Programming in Python for Business Analytics (BMAN73701)¶

Lab Session: Week 1 - Introduction¶

Get familiar with your environment¶

It's important to first understand your environment. Although you are free to use whatever tools you want, we expect at first that you will be using Spyder. Some aspects will be explained below so that you can become familiar, so then we can start writing some code!

We are using Python 3, so please check that it is the Python version that Spyder will be using. You can also check this in the console, as it will say what Python version is being used, as shown in the screenshot.

Working Directory¶

The working directory is the directory, or folder, that you are currently in. This is where you will be saving files, and later when we get on to file paths and loading files, it will be important to understand this.

For now, just have the working directory somewhere sensible (so whatever folder you have for this course, and if you have a folder for each week etc.). It can default to a temporary folder, where you run the risk of losing any work you have saved, so this is important.

Script Editor vs. Interpreter/Console¶

In the image above you can see two areas labelled, the script editor and interpreter/console. You can write Python code in both, but they have different purposes that are important to know.

The script editor allows you to write Python files (.py files), which is the main way of writing code. Much like a Microsoft Word document, it allows you to write text, save it, edit it etc. without losing content. When you run (or execute) this file, all of the code in the script will run. We recommend that you have a single .py file for each question, though this is up to you.

The interpreter, or console, is more for testing very short pieces of code, and should not be used for whole lab questions. It is best used for testing small things, ideally code that is just a single line. If the console is restarted, or if Spyder is closed, anything in the console will be lost.

You can see that there is a tab for *Python console* and another for *IPython console*. The latter has more features than the former, but this is not important. You just need to know that they are different, and as they are different consoles what you type in one is not available to the other.

Object Inspector¶

The *Variable explorer* tab can be used to see what variables are currently stored in memory and what values they have. This can be useful when debugging (i.e. trying to figure out what the code is doing, primarily what it is doing wrong!). The *File explorer* can be used just like your usual folder window, so you can navigate through your files and see what files are where. The *Help* button can be useful if you bring up any help documentation (which is available for some functions and can be useful/quicker than using Google to find the documentation).

If you type `help(print)` into the console, then you will get the help documentation on the print function. Try this now and see what it says!

Your first pieces of code - Hello World!¶

As we've shown, there are two ways of writing code - into a script, or into the console. For this first question, we're going to combine the two.

In the console, typing `"hello world!"` will print `'hello world!'`. The console prints by default, we don't need to tell it to. If you tried typing `"hello world!"` into a script and running it, what would happen?

Hmmm, nothing printed. Let's trying using the `print` function to actually print this. In your script, try `print("hello world!")`, then run again.

Great! Now, you can see that if you want anything printed from a script, you have to *explicitly* use the print function.

Exercise 1:¶

a) Print `hello hello hello` using 3 different methods

For this question, using the console to try different methods, and then typing the ones that work into a script is a great idea. Once you have 3 working methods in the script, when running it you should get:

```
hello hello hello hello hello hello hello hello hello
```

In [4]:

```
print("hello hello hello")
print("hello "*3)
print("hello","hello","hello")
```

```
hello hello hello
hello hello hello
hello hello hello
```

b) Print `hello world!` (i.e. `hello` is on one line, and `world!` is on the next)

In [5]:

```
print("hello\nworld!")
```

```
hello
world!
```

Python as a calculator¶

You can use Python as a calculator, using common math operators such as addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), modulo (`%`), and power (`**`). These are built-in to the language, which means we can use them straight away without needing to import any additional functionality.

Try typing into the console some different maths, e.g.:

```
3+4       # = 7
2**4      # = 6
5%2       # = 1
36**0.5   # = 6.0
```

Raising to the power of 0.5 is one way to get the square root, but there is another. We can import the `math` module, which gives us some additional functions that we can use. To import the math module, you need to type `import math`. Remember, if you do this in the console, you will need to type this every time after you close the console. If you type this into a script, **by convention all import statements are at the top of the .py file** (i.e. from line 1).

If we want to use a function from the math module, we do this by typing `math.function_name`, where `function_name` is the name of the function we want to use.

For example, typing `math.sqrt(36)` will give us the square root of 36. Try the following statements:

```
math.log(32,2) # = 5.0
math.cos(0)    # = 1.0
math.pow(2,4)  # = 16.0
```

N.B.: You do not need to memorise all the functions in the math module or anywhere else, you just need to know that they exist and how to use them.

Now, we want to perform two calculations, using the result of the first. We need to assign the result to a variable, i.e.:

```
a = 3*5
b = a / 10
print(b) # 1.5
```

Here, we call `=` the *assignment operator*, we are assigning the result of `3*5` to a **variable** `a`. This allows us to use `a` again.

Of course, you could just type `b = (3*5)/10`, but for more complex code this can become difficult to read and problematic.

Exercise 2:¶

a) Calculate `2-9*11`
b) Calculate `(2-9)*11` (BIDMAS/BODMAS etc. apply here too!)
c) Calculate the positive root of the following equation:

$34x^2 + 68x - 510$ $34x^2 + 68x - 510$

Recall that generally, $ax^2 + bx + c$ $ax^2 + bx + c$ *and the positive root*

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \quad x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$$

If you are stuck with c), remember that you just have 3 variables, and a general equation.

In [15]:

```
# week_1_ex2.py
import math

# 2a)
print(2-9*11)

# 2b)
print((2-9)*11)

# 2c)
a = 34
b = 68
c = -510
```

```
print((-b+math.sqrt(b**2-4*a*c))/(2*a))
```

```
-97
-77
3.0
```

Capture user input, use maths, print results¶

As we saw in the lecture, the `input()` function prompts the user for input, and returns whatever they type as a string. By assigning what is returned to a variable we can store whatever the user types.

Exercise 3¶

a) Prompt the user for their name, city, and country of residence and store these values
b) Print out a line greeting the user, using the information they have provided e.g. `"Nice to meet you Manuel from Cadiz, Spain."`
c) Prompt the user for their age
d) Print out a line complimenting the user by saying something like `"Wow! You look like you could be xxx!"`, where xxx is 80% of their real age, and you must make sure that it is an integer. For example, if their real age is 20, then xxx would be 16.

In [1]:

```
# week1_ex3.py

# 3a)
name = input("What is your name?: ")
city = input("What city do you live in?: ")
country = input("In which country is that city?: ")

# 3b)
# There are many ways of doing this
print("Hello there! Nice to meet you "+name+" from "+city+",
"+country)

# 3c)
age = input("Pardon my rudeness, but how old are you?: ")
```

```
# 3d)
print("Wow! You look like you could be", int(int(age)*0.8),
"!!")

What is your name?: Cameron
What city do you live in?: Manchester
In which country is that city?: UK
Hello there! Nice to meet you Cameron from Manchester, UK
Pardon my rudeness, but how old are you?: 26
Wow! You look like you could be 20 !!
```

In-place Operators and Errors¶

In-place operators are combinations of mathematical operators (`+`, `-` etc.) and the assignment operator (`=`). The mathematical operator is first, and then the assignment operator is second. See the example below:

```
# These two statements are equivalent
x += 4
x = x+4
```

As we said, for in-place operators the order is mathematical operator, and then assignment operator. However, what if we accidentally typed `x=+4` instead? Python does not give you an error, but it is not what you intended (you don't need to know but we call this a logic error). The same is true here for `x=-4`.

However, so we can see what an error looks like, this is what happens if you try to divide in-place but get the wrong order:

Here we've been told it is a *SyntaxError*, and shown specifically where this is occuring. As a reminder, syntax is the set of rules that define the language. In Python, `a/=b` is valid in the language, and `a=/b` is not. Try some different in-place operators to get a feel for how it works, and if you get an error try to understand why that is.

Exercise 4¶

The piece of code below is very, very wrong in lots of ways. There are many errors, and of several different types. You may be able to see some of the errors instantly, and fix them. Others you may not realise until you run the code. Do not be afraid to make big changes to the code - the only goal is the desired output shown below.

Python provides very informative and useful errors, so please read them and figure out where the problem is on your own first. **Being able to read and understand errors is key!** Below you will find the incorrect code, and the output that it should produce when fixed (and you input the numbers shown).

In [ ]:

```
# This code is incorrect - you need to fix it so that it
provides the desired output!
divide_by = 2
subtract_by = 3
multiply_by

var = input('Please provide an integer between 1 and 10: ')
var1 -= subtractby
print("Subtracting your value of var by '+subtract_by+' will
give "+ var1)

var2 = var1 / divide_by
print("Dividing var1 by "+divide_by+" will give "+var1)

2var = input("Please provide a real number between 0 and 1: ")
print("The ceiling of 2var is "+math.ceiling(2var))

input('Please provide an integer between 1 and 10: ')
input('Please provide an integer between 1 and 10: ')
```

The desired output is:

```
Please provide an integer between 1 and 10: 1
Subtracting your value of 1 by 3 will give -2
Dividing -2 by 2 will give -1.0
Please provide a real number between 0 and 1: 0.5
The ceiling of 0.5 is 1
Please provide an integer between 1 and 10: 2
Please provide an integer between 1 and 10: 3
Multiplying 2 by 3 will give 222
```

```
Multiplying 2 by 3 will give 6
Multiplying 2.0 by 3 will give 6.0
```

In [5]:

```
import math

divide_by = 2
subtract_by = 3

var = input('Please provide an integer between 1 and 10: ')
var1 = int(var) - subtract_by
print("Subtracting your value of", var, "by", subtract_by,
"will give", var1)
# Alternative way to print
# print("Subtracting your value of "+ str(var) +" by "+
str(subtract_by) +" will give "+ str(var1))

var2 = var1 / divide_by
print("Dividing", var1, "by", divide_by, "will give", var2)

var3 = input("Please provide a real number between 0 and 1: ")
print("The ceiling of", var3, "is", math.ceil(float(var3)))

var4 = input('Please provide an integer between 1 and 10: ')
var5 = input('Please provide an integer between 1 and 10: ')

print("Multiplying", var4, "by", var5, "will give",
var4*int(var5))
print("Multiplying", var4, "by", var5, "will give",
int(var4)*int(var5))
print("Multiplying", float(var4), "by", var5, "will give",
float(var4)*int(var5))

Please provide an integer between 1 and 10: 1
Subtracting your value of 1 by 3 will give -2
Dividing -2 by 2 will give -1.0
Please provide a real number between 0 and 1: 0.5
The ceiling of 0.5 is 1
Please provide an integer between 1 and 10: 2
Please provide an integer between 1 and 10: 3
Multiplying 2 by 3 will give 222
Multiplying 2 by 3 will give 6
Multiplying 2.0 by 3 will give 6.0
```

Further Resources¶

These tutorials and others you find online can be comprehensive, and cover both material that we will not, and also miss material that we will cover. Use them as supplementary material.

- Official Python tutorial
- Great Python tutorial - Automate the Boring Stuff
- Learn Python the Hard Way
- https://www.learnpython.org/

Remember that programming comes with practice! No amount of reading things online, the lecture slides or this lab will be better than just sitting down and playing around with Python. Everybody learns in different ways, so different explanations for certain concepts with have different levels of success. So if a certain explanation doesn't make sense to you, and after thinking and playing around with Python still doesn't, try another resource (or of course, one of us!).

For the curious: How was this document made?¶

This document is a Jupyter notebook converted into a .html file. Jupyter notebooks are quite useful for quickly writing some small pieces of code and sharing them with others in a nicely formatted way. We do not recommend that you use these for your lab work for now, as there are some pitfalls with using them as someone new to Python. However, it is all up to you!