Python, Numpy, Scipy, Matplotlib, Pandas and More…

https://www.youtube.com/watch?v=GeKzBQnAq5I

Cecilia Jarne
cecilia.jarne@unq.edu.ar
Twitter: @ceciliajarne

https://www.youtube.com/watch?v=JJBp6UBcG64

It is an interpreted programming language that allows dynamic typing and is multiplatform.

https://www.python.org

Different programming paradigms can be used.

- Supports object orientation.
- Imperative programming.
- Functional programming.
- It has a clear syntax.
- Its functionality can be extended through different libraries.

- Easy to learn.
- A huge set of libraries.
- Excellent scientific support!!
- You can develop software pretty quickly.
- It has an open source license.
- A huge development community that you can really count on.

Zen of Python, by Tim Peters

- Pretty is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- If the implementation is hard to explain, it's a bad idea.

https://www.python.org/dev/peps/pep-0020/

Visual Studio Code

A quick start: the use of the Interpreter

To start (very quick!):

Install the Python interpreter on your computer.

(Linux distributions also frequently include Python and it is upgraded.)

Call interpreter:

```
python
```

It has an interactive mode to introduce instructions one by one and see the result:

```
>>> 1 + 1
2
>>> a = range(10)
>>> print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Also, we can use Google Colab for learning

https://colab.research.google.com/?utm_source=scs-index

Google colab

Or we can install a virtual environment locally

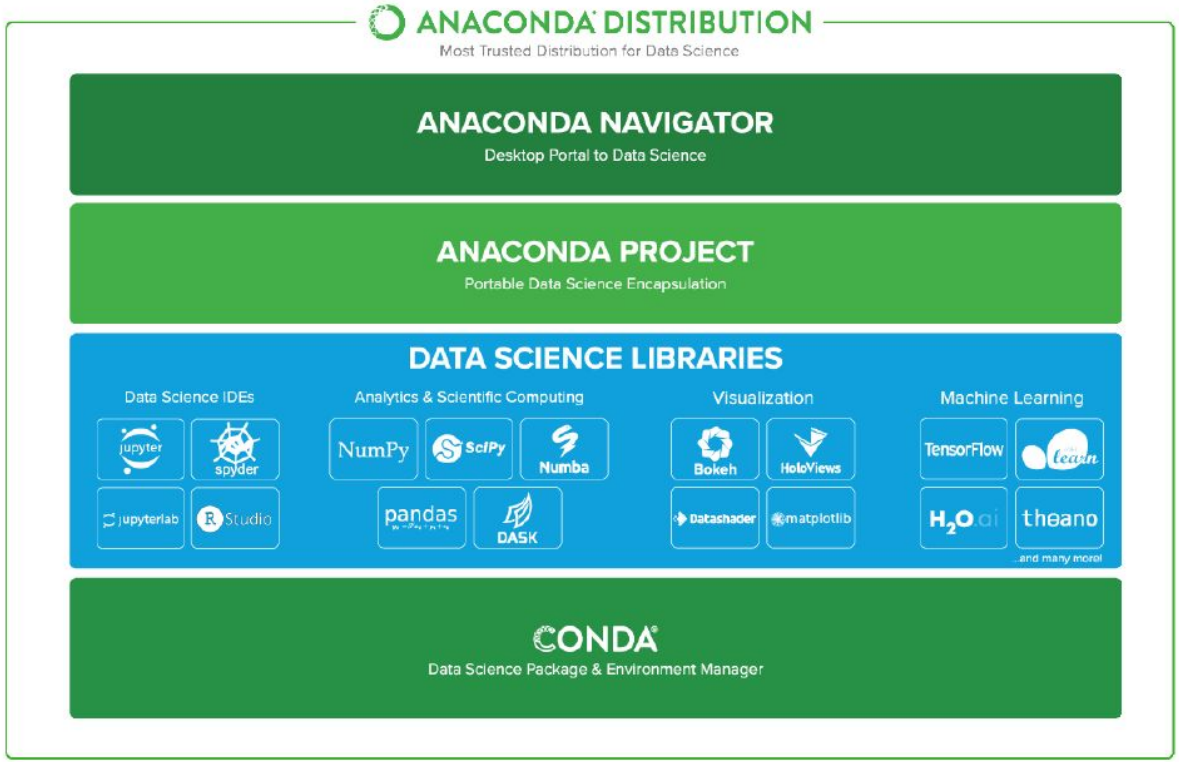
Virtual env

https://docs.python.org/3/library/venv.html

https://realpython.com/python-virtual-environments-a-primer/

Anaconda is a nice solution too:

https://www.anaconda.com/products/distribution




START

Basic ideas to start with python:

In a statically typed language, every variable name is bound:

- to a type (at compile time, by means of a data declaration).
- to an object.

In a dynamically typed language, every variable name is (unless it is null) is bound only to an object.

```
x = 1
x = "text" # dynamic typing :)
```

Data types:

https://docs.python.org/2/library/types.html

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Differences in the Python Syntax

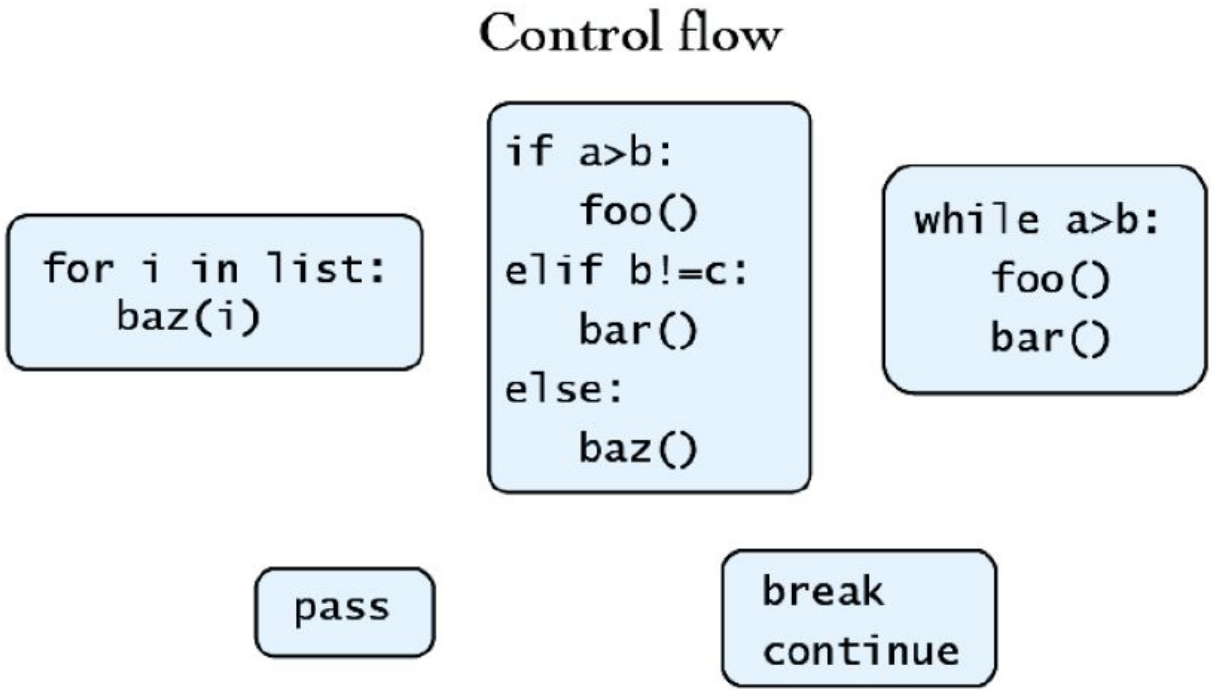
In C or C++: use of ; at the end of statements

```
if(a>b)
    foo();
    bar();
baz();
```

In python: indentation level of your statements is significant! Last statement is executed out of the conditional:

```
if(a>b):
    foo()
    bar()
baz()
```

Python Syntax: control flow instructions



Python Syntax

Function definition:

```
def function(x,y,z):
    x=3*y
    return x+y-z
```

Python Boolean and Math operators

Name	Function	symbol
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20

Operation	Result
x or y	if x is false, then y, else x
x and y	if x is false, then x, else y
not x	if x is false, then True, else False

Strings

It is possible to use ' or “ on string definition:

```
x= "this is a string"

y='this is also a string'
```

To print on screen we use the print function:

```
print(x)
```

Lists

- Very useful structure on python.

```
a=[1,'banana',1.2]
```

- Allow Index and slice access.

```
a[0] a[1] a[2:5] a[2:10:2]
```

- Comprehension list:

```
b=[x**2 for x in range(1,11)]
print(b)
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Some List Methods

```
list.append(x)

list.extend(iterable)

list.insert(i, x)

list.remove(x)

list.pop([i])

list.clear()

list.index(x[, start[, end]])

list.count(x)

list.sort(*, key=None, reverse=False)

list.reverse()

list.copy()
```

Three nice reason to use python in scientific programing

NumPy:

http://www.numpy.org/

SciPy:

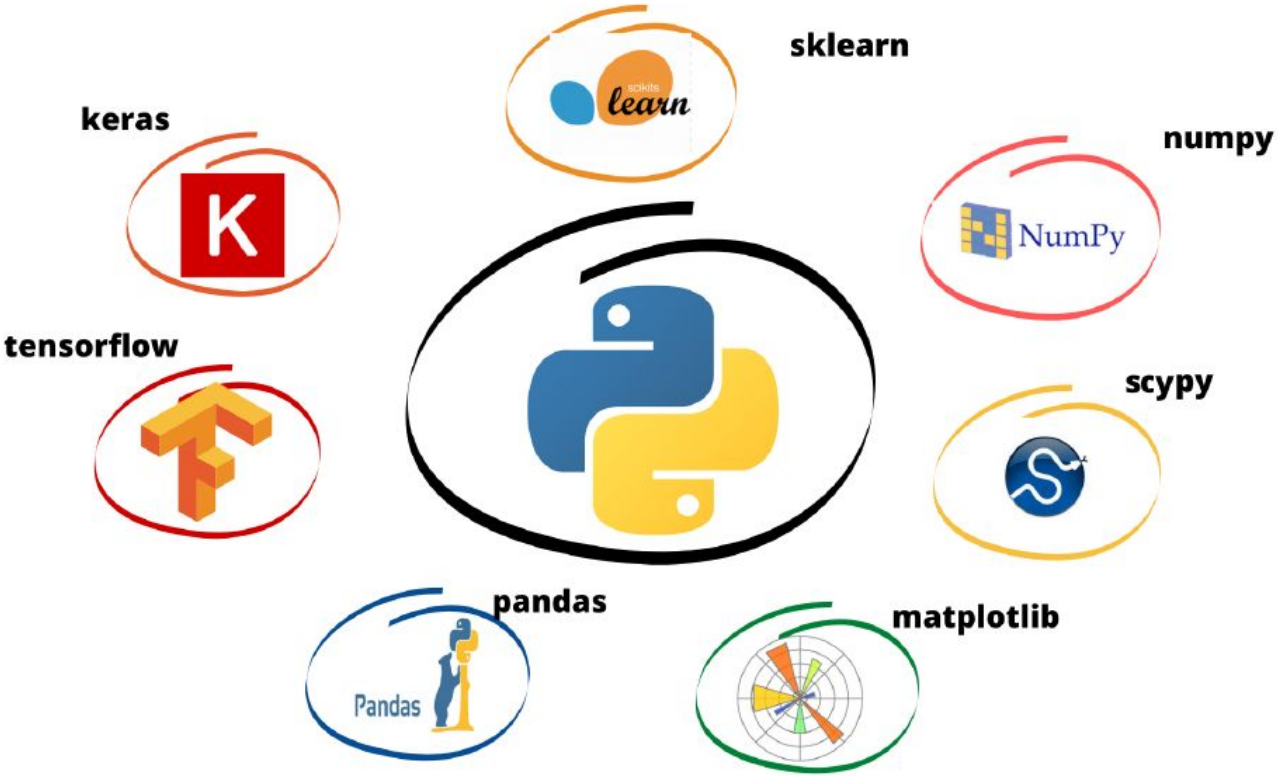
http://www.scipy.org/

MatPlotLib:

http://matplotlib.org/

All are open source!

And more


sklearn
keras
numpy
tensorflow
scypy
pandas
matplotlib
Pandas

Nice reason to use python in scientific programing

NumPy provides functionality to create, delete, manage and operate on large arrays of type raw" data (like Fortran and C/C++ arrays).

SciPy extends NumPy with a collection of useful algorithms like minimization, Fourier transforms, regression and many other applied mathematical techniques.

Both packages are add-on packages (not part of the Python standard library) containing Python code and compiled with (fftpack, BLAS).

MatPlotLib is a nice library to plot (but there are others as Seaborn or Bokeh).

How to import libraries?

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as pp
import matplotlib.pyplot as plt
```

Numpy

NumPy is the fundamental package for scientific computing in Python.

- A powerful N-dimensional array object.
- Sophisticated (broadcasting) functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities.
- Operations on matrices and vectors in NumPy are very efficient because they are linked to compiled in BLAS/LAPACK code.

NumPy functionality:

- Polynomial mathematics.
- Statistical computations.
- Pseudo random number generators.
- Discrete Fourier transforms.
- Size / shape / type testing of arrays.

Fundamental thing from Numpy: np.array

There are 5 general mechanisms for creating arrays:

- Conversion from other Python structures (e.g., lists, tuples).
- Intrinsic numpy array creation objects (e.g., arrange, ones, zeros, etc.)
- Reading arrays from disk, either from standard or custom formats.
- Creating arrays from raw bytes through the use of strings or buffers.
- Use of special library functions (e.g., random).

How important is Numpy?

Nature | Vol 585 | 17 September 2020 | 357

https://doi.org/10.1038/s41586-020-2649-2

Review

Array programming with NumPy

https://doi.org/10.1038/s41586-020-2649-2

Received: 21 February 2020

Accepted: 17 June 2020

Published online: 16 September 2020

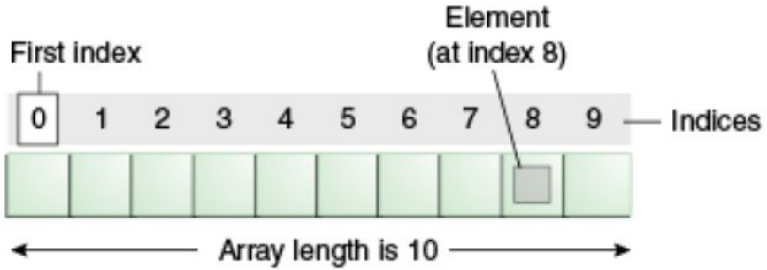
Open access

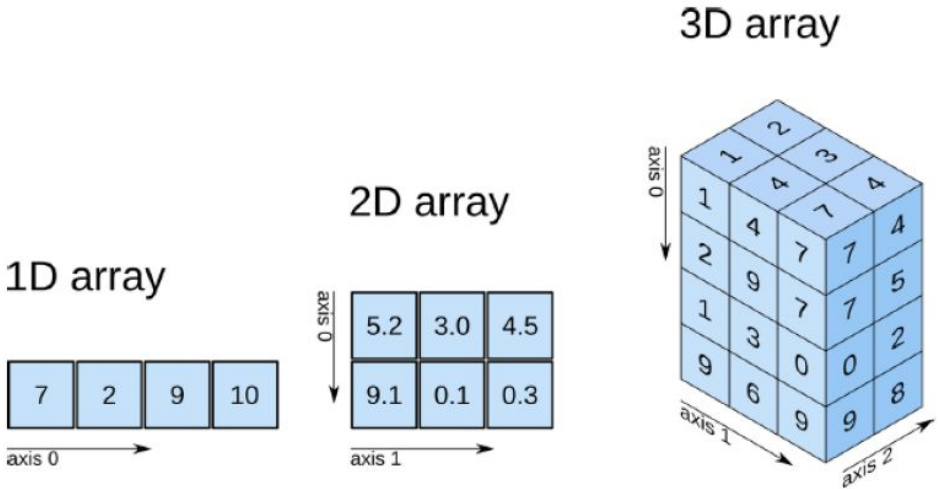
Check for updates

Charles R. Harris[1], K. Jarrod Millman[2,3,4✉], Stéfan J. van der Walt[2,4,5✉], Ralf Gommers[6✉], Pauli Virtanen[7,8], David Cournapeau[9], Eric Wieser[10], Julian Taylor[11], Sebastian Berg[4], Nathaniel J. Smith[12], Robert Kern[13], Matti Picus[4], Stephan Hoyer[14], Marten H. van Kerkwijk[15], Matthew Brett[2,16], Allan Haldane[17], Jaime Fernández del Río[18], Mark Wiebe[19,20], Pearu Peterson[6,21,22], Pierre Gérard-Marchant[23,24], Kevin Sheppard[25], Tyler Reddy[26], Warren Weckesser[4], Hameer Abbasi[6], Christoph Gohlke[27] & Travis E. Oliphant[6]

Array programming provides a powerful, compact and expressive syntax for accessing, manipulating and operating on data in vectors, matrices and higher-dimensional arrays. NumPy is the primary array programming library for the Python language. It has an essential role in research analysis pipelines in fields as diverse as physics, chemistry, astronomy, geoscience, biology, psychology, materials science, engineering, finance and economics. For example, in astronomy, NumPy was an important part of the software stack used in the discovery of gravitational waves[1] and in the first imaging of a black hole[2]. Here we review how a few fundamental array concepts lead to a simple and powerful programming paradigm for organizing, exploring and analysing scientific data. NumPy is the foundation upon which the scientific Python ecosystem is constructed. It is so pervasive that several projects, targeting audiences with specialized needs, have developed their own NumPy-like interfaces and array objects. Owing to its central position in the ecosystem, NumPy increasingly acts as an interoperability layer between such array computation libraries and, together with its application programming interface (API), provides a flexible framework to support the next decade of scientific and industrial analysis.

How is an Array?





Examples of how to create arrays


```
x = np.array([2, 3, 1, 0])
print(x)
[2 3 1 0]
```

```
np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.arange(2, 10, dtype=np.float)
array([2., 3., 4., 5., 6., 7., 8., 9.])
```

```
np.arange(2, 3, 0.1)
array([2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])
```

Special Functions


```
a = np.zeros((5, 2))
print(a)
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]
b = a.reshape((2, 5))
print(b)
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

Linear Algebra Operations: Transposition


```
x = np.array([[1.,2.,3.,4.],[1.,2.,3.,4.]])
print(x)
[[1. 2. 3. 4.]
 [1. 2. 3. 4.]]
b= x.T
print(b)
[[1. 1.]
 [2. 2.]
 [3. 3.]
 [4. 4.]]
```

Simple Linear Algebra Operations


```
x=np.array([1, 2,3])
y=np.array([11, -2,5])
print(x + y)
print(np.add(x, y))
[12   0   8]

[12   0   8]

#---------------

print(x - y)
print(np.subtract(x, y))
[-10    4   -2]
[-10    4   -2]
```

Simple Linear Algebra Operations


```
v=np.array([-1, 2,5])
w=np.array([1, 2,3])
print(v.dot(w))
print(np.dot(v, w))
18

18

----------

v1=np.array([[-1, 2,5],[1, 3,5]])
w1=np.array([1,10,3])
print(v1.dot(w1))
[34 46]
print(np.cross(v,w))
[-4  8 -4]
```

Opening a txt or cvs file


```
import numpy as np

file_name_you_want = np.loadtxt(fname,delimiter=" ")

print "First column element: ",file_name_you_want[0]
#to get the full column:

Transpose_your_file = file_name_you_want.T

print "First column: ", Transpose_your_file[0]
```

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

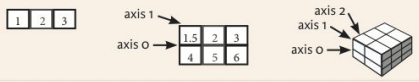
NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                  Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)  Create an array of ones
>>> d = np.arange(10,25,5)           Create an array of evenly
                                     spaced values (step value)
>>> np.linspace(0,2,9)               Create an array of evenly
                                     spaced values (number of samples)
>>> e = np.full((2,2),7)             Create a constant array
>>> f = np.eye(2)                    Create a 2X2 identity matrix
>>> np.random.random((2,2))          Create an array with random values
>>> np.empty((3,2))                  Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64       Signed 64-bit integer types
>>> np.float32     Standard double-precision floating point
>>> np.complex     Complex numbers represented by 128 floats
>>> np.bool        Boolean type storing TRUE and FALSE values
>>> np.object      Python object type
>>> np.string_     Fixed-length string type
>>> np.unicode_    Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape             Array dimensions
>>> len(a)              Length of array
>>> b.ndim              Number of array dimensions
>>> e.size              Number of array elements
>>> b.dtype             Data type of array elements
>>> b.dtype.name        Name of data type
>>> b.astype(int)       Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                        Subtraction
 array([[-0.5,  0. ,  0. ],
        [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                 Subtraction
>>> b + a                            Addition
 array([[ 2.5,  4. ,  6. ],
        [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                      Addition
>>> a / b                            Division
 array([[ 0.66666667,  1.        ,  1.        ],
        [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                   Division
>>> a * b                            Multiplication
 array([[  1.5,   4. ,   9. ],
        [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                 Multiplication
>>> np.exp(b)                        Exponentiation
>>> np.sqrt(b)                       Square root
>>> np.sin(a)                        Print sines of an array
>>> np.cos(b)                        Element-wise cosine
>>> np.log(a)                        Element-wise natural logarithm
>>> e.dot(f)                         Dot product
 array([[ 7.,  7.],
        [ 7.,  7.]])
```

Comparison

```
>>> a == b                           Element-wise comparison
 array([[False,  True,  True],
        [False, False, False]], dtype=bool)
>>> a < 2                            Element-wise comparison
 array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)             Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                          Array-wise sum
>>> a.min()                          Array-wise minimum value
>>> b.max(axis=0)                    Maximum value of an array row
>>> b.cumsum(axis=1)                 Cumulative sum of the elements
>>> a.mean()                         Mean
>>> b.median()                       Median
>>> a.corrcoef()                     Correlation coefficient
>>> np.std(b)                        Standard deviation
```

Copying Arrays

```
>>> h = a.view()                     Create a view of the array with the same data
>>> np.copy(a)                       Create a copy of the array
>>> h = a.copy()                     Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()                         Sort an array
>>> c.sort(axis=0)                   Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                             Select the element at the 2nd index
3
>>> b[1,2]                           Select the element at row 0 column 2
6.0                                  (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]                           Select items at index 0 and 1
array([1, 2])
>>> b[0:2,1]                         Select items at rows 0 and 1 in column 1
array([ 2.,  5.])
>>> b[:1]                            Select all items at row 0
array([[1.5, 2., 3.]])               (equivalent to b[0:1, :])
>>> c[1,...]                         Same as [1,:,:]
 array([[[ 3.,  2.,  1.],
         [ 4.,  5.,  6.]]])
>>> a[ : :-1]                        Reversed array a
array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                           Select elements from a less than 2
array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]    Select elements (1,0),(0,1),(1,2) and (0,0)
array([ 4. ,  2. ,  6. ,  1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]    Select a subset of the matrix's rows
array([[ 4. ,  5. ,  6. ,  4. ],     and columns
       [ 1.5,  2. ,  3. ,  1.5],
       [ 4. ,  5. ,  6. ,  4. ],
       [ 1.5,  2. ,  3. ,  1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)              Permute array dimensions
>>> i.T                              Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                        Flatten the array
>>> g.reshape(3,-2)                  Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))                  Return a new array with shape (2,6)
>>> np.append(h,g)                   Append items to an array
>>> np.insert(a, 1, 5)               Insert items in an array
>>> np.delete(a,[1])                 Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)     Concatenate arrays
 array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                 Stack arrays vertically (row-wise)
 array([[ 1. ,  2. ,  3. ],
        [ 1.5,  2. ,  3. ],
        [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                       Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                 Stack arrays horizontally (column-wise)
 array([[ 7.,  7.,  1.,  0.],
        [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))           Create stacked column-wise arrays
 array([[ 1, 10],
        [ 2, 15],
        [ 3, 20]])
>>> np.c_[a,d]                       Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                   Split the array horizontally at the 3rd
 [array([1]),array([2]),array([3])]  index
>>> np.vsplit(c,2)                   Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

DataCamp
Learn Python for Data Science Interactively

SciPy

One of the core packages. SciPy is built on top of NumPy and implements many specialized scientific computation tools:

- Manly user-friendly.
- Efficient numerical routines such as routines for numerical integration and optimization.
- Clustering.
- Fourier transforms.
- Numerical integration, interpolations. data I/O, LAPACK.

Also:

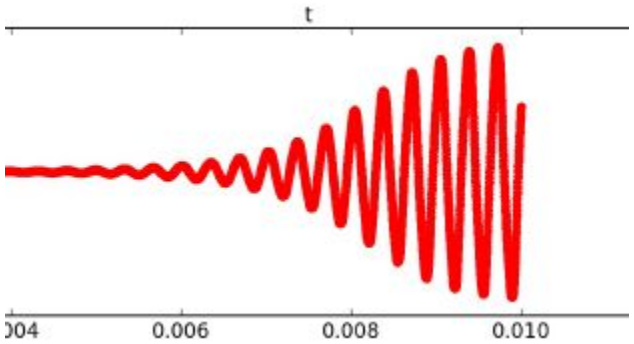
- sparse matrices, linear solvers, optimization.

- signal processing.

- statistical functions

Nice for numerical integration of equations

```
from scipy.integrate import odeint
dy/dt = func(y, t0, ...)
sol = odeint(func, X0, t)
```



https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

Or for example for a simple fit

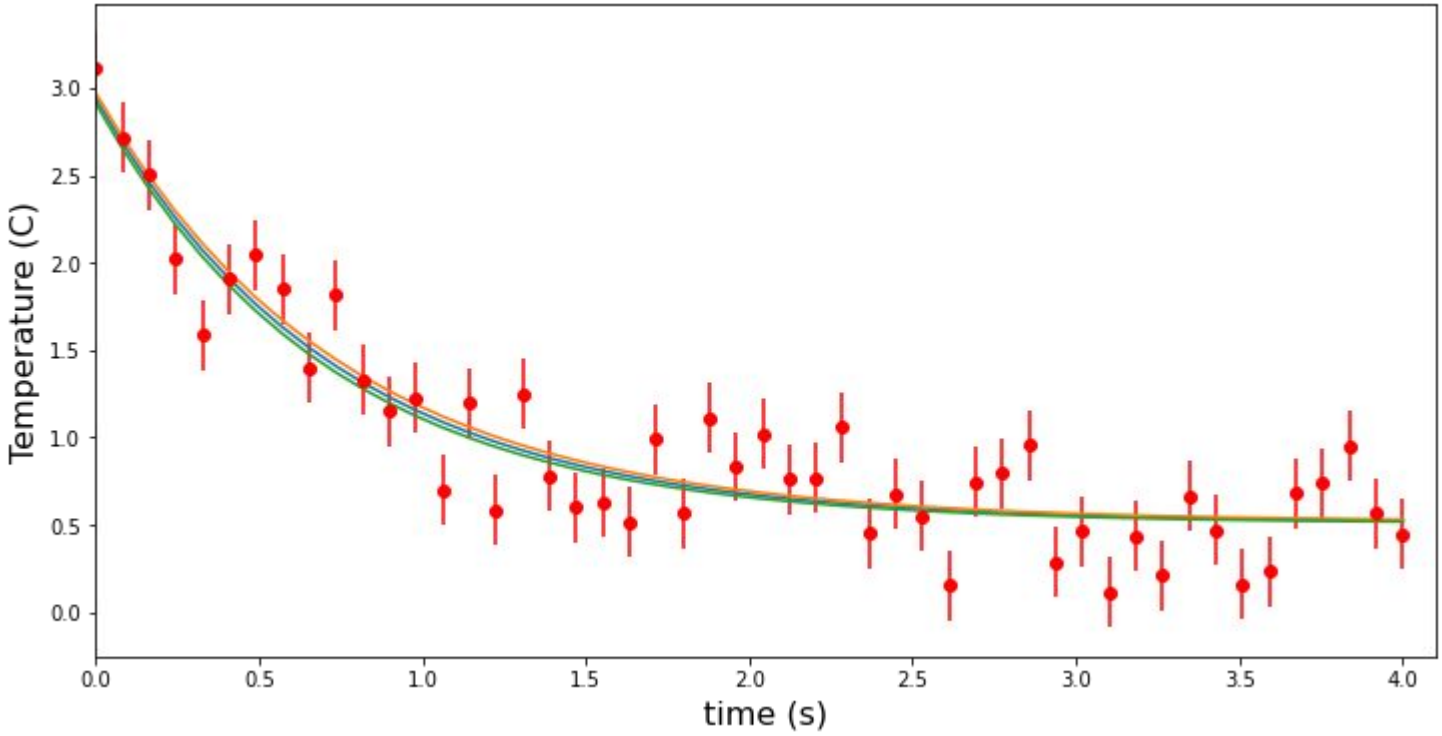
```
import numpy as np
import matplotlib.pyplot as pp
from scipy.optimize import curve_fit

def fitFunc(t, a, b, c):
   return a*np.exp(-b*t) + c

t = np.linspace(0,4,50)
temp = fitFunc(t, 2.5, 1.3, 0.5)
noisy = temp + 0.25*np.random.normal(size=len(temp))
fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)

pp.figure(figsize=(12, 6))
pp.ylabel('Temperature (C)', fontsize = 16)
pp.xlabel('time (s)', fontsize = 16)
pp.xlim(0,4.1)
pp.errorbar(t, noisy, fmt ='ro', yerr = 0.2)
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
pp.savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
pp.show()
```

And the result!



With the Fit parameters

[2.595658 1.74438726 0.69809511]

And covariance matrix:

[[0.02506636 0.01490486 -0.00068609]

2 [0.01490486 0.04178044 0.00641246]

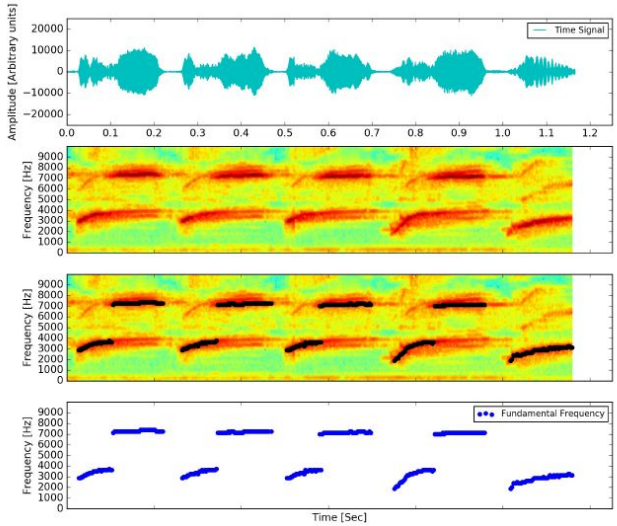
3 [-0.00068609 0.00641246 0.00257799]]

Also possible to apply for signal analysis and time series

https://doi.org/10.31527/analesafa.2018.29.2.51

https://doi.org/10.1016/j.mex.2018.12.011

https://doi.org/10.31527/analesafa.2019.30.3.68

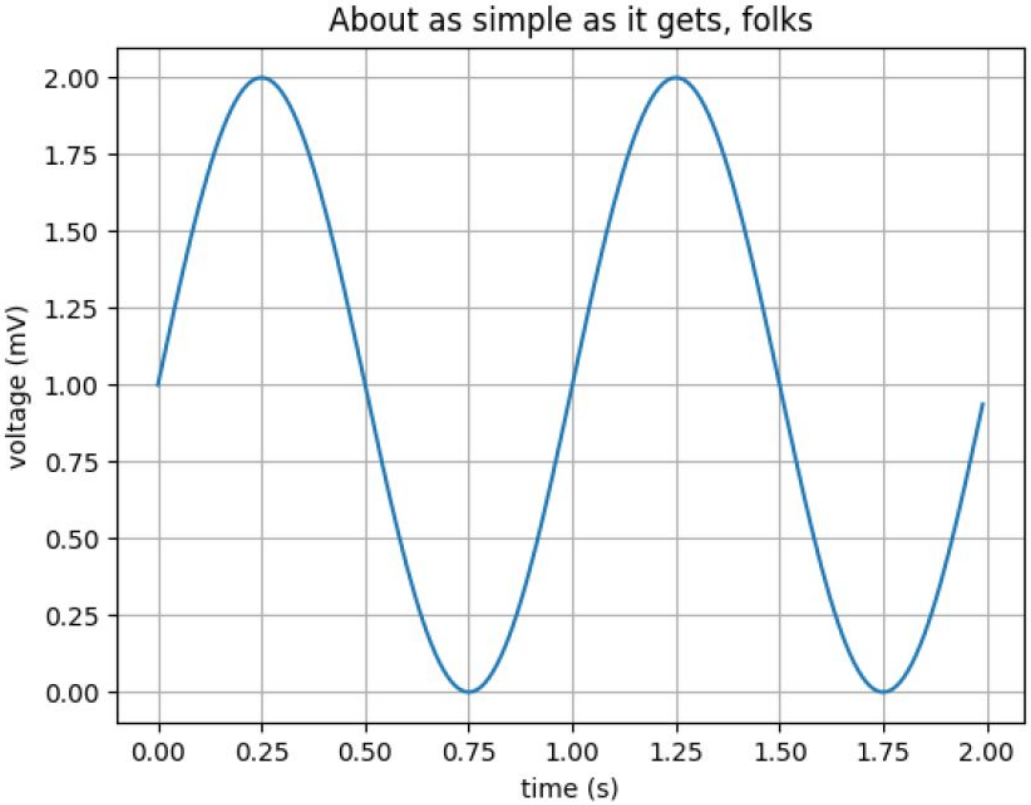


MatplotLib

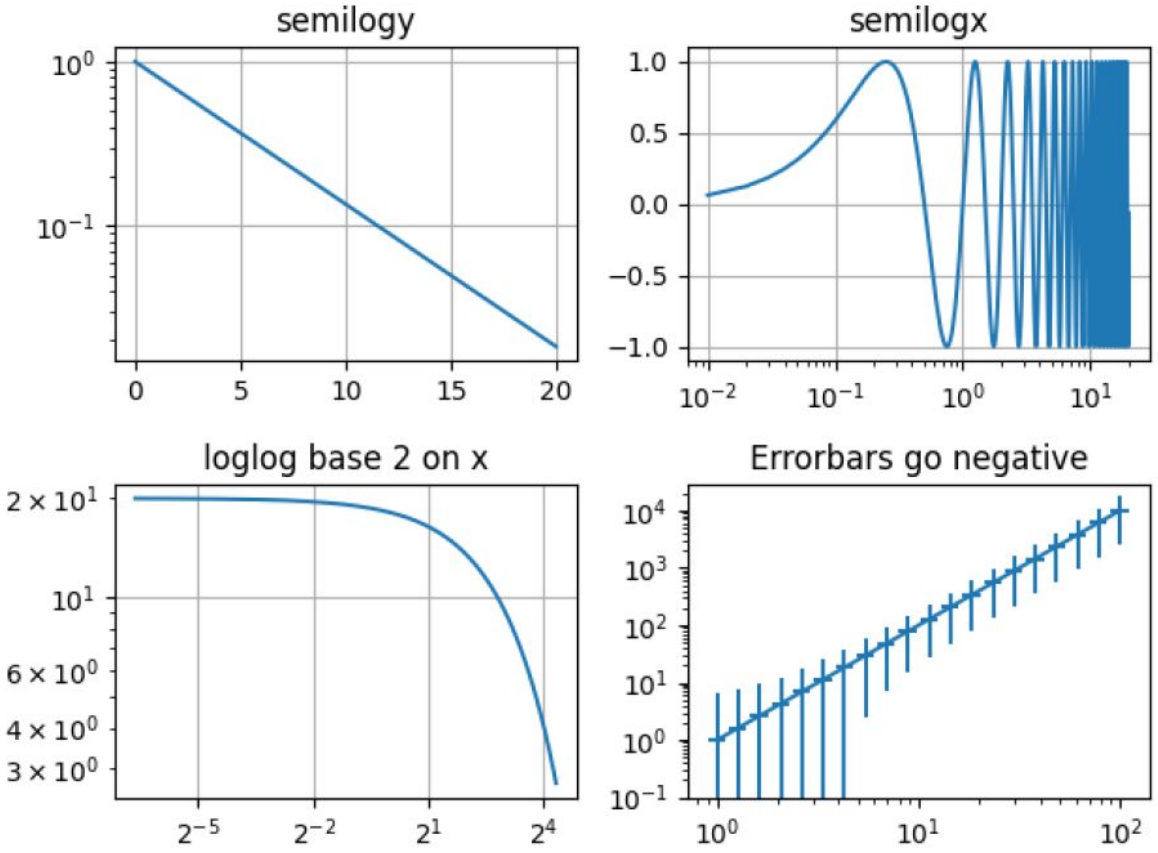
Is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Can be used in:

- Python scripts.
- The Python and IPython shell.
- The jupyter notebook.
- Web application servers.
- Graphical user interface toolkits.

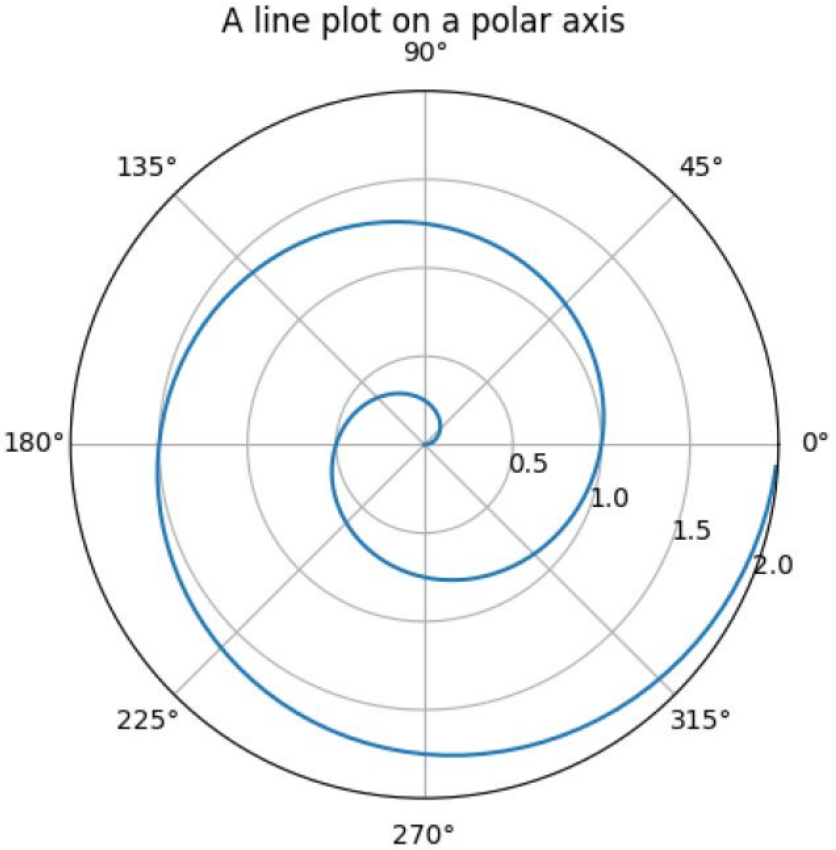
Different kind of plots: y vs. x



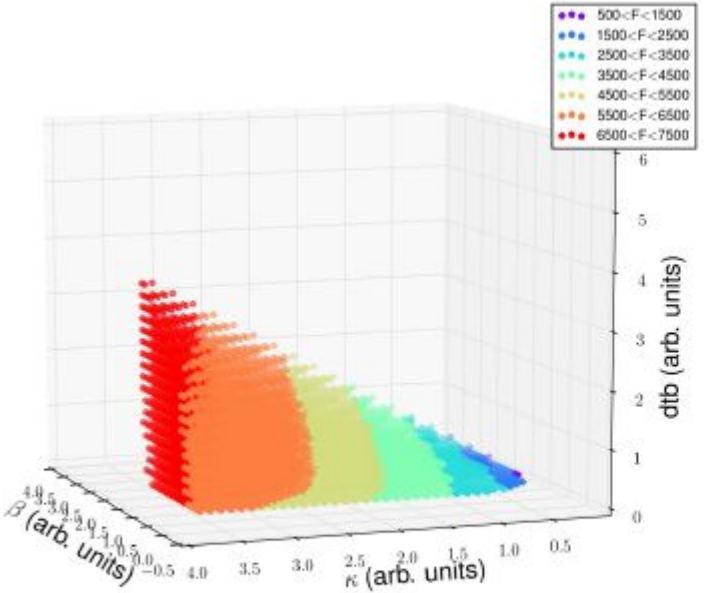
Different scales:

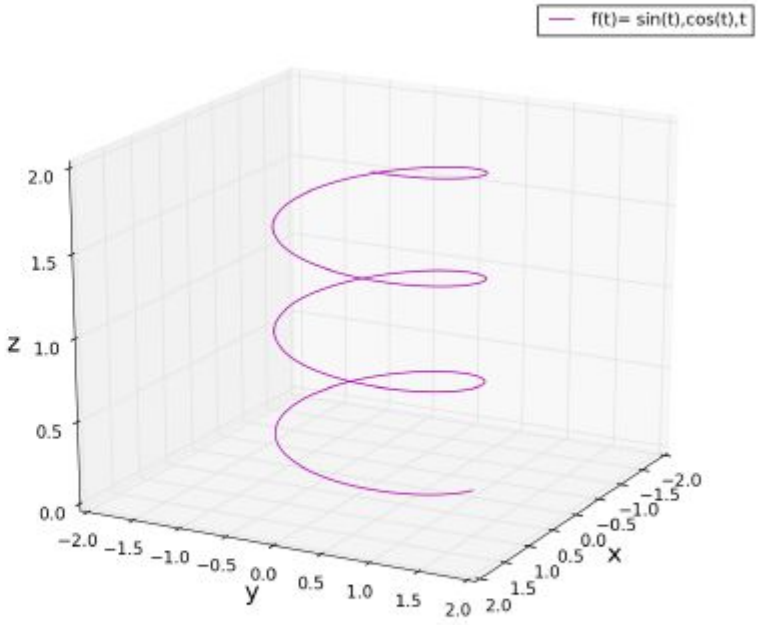


Different coordinates

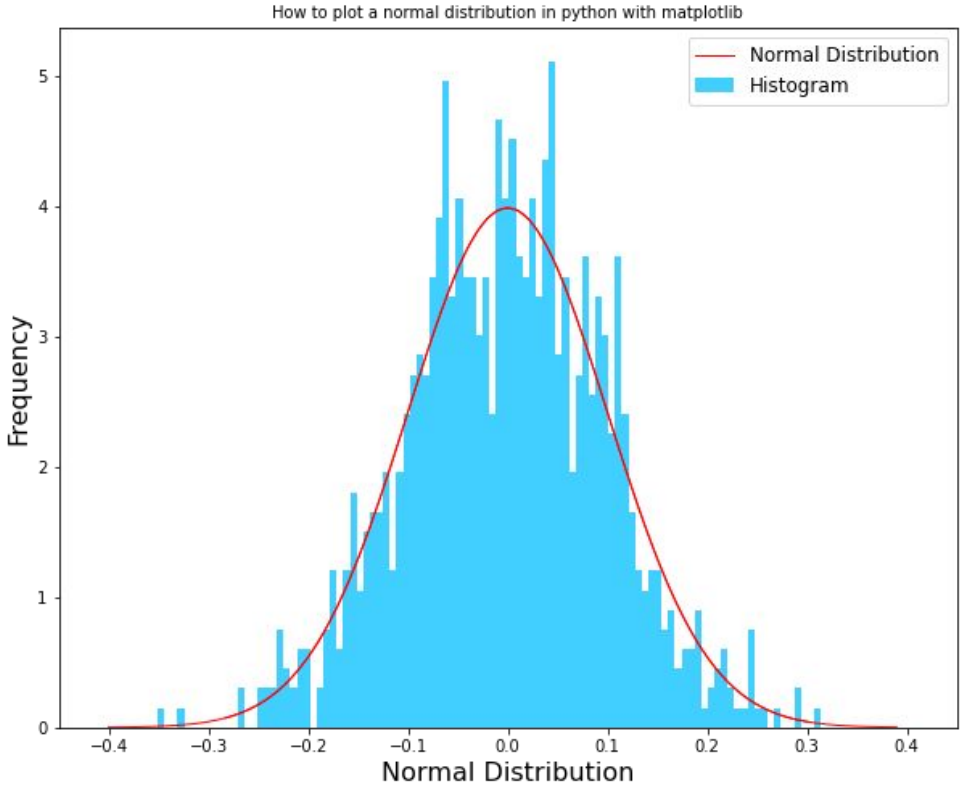


3d plots:


500<F<1500
1500<F<2500
2500<F<3500
3500<F<4500
4500<F<5500
5500<F<6500
6500<F<7500
dtb (arb. units)
β (arb. units)
κ (arb. units)


f(t)= sin(t),cos(t),t
z
y
x

Histograms and distributions



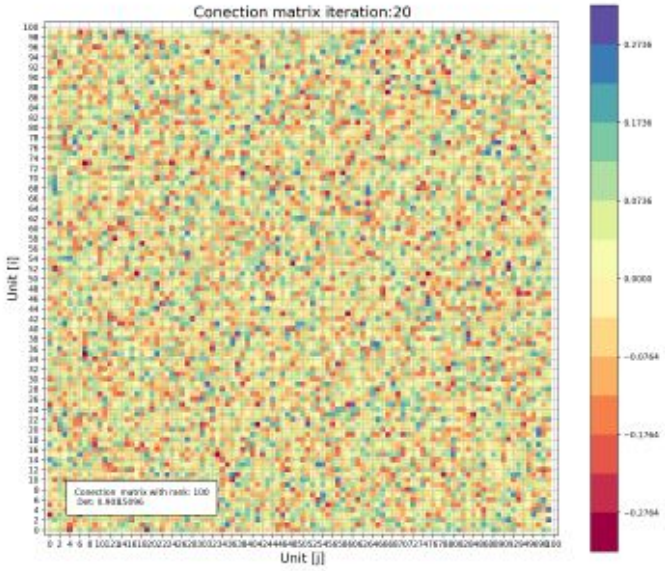
The code for the histogram:

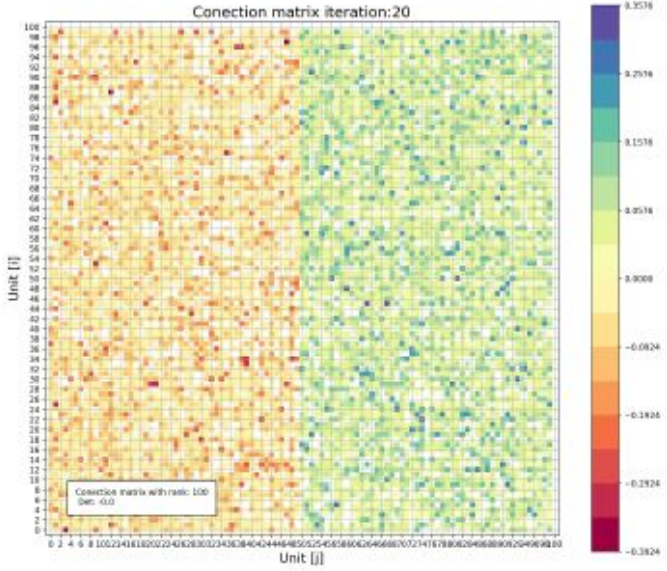
```
mu, sigma = 0, 0.1 # mean and standard deviation
s = np.random.normal(mu, sigma, 1000)
x_n = np.arange(-0.4,0.4,0.01)
y_n = norm.pdf( x_n, mu, sigma)

fig     = plt.figure(figsize=(10,8))
plt.title('Histogram Normal Distribution', fontsize = 18)
plt.hist(s, 100, density=1, facecolor='deepskyblue',label='Histogram',
alpha=0.75)
plt.plot(x_n, y_n, 'r-', linewidth=1,label='Normal Distribution')
```

Matrices:

Numpy Scipy Matplotlib



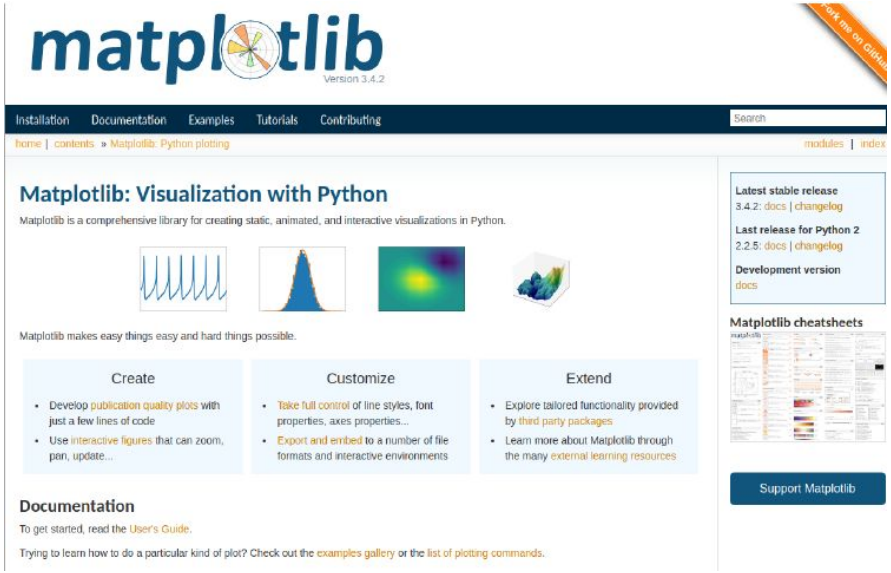


https://www.frontiersin.org/journals/systems-neuroscience/articles/10.3389/fnsys.2024.1269190/full

Please follow the examples in Documentation

there are a lot!

https://matplotlib.org/stable/gallery/index.html



More on visualization with python

https://clauswilke.com/dataviz/

https://www.python-graph-gallery.com/

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

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

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

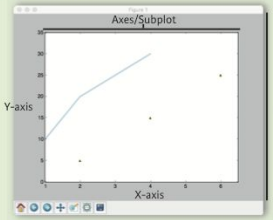
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                  Add padding to a plot
>>> ax.axis('equal')                         Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])    Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                      Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',          Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)     Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)                     Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                          Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])           Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                  Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                  Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')                Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')      Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                    Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)            Plot a 2D field of arrows
```

Data Distributions

```
>>> ax1.hist(y)                              Plot a histogram
>>> ax3.boxplot(y)                           Make a box and whisker plot
>>> ax3.violinplot(z)                        Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                   Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)                Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                  Plot contours
>>> axes2[2].contourf(data1)                 Plot filled contours
>>> axes2[2]= ax.clabel(CS)                  Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

DataCamp
Learn Python for Data Science Interactively

More on the np.arrays importance

ML frameworks as Scikitlearn, Tensorflow or Keras use this structures as input/output of it many functions

Scikit learn: Machine Learning in Python

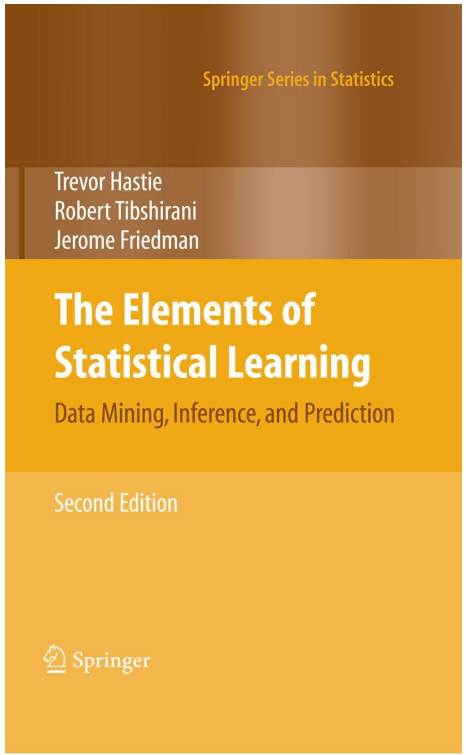
https://scikit-learn.org/stable/

Classification
Regression
Clustering
Dimensionality reduction
Model selection
Preprocessing

TensorFlow: Machine Learning and more:
https://www.tensorflow.org/

Donde estudio la teoría detrás de estos métodos estadísticos avanzados??

The Elements of Statistical Learning



http://pandas.pydata.org/

To import library

```
import pandas as pd


A unidimensional array with labels:

s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd']

print(s)

a    3
b   -5
c    7
d    4
dtype: int64
```

A bi-dimensional array

```
data = {'Country': ['Belgium', 'India', 'Brazil'], 'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
'Population': [11190846, 1303171035, 207847528]}

df = pd.DataFrame(data,columns=['Country', 'Capital', 'Population'])

print(df)
```

```
  Country    Capital  Population
0  Belgium   Brussels    11190846
1    India  New Delhi  1303171035
2   Brazil   Brasilia   207847528
```

To read a file and write it with pandas

```
pd.read_csv('file.csv', header=None, nrows=5)
```

```
df.to_csv('myDataFrame.csv')
```

To read multiple sheets of the same file

```
xlsx = pd.ExcelFile('file.xls')
df = pd.read_excel(xlsx, 'Sheet1')
```

To request help!

```
help(pd.Series.loc)

           mark ii           1       4
    viper      mark ii           7       1
               mark iii         16      36

    Single label. Note this returns a DataFrame with a single index.

    >>> df.loc['cobra']
             max_speed  shield
    mark i          12       2
    mark ii          0       4

    Single index tuple. Note this returns a Series.

    >>> df.loc[('cobra', 'mark ii')]
    max_speed    0
    shield       4
    Name: (cobra, mark ii), dtype: int64

    Single label for row and column. Similar to passing in a tuple, this
    returns a Series.

    >>> df.loc['cobra', 'mark i']....
```

To select one element

```
s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd']
s['b']
-5
```

To select a subset

```
data = {'Country': ['Belgium', 'India', 'Brazil'], 'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
'Population': [11190846, 1303171035, 207847528]}

df = pd.DataFrame(data,columns=['Country', 'Capital', 'Population'])


df[1:]
```

	Country	Capital	Population
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

Retrieving Series/DataFrame Information

```
#(rows, columns)
df.shape

#Describe index
df.index

#Describe DataFrame columns

df.columns
```

```
#Info on DataFrame

df.info()

#Number of non-NA values

df.count()
```

```
<class
'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column      Non-Null Count
Dtype
---  ------      --------------
-----
 0   Country     3 non-null
object
 1   Capital     3 non-null
object
 2   Population  3 non-null
int64
dtypes: int64(1), object(2)
memory usage: 200.0+ bytes
Country       3
Capital       3
Population    3
dtype: int64
```

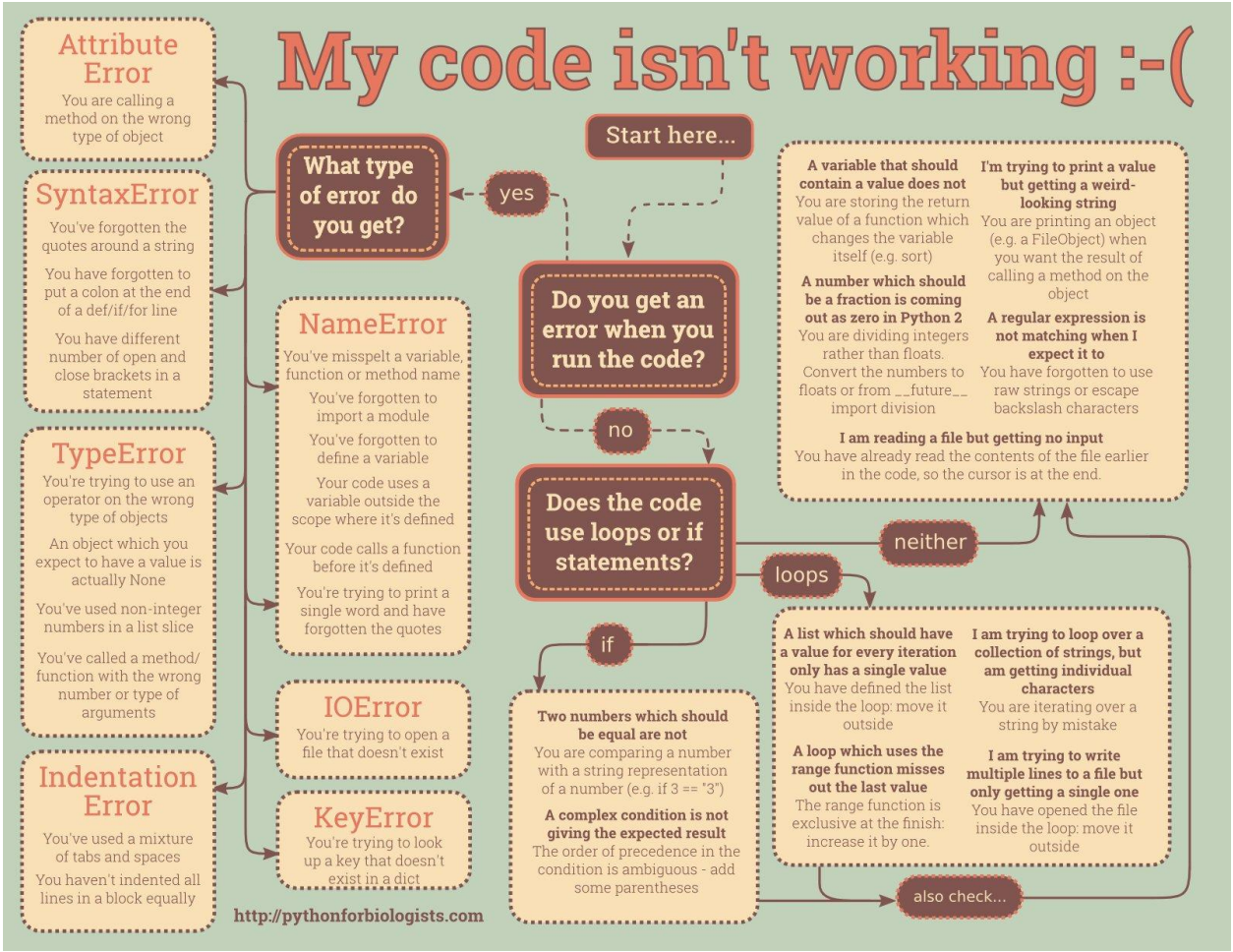
Applying Functions

```
f = lambda x: x*2
#Apply function
df.apply(f)

#Apply function element-wise
df.applynap(f)
```

Thanks!

https://github.com/wtpc/HO-python


My code isn't working :-(
Start here...
Do you get an error when you run the code?
yes
no
What type of error do you get?
Does the code use loops or if statements?
neither
loops
if
Attribute Error
You are calling a method on the wrong type of object
SyntaxError
You've forgotten the quotes around a string
You have forgotten to put a colon at the end of a def/if/for line
You have different number of open and close brackets in a statement
TypeError
You're trying to use an operator on the wrong type of objects
An object which you expect to have a value is actually None
You've used non-integer numbers in a list slice
You've called a method/ function with the wrong number or type of arguments
Indentation Error
You've used a mixture of tabs and spaces
You haven't indented all lines in a block equally
NameError
You've misspelt a variable, function or method name
You've forgotten to import a module
You've forgotten to define a variable
Your code uses a variable outside the scope where it's defined
Your code calls a function before it's defined
You're trying to print a single word and have forgotten the quotes
IOError
You're trying to open a file that doesn't exist
KeyError
You're trying to look up a key that doesn't exist in a dict
A variable that should contain a value does not
You are storing the return value of a function which changes the variable itself (e.g. sort)
I'm trying to print a value but getting a weird-looking string
You are printing an object (e.g. a FileObject) when you want the result of calling a method on the object
A number which should be a fraction is coming out as zero in Python 2
You are dividing integers rather than floats. Convert the numbers to floats or from __future__ import division
A regular expression is not matching when I expect it to
You have forgotten to use raw strings or escape backslash characters
I am reading a file but getting no input
You have already read the contents of the file earlier in the code, so the cursor is at the end.
A list which should have a value for every iteration only has a single value
You have defined the list inside the loop: move it outside
I am trying to loop over a collection of strings, but am getting individual characters
You are iterating over a string by mistake
A loop which uses the range function misses out the last value
The range function is exclusive at the finish: increase it by one.
I am trying to write multiple lines to a file but only getting a single one
You have opened the file inside the loop: move it outside
Two numbers which should be equal are not
You are comparing a number with a string representation of a number (e.g. if 3 == "3")
A complex condition is not giving the expected result
The order of precedence in the condition is ambiguous - add some parentheses
also check...
http://pythonforbiologists.com