Introduction to Python

MM 225: AI and Data Science

Week 1

In this course, we are assuming that all of you have been exposed to a programming language. Hence, the emphasis will be in teaching how to get things done in python – in terms of commands, operators, syntax, libraries and function calls. We assume that the ideas of loops, control, conditions, types of variables etc are known.

In other words, this is only a tutorial introduction and we will only teach those aspects that are needed for our course. In case you are interested in learning the complete python programming, we strongly recommend the python version of how to think like a computer scientist [1].

1 Getting started

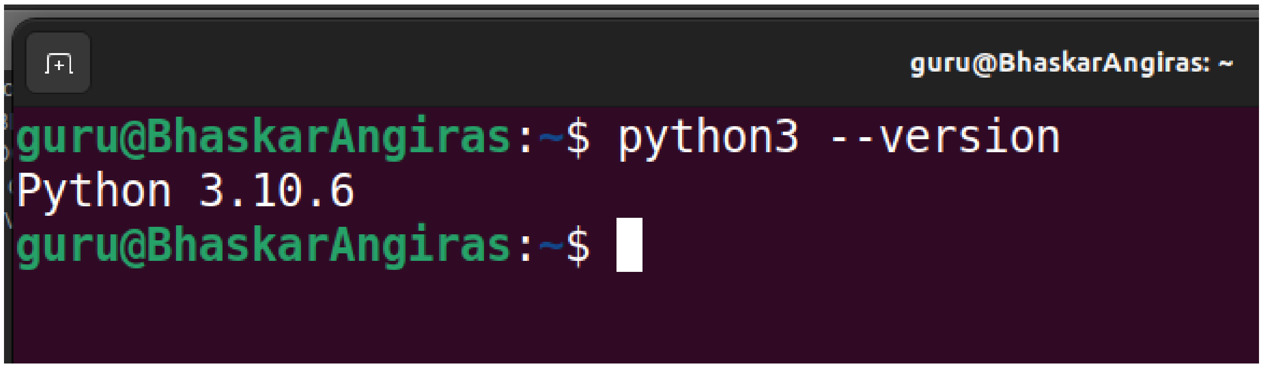
Let us get started with python.

1.1 Version

Open a terminal. Type `python3 --version` and enter. You will see the version of python in your computer. In my case, the version is 3.10.6.

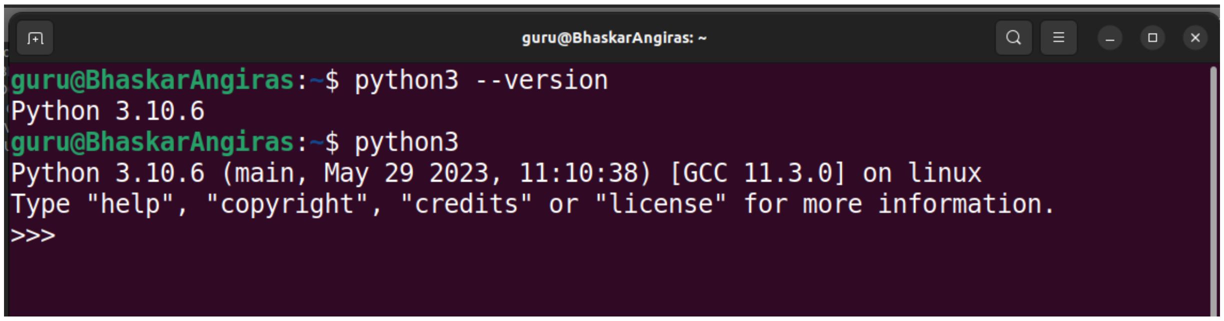
1.2 Help

One can get help from python. In order to do so, we open the python terminal by typing `python3` in the terminal and entering. Then you get the python terminal as shown. As indicate in the termianl, you can get the python version information as well as the OS along with some help. Let us try help and see what we get. Just typing `help` prompts us to use either `help()` or



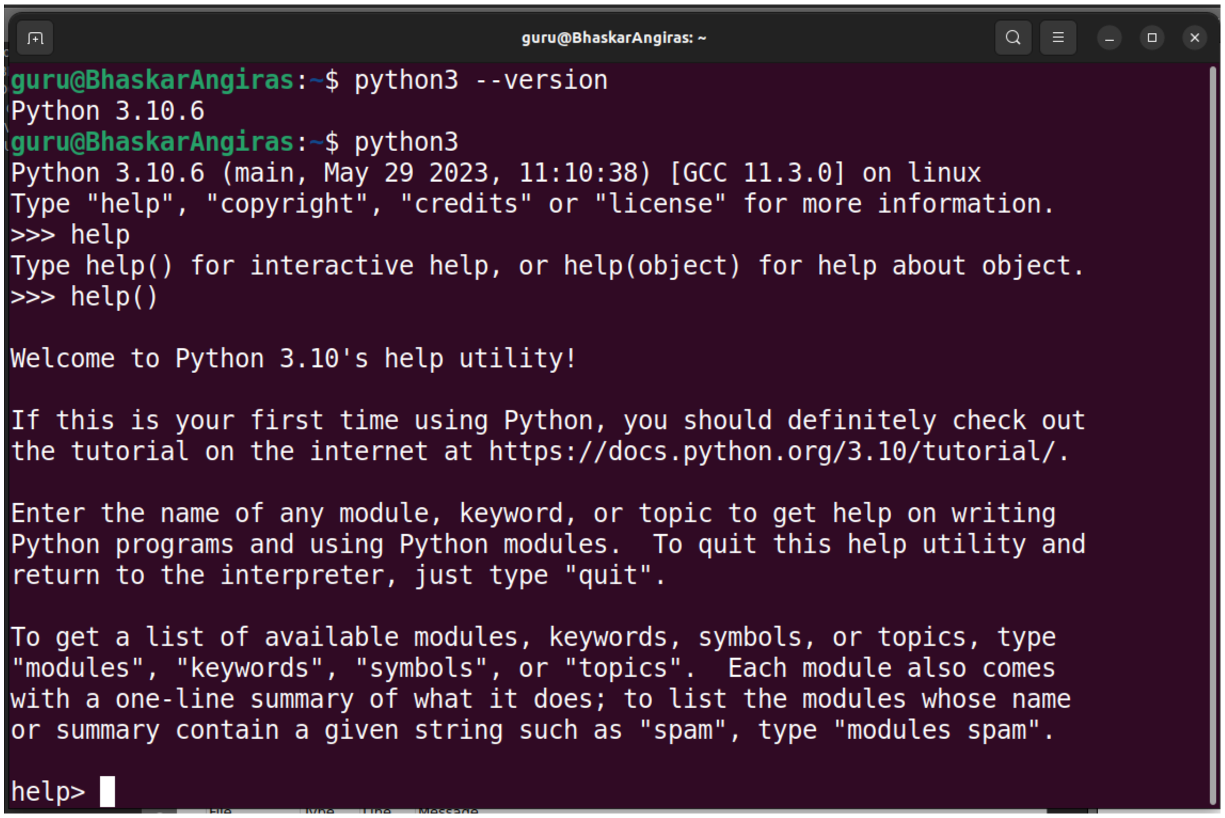
```
guru@BhaskarAngiras:~$ python3 --version
Python 3.10.6
guru@BhaskarAngiras:~$
```

Figure 1: The python version.



```
guru@BhaskarAngiras:~$ python3 --version
Python 3.10.6
guru@BhaskarAngiras:~$ python3
Python 3.10.6 (main, May 29 2023, 11:10:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

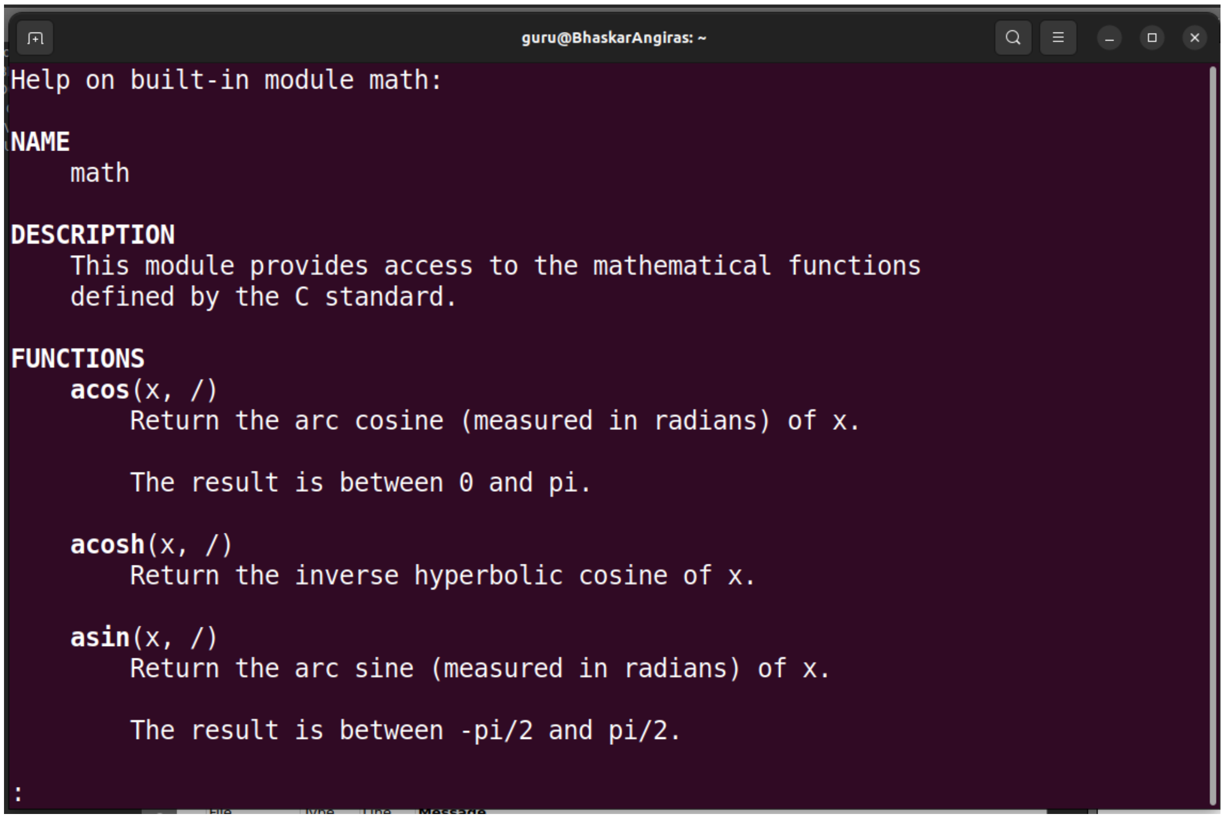
Figure 2: The python terminal.



```
guru@BhaskarAngiras:~$ python3 --version
Python 3.10.6
guru@BhaskarAngiras:~$ python3
Python 3.10.6 (main, May 29 2023, 11:10:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> help()

Welcome to Python 3.10's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the internet at https://docs.python.org/3.10/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help>
```

Figure 3: The python help prompt.

help(object). The result of help() is shown in the figure below. Typing math in the help prompt, for example, results in the following display: You



```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

        The result is between 0 and pi.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

        The result is between -pi/2 and pi/2.

:
```

Figure 4: The python help for math.

can leave the topic by typing q and enter. You can leave the help prompt by tying another q. We are back to python prompt after these two quits.

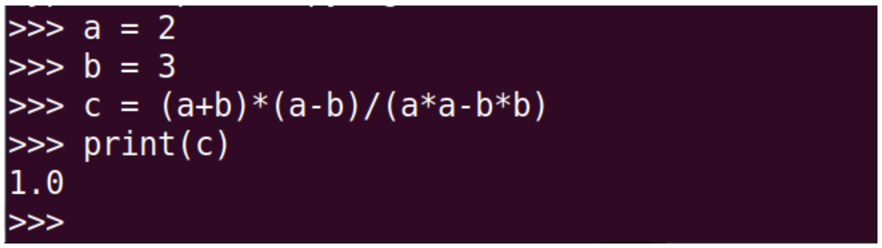
1.3 Python as calculator

Python can be used as a calculator. The following screen shot shown some simple mathematical operations carried out using python:

- **Algebra on numbers**

```
a = 2
```

```
b = 3
c = (a+b)*(a-b)/(a*a-b*b)
print(c)
```

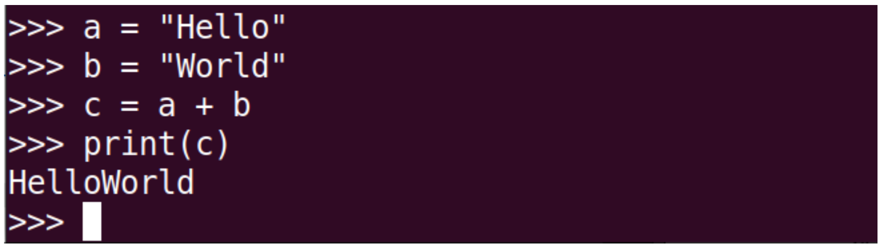


```
>>> a = 2
>>> b = 3
>>> c = (a+b)*(a-b)/(a*a-b*b)
>>> print(c)
1.0
>>>
```

Figure 5: Arithmetic operations using python. Note that `print` command can be used to print the result. As expected the result is unity.

- **Dealing with strings**

```
a = "Hello"
b = "World"
c = a + b
print(c)
```



```
>>> a = "Hello"
>>> b = "World"
>>> c = a + b
>>> print(c)
HelloWorld
>>>
```

Figure 6: Operations on strings using python. Note that `print` command results in this being your first Hello World program. Can you try and modify these commands a bit to intoduce the space between the two words and add an exclamation at the end.

1.4 A simple python script

Till now, we have interacted with python through the terminal. However, in most cases, we would like to interact with python in script mode. Let us write a short script. For example, consider the Stirling's approximation formula for factorial of n:

$$n! \approx n^n e^{-n} \sqrt{2\pi n} \tag{1}$$

Let us say we want to write a script that approximates $n!$ for, say, $n = 10$. Note that the computation involves π and the in-built `math` module of python contains this and many other mathematical functions. Hence, in our script we import `math` and use it.

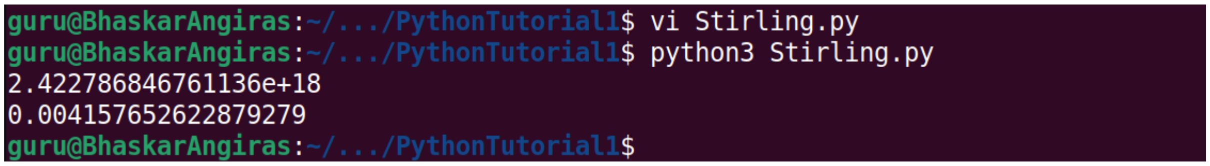
The script given in the box `Stirling.py` below is commented and is easy to follow. Note that when you try this, you need not type anything that starts with the # symbol. They are comments and meant for you to understand the script. However, when you write your scripts and submit them for evaluation, please comment them in as much detail as possible.

Stirling.py

```
# Script to use Stirling's formula to calculate n!
import math as m # To import math module
# Note the use of m to refer the the module
n = 20 # Define n
# Calculate n factorial and store it as fact
fact = pow(n,n)*pow(m.e,-n)*m.sqrt(2*m.pi*n)
print(fact) # Print n factorial
# Use in-built factorial
# function from math module and calculate the
# error in the approximation
print((m.factorial(n)-fact)/m.factorial(n))
```

Note the use of `m.e` and `m.pi` to obtain the values of e and π. We use the function `pow` for exponentiation and we use the `sqrt` function from `math` module for taking the square root. The in-built factorial function in the math module is, helpfully, called factorial!

In writing the script, the first step is to use your favourite text editor and save the script with `py` extension to indicate this is a python script. In this case, I have saved this file as `Stirling.py`. Then, the command `python3 Stirling.py` gives the result as shown in Fig. 7. As we see, the error in the approximation is of the order of 0.42%.



```
guru@BhaskarAngiras:~/.../PythonTutorial1$ vi Stirling.py
guru@BhaskarAngiras:~/.../PythonTutorial1$ python3 Stirling.py
2.422786846761136e+18
0.004157652622879279
guru@BhaskarAngiras:~/.../PythonTutorial1$
```

Figure 7: The Stirling.py file, written using the vim editor and executed. The result shows both the approximation for 20! and the error in the approximation.

1.4.1 Exercise

Can you modify the script to calculate 30! and the error in the approximation? Save the script and submit it in moodle.

2 Plotting

In this section, we will try out some simple plotting. We will use the `matplotlib` library for doing so.

2.1 Plotting $\sin(x)$

In this section, we will learn to make a simple plot. Along the way, we also will to learn about setting up `for` loops and setting up a list.

The script given in `PlotSine.py` plots $sin(x)$ in the range $(0, 2\pi)$. We use 1000 x-points for which the data is generated and plotted. We use the module `math` for the mathematical constants and function calls, and the module `matplotlib.pyplot` for plotting. Note that we also label the axes after plotting. The `show` command is essential for the plot to be shown on screen. The plot can be saved by clicking on the save icon on the plot window. However, we cal also save the figure using `savefig` command as shown in the script. Note that the `savefig` should precede the `show` command.

PlotSine.py

```
# Import the relevant modules
import math as m
import matplotlib.pyplot as plt
# Define the number of points
n = 1000
# Define the increment
delx = 2*m.pi/n
# Initialise the list for x and y
x = []
y = []
# Loop to calculate the angle and sin
# Note that we use indent instead of braces
# to indicate the beginning and end of the loop
for i in range(n+1):
    x.append(i*delx)
    y.append(m.sin(i*delx))
# Plot
plt.plot(x,y)
# Label the axes
plt.xlabel('Angle in radians')
plt.ylabel('sin(x)')
# Save the figure
plt.savefig('SinePlot.png')
# Show the plot
plt.show()
```

By running this script, we obtain the plot shown in Fig. 8.

2.1.1 Exercises

1. Can you modify the `PlotSine.py` script to plot $\cos(x)$ in the range $(0, 4\pi)$? Save the script and submit it in moodle.

2. Write a script to plot the error in approximating n! using Stirling's approximation as a function of n. Upload the script in moodle.

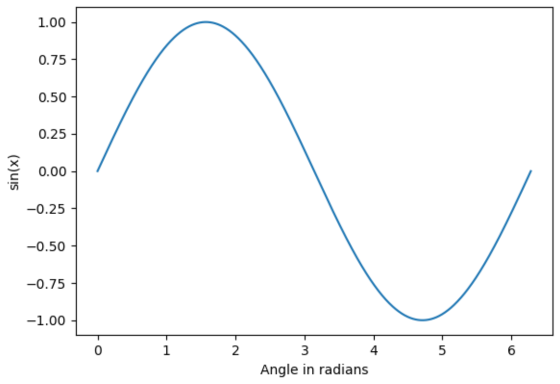


Figure 8: The plot obtained by executing PlotSine.py.

3. Write a script to plot the function

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2}$$

   in the range $-5 \leq x \leq 5$; assume $\mu = 0$, and plot for the cases $\sigma = 1$ and $\sigma = 2$ in the same plot. Label the axes as well as the curves. Save the plot. Upload the script and the plot in moodle.

4. Write a script to plot the function

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}} \tag{3}$$

   in the range $0 \leq x \leq 5$; assume $\sigma = 2$, and plot for the cases $\mu = 3$. Label the axes. Save the plot. When you run the code, do you see any error messages? What is it and why? Upload the script, the plot and your answer / explanation(s) in moodle.

2.2 More than one plot in the same figure

The following script plots $\sin(x)$ and $\cos(x)$ in the same figure. Note that in this case, we are not using the `for` loop or the list to plot. This is more straight forward. We are taking advantage of the `numpy` library function calls. Thus, there are several ways in which the same outcome can be achieved. It is useful to know as many of them as possible! I hope you can also follow the script in spite of the missing comments. Pay attnetion to how the variables are being generated in this case!

PlotSineCos.py

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0,6*np.pi,0.1)
y = np.sin(x)
z = np.cos(x)

plt.plot(x,y,x,z)
plt.xlabel('Angle in radians')
plt.ylabel('sin(x) and cos(x)')
plt.legend(['sin(x)','cos(x)'])

plt.show()
```

By running this script, we obtain the plot shown in Fig. 9.

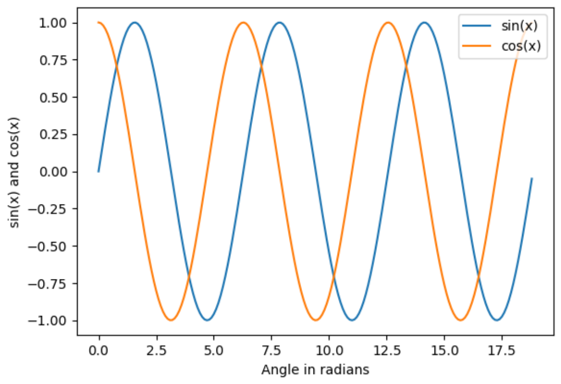


Figure 9: The plot obtained by executing PlotSineCos.py.

2.2.1 Exercises

1. Write a script to plot the n! and the approximate n! obtained using Stirling's approximation in the same figure as a function of n for n in the range 1 to 15. Upload the script and the generated plot in moodle.

2. Modify the `PlotSineCos.py` script to plot $\sin(x)$ and $\tan(x)$ in the same figure. Upload the script and your figure in moodle. Do you see any issues? How do you think this can be mitigated? Give your comments in moodle.

3 Coin matching game

Let us make the computer play a coin matching game. In this game, we make two players, G and H, show their coins. If coins match, that is, if both show head or tail, G gets both the coins. If not, that is, if it is one head and one tail, H gets both the coins. Let us write a script that makes G and H play the game for 5000 times and find out the net result.

CoinMatch.py

```
import random as r
h = 0
g = 0
for i in range(5000):
    G = r.random()
    H = r.random()
    if (G<0.5 and H<0.5): g = g+1
    elif(G>0.5 and H>0.5): g = g+1
    else: h = h+1
print(g)
print(h)
```

Note that we are generating a random number between 0 and 1 and if it is less than 0.5 (more than 0.5) we consider it as head (tail). Thus, we are assuming that both G and H show head 50% of the time.

3.1 Exercise

(a) Write and run this script. What do you see? (b) Suppose we want G to show heads 80% of the times while H still shows head 50% of the times. How will you modify the script? In that case, what happens? (c) Suppose we want G to show heads 80% of the times and H to show tails 80% of the times. How will you modify the script? In that case, what happens? (d) Suppose we want both G and H to show heads 80% of the times. How will you modify the script? In that case, what happens? (e) Based on the results of these games played by the computer, if you are playing this game in real life, what would be your strategy? Why?

Upload your scripts and answers in moodle.

References

[1] *Think python 2e: how to think like a computer scientist*, Allen B Downey, Second edition, Available for free download (along with some other resources) at `https://greenteapress.com/wp/think-python-2e/`