Basics

A **Python code/program/script** is a collection of commands in a file designed to be executed in a particular sequence in order to perform a specific task.

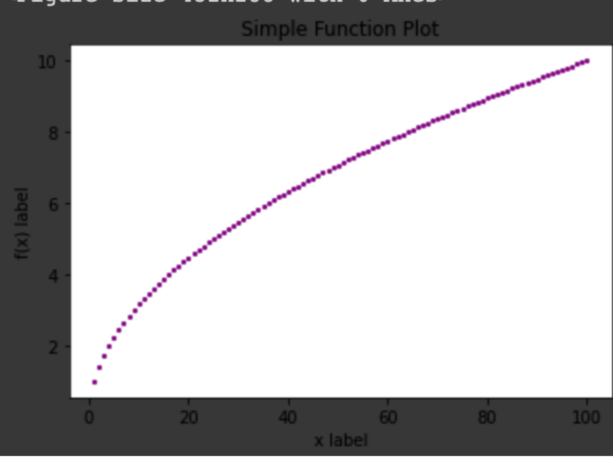
A guide for installing python can be found here, but you can always use Google Colaboratory ("**Colab**", for short) – an online platform that allows you to write and execute Python in your browser.
Benefits of using Colab:
- no need to install anything locally;
- it saves the file on **Google Drive** -- can be easily accessed from any device.

Note: works best in Google Chrome!

Here is a basic python program that plots the graph of the function $f: \mathbb{R} \to \mathbb{R}$, where $f(x) = \sqrt{x}$

```python
# Imports used for the task
import matplotlib.pyplot as plt
import math

# Example of defining a function
def f(x):
   return math.sqrt(x)

# Create an empty xOy figure
fig = plt.figure()  # an empty figure with no Axes
fig, ax = plt.subplots()  # a figure with a single Axes

# Calculate and plot f(x) for the first 100 natural consecutive numbers:
for x in range(1,101):
  ax.plot(x, f(x), color = 'purple', marker = 'o', linewidth=1, markersize= 2)
  # the command could also look as simple as:
  # ax.plot(x, f(x), 'bo')  # where b = the color blue, and o is the marker type

# Create the graph labels
plt.xlabel('x label')
plt.ylabel('f(x) label')
plt.title("Simple Function Plot")

# Print the final plot
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



As shown in the example above, the file can contain *imports*, *defined functions*, *built-in functions*, and so on. In order to become a python user you need to be aware of the integrated tools available for you to apply in your personal/school/work projects (*Data-Analysis, Data processing, etc.* …).

Python Input, Output and Import

Python Output

The print() function is used to output data to the screen. We can also output data to a file (useful when run the code on HPC).

```
print ("Python is cool.")
print ('I learn to code.')
```

```
Python is cool.
I learn to code.
```

From this example it's clear you can use both " " and ' ' in Python for a string.
The following example shows how to print a description of a variable together with that variable.

```
x = 2
print ("The value value of the variable is:", x)
```

```
The value value of the variable is: 2
```

Output formatting. If you want a more rigorous output you can do this by using str.format() method.

```
x = 5.5; y = 10.25
# use the variables in .format() to print their values
print('The value of x is {} and y is {}.'.format(x,y))

# use strings in .format() to replac in a printed sentence
print('The value of x is {} and y is {}.'.format("STRAWBERRY","PIE"))
```

```
The value of x is 5.5 and y is 10.25.
The value of x is STRAWBERRY and y is PIE.
```

Python Input

Until now, the value of variables was defined. To allow flexibility in the program, sometimes we might want to take the input from the user. In Python, the input() function allows this.
The syntax for *input()* is:

```
input('Your name is: ')
```

```
Your name is:
```

Where 'Your name is: ' can be replaced with what you need from the user. For example:

```
input('Insert a number: ')
```

```
Insert a number: 5
'5'
```

We can see that the entered value, 5, is taken by the program as a string.
It is important to know what kind of input you are expecting from the user.
If you need a *string* – the above method works, but if you need an *integer* or a *float* to proceed with further calculations, you have to encapsulate the input() into **int(**input(…)**)** or **float(**input(…)**)**.
Bonus you can directly calculate a string operation using eval() on the input() as in the example below:

```
eval(input('Insert an operation: '))
```

```
Insert an operation: 2+1
3
```

Python Import. Useful libraries/modules to import

Some commands might run without any imported libraries (for example "*print('Hello World!')*", or some basic calculations *a+b, a*b*) but most of the time you will need to use specific packages called *libraries*.

Here are some of the most common used libraries (*click on the name to access their official documentation*):

1. **Matplotlib** is a Python library used to write 2-dimensional graphs and plots.

- Often, *mathematic* or *scientific* applications require more than single axes in a representation.
- This library **helps us to build multiple plots at a time**.
- You can, however, use Matplotlib to **manipulate different characteristics of figures** as well (like shown in the example: *linewidth, marker type, color, etc.)*

How to call it:

```
import matplotlib.pyplot as plt
```

How to use it:

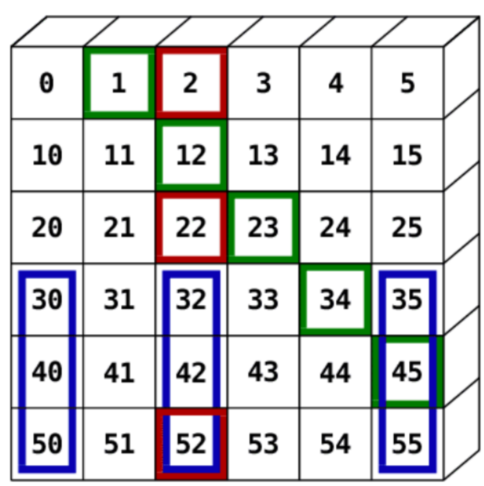
```
# Create an empty xOy figure
fig = plt.figure()  # an empty figure with no Axes
fig, ax = plt.subplots()  # a figure with a single Axes

# Calculate and plot x² for the first 100 natural consecutive numbers:
for x in range(1,101):
  ax.plot(x, x*x, 'bo')

# Create the graph labels
plt.xlabel('x label')
plt.ylabel('x² label')
plt.title("Plot Title")

# Print the final plot
plt.show()
```

Note: *you can change the highlighted part depending on your needs.*

2. **Numpy** provides good support for different dimensional array objects as well as for matrices.

- Not only confined to **provide arrays**, but it also provides a variety of tools to **manage these arrays**.
- It is fast, efficient, and really good for **managing matrics and arrays**.

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])

>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



- Numpy provides such functionalities that are comparable to MATLAB. They both allow users to get **faster with operations.**

How to call it:

```
import numpy as np
```

How to use it (more examples):

```
import numpy as np

# Defining the array (matrix)
arr = np.array( [[ 1, 2, 3],
                 [ 4, 2, 5]] )

# Printing the array
print("The array is: \n", arr)

# Printing shape of array
print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array
print("Size of array: ", arr.size)

# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)

# Printing type of arr object
print("Array is of type: ", type(arr))
```

```
The array is:
 [[1 2 3]
 [4 2 5]]
Shape of array:  (2, 3)
Size of array:  6
No. of dimensions:  2
Array is of type:  <class 'numpy.ndarray'>
```

Note: *.shape, .size, .ndim* are features of the numpy package.

3. **Scipy** is a python library that is used for mathematics, science, and engineering computation.
   - It can operate on an array of NumPy library.
   - Very suitable for **machine learning**, because contains a variety of sub-packages which help to solve the most common issue related to Scientific Computation.
   - It contains the following sub-packages:
     - File input/output - **scipy.io**
     - Special Function - **scipy.special**
     - Linear Algebra Operation - **scipy.linalg**
     - Interpolation - **scipy.interpolate**
     - Optimization and fit - **scipy.optimize**
     - Statistics and random numbers - **scipy.stats**
     - Numerical Integration - **scipy.integrate**
     - Fast Fourier transforms - **scipy.fftpack**
     - Signal Processing - **scipy.signal**
     - Image manipulation – **scipy.ndimage**

     Note: SciPy sub-packages need to be imported separately.

For example, SciPy special function includes *Cubic Root, Exponential, Log sum Exponential, Lambert, Permutation and Combinations, Gamma, Bessel, hypergeometric, Kelvin, beta, parabolic cylinder, Relative Error Exponential, etc. ...*

How to call and use it:

a). the special functions sub-package:

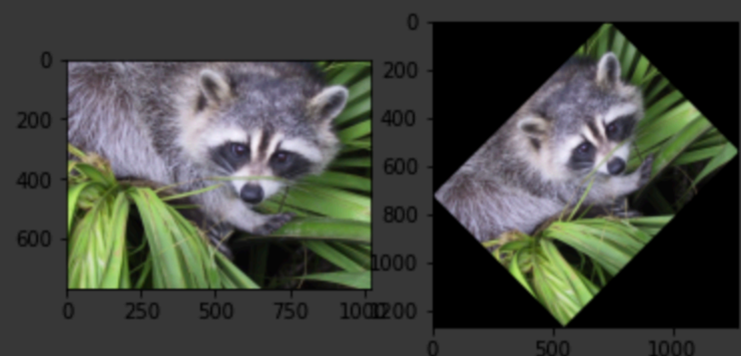
```python
from scipy.special import cbrt # for the Cubic Function
from scipy.special import exp10 # for the Exponential Function

# Calculate the cubic root of a given number
a = int(cbrt(27))
print ("The cubic root is: ", a)

# Calculate the exponential
e = int(exp10(3))
print ("The exponential is: ", e)
```

```
The cubic root is:  3
The exponential is:  1000
```

b). features of the library for image processing:

```python
from scipy import ndimage, misc
from matplotlib import pyplot as plt

# Get face image of panda from misc package
panda = misc.face()

# Rotatation function of scipy for image at 45 degree
panda_rotate = ndimage.rotate(panda, 45)

# Show the original image and the rotated image
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.imshow(panda)
ax2.imshow(panda_rotate)
```

```
<matplotlib.image.AxesImage at 0x7fa80b97fe48>
```



Numpy VS SciPy	Numpy	★ Numpy is used for mathematical or numerical calculations. ★ It is faster than other Python Libraries. ★ The most useful library for Data Science to perform basic calculations. ★ Performs the most basic operation like sorting, shaping, indexing, etc.
	SciPy	★ SciPy is built in top of the NumPy. ★ SciPy is a fully-featured version of Linear Algebra, while Numpy contains only a few features. ★ Most new Data Science features are available in Scipy rather than Numpy.

4. Pandas is a fast, demonstrative package that can be used to easily manipulate any type of data.
- Provides us with many Series and DataFrames.
- You can easily **organize, explore, represent, and manipulate data**.
- Pandas can support **Excel, CSV, JSON, HDF5**, and many other formats.
- In fact, **it allows you to merge different databases** at a time.
- It has a collection of built-in tools that allows you to both **read and write data** in databases, as well.

How to call and use it:

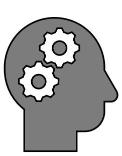
```
import pandas as pd
import pandas.util.testing as tm
import seaborn as sns

# Different ways to download and read your file
# 1). from a public link
iris = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv')
# 2). from your DropBox account
!wget https://www.dropbox.com/s/blablapath/unt.csv -q -nc
data = pd.read_csv('unt.csv')
# 3). if it is a sample dataset (for this you need to import seaborn package)
data = sns.load_dataset('iris')
data.head()

# List the columns' name of the .csv file
print ("These are your columns: \n", data.columns)

# Print the first part of the table to see how it looks like
data.head() # you can indicate between () the number of rows to be shown
```

```
These are your columns:
 Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Note: Use only one of the 3 methods. I wrote all the commands just to give an example for each of them.

Other modules and libraries can be found here.



Other Python Tutorials:
DataCamp has tons of great interactive Python Tutorials covering data manipulation, data visualization, statistics, machine learning, and more;
Read Python Tutorials and References course from After Hours Programming.

TIPS AND TRICKS

- ★ Try to write the commands on your own to get used to the syntax. Avoid the copy-paste.
- ★ Upload some toy dataset and start playing around – apply any function/method and see how it works and if you can handle it.
- ★ Got an error? Try to read carefully the output message, it might be intuitive. No? – Google it!