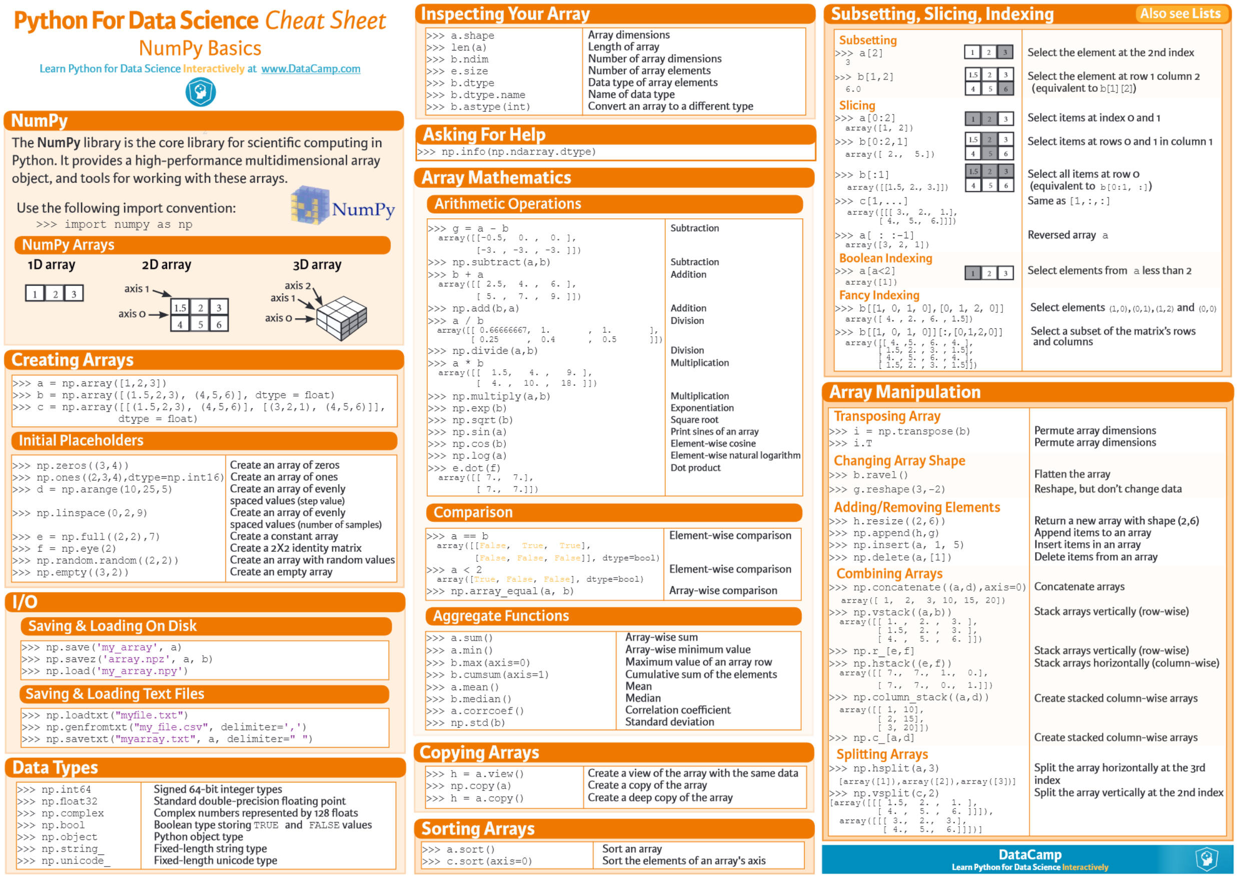
Numpy :



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

1D array — 2D array — 3D array

axis 0, axis 1, axis 2

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` `array([[-0.5, 0. , 0. ], [-3. , -3. , -3. ]])`	Subtraction
`>>> np.subtract(a,b)`	Subtraction
`>>> b + a` `array([[ 2.5, 4. , 6. ], [ 5. , 7. , 9. ]])`	Addition
`>>> np.add(b,a)`	Addition
`>>> a / b` `array([[ 0.66666667, 1. , 1. ], [ 0.25 , 0.4 , 0.5 ]])`	Division
`>>> np.divide(a,b)`	Division
`>>> a * b` `array([[ 1.5, 4. , 9. ], [ 4. , 10. , 18. ]])`	Multiplication
`>>> 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)` `array([[ 7., 7.], [ 7., 7.]])`	Dot product

Comparison

`>>> a == b` `array([[False, True, True], [False, False, False]], dtype=bool)`	Element-wise comparison
`>>> a < 2` `array([True, False, False], dtype=bool)`	Element-wise comparison
`>>> 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]` `3`	Select the element at the 2nd index
`>>> b[1,2]` `6.0`	Select the element at row 1 column 2 (equivalent to `b[1][2]`)
Slicing	
`>>> a[0:2]` `array([1, 2])`	Select items at index 0 and 1
`>>> b[0:2,1]` `array([ 2., 5.])`	Select items at rows 0 and 1 in column 1
`>>> b[:1]` `array([[1.5, 2., 3.]])`	Select all items at row 0 (equivalent to `b[0:1, :]`)
`>>> c[1,...]` `array([[[ 3., 2., 1.], [ 4., 5., 6.]]])`	Same as `[1,:,:]`
`>>> a[ : :-1]` `array([3, 2, 1])`	Reversed array `a`
Boolean Indexing	
`>>> a[a<2]` `array([1])`	Select elements from `a` less than 2
Fancy Indexing	
`>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]` `array([ 4. , 2. , 6. , 1.5])`	Select elements `(1,0)`,`(0,1)`,`(1,2)` and `(0,0)`
`>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]` `array([[ 4. ,5. , 6. , 4. ], [ 1.5, 2. , 3. , 1.5], [ 4. , 5. , 6. , 4. ], [ 1.5, 2. , 3. , 1.5]])`	Select a subset of the matrix's rows and columns

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)` `array([ 1, 2, 3, 10, 15, 20])`	Concatenate arrays
`>>> np.vstack((a,b))` `array([[ 1. , 2. , 3. ], [ 1.5, 2. , 3. ], [ 4. , 5. , 6. ]])`	Stack arrays vertically (row-wise)
`>>> np.r_[e,f]`	Stack arrays vertically (row-wise)
`>>> np.hstack((e,f))` `array([[ 7., 7., 1., 0.], [ 7., 7., 0., 1.]])`	Stack arrays horizontally (column-wise)
`>>> np.column_stack((a,d))` `array([[ 1, 10], [ 2, 15], [ 3, 20]])`	Create stacked column-wise arrays
`>>> np.c_[a,d]`	Create stacked column-wise arrays
Splitting Arrays	
`>>> np.hsplit(a,3)` `[array([1]),array([2]),array([3])]`	Split the array horizontally at the 3rd index
`>>> np.vsplit(c,2)` `[array([[[ 1.5, 2. , 1. ], [ 4. , 5. , 6. ]]]), array([[[ 3., 2., 3.], [ 4., 5., 6.]]])]`	Split the array vertically at the 2nd index

DataCamp
Learn Python for Data Science Interactively

Import Numpy :

```
import numpy as np
dir(np) #it returns what the library contains
```

Matrix :

$$A = \begin{pmatrix} 4 & 3 & 3 & 3 \\ 2 & 4 & 3 & 3 \\ 2 & 2 & 4 & \pi \\ 2 & 2 & 2 & 0.5 \end{pmatrix}$$

```
A=np.array([[4,3,3,3],[2, 4, 3, 3],[2,2,4,np.pi], [2, 2, 2, 0.5]]) # une matrice, est déclarée comme une liste de listes
print(A)
```

Dimension of Matrix :

```
print(A.shape)
#returns (4,4)
#or
np.shape(A)
```

Operations on matrix:

```
print(A[0,0])
print(A[3,3])
print(A[2,3])
#output
4.0
0.5
3.141592653589793
```

2. Afficher V la sous matrice de A formée par la deuxième colonne de A

`A[:,n:m]` est une sous matrice de A avec `:` represente toutes les lignes et `n:m` represente les colonnes de n à m-1.

```
V=A[:,1:2]
print(V)
#output
[[3.]
 [4.]
 [2.]
 [2.]]
************************************
Afficher W la sous matrice de A formée  par la deuxième et la troisième colonne de A.
W=A[:,1:3]
print(W)
#output
[[3. 3.]
 [4. 3.]
 [2. 4.]
 [2. 2.]]
```

Créer la matrice B, une copie de A, puis l'afficher.

```
B=A.copy()
```

💡 `B=A.copy()` entraine la création d'une nouvelle matrice `B` et laisser la matrice `A` intact.
`B=A` entraine que toute modification sur `B` enjendre la même modification sur `A`.

Remplacer π dans la matrice B par e.

```
B[2,3]=np.e
print(B)
```

Remplacer la quatrième ligne de la matrice B par le vecteur [1,1,1,1]

```
B[3,:]=[1,1,1,1]
print(B)
#output
[[4.         3.         3.         3.        ]
 [2.         4.         3.         3.        ]
 [2.         2.         4.         2.71828183]
 [1.         1.         1.         1.        ]]
```

Afficher la diagonale de A

```
print(np.diag(A))
```

Afficher la diagonale de A

```
print(np.diag(A,-1) )
```

Afficher la deuxième sur diagonale de A

```
print(np.diag(A,2) )
#output
[3. 3.]
```

Afficher la matrice triangulaire inférieure associée à A

```
print(np.tril(A))
#l=lower
#output
[[4.  0.  0.  0. ]
 [2.  4.  0.  0. ]
 [2.  2.  4.  0. ]
 [2.  2.  2.  0.5]]
```

Afficher la matrice triangulaire supérieure associée à A

```
print(np.triu(A))
#output
[[4.         3.         3.         3.        ]
 [0.         4.         3.         3.        ]
 [0.         0.         4.         3.14159265]
 [0.         0.         0.         0.5       ]]
```

Produit Matriciel :

matrice1.dot(matrice2)

Calculer les produits matriciels $A*V$, $A*W$, $A2$ et $A5$

```
A.dot(W)# A*V
#output
array([[36.        ],
       [34.        ],
       [28.28318531],
       [19.        ]])
A.dot(W)#A* W
#output
array([[36.        , 39.        ],
       [34.        , 36.        ],
       [28.28318531, 34.28318531],
       [19.        , 21.        ]])
A**2#  la matrice contenant les carrés des élmenents de A.
#output
array([[16.       ,  9.       ,  9.       ,  9.       ],
       [ 4.       , 16.       ,  9.       ,  9.       ],
       [ 4.       ,  4.       , 16.       ,  9.8696044],
       [ 4.       ,  4.       ,  4.       ,  0.25     ]])
A.dot(A) # ou A @ A ou np.linalg.matrix_power(A, 2) #    A^2
#output
array([[34.        , 36.        , 39.        , 31.92477796],
       [28.        , 34.        , 36.        , 28.92477796],
       [26.28318531, 28.28318531, 34.28318531, 26.13716694],
       [17.        , 19.        , 21.        , 18.53318531]])
A5 = np.linalg.matrix_power(A, 5)
print('A5=',A5)
#output
A5= [[39976.72454163 44560.18910578 49661.50322586 40080.90914335]
 [35855.10953339 39976.72454163 44560.18910578 35959.74806912]
 [32482.16861957 36199.34163287 40360.18720374 32569.71511652]
 [21197.78949703 23628.93327993 26337.14343345 21251.79436139]]
```

Afficher la matrice transposée de A

```
print(A.transpose())
#or
print(A.T)
```

Construire une matrice diagonale C contenant le vecteur [−2,1,2] sur la diagonale

```
U= [-2,1,2]
C=np.diagflat(U)
print(C)
#output
[[-2  0  0]
 [ 0  1  0]
 [ 0  0  2]]
```

Construire D une matrice diagonale dont sa diagonale est celle de la matrice A

```
D=np.diagflat(np.diag(A))
print(D)
#output
D=np.diagflat(np.diag(A))

print(D)

[[4.  0.  0.  0. ]
```

```
 [0.  4.  0.  0. ]
 [0.  0.  4.  0. ]
 [0.  0.  0.  0.5]]
```

Inverse :

```
np.linalg.inv(M)
```

Matrices particulières:

Matrice identité:

```
I=np.identity(4)
print(I)
#output
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
#second method
J= np.eye(3)
print(J)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Matrice nulle d'ordre $n*p$

```
O=np.zeros((3,2),int) # nombre de ligne, nombre de colonnes et de type entier
print(O)
#output
[[0 0]
 [0 0]
 [0 0]]
```

Matrice composée des 1 d'ordre $n*p$

```
U=np.ones((4,5),float)# nombre de ligne, nombre de colonnes et de type float
print(U)
#output
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

Résolution d'un système de Cramer.

Soit le système d'équations linéaires suivant : $\boldsymbol{MX} = \boldsymbol{b}$, avec :

$$\boldsymbol{M} = \begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix}, \ \boldsymbol{X} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, \text{ et } \boldsymbol{b} = \begin{pmatrix} 50 \\ 30 \\ 70 \\ 50 \end{pmatrix},$$

1. Ecrire les matrices $\boldsymbol{M}$ et $\boldsymbol{b}$.

```
M=np.array([[4,-1,-1,0],[-1,4,0,-1],[-1,0,4,-1],[0,-1,-1,4]])#les éléments de M doivent être de type float
b=np.array([50,30,70,50])
print(M)
print(b)
```

Montrer que le système est de Cramer.

```
#on vérifie que det(A)#0
np.linalg.det(M)
#output
192.0
```

Resoudre le systeme :

X=M^-1 b

```
# X=M^-1 b
X=np.linalg.inv(M).dot(b)
X
```

`np.arange()` et `np.linspace` :

Les fonctions `np.arange()` et `np.linspace` retournent un objet de type `numpy.ndarray`

`np.arange(n,m)` permet d'obtenir un tableau 1D de la valeur de départ `n` à la valeur de fin `m-1` avec un pas `1`

```
np.arange(2,8)
#output
array([2, 3, 4, 5, 6, 7])
```

`np.arange(m)` permet d'obtenir un tableau 1D de la valeur de départ `0` à la valeur de fin `m-1` avec un pas `1`

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

`np.arange(n,m,p)` permet d'obtenir un tableau 1D de la valeur de départ `n` à la valeur de fin `m-p` avec un pas `p`

```
np.arange(2,8,3)
```

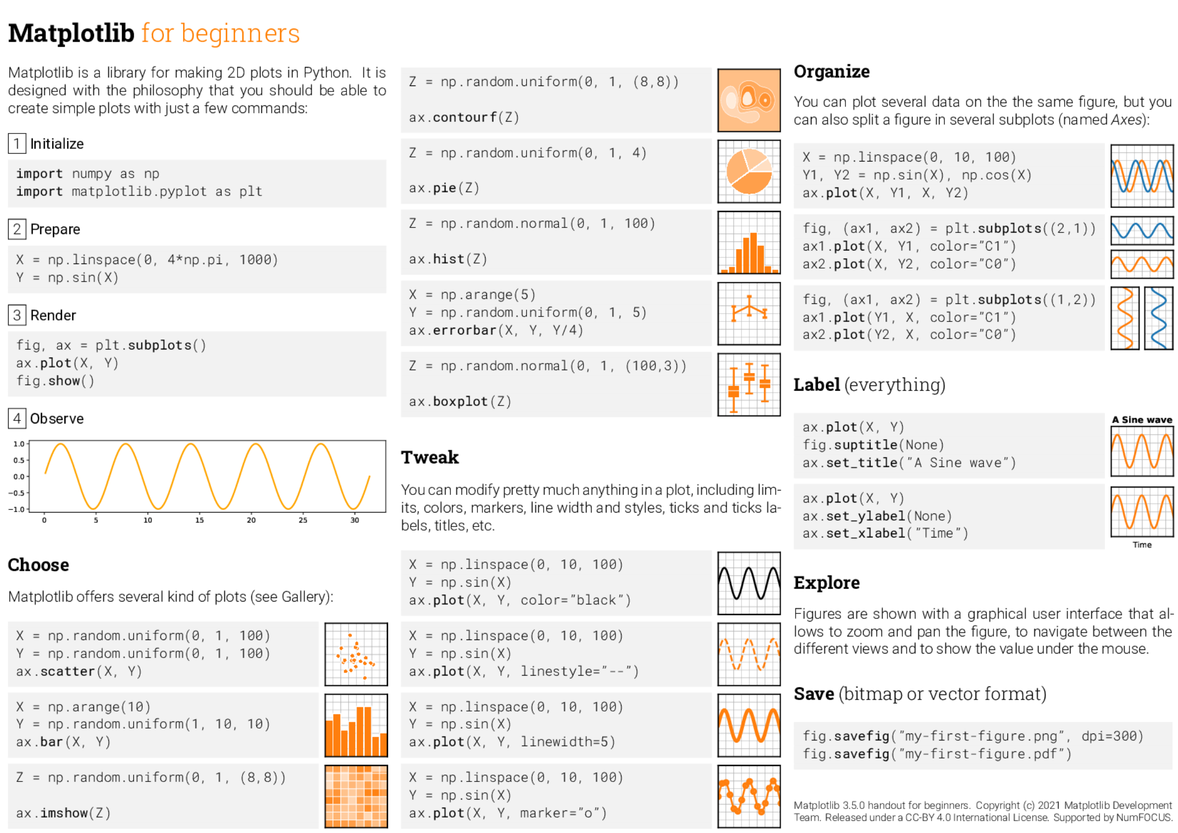
`np.linspace(n,m,p` permet d'obtenir un tableau 1D contenant p coefficient allant d'une valeur de départ `n` à une valeur de fin `m`

```
np.linspace(1,2,3)
array([1. , 1.5, 2. ])
```

Représentations graphiques sous Python

import matplotlib:

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



Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

```
import numpy as np
import matplotlib.pyplot as plt
```

2 Prepare

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
fig.show()
```

4 Observe

Choose

Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```

```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

```
Z = np.random.uniform(0, 1, (8,8))

ax.imshow(Z)
```

```
Z = np.random.uniform(0, 1, (8,8))

ax.contourf(Z)
```

```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```

```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```

```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```

```
Z = np.random.normal(0, 1, (100,3))

ax.boxplot(Z)
```

Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```

```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```

```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```

```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```

Organize

You can plot several data on the the same figure, but you can also split a figure in several subplots (named *Axes*):

```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```

```
fig, (ax1, ax2) = plt.subplots((2,1))
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

```
fig, (ax1, ax2) = plt.subplots((1,2))
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```

```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```

Explore

Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

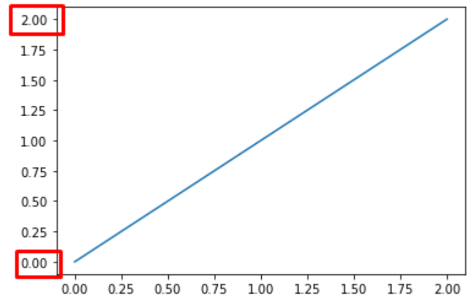
Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```


Tracer des lignes :

[0,1,2] sont les ordonees et les abscisses,sont automatiquement générées et vont de **0**à **len(liste) - 1**

```
plt.plot([0, 1, 2])
```

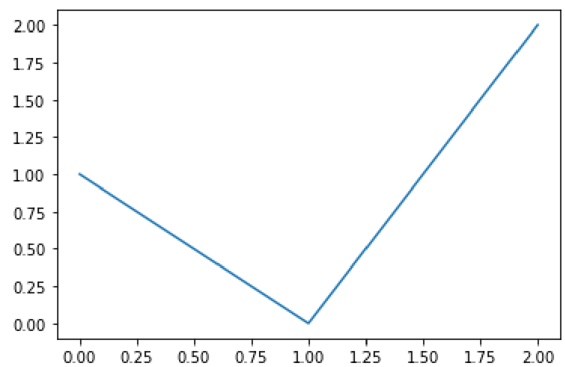


Tracer une figure :

on peut passer deux listes en arguments de la methode plot.

la premiere liste x=[0,1,2] correspond aux abscisses et y correspond à la liste de leurs ordonnées.

```
x = [0, 1, 2]
y = [1, 0, 2]
plt.plot(x, y)
```



Tracer une fonction mathématique

Tracer la fonction `sin()` sur [0,2π].

```
x = np.arrange(0,2*np.pi,0.1) #On crée un array qui va de 0 à 2pi exclu avec un pas de 0.01
y = np.sin(x)
plt.plot(x,y)
```

Résolution numérique de systèmes d'équations linéaires