Une introduction à Python pour scientifiques débutants

Frederic Paletou, Christophe Peymirat

Python scientifique

Une intro pour scientifiques débutants

frederic.paletou@univ-tlse3.fr

- Prise en main et rudiments
- Utilisation interactive
- **`numpy - scipy`**
- **`matplotlib`**
- Scripts
- Des boucles, des tests…
- Visualisation/sauvegarde graphiques
- Lecture/écriture de fichiers
- *Pickling* etc.

exemples : https://box.in2p3.fr/index.php/s/bqBNKH5zwz82ngQ

Généralités

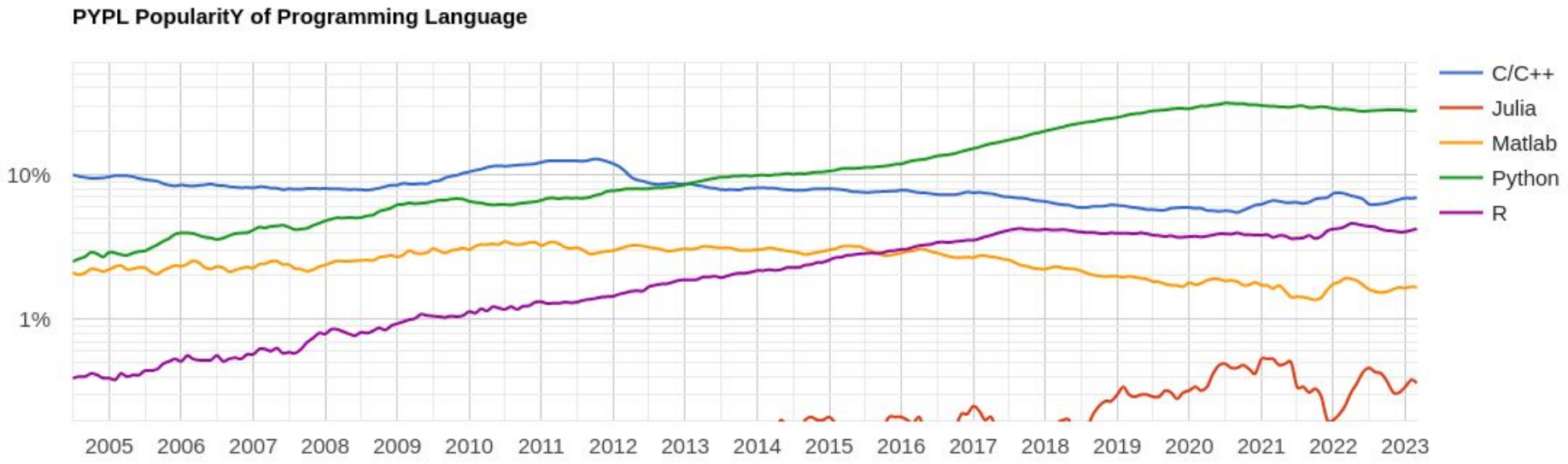
Observatoire Midi-Pyrénées
OMP

`Python` est un langage :

- **interprété**,
- à **typage dynamique** (fort),
  - l'exécution détectera des erreurs/conflits de typage
- **scriptable**,
- **« multi-paradigmes »** donc éventuellement « objet » (pas abordé ici…),
  - programmation impérative/structurée et fonctionnelle
- **multi-plateformes**,
- **sous licence libre**.

https://www.python.org/

Pourquoi Python ?

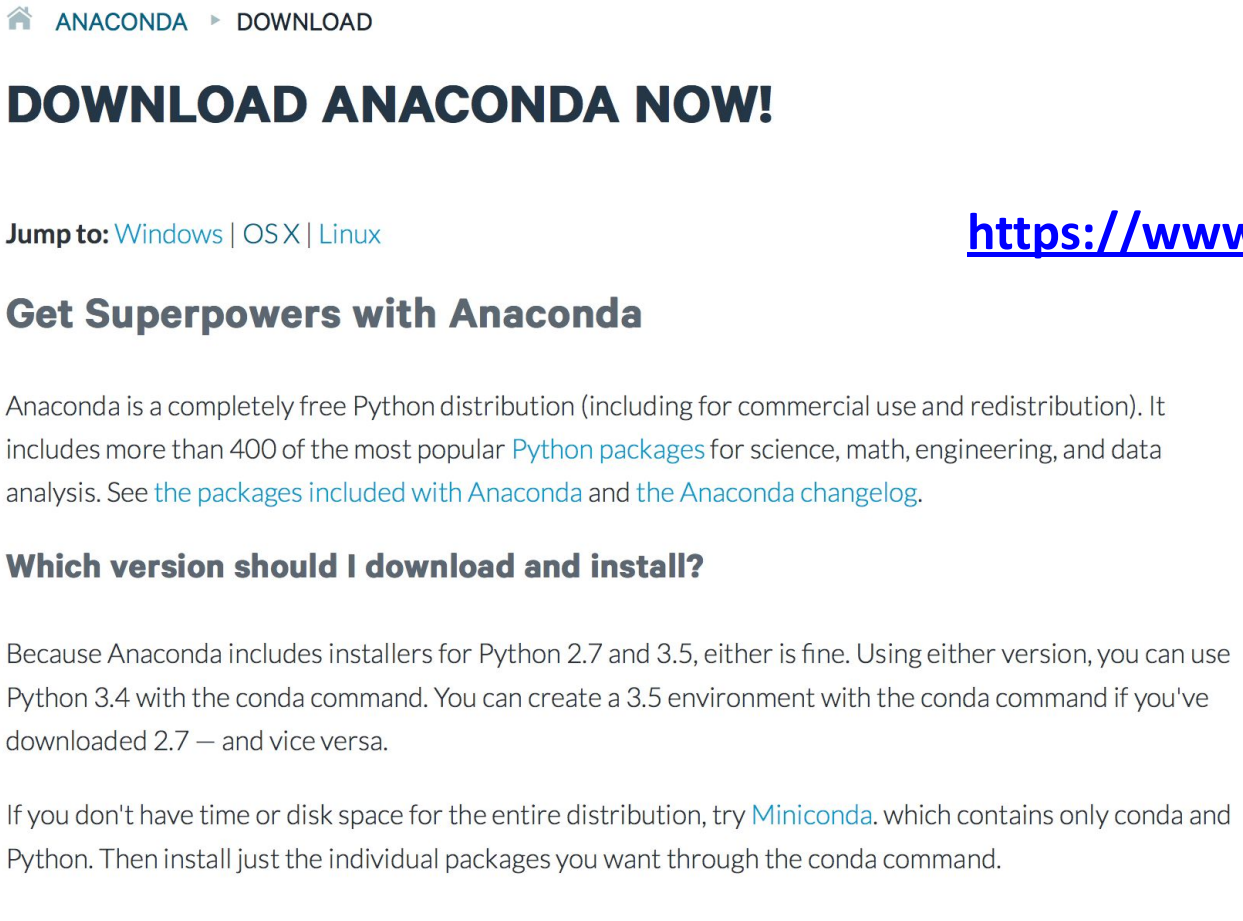


Toujours en croissance, notamment grace aux outils de **ML/DL** déployés ces dernières années pour Python !

Mais pour nous, **scientifiques**, il s'agit (surtout) de quelles **librairies** vont "augmenter" le langage à proprement parler...

Quoi installer ?

Il existe divers moyens d'installer les ressources que nous allons utiliser en priorité (`numpy/scipy` et `matplotlib`)... il est souvent recommandé d'utiliser :



ANACONDA ▸ DOWNLOAD

DOWNLOAD ANACONDA NOW!

Jump to: Windows | OS X | Linux

https://www.continuum.io/

Get Superpowers with Anaconda

Anaconda is a completely free Python distribution (including for commercial use and redistribution). It includes more than 400 of the most popular Python packages for science, math, engineering, and data analysis. See the packages included with Anaconda and the Anaconda changelog.

Which version should I download and install?

Because Anaconda includes installers for Python 2.7 and 3.5, either is fine. Using either version, you can use Python 3.4 with the conda command. You can create a 3.5 environment with the conda command if you've downloaded 2.7 — and vice versa.

If you don't have time or disk space for the entire distribution, try Miniconda. which contains only conda and Python. Then install just the individual packages you want through the conda command.

Quoi installer ?

Observatoire Midi-Pyrénées OMP

Au moins une bonne raison de passer d'emblée à Python 3 !

> Matplotlib 3.0 is Python 3 only.
>
> For Python 2 support, Matplotlib 2.2.x will be continued as a LTS release and updated with bugfixes until January 1, 2020.

Ou pas (encore), en utilisant en ligne :

- https://basthon.fr/
- https://repl.it/languages/python3
- https://cocalc.com/features/python

Démarrage

Ouvrez un **terminal**… ou bien utilisez en parallèle le ***Jupyter notebook*** distribué, et le cas écheant, en ligne avec https://basthon.fr/ p. ex.

Tapez : **`python`** *(ou bien :* **`python3`** *suivant votre installation)*

```
fpaletou@himba:~/ASTRO$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Vous êtes sous `Python` ! Essayez :

```
>>> a=2
>>> a+2
4
>>>
```

On **sort** par **Ctrl-D**

Typage dynamique

Observatoire Midi-Pyrénées OMP

```
>>> a=2
>>> b='zzz'
>>> a+b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and
'str'

>>> c='42'
>>> b+c
'zzz42'

>>> l=[0,1,2,3,4]
>>> a+l
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and
'list'
```

Usage scientifique

Observatoire Midi-Pyrénées

Bien que https://docs.python.org/2/library/numeric.html soit une librairie standard, **`Python`** n'est **pas** en l'état pleinement adapté à un usage scientifique…

Nécessite l'**ajout de certaines librairies** dont les plus **indispensables** sont :

`numpy`	http://www.scipy.org/
`matplotlib`	http://matplotlib.org/

`numpy` : pour la création/manipulation de vecteurs/matrices, faire de l'algèbre linéaire ou de l'analyse de Fourier, etc.

`matplotlib` : pour toutes les représentations graphiques, du `plot(x,y)` simple à la visualisation d'images...

Vous serez aussi certainement intéressés par d'**autres** librairies spécifiques e.g., **`scipy`**, `astropy`, **`pyngl`** (*NCAR Graphics*), `pyfits`, **`pandas`**, **`keras`** **`(DL)`** etc. suivant la nature de vos activités et de vos besoins.

`numpy`

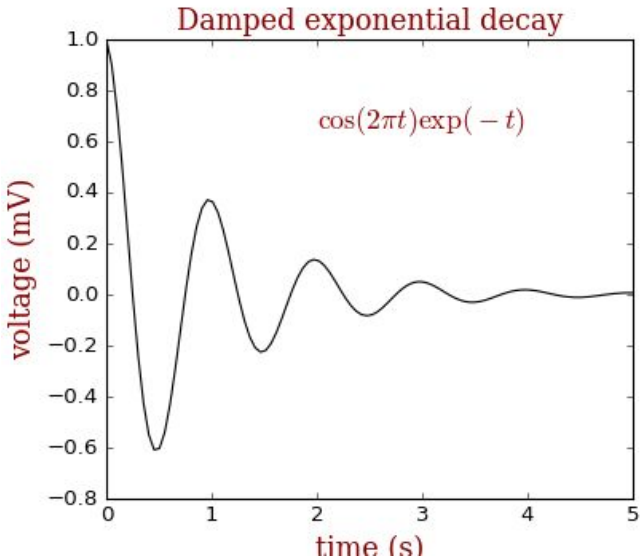
Observatoire Midi-Pyrénées OMP

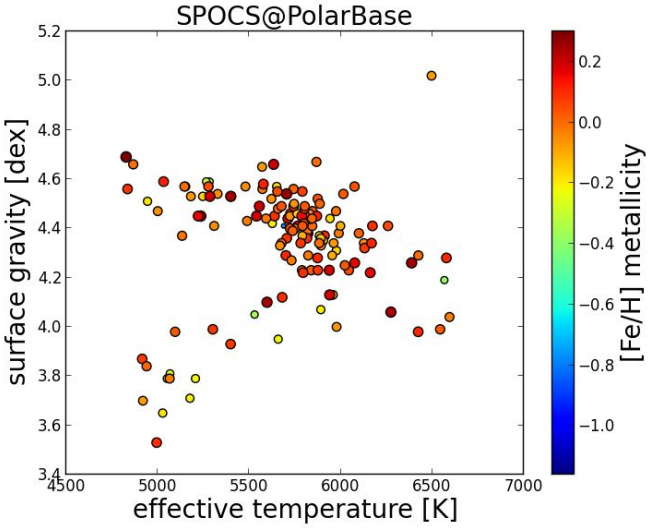
La librairie **`numpy`** permet :

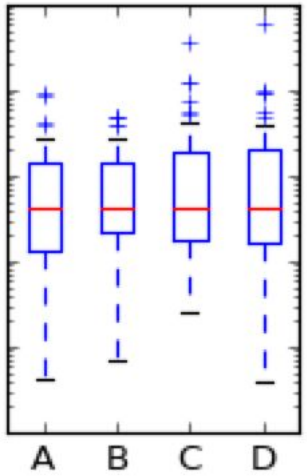
- la **création et manipulation de tableaux**,
- des **opérations de base** (e.g., produits vectoriel ou matriciel etc.),
- de faire de l'**algèbre linéaire** (**`numpy.linalg`**),
- et aussi :
  - **`numpy.random`**
  - **`numpy.fft`**
  - …

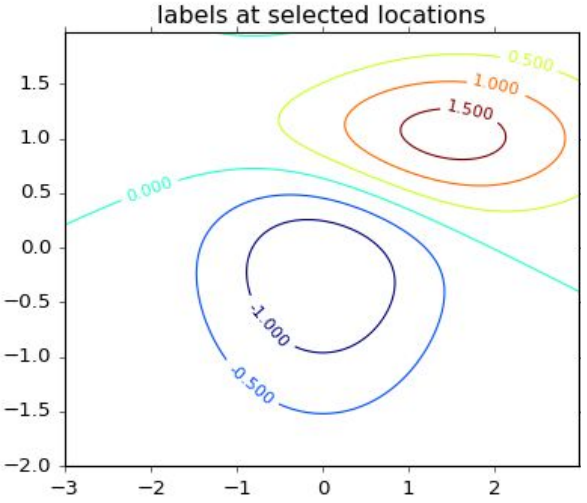
http://www.numpy.org/

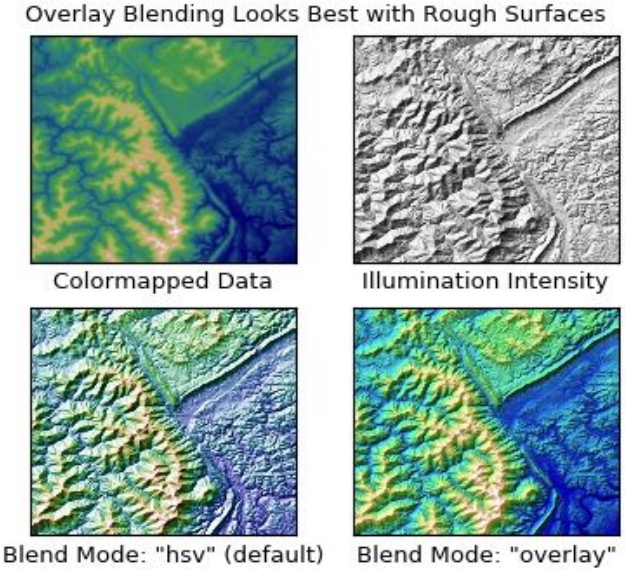
matplotlib











matplotlib.org

Librairies : `import`

Observatoire Midi-Pyrénées OMP

Dans la suite nous utiliserons fréquemment **`numpy`** et **`matplotlib`** que nous devrons systématiquement « **importer** » avant de pouvoir les utiliser !

Un exemple (avec une fonction a priori usuelle, *log*, dans le **notebook**) :

```
>>> pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
>>> import numpy as np
>>> np.pi
3.141592653589793
```

ou encore, pour créer un tableau 1D de 10 éléments, rempli initialement de 1...

```
>>> np.ones(10)
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

Librairies : **`import`**

Observatoire Midi-Pyrénées OMP

De la même façon, on ajoutera à **`Python`** les fonctionnalités **`pyplot`** de **`matplotlib`** par une commande du type :

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

Et l'utilisation de tout module de **`matplotlib`** se fera ensuite de la façon suivante :

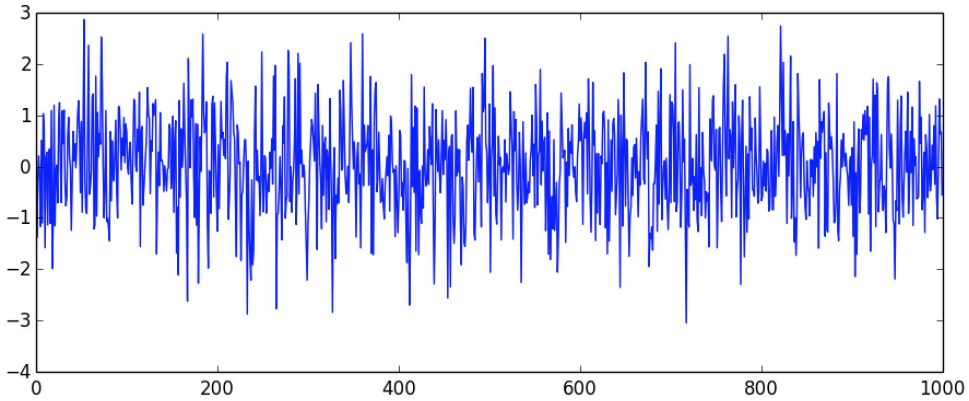
```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x=np.arange(10)
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> plt.plot(x,x*x)
[<matplotlib.lines.Line2D object at 0x10804a210>]
>>> plt.show()
```

Cette dernière commande permettant de visualiser le graphique… *on y reviendra*.

Librairies : **`import`**

Ces librairies proposent diverses « familles » de fonctions, regroupées selon les types de fonctionnalités proposées p. ex. **`random`** de **`numpy`** :

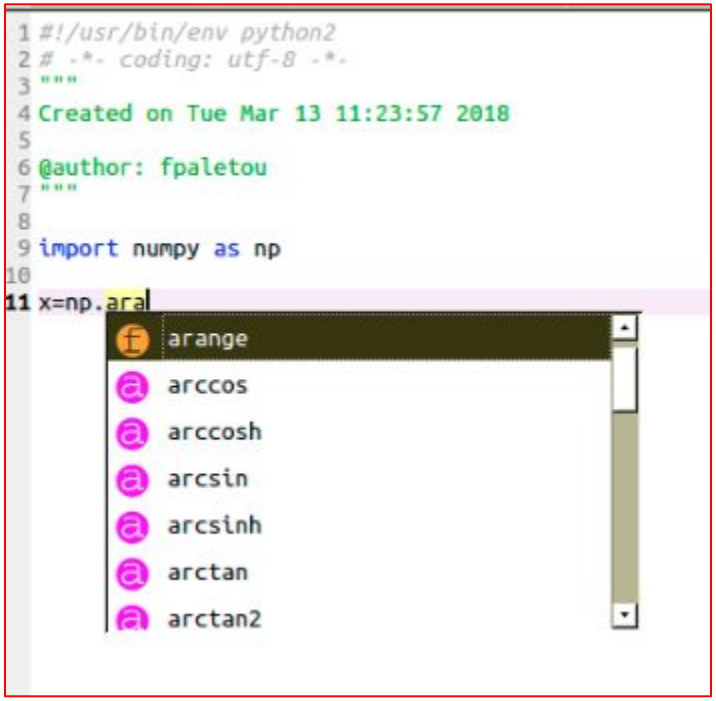
```
>>> import numpy as np
>>> s=np.random.randn(1000)
>>> np.mean(s)
0.021971490883817026
>>> np.std(s)
0.9637130692313359 3
>>> import matplotlib.pyplot as plt
>>> plt.plot(s)
[<matplotlib.lines.Line2D object at 0x106cdf290>]
>>> plt.show()
```

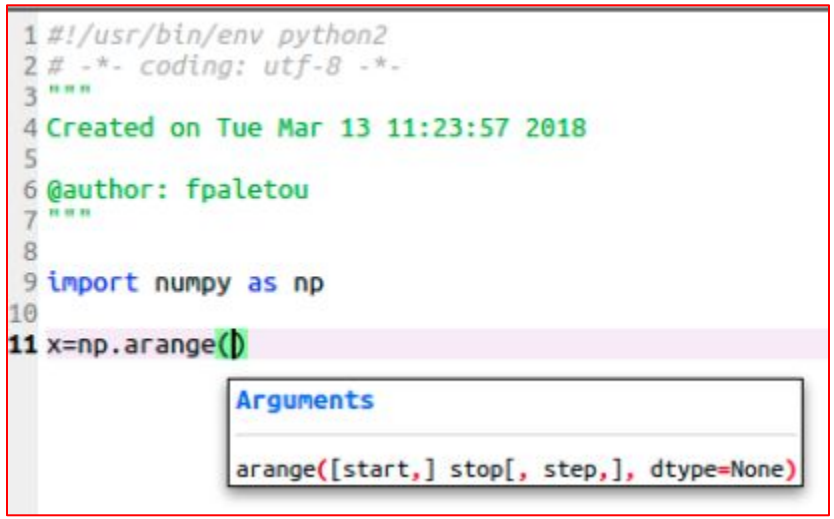


`randn` : *Return a sample (or samples) from the "standard normal" distribution.*

Auto-complétion

Un intérêt évident d'un outil comme **Spyder**… surtout en phase d'**apprentissage**





Création de tableaux

Observatoire Midi-Pyrénées

```
>>> x=np.arange(10)
>>> print(x)
[0 1 2 3 4 5 6 7 8 9]


>>> x=np.arange(1, 10)
>>> print(x)
[1 2 3 4 5 6 7 8 9]


>>> x=np.arange(1, 10, 0.5)
>>> print(x)
[ 1.   1.5  2.   2.5  3.   3.5  4.   4.5  5.   5.5  6.
6.5  7.   7.5  8.   8.5  9.   9.5]


>>> print(x[5])
3.5
```

Création de tableaux

Observatoire Midi-Pyrénées OMP

```
>>> x=np.zeros(10)
>>> print(x)
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]

>>> x=np.ones(10)
>>> print(x)
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

Tableaux multi-dimensionnels :

```
>>> M=np.zeros((4,4))
>>> print(M)
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
```

Manipulation de tableaux

Observatoire Midi-Pyrénées OMP

```
>>> x=np.arange(25)
```

On transforme un tableau 1D en matrice (5,5) :

```
>>> M=np.reshape(x,(5,5))

>>> print(M)
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

Accès à des sous-matrices :

```
>>> print(M[2,:])
[10 11 12 13 14]
```

Essayez maintenant :

```
>>> len(M)
5

>>> M.shape
(5, 5)

>>> M[1]
array([5, 6, 7, 8, 9])

>>> M[1][1]
6
```

« = » vs. copy

Observatoire Midi-Pyrénées

!!! Attention !!!

Soit :

```
>>> M
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])

>>> B=M[0]
>>> B[1]=42

>>> M
array([[ 0, 42,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Par contre...

```
>>> B=np.copy(M[0])
>>> B[1]=42

>>> B
array([ 0, 42,  2,  3])


>>> M
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Algèbre linéaire

Observatoire Midi-Pyrénées OMP

- Produit scalaire : **`dot(x,y)`**
- Produit matriciel : **même instruction !**

```
>>> x=np.arange(9)
>>> M=np.reshape(x,(3,3))
>>> M
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])

>>> a=np.arange(3)
>>> a
array([0, 1, 2])

>>> np.dot(M,a)
array([ 5, 14, 23])
```

```
>>> M=[ [-1., 3.],
...      [-5., 2.]]

print(np.linalg.det(M)*np.linalg.inv(M))
[[ 2. -3.]
 [ 5. -1.]]

>>> np.dot(np.linalg.inv(M),M)
array([[  1.00000000e+00,
-5.55111512e-17],
       [  5.55111512e-17,
1.00000000e+00]])
```

Algèbre linéaire

Observatoire Midi-Pyrénées OMP

Un grand nombre de fonctions sont disponibles @ **`numpy.linalg`**

- Multiplication par un scalaire (p) : **`p*M`**
- Transposée d'une matrice : **`transpose`**`(M)`
- Décomposition d'une matrice : **`svd`**`(M), etc…`
- Valeurs propres et vecteurs propres : **`eig`**`(M)`
- Solution de **A**x=**b** : **`solve`**`(A,b)`
- …

http://docs.scipy.org/doc/numpy/reference/routines.linalg.html

Scripts

Observatoire Midi-Pyrénées OMP

On peut aussi utiliser **`Python`** à partir de scripts rassemblant un certain nombre de commandes.

L'indentation des lignes d'instruction est critique !

En effet, les divers blocs (boucles, tests etc.) ne sont identifiés que par l'indentation (**pas** de commande **end** en particulier) **!**

On exécutera un script soit, depuis un terminal comme : **`python mon_script.py`**

ou encore : **`python -i mon_script.py`**

qui permet de **rester dans l'environnement `Python`** une fois les commandes effectuées, et de pouvoir continuer à travailler **interactivement**

En **restant sous l'environnement `Python(3)` et pour (re-)lancer un script** :

```
>>> exec(open("./filename").read())
```

Boucles `for`

Observatoire Midi-Pyrénées OMP

Un exemple :

```
>>> for i in np.arange(10):
...     print(i)  # attention en interactif placer un TAB avant la commande
...               # puis RETURN ici pour lancer la boucle...
0
1
2
3
4
5
6
7
8
9
```

À retenir que **`arange(10)`** commence par '**0**' tout comme les **indices** des tableaux créés avec **`numpy`**

Créer un tableau 2D

Observatoire Midi-Pyrénées OMP

Un exemple :

```
#---set (x,y) domains
x=np.arange(-12.,12.,0.165)
y=np.arange(-12.,12.,0.165)

#---make 2D function array
nx=len(x)
ny=len(y)
sinc2=np.zeros( (nx, ny), 'Float32')

#---make a 2D sinc-function
for i in np.arange(nx):
    for j in np.arange(ny):
        sinc2[i,j]=(np.sin(x[i])/x[i])*(np.sin(y[j])/y[j])
```

Notez qu'il **existe** une fonction **`numpy.sinc`** !

Visualisation à suivre…

Les listes

Observatoire Midi-Pyrénées OMP

Les objets **`list`** permettent la création **dynamique** de tableaux.

```
import numpy as np

#---make initial array
a=np.arange(15)
print(a)

l=[]

#---makes a list of even elements only
for i in a:
    if (np.mod(i,2) == 0):
        l.append(i)

print(l)
```

Ce sont aussi, comme les objets **`str`**, des **"objets itérables"** comme vont (aussi) le montrer les exemples à suivre…

Les listes

Observatoire Midi-Pyrénées OMP

On peut aussi **mélanger les types**… Soit par exemple une liste faite comme :

```
lst.append( ([0.,1./3,1./6,1./9],'A','BB','CCC') )
lst.append( ([0.,1./2,1./4,1./8],'B','CC','AAA') )
lst.append( ([0.,1./3,1./5,1./7],'C','AA','BBB') )
```

On peut alors la **manipuler** de la façon suivante :

```
>>> lst[1]
([0.0, 0.5, 0.25, 0.125], 'B', 'CC', 'AAA')

>>> lst[1][0]
[0.0, 0.5, 0.25, 0.125]

>>> len(lst[1][0])
4

>>> lst[1][1]+lst[1][3]
'BAAA'
```

Boucles `for`

Observatoire Midi-Pyrénées OMP

Avec des "**objets itérables**" comme…

```
>>> for letter in 'Python':
...     print('Current letter:',letter)
...
Current letter: P
Current letter: y
Current letter: t
Current letter: h
Current letter: o
Current letter: n
```

Ou encore…

```
>>> color=['r','g','b']
>>> for k in color:
...     print(k)
...
r
g
b
```

Formatage

```
import numpy as np

#--- LaTeX-table style output
a=1/3.
b=1/4.
c=1/5.

lst=[ ['object-1',a,b,c],
      ['object-2',b,c,a],
      ['object-3',c,a,b] ]

print('Method #1')
print('---------')
for i in np.arange(3):
    print(lst[i][0], ' & ', '%4.2f' % lst[i][1], ' & ', \
     '%4.2f' % lst[i][2], ' & ', '%4.2f' % lst[i][3], '\\\\')
```

```
object-1  &  0.33  &  0.25  &  0.20 \\
object-2  &  0.25  &  0.20  &  0.33 \\
object-3  &  0.20  &  0.33  &  0.25 \\
```

Formatage

```
print('Method #2')
print('---------')
for i in np.arange(3):
    print ("%10s & %4.2f & %4.2f & %4.2f \\\\" %  \
           (lst[i][0], lst[i][1], lst[i][2], lst[i][3]))
```

```
  object-1 & 0.33 & 0.25 & 0.20 \\
  object-2 & 0.25 & 0.20 & 0.33 \\
  object-3 & 0.20 & 0.33 & 0.25 \\
```

Attention en **Python 3** : **`print`** est désormais une **fonction** !!!

(pour ceux qui auraient encore du 2.8 à transformer...)

Boucles `while`

Observatoire Midi-Pyrénées OMP

Un exemple (classique) :

```
#--- seeking for the smallest float
eps=0.1

while (1. + eps > 1.):
    eps=eps/2.

print (eps)
```

Rappel : il n'y a pas d'autre indicateur de fin de boucle que le retour à l'indentation précédant cette boucle (**blocs identifiés par l'indentation !)**

```
>>> exec(open('epsilon.py').read())
8.881784197e-17
```

(permet de **relancer** un script sans sortir de l'environnement >>>)

Tests

Observatoire Midi-Pyrénées OMP

```
import sys
#--- warning! execute as: python SecDeg.py a b c
#--- where (a,b,c) are real

import numpy as np

#--- extract (a,b,c) before solving for:
#--- a*x^2 + b*x + c = 0
a=float(sys.argv[1])
b=float(sys.argv[2])
c=float(sys.argv[3])

delta=b*b-4.*a*c

if (delta < 0):
    print('Pas de solution (dans R) !')
else:
    x1=(-b+np.sqrt(delta))/2./a
    x2=(-b-np.sqrt(delta))/2./a
    print('solution(s):', x1, x2)
```

Nombres complexes

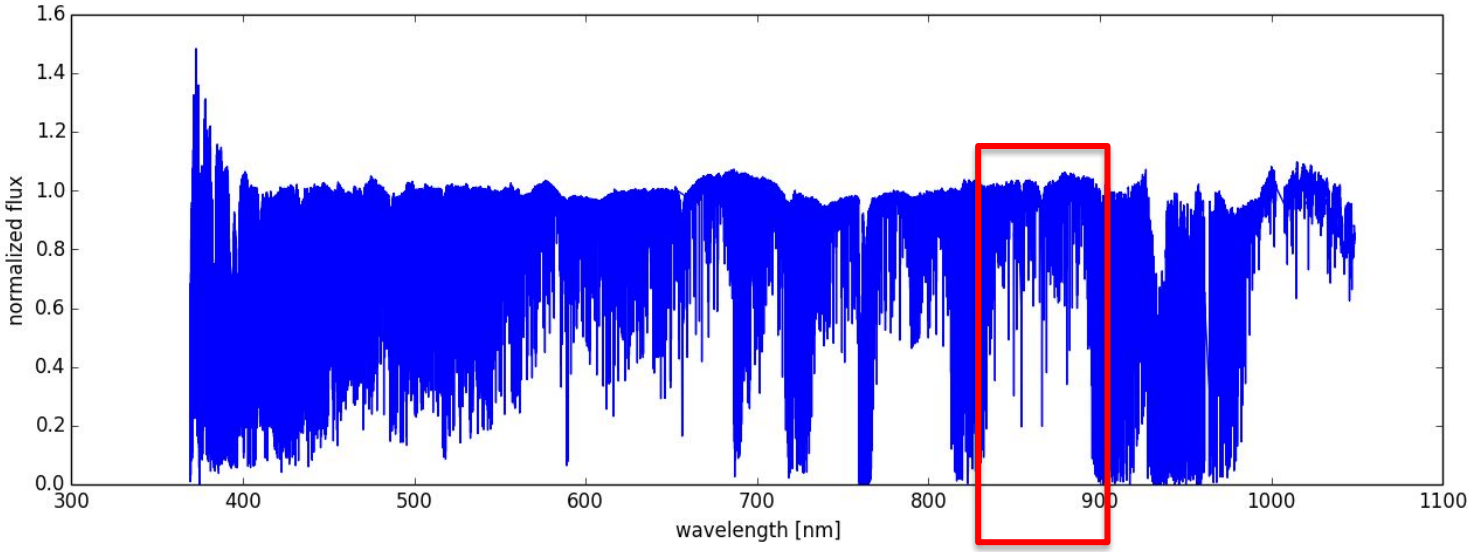
Observatoire Midi-Pyrénées OMP

```
import sys
#--- warning! execute as: python SecDegInC.py a b c
#--- (a,b,c) are real

import numpy as np

#--- extract (a,b,c) before solving for: a*x^2 + b*x + c = 0
a=np.complex(sys.argv[1])
b=np.complex(sys.argv[2])
c=np.complex(sys.argv[3])

delta=b*b-4.*a*c

x1=(-b+np.sqrt(delta))/2./a
x2=(-b-np.sqrt(delta))/2./a
print('solution(s):', x1, x2)
print(np.real(x1), np.imag(x1))
print(np.real(x2), np.imag(x2))
```

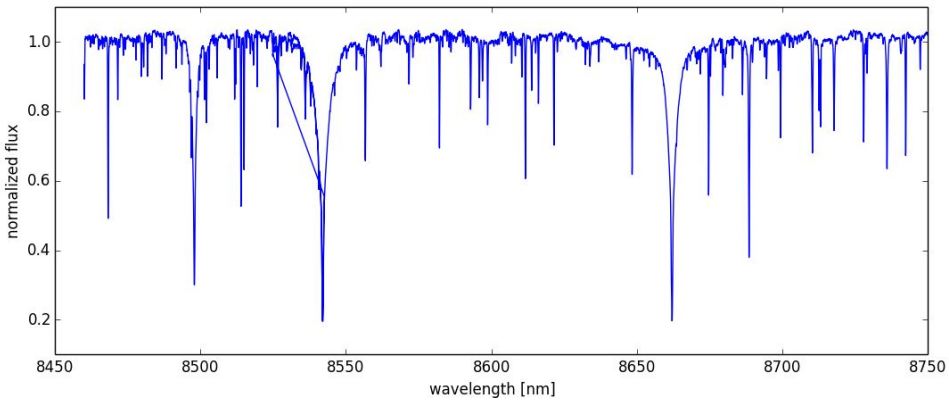
```
>>> print(np.sqrt(-2.+0j))
1.4142135623730951j
```

Sélection sur tests

Un spectre solaire observé par `Narval@TBL` (Pic du Midi)…




```
>>> critrvs=(lo>=846.0) & (lo<=875.0)
>>> lirt=10.*lo[critrvs]
>>> flirt=flo[critrvs]
```



`(lo,flo)` : *spectre complet*

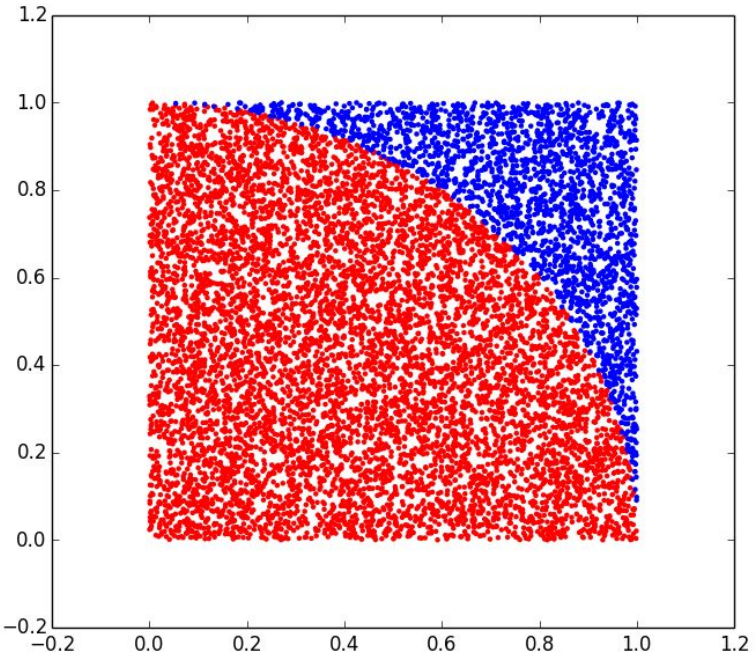
Sélection sur tests

```
>>> N=10000
>>> x=np.random.random(N)
>>> y=np.random.random(N)

>>> r2=x*x+y*y

>>> pi_est=4.*float(len(r2[np.where(r2<1)]))/float(N)

>>> pi_est
3.1572
```

Prétexte : estimation de π
par une méthode de **Monte-Carlo**

Sortie graphique associée :

`pyplot : scatter(x,y,[...])`



Définir une fonction

Observatoire Midi-Pyrénées OMP

```
#---ma fonction/procedure/subroutine
def echange(a,b):
    c=a
    a=b
    b=c
    return a,b

#---valeurs initiales
a=1
b=42
print(a,b)

#---swap
a,b=echange(a,b)
print(a,b)
```

Voir le script : **def_simple.py**

Créer sa bibliothèque

- Je mets **mes fonctions** dans un fichier spécifique p. ex. : **`ma_bib.py`**
- Dans le nouveau script, j'**importe** cette nouvelle librairie, ce qui va aussi **modifier la syntaxe de l'appel** :

```
#---importe ma fonction/procedure/subroutine
import ma_bib

#---valeurs initiales
a=1
b=42
print(a,b)

#---swap
a,b=ma_bib.echange(a,b)
print(a,b)
```

+ Création d'un fichier **`ma_bib.pyc`** !

Voir aussi : **`imp.reload`**(*module*)
(après : **`import imp`**)

Lecture de fichiers `ascii`

Nous allons utiliser un exemple avec les données historiques de **concentration de CO_2** mesurées au *Mauna Loa* (Hawaii) et distribuées par le NOAA @

https://gml.noaa.gov/ccgg/trends/data.html

```
#
# CO2 expressed as a mole fraction in dry air, micromol/mol, abbreviated as ppm
#
# year    mean     unc
  1959  315.97    0.12
  1960  316.91    0.12
  1961  317.64    0.12
  .......
```

*Nous n'avons retenu **que** 4 lignes de commentaires (commençant par #) par rapport au fichier d'origine.*

L'objectif est d'**extraire les concentrations (en colonne-2) vs. le temps** [year]

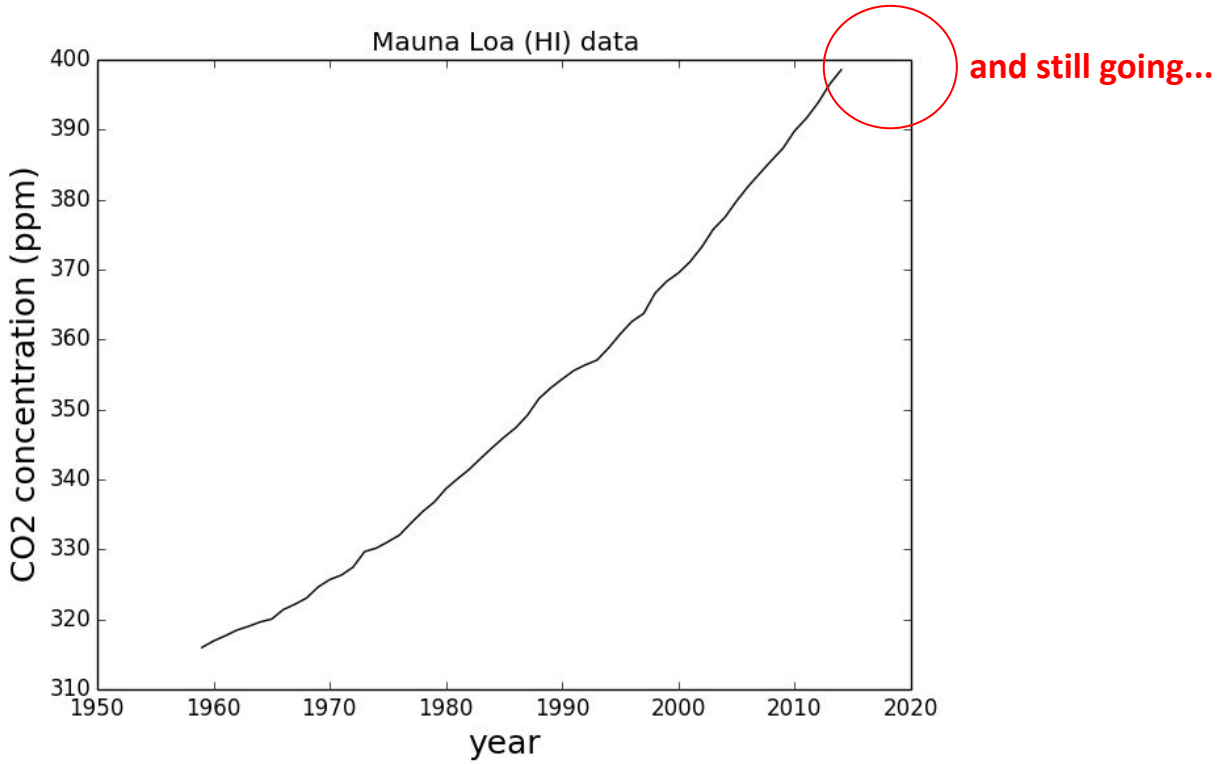
Lecture de fichiers **ascii**

Observatoire Midi-Pyrénées OMP

```
#--- ouvre le fichier ascii
f=open('CO2MaunaLoa.txt','r')

#--- tout lire
tout=f.readlines()
nlines=len(tout)

#--- fermer le fichier
f.close()

#--- initialiser deux listes : year/taux co2
year=[]
tco2=[]

#--- eliminer les 4 premieres lignes de commentaire
for i in np.arange(4,nlines):
    year.append( float(str.split(tout[i])[0]) )
    tco2.append( float(str.split(tout[i])[1]) )
```

CO_2 vs. année

Observatoire Midi-Pyrénées OMP



and still going...

```
plt.plot(year, tco2, 'k')
plt.title('Mauna Loa (HI) data')
plt.xlabel('year', fontsize=18)
plt.ylabel('CO2 concentration (ppm)', fontsize=18)
plt.show()
```

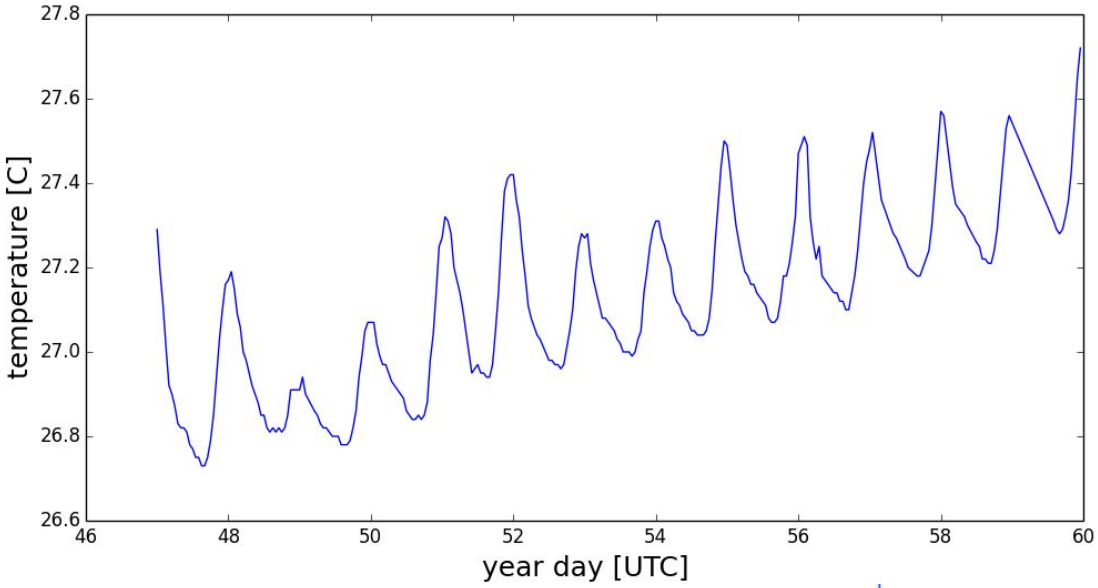
Lecture de fichiers `csv`

Un exemple à partir d'un fichier « *Carioca temperature data* » récupéré sur la page :

http://www.pmel.noaa.gov/co2/story/GasEx+2001+Data

```
>>> import csv
>>> flnm=open('carioca_temp.csv','rU')
>>> x=csv.reader(flnm)
>>> ll=[]
>>> for row in x:
...     ll.append(row)
...
>>> ll[0]
['%Year Day(UTC)', 'Latitude(deg)', 'Longitude(deg)',
'Carioca SST(degC)']
>>> ll[1]
['47.00000000', '-2.9992', '-125.2541', '27.29']
>>> len(ll)
273
>>> ll[272]
['59.95832176', '-2.3021', '-131.4159', '27.72']
```

Lecture de fichiers **csv**




```
day=[]
tmp=[]
for i in np.arange(1,len(ll)):
    day.append( float(ll[i][0]) )
    tmp.append( float(ll[i][3]) )
plt.plot(day, tmp)
plt.xlabel('year day [UTC]', fontsize=18)
plt.ylabel('temperature [C]', fontsize=18)
plt.show()
```

Lecture de fichiers "csv"

Observatoire Midi-Pyrénées OMP

À partir d'un fichier créé par **Excel** (**FR**) et exporté en *so-called* **csv** :

```
angle;cos(angle)
1;0,540302306
2;-0,416146837
3;-0,989992497
4;-0,653643621
5;0,283662185
6;0,960170287
```

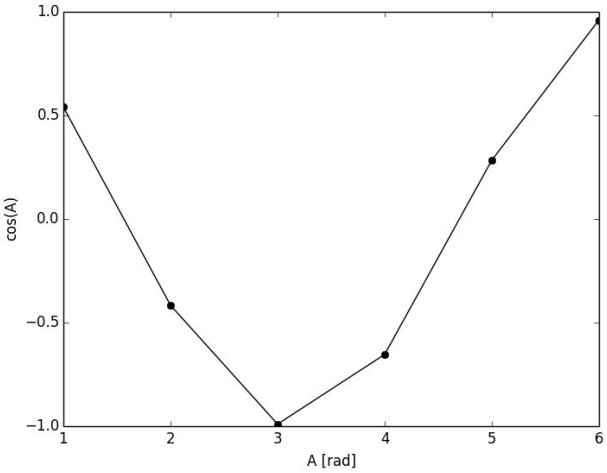
```
flnm=open('cosA.csv','rU')

tout=flnm.readlines()

A=[]
c=[]

for i in np.arange(1,len(tout)):
    A.append( float(tout[i].split(';')[0]) )
    c.append( ??? )
```

- **la colonne-2 est le cosinus de la colonne-1**
- **séparation par ';' (ou « ;- separated values » !)**
- **réels « à virgule » : 0,54…** au lieu de 0.54… **!**
- **relecture et exploitation sans utiliser le module csv ?**
- **méthode « `readlines` » : cf. le script `cosA.py`**

Manipuler "**;-sv(#,#)**"

Observatoire Midi-Pyrénées OMP

```
>>> tout[1]
'1;0,540302306\n'

>>> tout[1].split(';')
['1', '0,540302306\n']

>>> tout[1].split(';')[1]
'0,540302306\n'

>>> tout[1].split(';')[1].strip()
'0,540302306'

>>> tout[1].split(';')[1].strip().replace(',','.')
'0.540302306'

>>> float(tout[1].split(';')[1].strip().replace(',','.'))
0.540302306
```



Affichages multiples

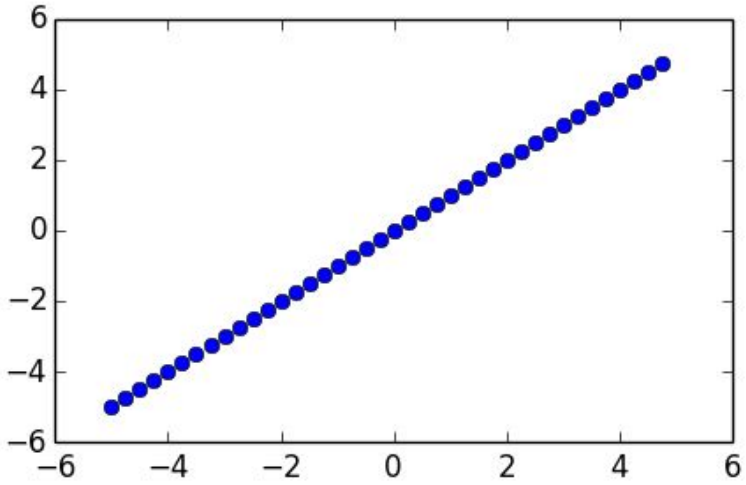
(plus ***linestyle***)

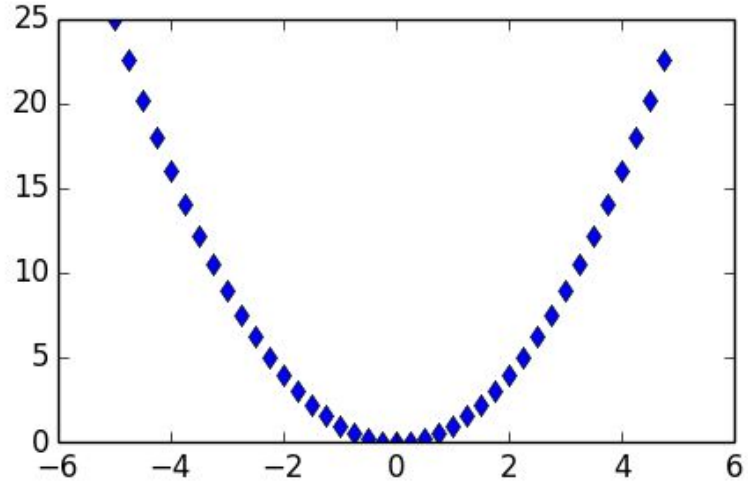
Observatoire Midi-Pyrénées OMP

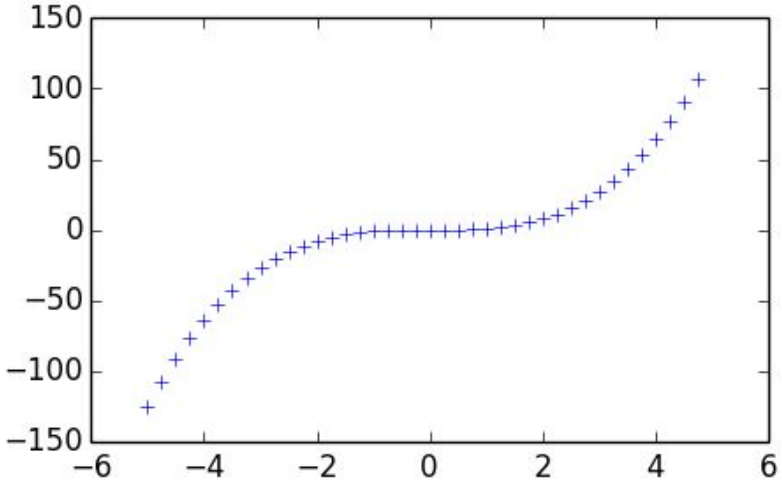
```
>>> x=np.arange(-5.,5.,0.25)
>>> plt.subplot(2,2,1)
<matplotlib.axes.AxesSubplot object at 0x10d957c90>
>>> plt.plot(x,x,'o')
[<matplotlib.lines.Line2D object at 0x10cfe2e50>]
>>> plt.subplot(2,2,2)
<matplotlib.axes.AxesSubplot object at 0x10cfe2e10>
>>> plt.plot(x,x*x,'d')
[<matplotlib.lines.Line2D object at 0x10d18f3d0>]
>>> plt.subplot(2,2,3)
<matplotlib.axes.AxesSubplot object at 0x10d18f390>
>>> plt.plot(x,x*x*x,'+')
[<matplotlib.lines.Line2D object at 0x10d1f0c90>]
>>> plt.subplot(2,2,4)
<matplotlib.axes.AxesSubplot object at 0x10d1f0ed0>
>>> plt.plot(x,x*x*x*x,'*')
[<matplotlib.lines.Line2D object at 0x10d286210>]
>>> plt.show()
```

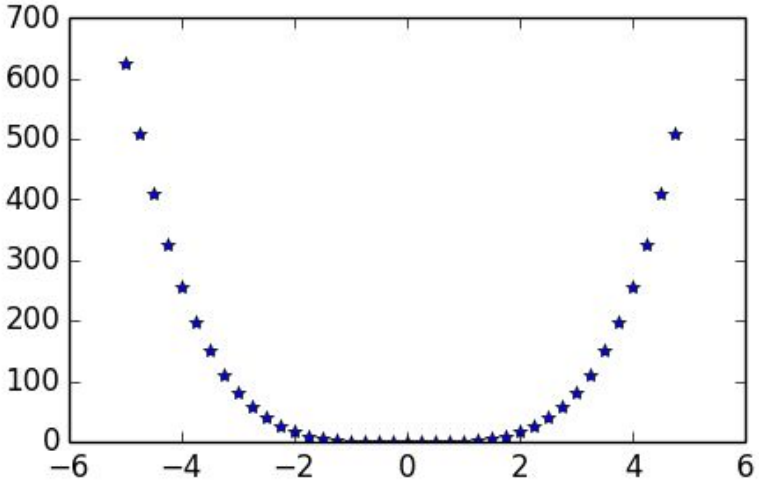
Affichages multiples

(plus ***linestyle***)







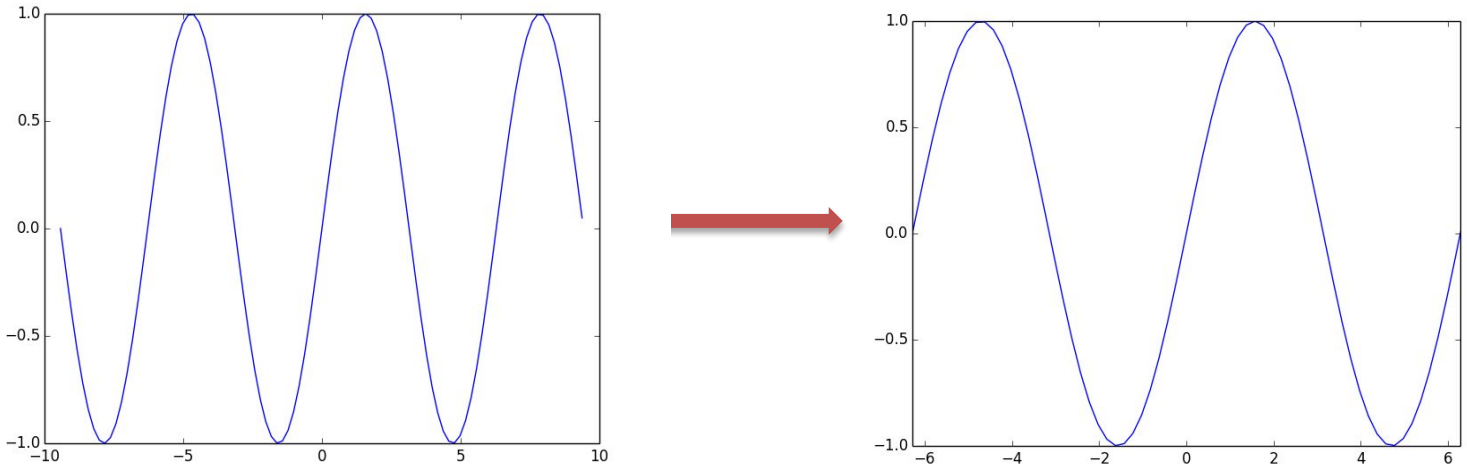


Contrôle des limites

```
>>> import numpy as np
>>> from matplotlib import pyplot as plt

>>> x=np.arange(-3.*np.pi,3.*np.pi,0.2)

>>> plt.xlim([-2*np.pi,2*np.pi])
(-6.283185307179586, 6.283185307179586)
>>> plt.plot(x,np.sin(x))
[<matplotlib.lines.Line2D object at 0x10e1b0990>]
>>> plt.show()
```

attention, voir aussi :

xylim_new.py



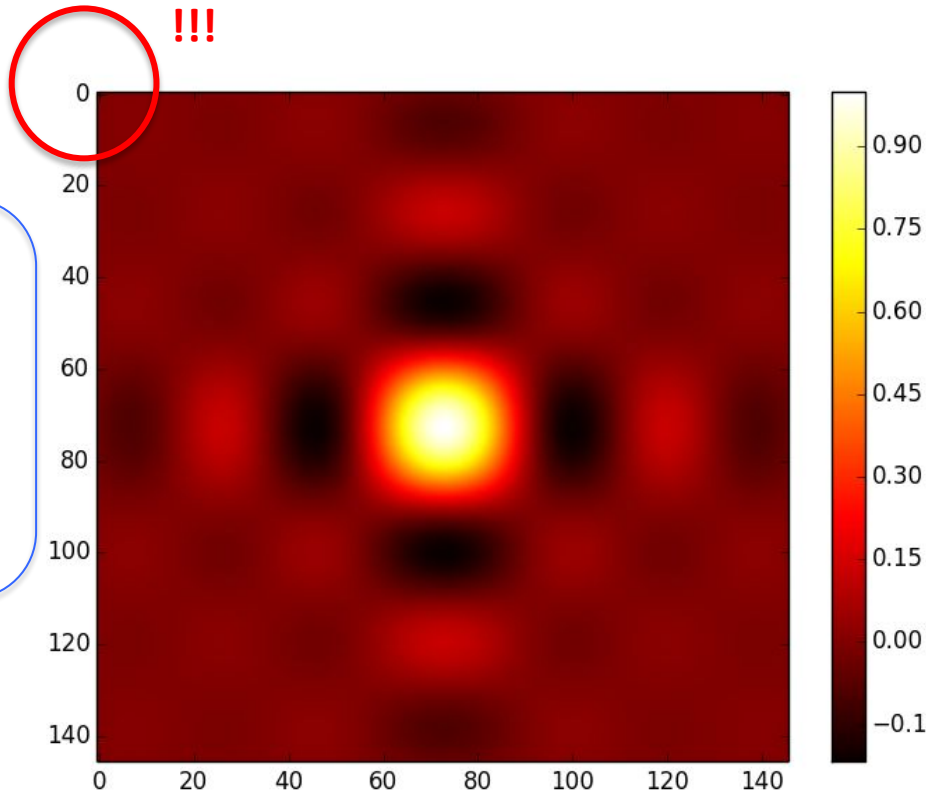
Visualiser une image

À partir de l'exemple (déjà vu) de fonction **`sinc`** en 2D :

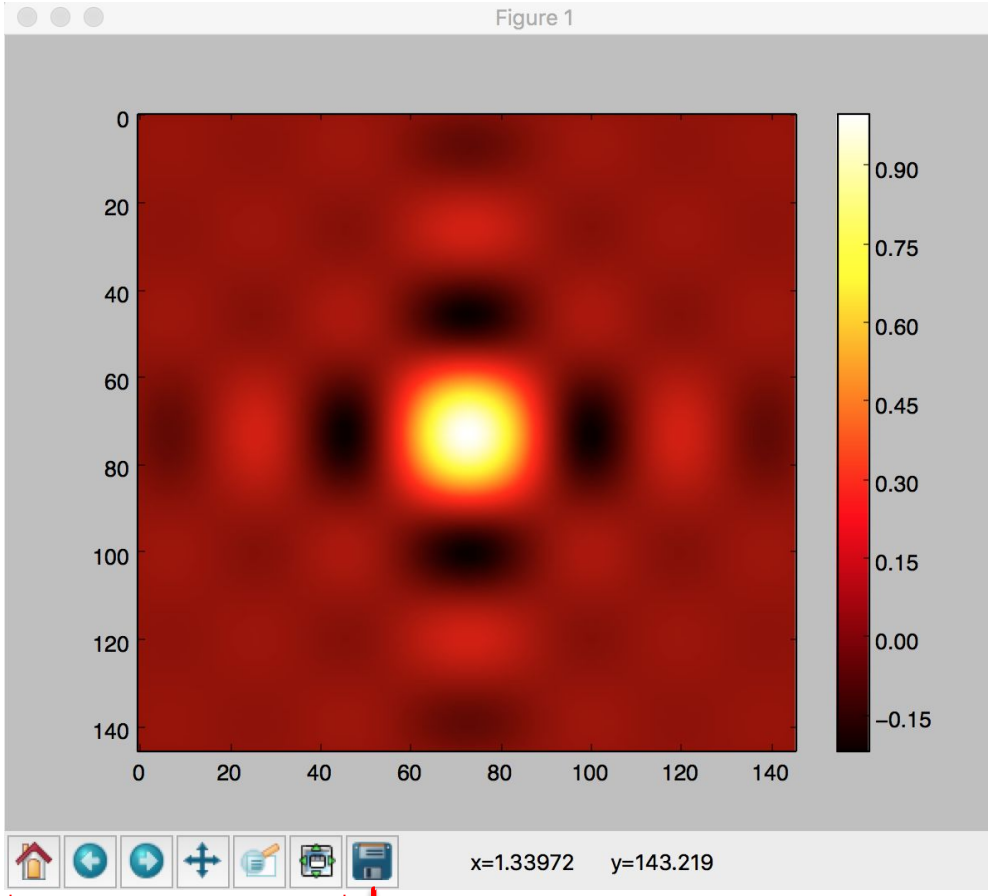
`sinc2(x,y) = sinc(x) * sinc(y) ;  x, y ∈ [-12, +12]`

(tableau créé plus haut)

```
#---display + (hot) colortable
#--- + colorbar
plt.hot()
plt.imshow(sinc2, aspect='auto')
plt.colorbar()
plt.show()
```



Exporter les graphiques



Manipulations interactives du graphique

Exporte un fichier **`png`**

(qui se souvient des disquettes 3.5" ?)

Voir aussi :

`savefig``()`

`@` **`matplotlib`**

Lire/traiter du PNG / JPG

```
import matplotlib.pyplot as plt

im=plt.imread('Sentinel.png')
print('Image shape is:', im.shape)

Image shape is: (10980, 10980, 3) #for RGB
```

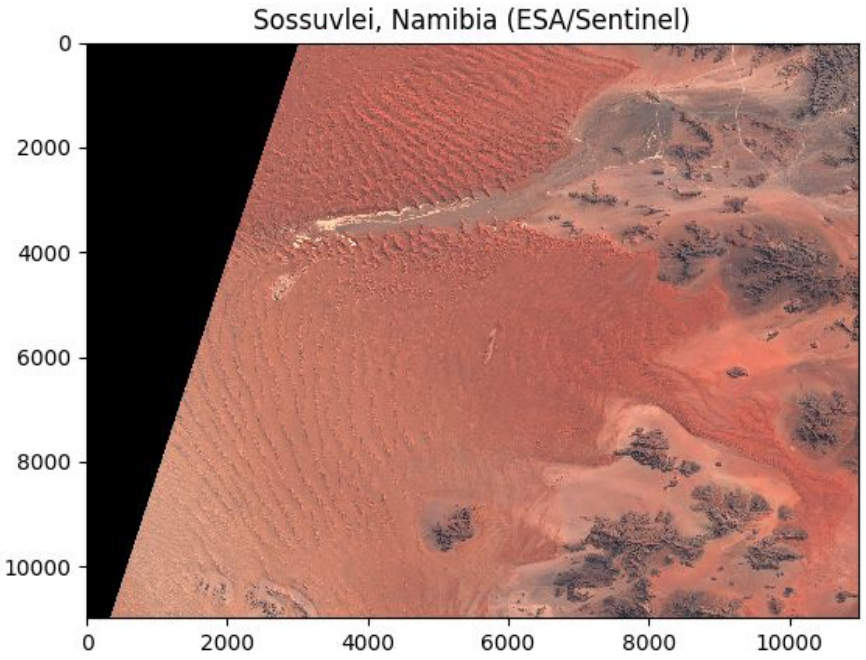
exemples simples

ReadSheldonPNG*.py

Visualisation d'une image ESA/Sentinel

cf. ReadSentinel*.py

+ jpeg / **jp2** (+ ***PIL*** + *installer les libjpeg adéquates)*



Animations : plot 2D

Observatoire Midi-Pyrénées OMP

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

x=np.arange(100)
y=np.random.rand(100)

for i in np.arange(10):
    if i == 0:
        # extract object from 1 elt. list!
        pp,=plt.plot(x,y)
        t=plt.gca().set_title("frame #"+str(i))
    else:
        y=np.random.rand(100)
        pp.set_ydata(y)
        t.set_text("frame #"+str(i))
    print("step", i)
    plt.pause(0.25)
```

Animations : images

Observatoire Midi-Pyrénées OMP

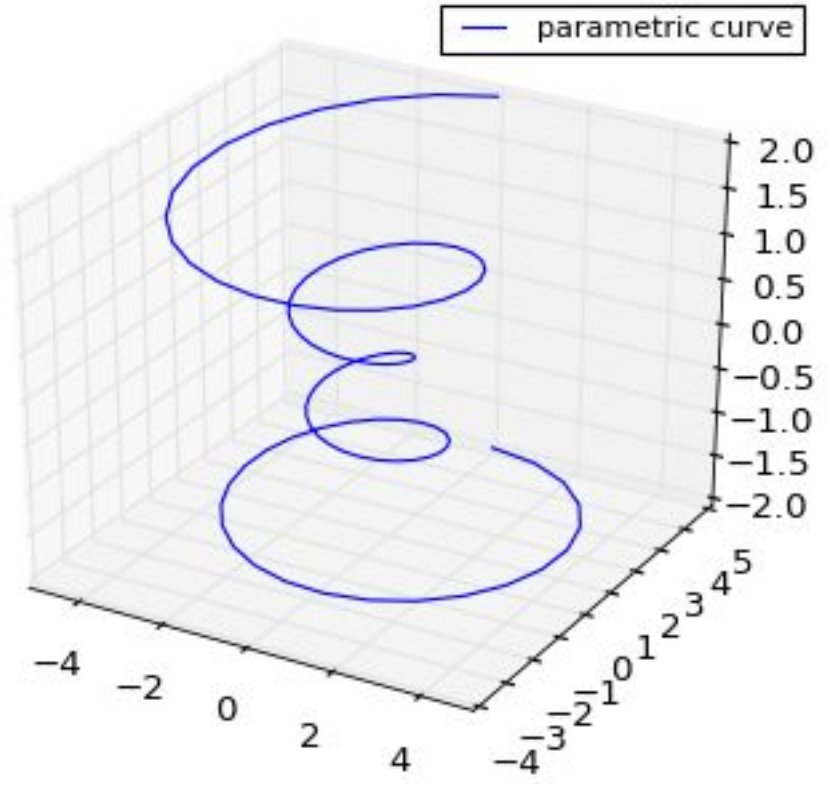
```
import numpy as np
import matplotlib.pyplot as plt

imas=np.random.rand(20,20)

for k in np.arange(10):
    if k == 0:
        m=plt.imshow(imas, aspect='auto',
interpolation='nearest', cmap='jet')
        t=plt.gca().set_title("frame #"+str(k))
    else:
        imas=np.random.rand(20,20)
        m.set_data(imas)
        t.set_text("frame #"+str(k))
    print("step",k)
    plt.pause(0.1)

                    ###- more @ matplotlib.animation
```

voir aussi :

MakeMP4.py

3D

`matplotlib` permet aussi de produire des représentations en 3D…



https://matplotlib.org/stable/gallery/mplot3d/index.html

Pickling

Ce module permet la **sauvegarde** ou encore l'**échange** de **résultats**, partiels ou définitifs, de façon très pratique *(similaire aux fichiers .sav d'IDL pour les… nostalgiques)*.

Il faut d'abord incorporer dans ses commandes : **`import pickle`**

```
flnm=open('sauvegarde.dat','wb')
pickle.dump(mon_objet,flnm)
flnm.close()
```

wb indique que l'on va **écrire** `(w)` en format **binaire** `(b)`

Mais **à vous** de transmettre convenablement le **format** des données sauvegardées (binaire) pour en assurer la bonne réutilisation...

La **relecture** se fera comme :

```
flnm=open('sauvegarde.dat','rb')
relecture=pickle.load(flnm, encoding='bytes')
flnm.close()
```

(Un)pickling

Observatoire Midi-Pyrénées OMP

Il existe des petites **différences** dans les commandes, en particulier le **load** en passant de Python 2 à **Python 3**...

Unpickling Python 2 objects in Python 3

You might sometimes come across objects that were pickled in Python 2 while running Python 3. This can be a hassle to unpickle.

You could either unpickle it by running Python 2, or do it in Python 3 with `encoding='latin1'` in the `load()` function.

```
infile = open(filename,'rb')
new_dict = pickle.load(infile, encoding='latin1')
```

This will not work if your objects contains NumPy arrays. In that case, you could also try using `encoding='bytes'`:

```
infile = open(filename,'rb')
new_dict = pickle.load(infile, encoding='bytes')
```

Données formatées

Observatoire Midi-Pyrénées OMP

`pyfits` est dédiée à la lecture/écriture de fichiers **`FITS`** (astro) :

https://pyfits.readthedocs.io/en/latest/

Il existe aussi des ressources spécifiques à d'autres formats de données d'usage courant dans les divers laboratoires de l'OMP :

- **HDF5 :** http://www.h5py.org/
- **NetCDF :** http://www.pyngl.ucar.edu/Nio.shtml
- **GDAL** : https://pypi.python.org/pypi/GDAL/
- **ObsPy** : https://docs.obspy.org/ etc...

os

Ce module permet d'exécuter des **commandes systèmes** sous python.

Un exemple simple :

```
>>> import os
>>> os.listdir(os.curdir)
['co2.py', 'CO2MaunaLoa.txt', 'def_simple.py', 'epsilon.py',
'import_ma_bib.py', 'ma_bib.py', 'SecDeg.py', 'sinc2D.py']
```

permet de récupérer les **noms de tous les fichiers** présents dans le répertoire courant (`curdir`) - dans cet exemple, une liste des ressources qui sont distribuées ici… - dans une **liste** manipulable.

Voir aussi, selon les besoins :

- **`sys`**
- **`time`**
- **…**

Aller plus loin…

- **Beaucoup d'utilisateurs apprécient l'environnement `jupyter`**
  - https://jupyter.org/ (et ses *notebooks*)

- **Visualisations alternatives/complexes**
  - http://yhat.github.io/ggpy/ (graphiques "à la R")
  - https://napari.org/stable/
  - https://docs.bokeh.org/en/latest/
  - https://docs.enthought.com/mayavi/mayavi/
  - …

- **Programmation objet**
  - https://courspython.com/classes-et-objets.html
  - autres ateliers : L. Risser (IMT) en 2017 notamment

- **Calcul parallèle**
  - http://www.parallelpython.com/
  - multi-**threading** (cf. F. Niño, Legos)
  - ...

Aller (encore) plus loin…

https://calcul.math.cnrs.fr/index.html

http://www.scipy-lectures.org/

https://www.labri.fr/perso/nrougier/

http://scikit-learn.org/stable/ **(machine learning)**

etc...

exemples : https://box.in2p3.fr/index.php/s/bqBNKH5zwz82ngQ