Python & SciPy Crash course

Summary

Set up (next)

Before being able to perform some Python and SciPy - or scientific - programming, we need to finish (properly) our Python ecosystem setting-up we have started in the previous section of this module.

Step 0: Create and activate a virtual environment for the course

Isolate our course into its own **venv** (e.g. virtualenv) using its own version of Python (if needed) and its own version of any other package:

```
$ cd <RPO_TOPDIR>/Module-03/
$ python -m venv .venv
$ ls -al
...
drwxr-xr-x 5 kla kla   4096 Feb 28 09:28 .venv
```

A virtual environment named .venv is created in the local directory. Then you need to activate it:

```
$ source .venv/bin/activate
(.venv) kla@assieoussou../Module-03$
```

As you can see by the appearance of (.venv) prefix, the environment is activated now.

Activate[1] the REPL (Read, Evaluate, Print, and Loop):

[1] Always ensure your virtual environment is active!

```
(.venv) $ python
Python 3.12.0 (main, Sep 12 2024, 07:03:03) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python Crash Course

To start coding some basic Python, the REPL is enough. No need to install additional packages.

Here, we will explore the basics of Python. But, you need to practice more by yourself at home because Python - even being simple - is very huge…

Assign strings, numbers, …

```
>>> data = "hello world!"
>>> print(data)
hello world!
>>> print(len(data))
12
>>> print(data[2])
l
```

```
>>> val = 123.5
>>> print(val)
123.5
>>> type(val)
<class 'float'>
```

```
>>> # multiple assignment
>>> a, b, c, d = 1, 2, 3, None
>>> print(a,b,c, d)
1 2 3 None
```

Flow control

```
# if-then-else
>>> size = 99
```

```
>>> if size == 99:
...     print("That is high!")
... elif size > 200:
...     print("That is too high!")
... else:
...     print("That is a normal size!")
...
That is high!
```

```
>>> # for loop
>>> for i in range(10):
...     print(i)
...
0
1
2
3
4
5
6
7
8
9
>>>
```

```
>>> # while loop
>>> i = 0
>>> while i < 10:
...     print(i)
...     i += 1
...
0
1
2
3
4
5
6
7
8
9
>>>
```

The most used Python (abstract) data structures

```
>>> # tuple
>>> t = (1, 2, 3)
>>> print(t)
(1, 2, 3)
>>> type(t)
<class 'tuple'>
```

```
>>> # list
>>> l = [1, 2, 3]
>>> print("zeroth element: %d" % l[0])
zeroth element: 1
>>> l.append(4)
>>> l
[1, 2, 3, 4]
>>> type(l)
<class 'list'>
```

```
>>> # dictionary
>>> # dictionary: mapping names to values (key-value pairs)
>>> d = {"a": 1, "b": 2, "c": 3}
>>> d["b"]
2
>>> print("keys of d = %s" % d.keys())
keys of d = dict_keys(['a', 'b', 'c'])
>>> type(d.keys())
<class 'dict_keys'>
>>> print("values of d = %s" % d.values())
values of d = dict_values([1, 2, 3])
>>> type(d.values())
<class 'dict_values'>
>>> type(d)
<class 'dict'>
```

Integrated help:

```
>>> help(dict)
...
class dict(object)
 |  dict() -> new empty dictionary
```

```
 |  dict(mapping) -> new dictionary initialized from a mapping object's
 |      (key, value) pairs
...
```

```
>>> # functions
>>> def sum_of(x, y):
...     return x + y
...
>>> # test
>>> res = sum_of(2, 3)
>>> res
5
```

SciPy Crash Course

Unlike Python, even learning the basics of SciPy requires us to install additional packages:

- NumPy: numpy
- Matplotlib: matplotlib
- Pandas: pandas
-

NumPy basics

You need to install it before, if not:

```
>>> import numpy
...
ModuleNotFoundError: No module named 'numpy'
```

Exit the REPL and install it using pip:

```
>>> quit()
(.venv) $ pip install numpy
```

Back into the REPL:

```
# create an array from list
>>> import numpy
>>> l = [1, 2, 3]
```

```
>>> a = numpy.array(1)
>>> a
array([1, 2, 3])
>>> a.shape
(3,)
```

Accessing array values and slicing:

```
>>> # accessing array values
>>> import numpy
>>> list_of_list = [[1, 2, 3], [4, 5, 6]]
>>> list_of_list
[[1, 2, 3], [4, 5, 6]]
>>> double_dim_array = numpy.array(list_of_list)
>>> double_dim_array
array([[1, 2, 3],
       [4, 5, 6]])
>>> double_dim_array.shape
(2, 3)
>>> print("first row = %s" % double_dim_array[0])
first row = [1 2 3]
>>> print("last row = %s" % double_dim_array[-1])
last row = [4 5 6]
>>> print("first col = %s" % double_dim_array[:,0])
first col = [1 4]
>>> print("last col = %s" % double_dim_array[:,-1])
last col = [3 6]
```

Arithmetic and operations on arrays:

```
>>> import numpy as np
>>> a1 = np.array([2, 2, 2])
>>> a2 = np.array([3, 3, 3])
>>> a3 = a1 + a2
>>> a4 = a1 * a2
>>> a1, a2, a3, a4
(array([2, 2, 2]), array([3, 3, 3]), array([5, 5, 5]), array([6, 6, 6]))
```

Matplotlib basics: .plot(), .show()

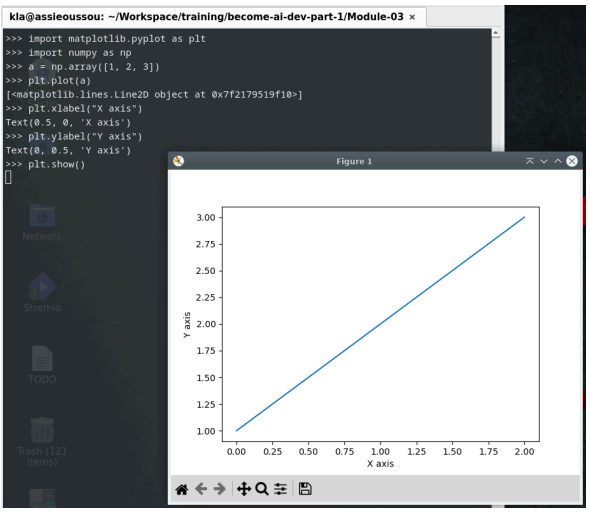
Same as for NumPy, you need to install Matplotlib also using pip:

```
(.venv) $ pip install matplotlib
```

Back in the REPL:

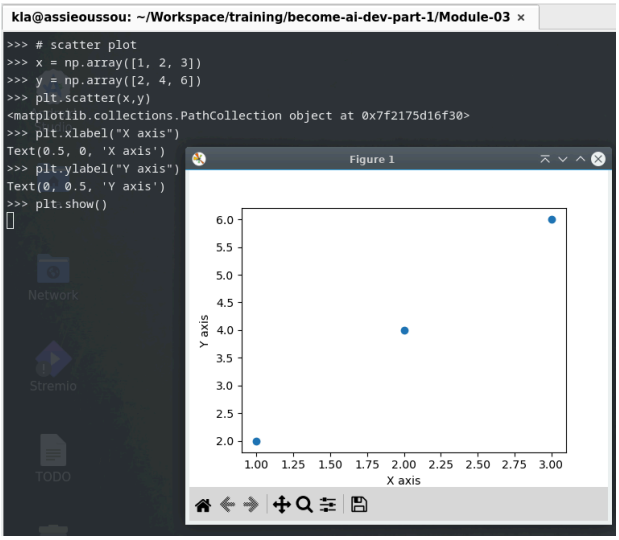
```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> plt.plot(a)
>>> plt.xlabel("X axis")
>>> plt.ylabel("Y axis")
>>> plt.show()
```

The plot is displayed in a dedicated window:



Scatter plot:

```
>>> # scatter plot
>>> x = np.array([1, 2, 3])
>>> y = np.array([2, 4, 6])
>>> plt.scatter(x,y)
>>> plt.xlabel("X axis")
>>> plt.ylabel("Y axis")
>>> plt.show()
```



Pandas basics: Series and Dataframes

Pandas provides high level data structures and functionalities to facilitate the handling of data.

Same as for NumPy, and Matplotlib, you need to install Pandas also using pip:

```
(.venv) $ pip install pandas
```

Back in the REPL:

```
# create a Series of integers with named rows
>>> import numpy as np
>>> import pandas as pd
>>> a = np.array([1, 2, 3])
>>> rownames = ["a", "b", "c"]
>>> s = pd.Series(a, index=rownames)
>>> s
a    1
b    2
c    3
dtype: int64
```

```
>>> # access data
>>> print(s.iloc[0])
1
>>> print(s["a"])
```

```
1
```

```
>>> # Dataframe
>>> import numpy as np
>>> import pandas as pd
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> rownames = ["a", "b"]
>>> colnames = ["one", "two", "three"]
>>> df = pd.DataFrame(a, index=rownames, columns=colnames)
>>> print(df)
   one  two  three
a    1    2      3
b    4    5      6
```

```
>>> # index data using col names
>>> df["one"]
a    1
b    4
Name: one, dtype: int64
>>> df.one
a    1
b    4
Name: one, dtype: int64
```

END OF DOCUMENT.