PYTHON - An introduction (towards applications in astrophysics)
By Sandra (march 2019)

```
################################################################################
# One of the most popular programming languages for general use. It is easy to interpret,
# it takes syntax from C, but not as fast as Fortran or similar. Wide number of applications
# which are public. It is Free!
#
# Other advantage of ipython: debagger, terminal interface, graphic environment, execute
# external script, list of commands saved in file ipython_log.py.
################################################################################
```

Documentations for Intro:

** Course on Data Analysis with Python for PhD Students, by Leonardo Primavera (in Italian):

http://www.fis.unical.it/astroplasmi/primavera/dottorato/dottorato.html

** Tutorial for beginners:

http://www.afterhoursprogramming.com/tutorial/Python/Introduction/

** Interactive tutorial (install Notebook first, than start using it locally):

http://jupyter.org

** Tutorial for plotting, VERY EASY & WELL DONE:

http://matplotlib.org/users/pyplot_tutorial.html

Documentations for Astro:

** Documentation for Astropy package for use in astronomy:

http://astropy.readthedocs.org/en/stable/index.html

```
        ** General advises:

http://python4astronomers.github.io/

        ** General advices and tutorials:

http://www.astropython.org

=====================================================================
        Open Python by typing from Unix terminal:

$ python                                  # the simplest way to access

$ ipython notebook                        # for interactive use of iPython on the web

$ ipython –pylab                          # preferred, version 2.7
$ ipython3 –pylab                         # latest version 3.7
In [1]: %pylab tk                         # Necessary to change graphic interphase

        To exit: Ctrl-D
        or: exit (if ipython is used)
        or: quit (if ipython is used)
---------------------------------------------------------------------
 Important libraries to import (NumPy, SciPy, Matplotlib, Astropy, Math):

In [1]: import numpy as np                # np short for numpy, for scientific computing
In [2]: import matplotlib.pyplot as plt # graphics interface, already imported if –pylab is used
In [3]: import scipy.stats as st          # stats scripts
In [4]: import math as m                  # library for math functions
In [5]: import astropy as astr            # import astropy for astronomy applications
---------------------------------------------------------------------
        HELP:

In [1]: dir(m)                            # lists all functions in math
```

```
In [2]: help(m)                          # lists all functions in math
In [3]: help()                           # general interactive help
help> quit
-----------------------------------------------
        Print list of commands used:

In [1]: history
In [2]: history -n                       # numbered list of commands
In [3]: history 3                        # it shows command #3
In [4]: %rerun 3                         # rerun command #3

        Search command already used:

In [5]:  Ctrl-r                          # type command to search
-----------------------------------------------
        Some examples of functions in Math:

In [1]: print (m.pi)                     # print p-greco
3.14159265359
In [2]: pi                               # print p-greco
3.14159265359


In [3]: print (m.sin(pi))                # should give you zero, (sin(pi)=0,
1.22464679915e-16                        # but returns number closest to zero

In [4]: sin(pi)                          # same as before
In [5]: np.sin(pi)                       # same as before
-----------------------------------------------
                Simple plotting functions:

In [1]: x = [0,1]                                  # it defines x range
In [2]: y=[10,30]                                  # it defines y range
In [3]: plot(x,y)                                  # plot line connecting x to y
In [4]: plt.plot(x,y)                              # same as before
```

```
In [5]: plt.plot([0,1],[10,30])                    # same as before
In [6]: plt.plot([0,1],[10,30],'--')               # same as before with dashed line
In [7]: plot([0,1],[10,20])                        # same as before
```

Plot in log scale (positive numbers):

```
In [8]: x = [10,100]                               # x range
In [9]: y=[10,30]                                  # y range
In [10]: plt.loglog(x, y, basex=10, basey=10)      # plots in log-scale base 10 for x & y
```

Labels for x-axis and y-axis, plus title on top of the plot:

```
In [11]: xlabel ('Temperature')
In [12]: ylabel ('Density')
In [13]: plt.title('This is my beautiful plot')
In [14]: plt.text(3, 5, '$\mu=100,\ \sigma=15$')   # plot labels inside the graph
In [15]: plt.grid(True)                                    # plot grid
In [16]: plt.axis([50,60,13,100])                          # plot limited portion of graph
```

Clear display:

```
In [16]: clf()
```

Read table, show values and plot:

```
In [1]: x,y=loadtxt('test.txt',unpack=True,usecols=(0,1))  # reads cols 1 & 2 test.txt
In [2]: print (x,y)                                        # print values
In [3]: x[3]                                               # print 4th element in array
In [4]: plot(x,y)                                          # connects points
In [5]: plot(x,y,'ro')                                     # red circles
In [6]: plot(x,sin(y))                                     # plot x vs. sin(y)
In [7]: plot(x,y,'g^')                                     # green triangles
In [8]: plot(x,y,'.')                                      # black points
In [9]: plot(x,y,'h')                                      # large hexagon
```

Plot with error bars:

```
In [10]: x,dx,y,dy=loadtxt('test2.txt',unpack=True,usecols=(0,1,2,3))
In [11]: plt.errorbar(x,y,dy,dx,'go')                                    # to note the order of array
In [11]: plt.errorbar(x,y,dy,1,'r^')                                     # error bar of x values is
fixed to 1
In [12]: plt.errorbar(x,y, xerr=dx, yerr=dy, fmt='b^', ecolor='y')       # same but errorbar and
symbol have different colors
In [13]: plt.errorbar(x,y,dy,None,'o')                                   # error bars only for x
points
```

Print vector to a file:

```
In [1]: x=arange(10,20,0.1)              # creates vector
In [2]: x.tofile("myfile",sep=' ')       # elements written in file separated by 1 space
In [3]: x.tofile("myfile",sep='\n')      # elements separated by one line
```

Create random numbers:

```
In [1]: r=randn(100)                     # 1D array r with 100 random numbers
In [2]: r=randn(3,4)                     # 2D table r with 3 raws and 4 columns
In [3]: r[0,0]                           # print first element (first raw, first column)
In [4]: r[2,1]                           # print element (3,2)
In [5]: r[2][1]                          # same as before

In [6]: a=np.random.randn(5)             # 5 random values
In [7]: b=np.random.randn(2,3)           # matrix 2x3 random values
In [8]: print(b)                         # print matrix
In [9]: b[1,2]                           # print last element of matrix above
```

Plot histogram from table:

```
In [1]: x,y=loadtxt('test.txt',unpack=True,usecols=(0,1))  # read file
In [2]: width = 0.5                                        # width of the bin is 0.5
```

```
In [3]: plt.bar(x,y, width, color='r')                    # histogram x,y is red
In [4]: plt.bar((x+1)/2, y, width*2, color='b')           # change x,width and color to blue

In [5]: hist(y)                                           # plot histogram of column y
```

Alternative plot histogram (from random numbers)

```
In [1]: mu, sigma = 100, 15
In [2]: x = mu + sigma * np.random.randn(10000)                    # creates array with 1000 random
numbers
In [3]: n, bins, patches = plt.hist(x, 50, facecolor='g')  # histogram of the data in 50 bins (in green)
```

Polynomial fit of x,y of any degree:

```
In [1]: x,y=loadtxt('test.txt', unpack=True,usecols=(0,1))
In [2]: plot(x,y,'o')                                     # plot numbers
In [3]: polyfit(x,y,1)                                    # degree=1, output 2 coefficients
Out[4]: array([ 0.42848485, -0.14666667])         # output coefficients

In [5]: plot(x,0.42848485*x-0.14666667,'-')               # plot line with coefficients

In [6]: polyfit(x,y,2)                                    # degree=2, output 3 coefficients
Out[7]: array([-0.02689394,  0.72431818, -0.73833333])

In [8]: plot(x,-0.02689394*x**2+0.72431818*x-0.73833333,'r--')     # plot 2-degree line

In [9]: coeffs = polyfit(x,y,2)                           # print coefficients of 2-degree line in array
In [10]: plot(x,x*x*coeffs[0]+x*coeffs[1]+coeffs[2])      # plot 2-degree fit
```

Calculates coefficients, writes in array and plots:

```
In [10]: coeffs = numpy.polyfit(x,y,2)
In [11]: plot(x,y,'o')
In [12]: plot(x,coeffs[0]*x**2+coeffs[1]*x+coeffs[2],'b-')
```

```
In [13]: corr = numpy.corrcoef(x,y)[0,1]  # create matrix with correlation coefficients and print value
in (0,1)
In [14]: print (corr)                              # print significance of correlation
```

Sum elements in array or matrix:

```
In [1]: x,y=loadtxt('test.txt',unpack=True,usecols=(0,1))
In [2]: sum = np.sum(y)                            # sum elements in array y
In [3]: print (sum)                                # print result

In [4]: data = numpy.loadtxt('test.txt')
In [5]: sum = np.sum(data[:,0])                    # sum first column
In [6]: sum = np.sum(data[:,1])                    # sum second column
```

To save plot in display in PDF or PS file:

```
In [7]: plt.savefig('figure.pdf')
In [8]: plt.savefig('figure.ps')
```

Display figure in foreground:

```
In [1]: plt.show()
```

Create new display for another plot:

```
In [2]: plt.figure("This is my new figure")
```

To execute a script file within Python:

```
In [1]: execfile("test.py")                                  # for example test.py
```

If using version of Python 3, execfile doesn't work, then:

```
In [2]: exec(compile(open('test.py').read(), 'test.py','exec'))
```

```
        Or, also:

In [3]: import runpy
In [4]: file_globals = runpy.run_path(“test.py")

        Finally, also (faster):

In [5]: exec(open('test.py').read())

        Or from outside Python:

$ python test.py
-----------------------------------------------
        Save list of used commands in a file (called ipython_log.py):

In [1]: logstart                 # starts writing log file (file name optional)
In [2]: logstop                  # stops writing log file
In [3]: logstart ?               # all possible options
In [4]: logstart -o              # add output which is added in the same log file
-----------------------------------------------------
        Print comment on the screen (‘ ‘ or “ “ are almost the same):

In [1]: print (‘Good morning beautiful lady')
In [2]: s1 = ‘ciao’
In [3]: print (s1)
ciao

In [4]: print (s1*3)                                    # prints 3 times “ciao”
ciaociaociao

In [5]: s2 = bella
In [6]: print (s1+s2)
ciaobella

In [7]: s = ‘ciao\tbella’
```

```
In [8]: print (s)
ciao            bella

NOTE: if Python 3 is used, brackets must be used: print (s)
-----------------------------------------------------
                        Working with lists:

In [1]: l=[1,2,3],'ciao'
In [2]: print (l)
([1, 2, 3], 'ciao')

In [3]: print (l[0])                    # print first list
[1, 2, 3]

In [4]: print (l[1])
ciao                                    # print second list

In [5]: l = [1,2,3],'ciao',’bella’      # print second element of first list:
In [6]: print (l[0][1])
2

In [7]:  print (l[2])
bella

                        Lists can be modified:

In [8]: l=1,2,3,4,5
In [9]: print (l[2:])
(3, 4, 5)

In [10]: l = [1,2,3,4,5]          # it doesn’t work if you omit [ ]
In [11]: l[2]=100
In [12]: print (l)
[1, 2, 100, 4, 5]
-----------------------------------------------------
```

To create an array:

```
In [1]: x=arange(10)              # integer numbers from 0 to 9
In [2]: x=arange(10,20)           # 10 integers starting from 10 to 19, steps of 1
In [3]: x=arange(10,20,3)         # array starting from 10 in steps of 3, ending before 20
In [4]: x=arange(1,10,0.1)        # array starting from 1 in steps of 0.1, ending at 10
```

Variable types: integer, string, floating…:

```
In [1]: a = pi
In [2]: type (a)
Out[2]: float

In [3]: a=int(pi)
In [4]: print (a)
3

In [5]: f = float(a)
In [6]: print (f)
3.0

In [7]: name = 'Anna'
In [8]: type (name)
Out: str

In [9]: a = 0.5+2j
In [10]: print (a)
(0.5+2j)                  # print real and complex numbers

In [11]: type(a)
Out: complex

In [12]: s=10
In [13]: type(s)
Out: int
```

```
In [14]: s=str(10)
In [15]: type(s)
Out: str                 # assign s the type string, although it is a number
```

Reading ASCII file and printing values:

```
In [1]: import astropy.io.ascii as ascii
In [2]: ascii.read('test.txt')
Out[2]:
<Table length=8>
  col1   col2
float64 int64
------- -----
    1.0     2
    2.0     3
    2.0     5
    3.5     5
    5.0     6
    7.4    12
    9.0    13
   12.0    15
```

Or (similar to before):

```
In [3]: import numpy as np

In [4]: data = np.loadtxt('test.txt', unpack = True)       # to unpack the table into different arrays
In [5]: print (data[:,1])                                  # show second raw

In [6]: data = np.loadtxt('test.txt')
In [7]: print (data[:,1])                                  # show second row (different from before)
In [8]: print (data[2,:])                                  # read third column
```

Reading and printing numbers from a string:

```
In [1]: i = 2
In [2]: x = sqrt(i)
In [3]: string = “These are your numbers %d %f” % (i,x)    # operator % connects the 2 parts
In [4]: print (string)
These are your numbers 2 1.414214
=========================================================
                OPERATORS IF, ELSE, ELIF, FOR, WHILE
---------------------------------------------------------
                IF, ELIF, ELSE   (it doesn’t have to conclude with else or elif):

In [1]: x = 100**.5
In [2]: if x == 10.0:
...     print ("x=", x)
... else:
...    print "This is not 10"
...
x = 10.0

In [3]: x = sqrt(2)
In [4]: if x == 10:
    print ("x = “, x)
elif x <= 10:
    print ("x <= 10")
else:
    print ("x > 10")
   .....:
x <= 10
---------------------------------------------------------
                FOR:

In [1]: for var in range(5):
...    print (var)
...
0
```

```
1
2
3
4

In [2]: l=1,2,3,'ciao'
In [3]: for i in l:
...    print (i*2)
...
2
4
6
ciaociao

In [4]: for i in range(1,10,2):           # start, end, step (integer is expected)
    print (i)                             # print 5 numbers increment by 2
   ....:
1
3
5
7
9

In [5]: l1 = 1,2,3
In [6]: l2 = "c1","c2","c3"
In [7]: for a,b in zip(l1,l2):            # operator “zip” concats l1 and l2
...     print (a,b)
...
1 c1
2 c2
3 c3
--------------------------------------------------
                WHILE:

In [1]: x = 1                             # initialise value of x
```

```
In [2]: while x<10:
...   print ("The count is:",x)
...   x = x+1                              # increment by 1
...
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
The count is: 9
The count is: 10
```