

CMP 4266
Computer Programming UG1
Lab Session 1: Introduction to Python and Spyder

1. Objectives

- a. Use of programmer's logbooks and why these are needed
- b. Start with Python using IDE Spyder (Integrated DeveLopment Environment).
- c. Use the Python command line to explore some arithmetic and string expressions
- d. Create and run simple Python scripts
- e. Investigate and if possible, install Python and a simple development environment (e.g. Spyder IDE) on students own Windows, Linux or Apple computer. (as Homework)

2. Starting Python using Spyder IDE

On Windows use the search bar to look for “Anaconda Navigator”, once you select this item, to run Anaconda Navigator. This is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments and channels without using command-line commands.

Once Anaconda initialised, a window will appear containing the following applications by default in Navigator:

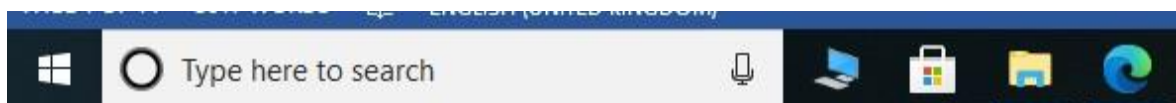
- JupyterLab
- Jupyter Notebook
- QtConsole
- **Spyder**
- VSCode
- Etc..

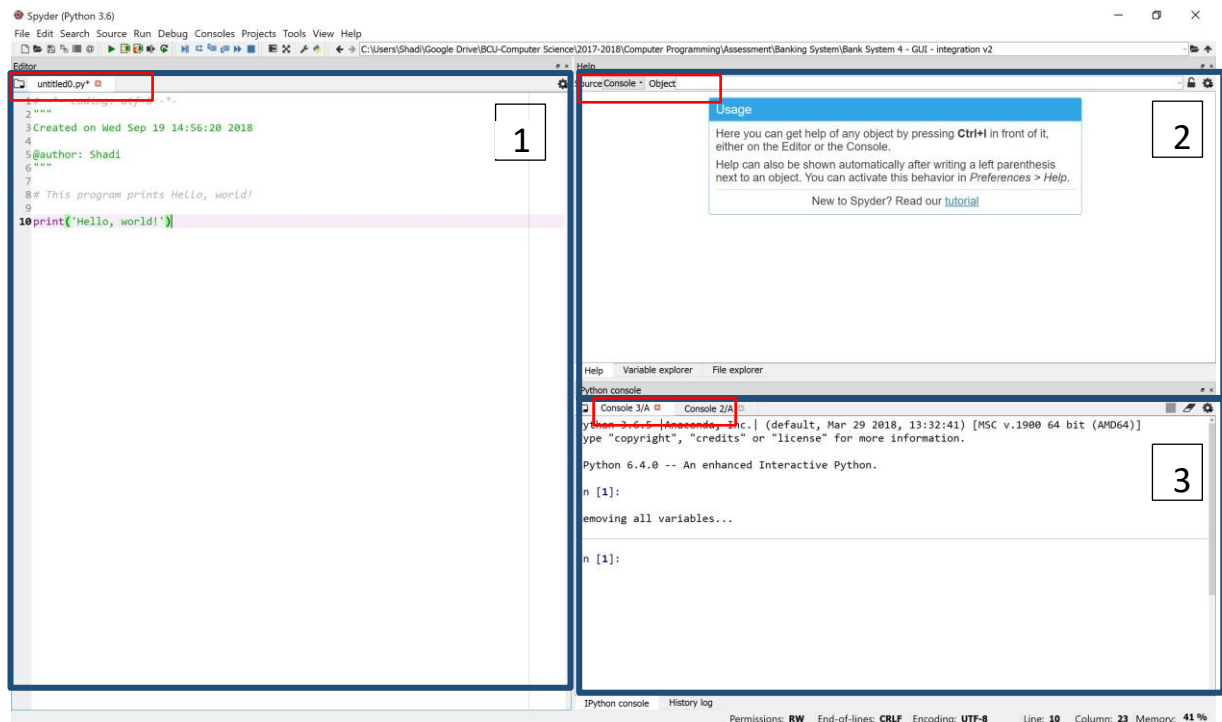
For now, we are only interested to computer programming solutions using Python, hence, we will need to select Spyder IDE to help us with the coding.

Type spyder on the windows search bar, and then click



to run Spyder.





The first thing to note is how the Spyder app is organized. The application includes multiple separate windows (marked with blue rectangles), each of which has its own tabs (marked with red rectangles). You can change which windows you prefer

To write Python code, you can do this using interactive mode or script mode. The later will be discussed later at Section 5.

The interactive mode can be used to write and test simple Python source code. In Window no. 3, python will be waiting for you to type. The Editor allows you to write sequences of commands, which together make up a program. The History Log stores the last 100 commands you've typed into the Console. The Object inspector/Variable explorer/File explorer windows are purely informational.

The **In [n]:** which indicates data inputs – it shows how the interpreter is waiting for you to type a Python statement. When you type a statement at the In [n] prompt and press the Enter key, the statement is immediately executed.

3. Using the Python command line to explore mathematical calculations

Python has numerous operators that can be used to perform mathematical calculations. Following table lists the math operators that are provided by the Python language [1].

Symbol	Operation	Description
+	Addition	Adds two numbers
-	Subtraction	Subtracts one number from another
*	Multiplication	Multiplies one number by another
/	Division	Divides one number by another and gives the result as a floating-point number
//	Integer division	Divides one number by another and gives the result as an integer
%	Remainder	Divides one number by another and gives the remainder
**	Exponent	Raises a number to a power

In the following sections you will use these operators in the SPYDER command prompt.

3.1 Adding and subtracting

Try typing `2+2` and press the enter key. (With all buffered command-line systems which provide *linguistic* user interfaces, a special key is needed to say that you have finished typing.) You should see the result of 4 appear:

```
In[n]: 2+2
Out[n]: 4
```

OK, so we can use the Python command line as a simple calculator.

```
In[n]: 5.6-3.1
Out[n]: 2.4999999999999996
```

Shouldn't that have been exactly 2.5? Well, computers count in twos, and we count in tens. Computers don't always store numbers with fractions exactly, and we can't always on paper or in our organic memories either. How many decimal places are needed to store 1/3rd exactly? There's no such thing as infinite memory, so we have to allocate a reasonable quantity.

3.2 Multiplication, two kinds of division and remainders

```
2*3
```

```
Output
```

```
6
```

```
2*3+4
```

```
Output
```

```
10
```

```
2+3*4
```

Output

14

Shouldn't the last one have given us 20? Why didn't it? Add round brackets () so that 2 plus 3 times 4 gives 20 evaluating the whole expression from left to right, instead of multiply then add.

10/3.0

Output

3.3333333333333335

10/3

Output

3

When you started school you almost certainly learned 2 *different kinds of division*. Which of these did you learn first? The kind which results in whole numbers or the kind which results in fractions? In computing languages we call the whole numbers *integers* and numbers which can contain fractions *floating point numbers* or *floats*, (and sometimes more precise ones are called *double precision floating point* or *doubles*). Different kinds of storage are allocated for these different kinds of number.

With the kind of division which results in whole numbers, there's another result called the remainder. What's the remainder on dividing 10 by 3 ? In Python we use another operator to get this remainder result, the *modulus* operator which uses the % character.

10%3

1

Python provides ** operator to raise a number to some power.

2 ** 2

Output

4

5**0.5

Output

2.23606797749979

Carry out your own experiments using + - * / and % operators and see if you can work out the order of evaluation between these. Try to predict the results by writing down on paper the results of the following expressions *before* testing them. Then test them in Python to see if you wrote down the correct answers. Write in your logbook what you have learned from any differences between what you predicted and what you actually observed.

$4 + 5 * 3 - 2$

$6 + 4 / 2$

$7 + 5 / 2$

$5 + 7.0 / 2$

$11 \% 5 + 1 * 3$

$11 \% (5 + 1) * 3$

$3 - -8$

$1 \% 3$

$0 \% 3$

$1/0$

$1.0/0.0$

$3 \% 0$

$3 \% 2.5$

4. Variables, String Expressions and Functions

In computer program a **variable** is a name that represents a value stored in the computer memory. Variables are used to access and manipulate data stored in memory.

In Python assignment statements are used to create a variable and make it reference data. General format is *variable = expression*.

`a=2`

`b=3`

`a+b`

Output

5

`a=2`

```
b=3
```

```
a=4
```

```
a+b
```

```
Output
```

```
7
```

How can **a** be equal to both 2 and 4 at the same time? That wouldn't make sense in traditional algebra. In most programming languages a statement like `a=2` isn't an *assertion* (*stating*) that `a` is equal to 2, it *assigns* the value 2 to `a`. This doesn't stop you *assigning* a different value later, when the previous value stored by the variable disappears. The command line statements are acted upon one at a time, from top to bottom.

```
a=5
```

```
b=8
```

```
a+b
```

```
Output
```

```
13
```

String values can be stored in a variable in similar way, but string values needs to be written within a pair of double quote or single quote,

```
a='hello'
```

```
b="world"
```

```
a+b
```

```
Output
```

```
'helloworld'
```

We can add strings with Python too. Some languages don't use `+` for this job, some do.

```
b[0]
```

```
Output
```

```
'w'
```

```
b[1]
```

```
Output
```

```
'o'
```

```
b[2]
```

```
Output
```

```
'r'
```

```
b[4]
```

```
Output
```

```
'd'
```

We can get at individual characters within strings using `[]` and *index* numbers, starting counting at 0 from the left, or down from -1 from the right. So if there are 5 characters in a string the valid indices are between 0 and 4 inclusive.

Try indexing a 5 letter string of your own with the following indices: -6, -5, -4, ... -1, 0, 1, ... 4, 5

A function is a piece of prewritten code that performs an operation. Python has numerous built-in functions that perform various operations. Perhaps the most fundamental built-in function is the print function, which displays output on the screen.

Here is an example of a statement that executes the print function:

```
print('Hello world')
```

if you type this statement and press the Enter key, the message Hello world is displayed. Here is an example:

```
print('Programming is fun')
```

```
Output
```

```
Programming is fun
```

Let us consider another example for the function print.

```
2+2
```

```
Output
```

```
4
```

Instead after the prompt type **print(2+2)**

```
print(2+2)
```

```
Output
```

```
4
```

If we can get the same result without the print, why use it ? Without only works on the command line. When we copy and paste what we learn from our command line into a Python program which we can edit and run many times without retyping it we'll need **print**.

5. Creating and running simple Python scripts

We've explored a useful chunk of Python programming already but without having to write a single program! In less *linguistically interactive* programming environments (e.g. C# or C Sharp), this much work could have taken us 3 weeks of 1st year classes, due to the hours we would have had to spend creating projects, editing and compiling software and correcting errors ! Just in order to do a few simple experiments. But learning any programming language or technique always needs many small and quick experiments.

But we now need to change tack a little, or we'd start wasting our time for a different reason. We don't want to have to retype what we've got working in order to run it many times. So we'll now need to edit a program file and save and run it. If we need to change it again, we can just load and edit it using the same editor. Spyder has a very suitable text editor, designed specifically for Python programs.


Using the scripting mode, you need to write your code inside the Editor Window no. 1 on the left. Once the student complete with coding, you need to save the file first in order to execute it. Click file->Save. Input a sensible name for your programme, select the destination location on which you can store your files. Ensure you select the type as "Python file". Note, Spyder is IDE, which means you can use it to write code for other programming languages such as C and C++. Hence, it is important to define the type of your file as "Python files".

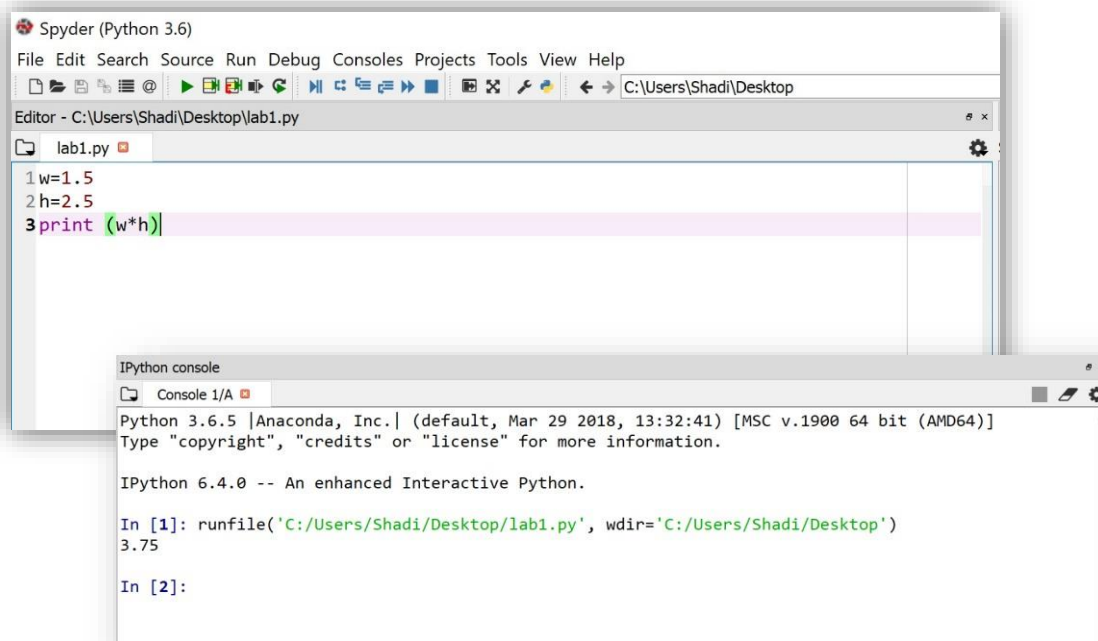
Once the file has been saved, you can now execute the python code by pressing the  button.

Type the following python code inside the Editor Window:

```
w=1.5
h=2.5
print (w*h)
```

Click file -> Save As, go to your Z: drive and create a folder such as CMP4266 and a folder under that called Lab1, save the file in this folder. When saving the file you should select file type as "Python files", which identifies it as a Python program (e.g. t1.py). Check you can open this file using Notepad using your Windows start->computer, click on the Z: drive, and the file manager this gives you, going into the folders you created. If you can open the same file you have saved in Notepad, you know where you have stored it. Close notepad - Spyder has much better editing facilities for Python source code files.

In Spyder, click the  to run your first programme.



You could use F5 as a shortcut instead. If you haven't made any punctuation errors, you should see the result of running the program, 3.75 on the Spyder command line. If this doesn't run, check punctuation on your program, correct any errors, save and run again until it works. If it doesn't raise your hand and work through this until it works with your tutor or another student who may be able to help you.

6. More on Strings and Strings Literals

When programmers execute a function, they say that they are calling the function. When you call the print function, you type the word print, followed by a set of parentheses. Inside the parentheses, you type an argument, which is the data that you want displayed on the screen.

In the previous example, the argument is 'Hello world'. Notice that the quote marks are not displayed when the statement executes. The quote marks simply specify the beginning and the end of the text that you wish to display.

```
print('Kate Austen')
print('123 Full Circle Drive')
print('Asheville, NC 28899')
```

Output

```
Kate Austen
123 Full Circle Drive
Asheville, NC 28899
```

It is important to understand that the statements in this program execute in the order that they appear, from the top of the program to the bottom. When you run this program, the first statement will execute, followed by the second statement, and followed by the third statement.

String Literals: Programs almost always work with data of some type. For example, following three pieces of data:

```
'Kate Austen'
```

```
'123 Full Circle Drive
```

```
'Asheville, NC 28899'
```

These pieces of data are sequences of characters. In programming terms, a sequence of characters that is used as data is called a string. When a string appears in the actual code of a program it is called a string literal. In Python code, string literals must be enclosed in quote marks.

As mentioned earlier, the quote marks simply mark where the string data begins and ends. In Python you can enclose string literals in a set of single-quote marks (') or a set of double quote marks ("). The string literals in the above code are enclosed in single-quote marks, but the program could also be written as shown below.

```
print("Kate Austen")
print("123 Full Circle Drive")
print("Asheville, NC 28899")
```

Output

```
Kate Austen
123 Full Circle Drive
Asheville, NC 28899
```

If you want a string literal to contain either a single-quote or an apostrophe as part of the string, you can enclose the string literal in double-quote marks. For example, the following script prints two strings that contain apostrophes.

```
print("Don't fear!")
print("I'm here!")
```

Output

```
Don't fear!
```

I'm here!

Likewise, you can use single-quote marks to enclose a string literal that contains double-quotes as part of the string. The following code shows an example.

```
print('Your assignment is to read "Hamlet" by tomorrow.')
```

Python also allows you to enclose string literals in triple quotes (either `"""` or `'''`). Triple-quoted strings can contain both single quotes and double quotes as part of the string. The following statement shows an example:

```
print("""I'm reading "Hamlet" tonight.""")
```

This statement will print:

Output

I'm reading "Hamlet" tonight.

Triple quotes can also be used to surround multiline strings, something for which single and double quotes cannot be used. Try the following code:

```
print("""One  
Two  
Three""")
```

Great. Now you have learnt something about printing multiline strings in Python.

7. Comments:

Comments are short notes placed in different parts of a program, explaining how those parts of the program work. Although comments are a critical part of a program, they are ignored by the Python interpreter. Comments are intended for any person reading a program's code, not the computer.

In Python you begin a comment with the `#` character. When the Python interpreter sees a `#` character, it ignores everything from that character to the end of the line. For example, look at the below programme. The first two lines are comments that briefly explain the program's purpose.

```
# This program displays a person's  
# name and address.  
print('Kate Austen')
```

```
print('123 Full Circle Drive')
print('Asheville, NC 28899')
```

The output will look like this

Output

```
Kate Austen
123 Full Circle Drive
Asheville, NC 28899
```

Programmers commonly write end-line comments in their code. An end-line comment is a comment that appears at the end of a line of code. It usually explains the statement that appears in that line. The programme below shows an example. Each line ends with a comment that briefly explains what the line does.

```
print('Kate Austen')      # Display the name.
print('123 Full Circle Drive') # Display the address.
print('Asheville, NC 28899') # Display the city, state, and ZIP.
```

Is the output of the above code differ from the previous example?

8. Exit Python: Time to leave the python command-line. You can either exit using `quit()` or EOF (Ctrl-d).

9. Students Exercises

1) Roots of a Quadratic Equation

Consider the quadratic equation:

$$A * x^{**2} + B * x + C = 0$$

where x is the unknown and A , B and C are constants (with A not equal to 0). Note that here we use the Python notation for exponents (two asterisks). A quadratic equation has two solutions (called roots), which may not be distinct values and which may not be real values.

The two roots of a quadratic equation may be calculated using the quadratic formula. See the brief article at Wolfram MathWorld if you don't recall the formula:

<http://mathworld.wolfram.com/QuadraticFormula.html>

- a) Discuss the pseudocode
 1. Use variables: A , B , C root1 and root2 of type floating-point
 2. Input values for A , B and C
 3. Print the inputted values for A , B and C
 4. Calculate $d = \sqrt{B^2 - 4 * A * C}$
 5. Calculate $root1 = (-B + d) / (2 * A)$

6. Calculate $\text{root2} = (-B - d) / (2 * A)$
 7. Print root1, root2
 8. End Programme.
- b) Copy the file named “lab01.py” into your computer or you can directly work on it.
- c) Modify that program to compute the two roots of a quadratic equation, as described in the pseudocode.
- d) Test your completed program using the following values:

A = 1, B = 0, C = -4
 Root #1 = _____ (should be 2.0)
 Root #2 = _____ (should be -2.0)

A = 1, B = 5, C = -36
 Root #1 = _____
 Root #2 = _____

A = 2, B = 7.5, C = 6
 Root #1 = _____
 Root #2 = _____

A = 0, B = 3.5, C = 8 # this test case fails and generates an error (why?):
 Root #1 = _____
 Root #2 = _____

2. Population standard deviation

Write a Python script that calculates the sum, average and **population standard deviation** of 5 numbers stored in the variables a,b,c,d,e The formula to calculate the standard deviation of a number can be found below. To ensure that your calculation is correct, use the following 5 numbers 9, 8, 12, 33, 5 and assign then into the 5 variables a, b, c, d, e. The population standard deviation should equal to 10.051865498503252.

The formula to calculate the population standard deviation:

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$$

Here's how to calculate population standard deviation:

Step 1: Calculate the mean of the data—this is μ in the formula.

Step 2: Subtract the mean from each data point, and then square the result to make it positive.

Step 3: Add the squared deviations together.

Step 4: Divide the sum by the number of data points in the population. The result is called the variance.

Step 5: Take the square root of the variance to get the standard deviation.

10. Moodle submission

At the end of this lab session, zip the python files that you created in Sections 9, and then upload it on Moodle using the provided submission link. **Note:** It is important to complete the weekly labs in particular labs 1, 2, 3, 4, 5 and 6 because two lab Exercises commencing W1-W6 will be randomly picked and will be marked at 30% of the overall mark for this module.

11. Homework

Before obtaining and downloading Python development environments onto your own computer, make sure you have found and directly access the reputable developer download. Don't just go to the first site you search for which is offering a download, because many sites which use search engine optimisation will give you more than you intended. Much malware, adware, spyware and other Trojans are distributed by these means. Spyder works on any likely PC operating system. Python and Spyder can be obtained from <https://www.spyder-ide.org/> and through which you can download Spyder with Anaconda distribution.

If you intend running Spyder on Linux, most distributions package Python as standard, but you may need to install Spyder using the correct distribution specific software package (e.g. Ubuntu, Debian or Fedora). Python and Spyder also work on Apple computers, but you may need compatible downloads and libraries (e.g. Tkinter) first.