Introduction to Python

May 16, 2017

1 Introduction to Python

1.1 Basic interaction

Python can be used as a calculator since there is no need to declare types and most of the operations behave as expected (like the int division). The power operator is not ^ but **.

```
In [1]: 2 + 2

Out[1]: 4

In [2]: 3 / 2

Out[2]: 1.5

In [3]: 2 ** 8

Out[3]: 256
```

Variables can be defined freely and change type. There is a very handy print function (this is very different from Python2!). The format function can be used to customize the output. More at https://pyformat.info/

```
In [4]: a = 42
        b = 256
        z = 2 + 3j
        w = 5 - 6j
        print("I multiply", a, "and", b, "and I get", a * b)
        print("Compex numbers!", z + w)
        print("Real:", z.real)
        # Variables as objects (in Python everything is an object)
        print("Abs:", abs(z))
```

```
I multiply 42 and 256 and I get 10752
Compex numbers! (7-3j)
Real: 2.0
Abs: 3.605551275463989
```

```
In [5]: almost_pi = 3.14
        better_pi =  3.14159265358979323846264338327950288419716939937510
        c = 299792458
        print("Look at his scientific notation {:.2E} or ar this nice rounding {:.3f}".format(c
```

```
Look at his scientific notation 3.00E+08 or ar this nice rounding 3.142
```

Note that Python does not require semicolons to terminate an instruction (but they don't harm) but require the indendation to be respected. (After for, if, while, def, class, ...)

```
In [6]: for i in range(5):
            if (not i%2 == 0 or i == 0):
                print(i)
```

```
0
1
3
```

1.2 Strucutred Data

It's easy to work with variables of different nature. There are three kinds of structured variable: tuple (), lists [], and dicts {}. Tuples are immutable (ofter output of functions is given as a tuple). Lists are the usual arrays (multidimensional). Dictionaries are associative arrays with keywords.

```
In [9]: a = 5
        a = "Hello, World"
        # Multiple assignation
        b, c = "Hello", "World"
        print(a)
        print(b, c)

        tuple_example = (1,2,3)
        print("Tuple", tuple_example[0])
        # tuple_example[1] = 3

        list_example = [1,2,3]
        print("List 1", list_example[0])
        list_example[1] = 4
        print("List 2", list_example[1])

        dict_example = {'one' : 1,
                        'two' : 2,
                        'three' : 3
                       }
        print("Dict", dict_example['one'])
```

```
Hello, World
Hello World
Tuple 1
List 1 1
List 2 4
Dict 1
```

Lists are very useful as most of the methods are build it, like for sorting, reversing, inserting, deleting, slicing, ...

```
In [10]: random_numbers = [1,64,78,13,54,34, "Ravioli"]
         print("Length:", len(random_numbers))
         true_random = random_numbers[0:5]
         print("Sliced:", true_random)
         print("Sorted:", sorted(true_random))
         print("Max:", max(true_random))

         random_numbers.remove("Ravioli")
         print("Removed:", random_numbers)

         multi_list = ["A string", ["a", "list"], ("A", "Tuple"), 5]

         print("Concatenated list", random_numbers + multi_list)
```

```
Length: 7
Sliced: [1, 64, 78, 13, 54]
Sorted: [1, 13, 54, 64, 78]
Max: 78
Removed: [1, 64, 78, 13, 54, 34]
Concatenated list [1, 64, 78, 13, 54, 34, 'A string', ['a', 'list'], ('A', 'Tuple'), 5]
```

CAVEAT: List can be dangerous and have unexpected behavior due to the default copy method (like pointers pointing to the same area of memory)

```
In [11]: cool_numbers = [0, 11, 42]
         other_numbers = cool_numbers

         print(other_numbers)

         cool_numbers.append(300)

         print(other_numbers)
```

```
[0, 11, 42]
[0, 11, 42, 300]
```

To avoid this problem usually slicing is used.

```
In [13]: cool_numbers = [0, 11, 42]
         other_numbers = cool_numbers[:]

         print(other_numbers)

         cool_numbers.append(300)

         print(other_numbers)
```

```
[0, 11, 42]
[0, 11, 42]
```

String are considered list and slicing can be applied on strings, with a sleek behavior with respect to indeces:

```
In [14]: s = "GNU Emacs"
         # No problem with "wrong" index
         print(s[4:100])
         # Backwards!
         print(s[-9:-6])
```

```
Emacs
GNU
```

With a for loop it is possible to iterate over lists. (But attention not to modify the list over which for is iterating!)

```
In [15]: for num in cool_numbers:
             print("I like the number", num)
```

```
I like the number 0
I like the number 11
I like the number 42
I like the number 300
```

List can generate other list via *list comprehension* which is a functional way to operate on a list or a subset defined by if statements.

```
In [16]: numbers = [0, 1, 2, 3, 4, 5, 6, 7]

         # Numbers via list comprehension

         numbers = [i for i in range(0,8)]
         print("Numbers:", numbers)
```

```
even = [x for x in numbers if x%2 == 0]
odd = [x for x in numbers if not x in even]
print("Even:", even)
print("Odd:", odd)
```

```
Numbers: [0, 1, 2, 3, 4, 5, 6, 7]
Even: [0, 2, 4, 6]
Odd: [1, 3, 5, 7]
```

1.3 Functions

Python can have user-defined functions. There are some details about *passing by reference* or *passing by value* (what Python actually does is *passing by assignment*, details here: https://docs.python.org/3/faq/programming.html#how-do-i-write-a-function-with-output-parameters-call-by-reference). There are no return and arguments type but there is no overloading.

```
In [18]: def say_hello(to = "Gabriele"):
             print("Hello", to)

         say_hello()
         say_hello("Albert")


         def sum_and_difference(a, b):
             return (a + b, a - b)

         (sum, diff) = sum_and_difference(10, 15)
         print("Sum: {}, Diff: {}".format(sum, diff))

         def usless_box(a,b,c,d,e,f):
             return a,b,c,d,e,f

         first, _, _, _, _, _ = usless_box(100, 0, 1, 2, 3, 4)

         print(first)
```

```
Hello Gabriele
Hello Albert
Sum: 25, Diff: -5
100
```

A very useful construct is try-except that can be used to handle errors.

```
In [19]: hey = "String"
         ohi = 6
```

```
try:
    print(hey/3)
except:
    print("Error in hey!")

try:
    print(ohi/3)
except:
    print("Error in ohi!")
```

```
Error in hey!
2.0
```

NOTE: Prefer this name convenction (no CamelCase) and space over tabs

There is full support to OOP with Ineheritance, Encapsulation and Polymorphism. (https://docs.python.org/3/tutorial/classes.html)

1.4 Shipped with battery included

For Python there exist a huge number of *modules* that extend the potentiality of Python. Here are some examples:

1.4.1 OS

os is a module for interacting with the system and with files

```
In [20]: # Modules have to be imported
         # In this way I import thw whole module
         import os
         # To access an object inside the module I have to prepend the name

         # In this way I import only a function but I don't have to prepend the
         # module's name
         from os import getcwd

         print(os.getcwd())
         print(getcwd())
```

```
/home/sbozzolo/misc/Linux4Physics/seminar_3
/home/sbozzolo/misc/Linux4Physics/seminar_3
```

os with Python's capability for manipulating string is a very simple way to interact with files and dir

```
In [21]: dir = "test"
         files = os.listdir(dir)
```

```
print(files)

# Sorting
files.sort()
print(files)

# I take the subset starting with d and not ending with 10 and that are not directori

dfiles = [f for f in files if f.startswith("d") and not f.endswith("10") and not os.pa

print(dfiles)

for f in dfiles:
    data = f.split("_")
    n1 = data[1]
    n2 = data[2]
    print("From the name of the file {} I have extrected {} {}".format(f, n1, n2))
```

```
['d1_2_1', 'e2_4_2', 'e1_2_1', 'e0_0_0', 'd4_8_4', 'e4_8_4', 'e6_12_6', 'd5_10_5', 'e10_20_10'
['d0_0_0', 'd10_20_10', 'd1_2_1', 'd2_4_2', 'd3_6_3', 'd4_8_4', 'd5_10_5', 'd6_12_6', 'd7_14_7
['d0_0_0', 'd1_2_1', 'd2_4_2', 'd3_6_3', 'd4_8_4', 'd5_10_5', 'd6_12_6', 'd7_14_7', 'd8_16_8',
From the name of the file d0_0_0 I have extrected 0 0
From the name of the file d1_2_1 I have extrected 2 1
From the name of the file d2_4_2 I have extrected 4 2
From the name of the file d3_6_3 I have extrected 6 3
From the name of the file d4_8_4 I have extrected 8 4
From the name of the file d5_10_5 I have extrected 10 5
From the name of the file d6_12_6 I have extrected 12 6
From the name of the file d7_14_7 I have extrected 14 7
From the name of the file d8_16_8 I have extrected 16 8
From the name of the file d9_18_9 I have extrected 18 9
```

1.4.2 Sys (and argparse)

sys is another module for interactive with the system or to obtain information about it, in particular by means of the command line. argparse is a module for defining flags and arguments.

```
In [22]: import sys

         # sys provides the simplest way to pass command line arguments to a python script
         print(sys.argv[0])


         # argparse is more flexible but requires also more setup
```

```
/usr/lib/python3.6/site-packages/ipykernel_launcher.py
```

1.4.3 NumPy

Numpy is a module that provides a framework for numerical application. It defines new type of data highly optimized (NumPy is written in C) and provides simple interfaces for importing data from files and manipulate them. It is well integrated with the other scientific libraries for Python as it serves as base in many cases (SciPy, Matplotlib, Pandas, ...) Its fundamental object is the numpy array. With good (enough) documentation!

In [24]:
```python
# Standard import
import numpy as np

# Array from list
num = [0,1,2]
print("List:", num)

x = np.array(num)
print("Array:", x)

y = np.random.randint(3, size = (3))
print("Random", y)

z = np.array([x,y])
print("z:", z)
print("Shape", z.shape)
zres = z.reshape(3,2)
print("z reshaped:", zres)
# Attention: numpy does not alter any object!

# Operation behave well on arrays
y3 = y + 3
print("y + 3:", y3)
print("y squared:", y**2)

# Many built-in operations
print("Scalar product:", np.dot(x,y))

# Handy way to create an equispaced array
xx = np.linspace(0, 15, 16)
print("xx:", xx)

yy = np.array([x**2 for x in xx])
print("yy:", yy)

zz = yy.reshape(4,4)
print("zz", zz)
print("Eigenvalues:", np.linalg.eigvals(zz))
```

```
List: [0, 1, 2]
Array: [0 1 2]
```

```
Random [0 0 2]
z: [[0 1 2]
 [0 0 2]]
Shape (2, 3)
z reshaped: [[0 1]
 [2 0]
 [0 2]]
y + 3: [3 3 5]
y squared: [0 0 4]
Scalar product: 4
xx: [  0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.  11.  12.  13.  14.
  15.]
yy: [   0.    1.    4.    9.   16.   25.   36.   49.   64.   81.  100.  121.
  144.  169.  196.  225.]
zz [[   0.    1.    4.    9.]
 [  16.   25.   36.   49.]
 [  64.   81.  100.  121.]
 [ 144.  169.  196.  225.]]
Eigenvalues: [  3.65926730e+02  -1.75236436e+01   1.59691356e+00  -1.24290102e-14]
```

NumPy offers tools for: - Linear algebra - Logic functions - Datatypes - Constant of nature - Matematical functions (also special, as Hermite, Legendre...) - Polynomials - Statistics - Sorting, searching and counting - Fourier Transform - Random generation - Integration with C/C++ and Fortran code

In [25]:
```python
# Example: Polynomail x^2 + 2 x + 1
p = np.poly1d([1, 2, 1])
print(p)

# Evaluate it at 1
print("p(1):", p(1))

# Find the roots
print("Roots:", p.r)

# Take derivative
print("Deriv:", np.polyder(p))
```

```
   2
1 x + 2 x + 1
p(1): 4
Roots: [-1. -1.]
Deriv:
2 x + 2
```

Interaction with files is really simple

```
In [26]: arr = np.random.random(10)

         # Prints a single column file, for arrays print many columns
         np.savetxt("array.dat", arr)

         files = os.listdir(".")

         print([f for f in files if f == "array.dat"])

         data = np.loadtxt("array.dat")

         print(data)
```

```
['array.dat']
[ 0.45195533  0.67631487  0.65834425  0.79200876  0.70282038  0.36252459
  0.77412305  0.39734459  0.03441047  0.95842789]
```

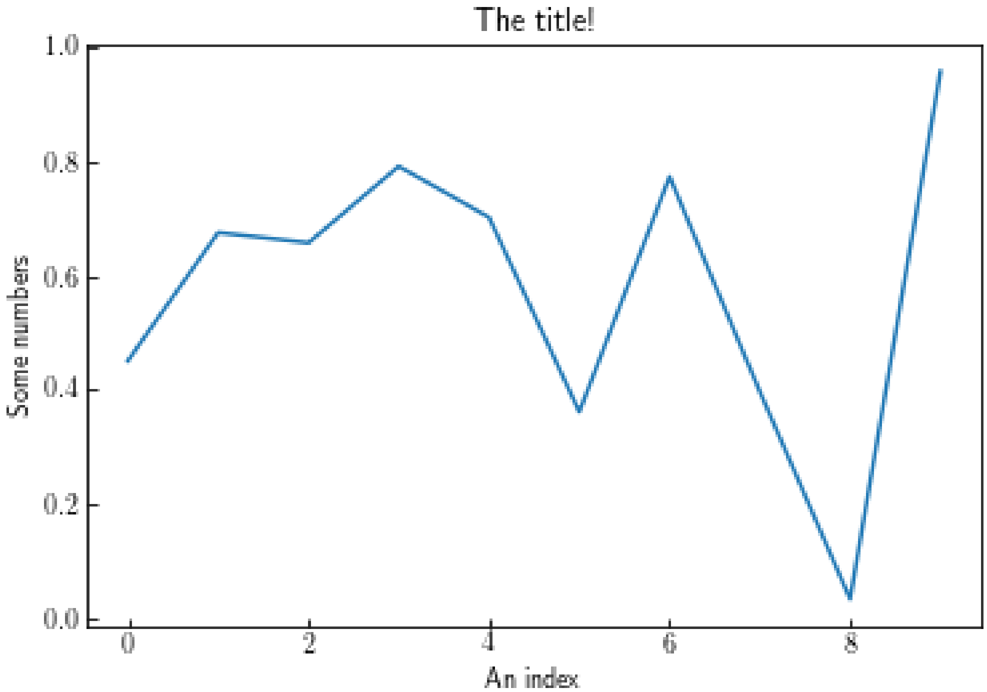
It is possible to save data compressed in a gzip by appending tar.gz to the name of the file (in this case array.dat.tar.gz).

REMEMBER: - To create: `tar cvzf archive.tar.gz folder` - To extract: `tar xvzf archive.tar.gz`

1.4.4 Matplotlib

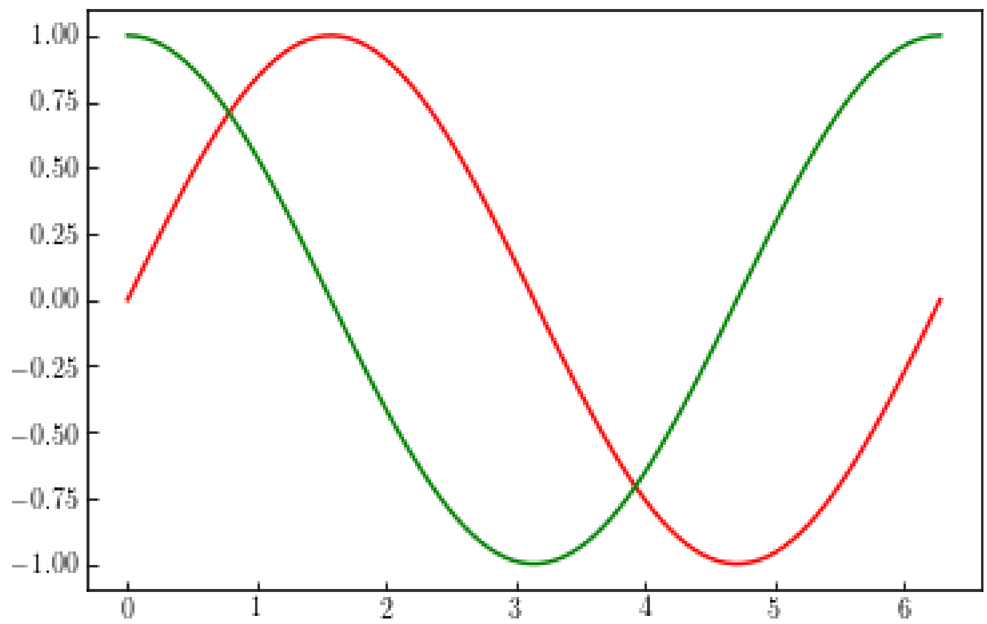
Matplotlib is the tool for plotting and graphics

```
In [27]: import matplotlib.pyplot as plt

         plt.plot(arr)
         plt.ylabel('Some numbers')
         plt.xlabel('An index')
         plt.title("The title!")
         plt.show()
```



Matplotlib has a seamless integration with NumPy

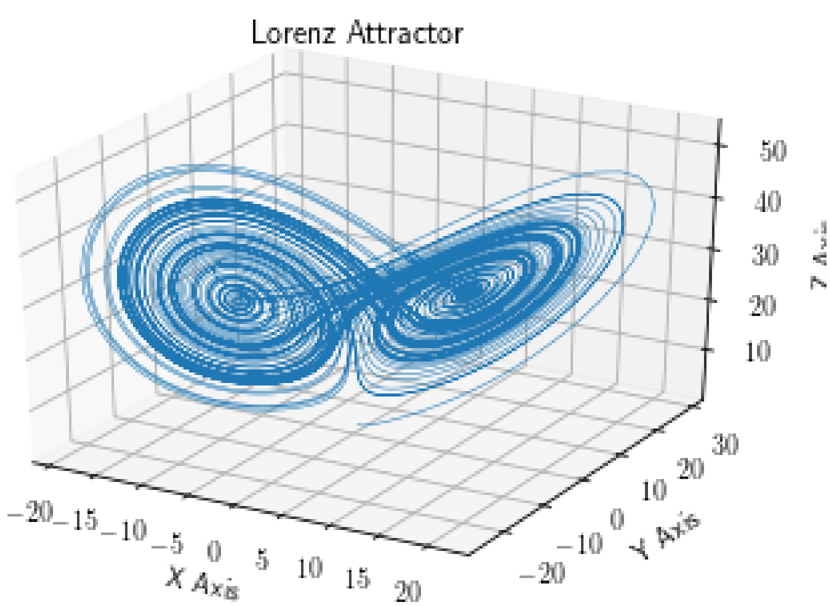
```
In [28]: x = np.linspace(0,2 * np.pi, 100)
         y = np.sin(x)
         z = np.cos(x)

         plt.plot(x, y, "r-", x, z, "g-")
         plt.show()
```



Matplotlib has a great library of examples (https://matplotlib.org/examples/) that in particular contains many of the most common plots (histograms, contour, scatter, pie, ...)

In [29]:
```
# Plot of the Lorenz Attractor based on Edward Lorenz's 1963 "Deterministic
# Nonperiodic Flow" publication.
# http://journals.ametsoc.org/doi/abs/10.1175/1520-0469%281963%29020%3C0130%3ADNF%3E2
#
# Note: Because this is a simple non-linear ODE, it would be more easily
#       done using SciPy's ode solver, but this approach depends only
#       upon NumPy.

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


def lorenz(x, y, z, s=10, r=28, b=2.667):
    x_dot = s*(y - x)
    y_dot = r*x - y - x*z
    z_dot = x*y - b*z
    return x_dot, y_dot, z_dot


dt = 0.01
stepCnt = 10000

# Need one more for the initial values
```

```python
xs = np.empty((stepCnt + 1,))
ys = np.empty((stepCnt + 1,))
zs = np.empty((stepCnt + 1,))

# Setting initial values
xs[0], ys[0], zs[0] = (0., 1., 1.05)

# Stepping through "time".
for i in range(stepCnt):
    # Derivatives of the X, Y, Z state
    x_dot, y_dot, z_dot = lorenz(xs[i], ys[i], zs[i])
    xs[i + 1] = xs[i] + (x_dot * dt)
    ys[i + 1] = ys[i] + (y_dot * dt)
    zs[i + 1] = zs[i] + (z_dot * dt)

fig = plt.figure()
ax = fig.gca(projection='3d')

ax.plot(xs, ys, zs, lw=0.5)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
ax.set_title("Lorenz Attractor")

plt.show()
```



1.4.5 SciPy

SciPy is a module that relies on NumPy and provides many ready-made tools used in science. Examples: - Optimization - Integration - Interpolation - Signal processing - Statistics

Example, minimize: $f(\mathbf{x}) = \sum_{i=1}^{N-1} 100\left(x_i - x_{i-1}^2\right)^2 + \left(1 - x_{i-1}\right)^2.$

```
In [1]: import numpy as np
        from scipy.optimize import minimize

        def rosen(x):
            """The Rosenbrock function"""
            return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)

        x0 = np.array([1.3, 0.7, 0.8, 1.9, 1.2])
        res = minimize(rosen, x0, method='nelder-mead', options={'xtol': 1e-8, 'disp': True})

        print(res.x)
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 339
         Function evaluations: 571
[ 1.  1.  1.  1.  1.]
```

2 A complete example -- Dice rolls

Scripted!