

Big Data Lab – Week 2: Working with Python

In this lab you will practice working with Python from the command line and using an IDE, and practice using some basic Python syntax and features, including features that are useful in data analytics.

You can refer to the following sources:

- **Linux cheat sheet** (posted on GCULearn)
- **Python cheat sheet** (posted on GCULearn)
- **Python tutorial** (<https://www.tutorialspoint.com/python/>)

Task 1: Working with the shell

This exercise is based on Python 3, which is installed in the Linux Mint Virtual machine in the labs. If you have downloaded and installed Anaconda 3 on your own machine, Python 3 has also been installed.

To run the VM on a lab PC on campus, click **This PC** on the desktop and then navigate to the **VMware** folder on the **C** drive.

Click on the folder **SCEBE-LinuxMint_17.xxxx**

Inside this folder you are looking for the executable file in the file set in front of you.

Using the following username and password to log into the VM:

Username: **bdtech**
Password: **bdtech**

1. Check the version of Python

a. If you are using the VM

Open a terminal window. Enter the following command:

```
source activate py37  
python --version
```

You may have to enter the full path to the Python executable, for example:


```
/home/bdtech/anaconda2/bin/python --version
```

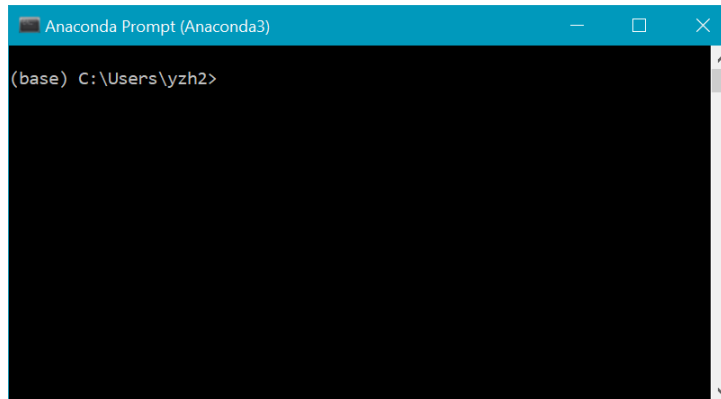
You should see a message confirming the version of Python that is installed, e.g.

```
Python 3.7.3
```

- b. If you are using locally installed Anaconda 3

You can start a python shell from Anaconda Prompt.

Click the Windows Start button  and navigate to the Anaconda3 folder, click the arrow on the right end the menu to open the list, then click **Anaconda Prompt (Anaconda3)** to open an Anaconda Prompt window similar to:



In an Anaconda Prompt window. Enter the following command:

```
python --version
```

You should see a message confirming the version of Python that is installed, e.g.

```
Python 3.8.8
```

2. Start a Python shell

Create a folder where you want to store your working files for this lab and navigate to this in the command window. Enter the command (with full path if necessary):

```
python
```

This should start the **Python shell**. When it is ready you should see a message and a '>>>' prompt, e.g.

```
(base) C:\Users\yzh2>cd d:\labs\lab01
(base) C:\Users\yzh2>d:
(base) d:\Labs\Lab01>python
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

3. Quite the Python shell

When you are finished with the shell and want to return to the OS command prompt, enter the command:

```
>>> exit()
```

4. Initialise a variable and see the value of the variable

Start a python shell again if you quit just now. Enter a simple variable initialisation statement, e.g.

```
>>> a = 5
```

You can see the value of the variable by entering an expression that is simply the variable name:

```
>>> a
5
```

5. Print to the output

The shell shows the result of each expression, but you can also explicitly print to the output. Enter the following:

```
>>> print(a)
```

6. Try changing the value of a by entering:

```
>>> a = 10
```

7. Declare another variable b with the value 7.

Enter an expression that adds a and b.

Enter a statement that prints the result of adding a and b.

8. Define a function

Enter the following function definition. This function calculates the ***n*th power** of a number, e.g. $2^3 = 8$. Note that the shell allows you to enter multiline statements like this and recognises when the statement is completed.

Python needs **indentation** to recognise when a statement is inside a function or inside another statement such as a for loop, so you should use the tab key (or the space key) when entering the code to indent as shown.

```
>>> def power (value, power):
...     result = 1
...     for i in range(0, power):
...         result = result*value
...     return result
```

```
...
>>>
```

Pay careful attention to the syntax, if you don't get it right you will have to start again!

9. Enter an expression that calls the power function, e.g.

```
>>> power(2,3)
```

Did you get the correct result? Try some more calls to power, with different parameter values. Do you always get the result you expect?

10. Define a module/class

You can define modules (classes) in the shell. Enter the following definition for a class **Welcome**, making sure you use the tab key (or the space key) to get the indentation as shown. Press the return key to return to the `>>>` prompt when finished.

Note the double underscore characters around the constructor **init**.

```
>>> class Welcome:
...     def __init__(self, name):
...         self.name = name
...     def sayHello(self):
...         print ("Hello "+self.name)
...
>>>
```

Create an instance of Welcome and call its **sayHello** method

```
>>> tt = Welcome("Alex")
>>> tt.sayHello()
Hello Alex
>>>
```

Task2: Running scripts

The shell is useful for experimenting interactively with Python. However, it can be more convenient sometimes to write your Python code in a file in a text editor, and run it as a script. That way, if you have made an error in your code, you can just edit it and run the script again.

1. Use a text editor to create a file `script_1.py` in your working folder.

If you are running the VM, you can use **gedit** by enter the following command in a terminal window or at the command prompt:

gedit

If you are using a locally installed Anaconda, you can use **Notepad** or any other editors available on your computer.

2. In the editor, enter the definition of the power method from Task 1, and add some print statements which print the result of calling the function with various values. Save the script file as **script_1.py** in your working folder.
3. In a terminal window (for VM) or a Prompt window (for Anaconda), run the script with the command:

```
python script_1.py
```

You should see the output from the print statements in your script.

Note: this will not work if your current folder is not the one where your script file was saved.

4. Create a new file **cpoint_1.py** in your working folder and use your text editor to enter the following in that file:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def move(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy
    def toString(self):
        return "(" + str(self.x) + ", " + str(self.y) + ")"

pt = Point(1, 2)
print(pt.toString())
pt.move(10, 10)
print(pt.toString())
```

This defines a class called **Point** that represents a point in 2D space, and code to create an instance of *Point* and call its methods.

5. Run this code from the OS command prompt with the command:

```
python cpoint_1.py
```

Do you see the output you expect?

Task 3: Working with an advanced IDE

Some developers like to work with the command prompt. Others prefer to use an advanced IDE, which provides many features to help developer productivity, including organisation of work into projects, visual debugging and autocomplete to give suggestions that can speed up code entry. Here, you will look at the **Spyder** IDE.

Spyder IDE (which is contained in Anaconda Navigator by default) has been installed and configured in the Linux Mint Virtual Machine on the lab PCs. Also, if you are using a locally installed Anaconda, **Spyder** IDE should also be installed along with Anaconda.

1. Launch the *Spyder* IDE

a. If you are using the VM


Open a terminal window, enter the following command:

```
spyder
```

You will see the *spyder* window.

Note, try `spyder --new-instance` if the above command does not work.

b. If you are using locally installed Anaconda 3

Click the Windows Start button  and navigate to the Anaconda3 folder, click the arrow on the right end of the menu to open the list, then click **Spyder (Anaconda3)** to launch the Spyder IDE.

2. Enter the same code you created for ***script_1.py*** from Task 2 in the Editor window. Click the green Run button at the top of the script window to run the script. You should see the output in the console window.

```
script_1.py* X
1  def power(value, power):
2      result = 1
3      for i in range(0, power):
4          result = result * value
5      return result
6
7  print(power(2,3))
8  print(power(10,4))
9
```

```
Console 1/A X
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 7.22.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/Labs/Lab01/script_1.py', wdir='D:/Labs/Lab01')
8
10000

In [2]:
```

3. IDEs are particularly useful when working with larger projects where you want to organise your code into multiple modules. Create a new Python file called **cpoint.py**. Enter the same code you created for the Point class in Task 2. This creates a module.
4. Create another Python file called **app.py**. This will contain code that creates an instance of Point and calls its methods, so it needs to import the module classes. Enter the following code:

```
import cpoint

pt = cpoint.Point(1, 2)
print(pt.toString())
pt.move(10, 10)
print(pt.toString())
```

Note that to create an instance of Point you have to provide not just the name of the class, but also the module which contains it and which you import with the import statement.

5. Run the script **app.py** and make sure you get the output you expect in the console.

Task 4. Working with Jupyter Notebooks


Jupyter notebooks are a powerful way to write and iterate on your Python code for data analysis. You can write lines of code and run them at a time, rather than writing an entire program. The Jupyter Notebook is a web application in which you can create and share documents that contain live code, equations, visualization as well as text.

1. Launch the Jupyter Notebook environment (which is contained in Anaconda)

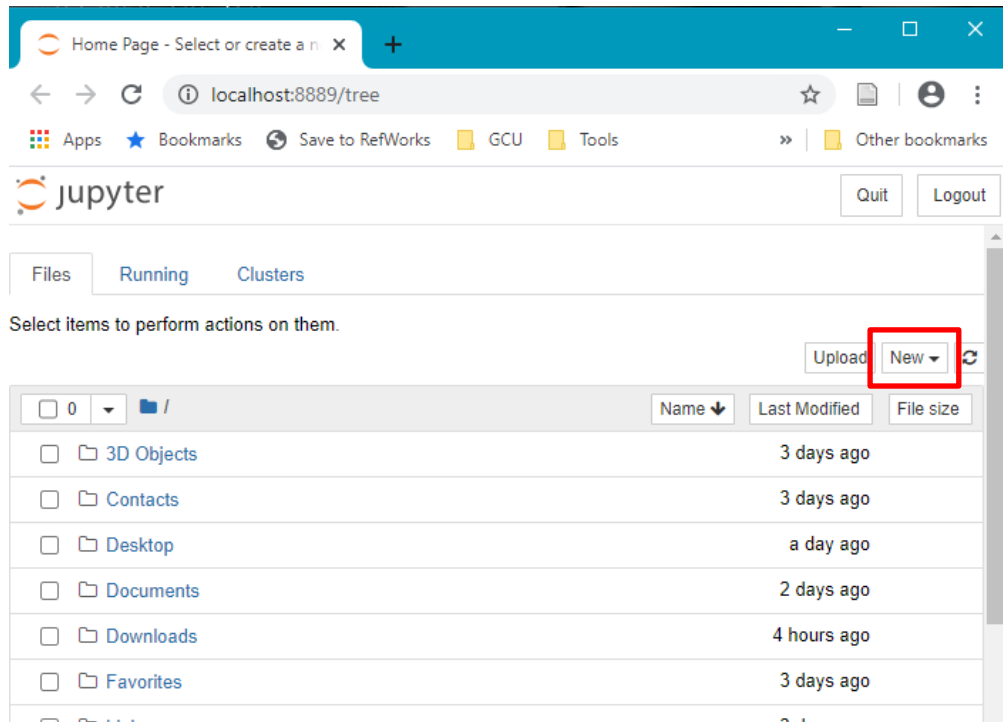
On the Linux Mint VM, you can enter:

```
jupyter notebook
```

at the command prompt.

If you are using a locally installed Anaconda, Click the Windows Start button  and navigate to the Anaconda3 folder, click the arrow on the right end of the menu to open the list, then click **Jupyter Notebook (Anaconda3)** to launch the Jupyter Notebook environment.

A new tab will be opened in the default web browser and show the notebook home page, which should look something like the following screenshot.



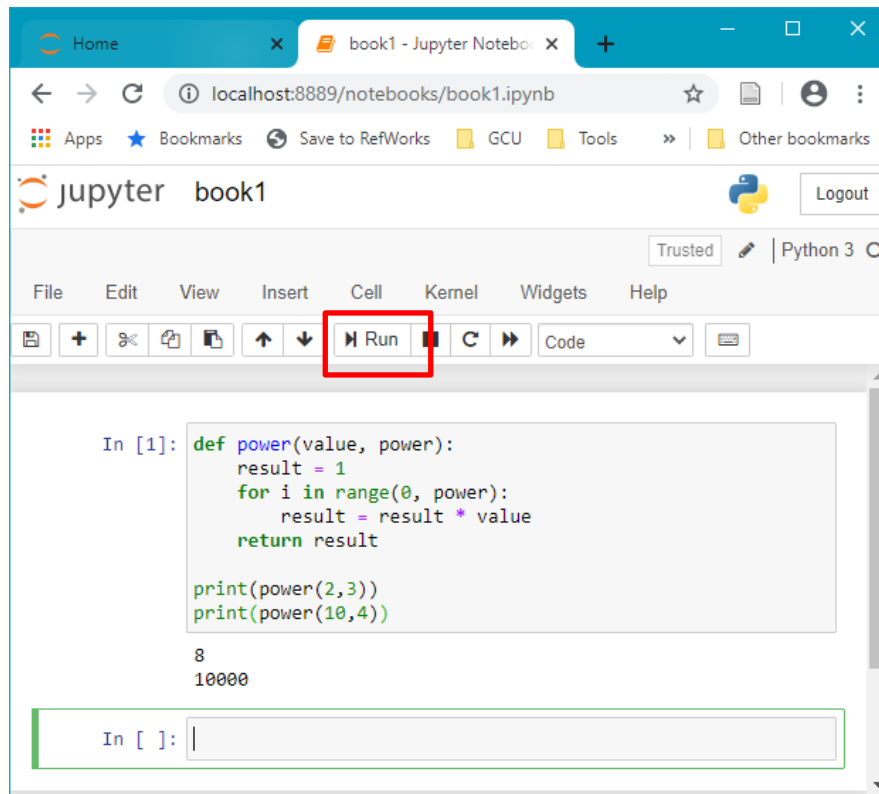
This is the Notebook Dashboard, designed for managing your Jupyter Notebooks. The URL for the dashboard is <http://localhost:8888/tree>. You can navigate in the list to your working directory.

2. Create a new notebook

Click the **New** button on the right-hand side of the page and select “**Notebook > Python 3**”. This should create notebook called Untitled1 and opened in a new tab in the browser. The notebook will contain a single cell. Choose the “File > Rename...” option from the menu in the notebook and rename it to **book1**. If you switch back to the dashboard, you will see the new file **book1.ipynb** and you should see some green text that tells you this notebook is running.

3. Add code to the notebook

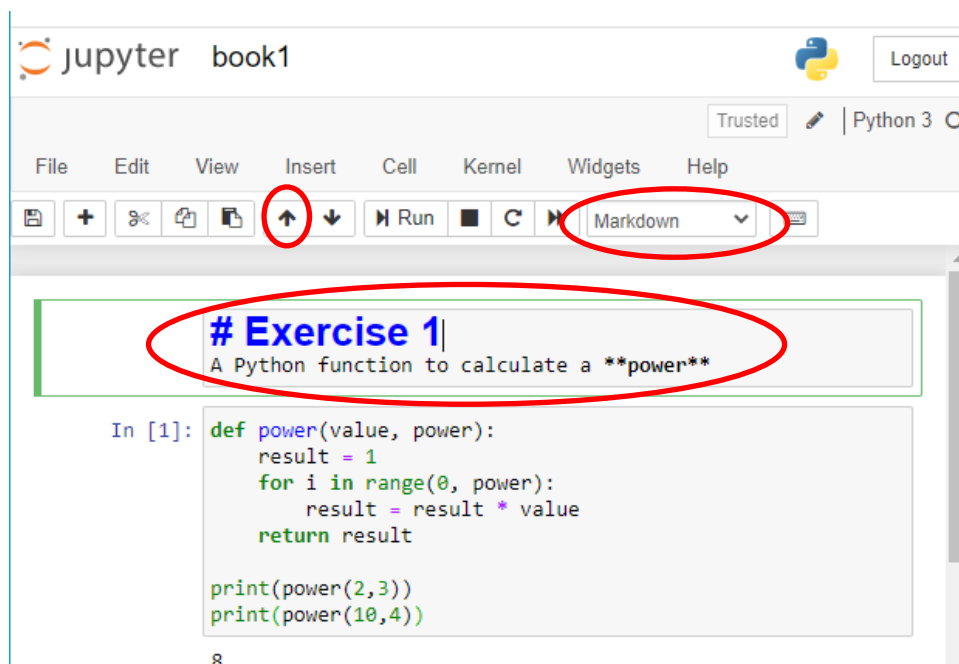
Enter the same code you created for **script_1.py** from Task 2 in the notebook cell. Click the run button on the notebook toolbar – this will run the currently selected cell, which will be the only cell at this point. You should see the output from the code and a new cell should be created.



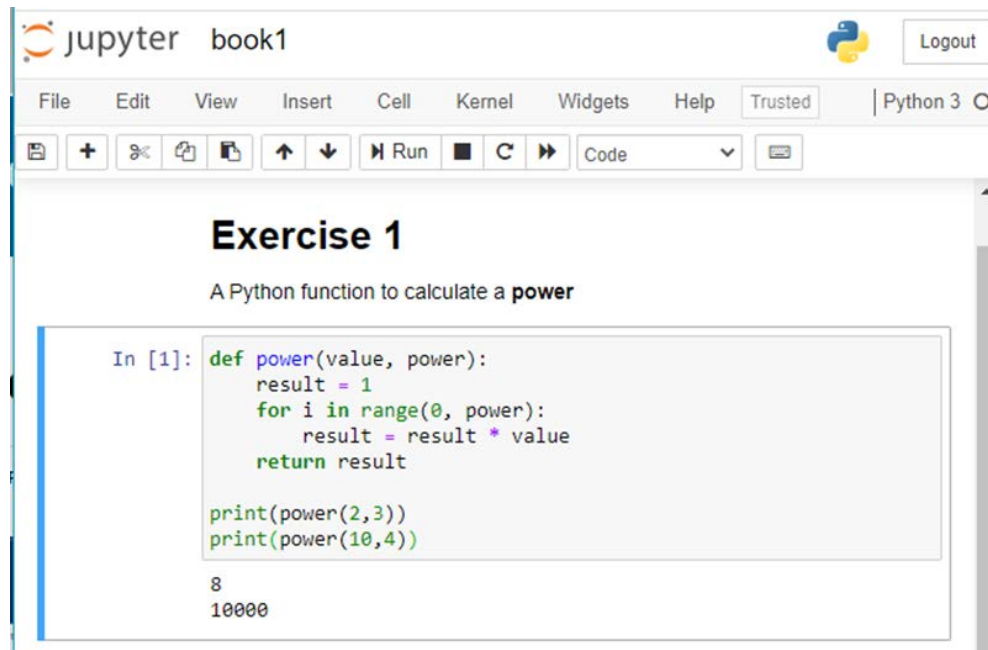
4. Create a text-only cell

In addition to running lines of code, you can also include **text-only** cells that use *Markdown* to format and organize your notebooks.

When a cell is created, it will default to being a **code** cell. Select *Markdown* instead of *Code* from the dropdown box in the toolbar, and click the *up-arrow* button to move the cell to the top. Then enter the text as shown in the cell below.



Run the markdown cell to see the formatted result. You will get something like:



Task 5. Working with Google Colab

You can also use Jupyter Notebook through Google Colaboratory (Colab). Google Colab notebooks are Jupyter notebooks that run in the cloud and are highly integrated with Google Drive.

1. As mentioned in the “Software Requirements for Labs and Coursework”, create a new google account using your caledonian.ac.uk email address at:

[Http://google.com/accounts/newAccount](http://google.com/accounts/newAccount)

2. Using the following URL to access Google Drive and login using your new Google account.


<https://drive.google.com/>

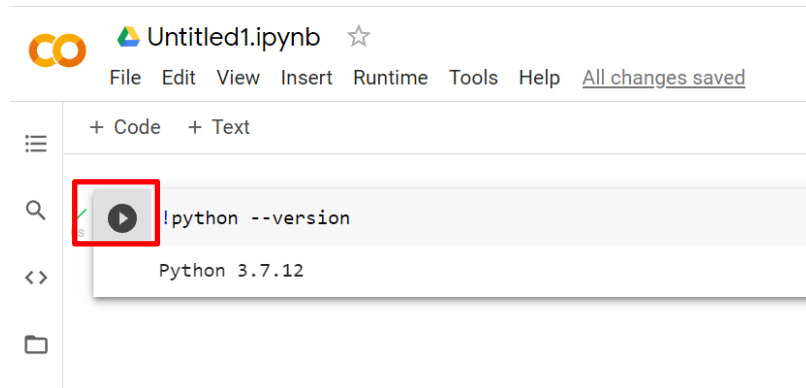
3. Using the following URL to access **Google Colab** and login using your new Google account.

<https://colab.research.google.com/>

4. In **Google Colab**, access the File menu and create a new notebook. In the empty code cell add the following.

```
!python --version
```

Execute the cell by clicking the  icon and check that the Python version number is displayed, which should look something like the following screenshot.



5. Go to Google Drive and locate the new notebook file “*Untitled1.ipynb*”.
6. Download from GCULearn the file called “**Colab_Overview.ipynb**” to your Google Drive. This file is present in the Week 1 folder. This was file used in the demonstration at the beginning of the lab session. Place it in the same location as the previous new notebook file you created.
7. Open the notebook file and work through it in **Colab** to ensure you understand all that it covers. Ensure that you tweak the code to help you gain a better understanding of what it does.

Task 6: Practicing Python

You have now tried a number of tools and ways of working with Python. In this task you will now practice writing some Python expressions yourself. Unless stated otherwise, you can use whichever of these tools you prefer to attempt the following exercises.

1. Write and test a function called **square** that takes one parameter and returns the square of the parameter
2. Write and test a function called **distance** that takes parameters x_1 , y_1 , x_2 and y_2 , all of type Double as parameters. These represent the coordinates of two points in 2D space (x_1 , y_1) and (x_2 , y_2). Your function should return the distance between these points, using the distance formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

For example, `distance(0,0,1,1)` should return `1.414` (approximately).

You can make use of your square function in the distance function. You will also need to use Python's `sqrt` function, which is in the `math` module, so you will need to use the following statement and refer to the function as `math.sqrt`.

```
import math
```

3. Create list of integers using the following statement

```
numbers = [4,7,3,2,6]
```

Write a function called **maximum** that takes a list of numbers as a parameter and returns the highest value in the list.

4. For this exercise you should work with the `cpoint.py` module and `app.py` script you created in Task 3. Create an additional class in `ccircle.py` called **Circle** with two constructor parameters:
 - a. centre, this will be an object of type *Point*
 - b. radius, this will be a number

Add a method called **area** which returns the area of the circle (πr^2).

Add a method **toString**, similar to the one in *Point*, that returns the details of the **Circle** as a string, for example `(10,10), radius = 5`. You may want to call the `toString` method of *Point* in this method.

5. Test your **Circle** class by adding the following code to `app.py` you created previously and run it:

```
circ = classes.Circle(pt,5)
print(circ.toString())
```

Test further by adding code to print the area and radius of the circle, to change the radius to 10 and to call **toString** again.