> cake_price = 2.50
>>> bill = cake_price * number_cake

Traceback (most recent call last):
  File "<pyshell#55>", line 1, in <module>
    bill = cake_price * number_cake
NameError: name 'number_cake' is not defined
>>>
```

Figure 6

It is very important for you to recognise the type of error and try to fix it. Python interpreter try its best to help you in this task. Here it points to the line containing an error <bill = ….> followed by the type of error <NameError:…>.

In our case it tells us that number_cake is not defined. Indeed we misspelled the name of the variable forgetting the 's' at the end. When you encounter an error that you are not familiar with I encourage you to search an explanation online. If it proves unsuccessful, ask myself or one of the PTAs.

Another data type

So far we have seen two types of numbers, does Python have any other type of values? The answer to that question is yes, Python does have quite a few more types. The next one we are going to see represents a series of character to form words or sentences for example. Values of this type are called string. Figure 7 shows a string representing a sentence. Python has two ways of representing a string, using single quotes or double quotes. In the same way as for numbers, string can be assigned to variables (Figure 8) and we can use the command `print` to print the string

```
>>> 'this is a succession of characters.'
'this is a succession of characters.'
>>> "this is a succession of characters."
'this is a succession of characters.'
>>> type('this is a succession of characters.')
<class 'str'>
>>>
```

Figure 7

value on the console. For example the statement `print words` display the content of the variable words on the console.

```
>>> print('this is a succession of characters.')
this is a succession of characters.
>>> words = 'this is a succession of characters.'
>>> words
'this is a succession of characters.'
>>> print(words)
this is a succession of characters.
>>> sentence = 'This is another sentence.'
>>> print(words + sentence)
this is a succession of characters.This is another sentence.
>>> print(words, sentence)
this is a succession of characters. This is another sentence.
>>>
```

Figure 8

To concatenate two strings, e.g. build one string from two substrings, we can use the addition operator +. For example:

```
longString = words+sentence
```

We can also print several strings in the same statement by separating them with a comma. Note that by doing so a space will be added between the two strings.

More errors

```
>>> another_sentence = "Black Knight: All right, we'll call it a draw.'
SyntaxError: EOL while scanning string literal
>>>
>>> another_sentence = 'Black Knight: All right, we'll call it a draw.'
SyntaxError: invalid syntax
>>> another_sentence = "Black Knight: All right, we'll call it a draw."
>>>
```

Figure 9

You should be careful when declaring strings that you are using the same delimiter at the start and end of the string (first error in Figure 9). What about the second error? How can I do if I want to use single quote or double quotes in a string? Look on the web to find the answer.

First attempt at a small program

Let's try to create a small program that compute a bill for a cake shop. We would like to get the number of cakes, the price of a single cake and then print the total price of the bill. Figure 10 shows such a small program where the number of cake is four.

```
>>> number_cakes = 4
>>> cake_price = 2.50
>>> bill = cake_price * number_cakes
>>> print('The price of', number_cakes, 'cake(s) is', bill, 'pounds.')
The price of 4 cake(s) is 10.0 pounds.
>>>
```

Figure 10

Now if I want to compute a bill for seven cakes I need to rewrite the code as shown in Figure 11. As you can see the only change between the two scripts is the assignment of the value 7 to the variable `number_cakes`.

```
>>> number_cakes = 7
>>> cake_price = 2.50
>>> bill = cake_price * number_cakes
>>> print('The price of', number_cakes, 'cake(s) is', bill, 'pounds.')
The price of 7 cake(s) is 17.5 pounds.
>>>
```

Figure 11

And if I want to do that again I will have to retype everything one more time. This is clearly not efficient. What can I do to change that? This is the subject of the next section.