



DIRECTION DES SCIENCES DE LA MATIERE

Département de Recherche sur l'Etat Condensé, les Atomes et les Molécules

Laboratoire Interdisciplinaire sur l'Organisation Nanométrique et Supramoléculaire

# Initiation à l'utilisation scientifique de Python

# Plan

- Philosophie de Python
- Quelques éléments de syntaxe
  - Types
  - Conditions
  - Boucles
  - Fichiers
  - Fonctions
- Modules
  - Son propre module
  - Modules scientifiques
  - Tableaux
- Tracé de courbes
  - Introduction à Gnuplot
  - Interface avec Python

# Philosophie de Python

- Langage de programmation « simple » qui permet de se concentrer sur l'application scientifique et pas la syntaxe
- Interface avec d'autres langages (Fortran, C,...)
- Interfaçage avec de nombreuses bibliothèques graphiques (Tkinter par défaut)
- Portable (utilisable sous unix, mac, pc,...)
- Orienté Objet et donc évolutif
- Open Source et gratuit

# Philosophie de Python : Interpréteur de commande

additionner 2 valeurs

```
>>> 1 + 1
```

```
2
```

affecter une variable

```
>>> a = 1
```

```
>>> a
```

```
1
```

Chaîne de caractère

```
>>> s = "hello world"
```

```
>>> print s
```

```
hello world
```

On peut aussi créer un programme sans l'interpréteur

Il suffit de lancer l'interpréteur python :

```
Python.exe monprogramme.py
```



Calculatrice scientifique très puissante

# Philosophie de Python : Modulaire

- Python est orienté objet
- De nombreux modules sont disponibles :
  - Scientifique
  - Imagerie
  - Réseaux
  - Interfaces graphiques
  - ...
- Documentés facilement (grâce à pyDoc)

# Aspects du langage

- Indentation importante:

```
If n in range(1,10):  
    → Print n  
Print 'fini'
```

- Documentation
  - Commentaires précédés de #
  - Ou bien """

# Affectation des variables

- Pas de déclaration des variables

- Comment affecter une variable ?

```
>>>x=[1,2,3]
```

- Une variable se réfère à une zone mémoire

```
>>>Y=X
```

- Deux variables peuvent de référer à la même zone mémoire

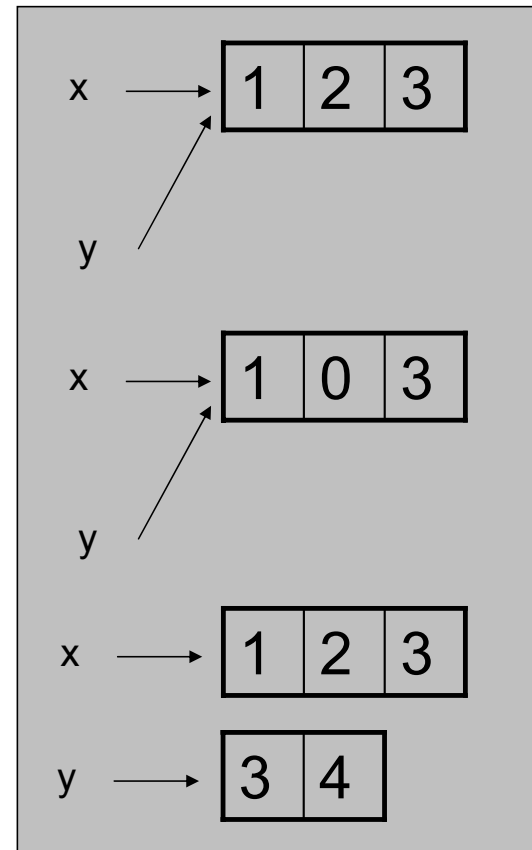
```
>>>X[1]=0
```

```
>>>Print y
```

```
[1,0,2]
```

- Réaffecter une variable :

```
>>>y=[3,4]
```



# booleens

- True ou False

```
>>>test=true
```

```
>>>if test :
```

```
    Print « vrai »
```

Vrai



# Nombres

Entier (int)

0, 1, 2, 3, -1, -2, -3

Real (float)

0., 3.1415926, -2.05e30, 1e-4

(doit contenir . ou exposant)

Complex

1j, -2.5j, 3+4j

- Addition

3+4, 42.+3, 1+0j

- soustraction

2-5, 3.-1, 3j-7.5

- Multiplication

4\*3, 2\*3.14, 1j\*3j

- Division

1/3, 1./3., 5/3j

- Puissance

1.5\*\*3, 2j\*\*2, 2\*\*-0.5

# Chaines de caractères (string)

- `'abc'` ou `'abc'`
- `'\n'` retour à la ligne
- `'abc'+ 'def'` `'abcdef'`
- `3*'abc'` `'abcabcabc'`
- `'ab cd e'.split()` `['ab', 'cd', 'e']`
- `'1,2,3'.split(',')` `['1', ' 2', ' 3']`
- `','.join(['1','2'])` `'1,2'` ajoute , entre les éléments
- `' a b c '.strip()` `'a b c'` enlève les espaces de fin et début
- `'text'.find('ex')` 1 #recherche
- `'Abc'.upper()` `'ABC'`
- `'Abc'.lower()` `'abc'`
- **Conversion vers des nombres:** `int('2')`, `float('2.1')`
- **Conversion vers du texte:** `str(3)`, `str([1, 2, 3])`

# Listes ou séquences (list)

- **Création**

```
>>>a=[1,2,3, 'blabla', [9,8]]
```

- **Concatenation (pas addition... voir array)**

```
>>>>[1,2,3]+[4,5]
```

```
[1,2,3,4,5]
```

- **Ajout d'un élément**

```
>>>a.append('test')
```

```
[1,2,3, 'blabla', [9,8], 'test']
```

- **Longueur**

```
>>>len(a)
```

```
7
```

- **range([start,] stop[, step]) -> list of integers**

Return a list containing an arithmetic progression of integers.

```
>>>range(5)
```

```
[0,1,2,3,4]
```

- **Indexation simple**

```
>>>a[0]
```

```
1
```

- **Indexation multiple**

```
>>>a[4][1]
```

```
8
```

- **Définition d'un élément**

```
>>> a[1]=1
```

```
>>> a
```

```
[1, 1, 3, 'blabla', [9, 8], 'test']
```

- **Indices négatifs**

```
>>> a[-1]
```

```
'test'
```

```
>>> a[-2]
```

```
[9, 8]
```

- **Parties d'une liste (liste[lower:upper])**

```
>>>a[1:3]
```

```
[2,3,'blabla']
```

```
>>> a[:3]
```

```
[1, 1, 3]
```

Les indices commencent toujours à 0

# >>>Help(list)

**append(...)**

L.append(object) -- append object to end

**count(...)**

L.count(value) -> integer -- return number of occurrences of value

**extend(...)**

L.extend(iterable) -- extend list by appending elements from the iterable

**index(...)**

L.index(value, [start, [stop]]) -> integer -- return first index of value

**insert(...)**

L.insert(index, object) -- insert object before index

**pop(...)**

L.pop([index]) -> item -- remove and return item at index (default last)

**remove(...)**

L.remove(value) -- remove first occurrence of value

**reverse(...)**

L.reverse() -- reverse \*IN PLACE\*

**sort(...)**

L.sort(cmpfunc=None) -- stable sort \*IN PLACE\*; cmpfunc(x, y) -> -1, 0, 1

Usage :

```
>>>Maliste.methode(var)
```

# Dictionnaires

## Dictionnaire : Association d'une valeur et d'une clé.

- **Création d'un dictionnaire**

```
>>> dico = {}
>>> dico['C'] = 'carbone'
>>> dico['H'] = 'hydrogène'
>>> dico['O'] = 'oxygène'
>>> print dico
{'C': 'carbone', 'H': 'hydrogène', 'O': 'oxygène'}
```

- **Utilisation**

```
>>>print dico['C']
Carbone
```

- **Création d'un nouveau dictionnaire**

```
>>> dico2={'N':'Azote','Fe':'Fer'}
```

- **Concaténation de dictionnaires**

```
>>> dico3=dico+dico2
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#18>", line 1, in -toplevel-
    dico3=dico+dico2
```

```
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

```
>>> dico.update(dico2)
```

```
>>> dico
```

```
{'H': 'hydrogène', 'C': 'carbone', 'Fe': 'Fer', 'O': 'oxygène', 'N': 'Azote'}
```

# >>>help(dict)

## **clear(...)**

D.clear() -> None. Remove all items from D.

## **copy(...)**

D.copy() -> a shallow copy of D

## **get(...)**

D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.

## **has\_key(...)**

D.has\_key(k) -> True if D has a key k, else False |

## **items(...)**

D.items() -> list of D's (key, value) pairs, as 2-tuples

## **iteritems(...)**

D.iteritems() -> an iterator over the (key, value) items of D

## **iterkeys(...)**

D.iterkeys() -> an iterator over the keys of D

## **itervalues(...)**

D.itervalues() -> an iterator over the values of D

## **keys(...)**

D.keys() -> list of D's keys

## **pop(...)**

D.pop(k[,d]) -> v, remove specified key and return the corresponding value  
If key is not found, d is returned if given, otherwise  
KeyError is raised

## **popitem(...)**

D.popitem() -> (k, v), remove and return some (key, value)  
pair as a  
2-tuple; but raise KeyError if D is empty

## **setdefault(...)**

D.setdefault(k[,d]) -> D.get(k,d), also set D[k]=d if k not in D

## **update(...)**

D.update(E) -> None. Update D from E: for k in E.keys():  
D[k] = E[k]

## **values(...)**

D.values() -> list of D's values

# Conditions

`if`

`<condition>:`

`<code>`

`elif`

`<condition>:`

`<code>`

`else:`

`<code>`

## Exemples :

```
test=True
if test:
    print 'vrai'
else:
    print 'faux'
->vrai
```

```
if i==1:
    print 'A'
elif i==2:
    print 'B'
elif i==3:
    print 'C'
else:
    print "?"
```

# Boucles for loop

**for <variable> in <list>:  
<code>**

```
>>> for i in range(5):  
    print i,  
0 1 2 3 4  
  
>>> for i in 'abcdef':  
    print i,  
a b c d e f  
  
l=['C','H','O','N']  
>>> for i in l:  
    print i,  
C H O N
```

```
>>> for i in dico:  
    print i,  
  
H C Fe O N  
>>> for i in dico:  
    print dico[i],  
  
hydrogène carbone Fer  
oxygène Azote
```



# Boucles while

**while <condition>:**  
**<instructions>**

Exécution répétée d'instructions en fonction d'une condition

```
>>> test=0
>>> while test<10:
    test=test+1
    print test,
```

1 2 3 4 5 6 7 8 9 10

```
>>> i = 0
>>> while 1:
    if i<3 :
        print i,
    else :
        break
    i=i+1
```

0 1 2

# Lecture des fichiers textes

- Rappel sur les fichiers textes :
  - Ouverture du fichier
  - Lecture ligne à ligne
  - La ligne est affectée à une variable texte
  - Fermeture du fichier

- En Python :

```
>>>f=open('nom_du_fichier.txt','r')
>>>lignes=f.readlines()
>>>f.close()
```

L'instruction `readlines` affecte toutes les lignes du fichier dans une liste.

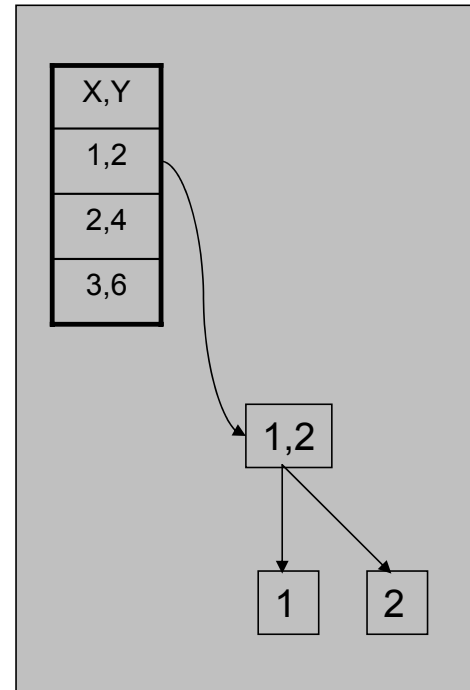
- Chaque élément de la liste doit désormais être traité.

*Dans le fichier il y a 2 colonnes x et y séparés par une ,*

```
>>>x=[ ] #liste
```

```
>>>y=[ ]
```

```
>>>for chaine in lignes[1:]:On ne prend pas la 1ere ligne
    elements=chaine.split(',')
    x.append(float(element[0]))
    y.append(float(element[1]))
```



# Écriture des Fichiers

- Rappel sur les fichiers textes :
  - Ouverture du fichier
  - écriture ligne à ligne
  - Fermeture du fichier

- En Python :

```
>>>f=open('nom_du_fichier.txt','w')
>>>f.write('ligne1\n')
>>>f.write('ligne2\n')
>>>f.close()
```

# Fonctions

- **Syntaxe :**

```
def nomdelafunction(arguments):  
    instructions  
    return quelquechose
```

```
>>> def addition(a,b):  
        c=a+b  
        return c  
  
>>> addition(2,3)  
5  
>>> addition('nom', 'prenom')  
'nomprenom'  
>>> addition(1.25, 5.36)  
6.6100000000000003
```

# Modules

- Créer son propre module:

Exo1.py :

```
def addition(a,b):  
    c=a+b  
    return c
```

- Utiliser un module

```
>>>import exo1  
>>>exo1.addition(1,2)  
3
```

- Recharger le module

```
>>>reload(exo1)
```

- Utiliser quelques parties d'un module

```
>>>from Numeric import array
```

- Utiliser tous les composants d'un module

```
>>>from Numeric import *  
>>>import Numeric as N  
>>> N.sin(3.14)  
0.0015926529164868282
```

```
Un module qui s'exécute :  
# Ajouter ce code à la fin  
if __name__ == '__main__':  
    test()
```

Plus propre parce que l'on sait quelle module est utilisé...

# Modules numeric et scientifique

- Numerical Python

- Nouveau type array (tableau)
- Fonctions mathématiques universelles permettant de les manipuler

```
>>>import Numeric
```

- Scientific Python

- Collection de modules pour le calcul scientifique puissantes et complètes

```
>>>import Scientific
```

- Scipy

- « Fourre-tout » de modules scientifiques
- Fonctionnalités de plotting (?!)
- Interface genre matlab

# Fonctions de Scientific

- PACKAGE CONTENTS
- BSP (package)
- DictWithDefault
- Functions (package)
- Geometry (package)
- IO (package)
- Installation
- MPI (package)
- Mathematica
- NumberDict
- Physics (package)
- Signals (package)
- Statistics (package)
- Threading (package)
- TkWidgets (package)
- Visualization (package)

# Type tableau (array)

- **Attention, c'est différent d'une liste (sequence)**

- **Création d'un tableau :**

```
>>> N.array([1,2,3,4])
array([1, 2, 3, 4])
>>> N.array([1,2,3,4.])
array([ 1.,  2.,  3.,  4.])
>>> N.arange(0,20,2)
array([ 0,  2,  4,  6,  8, 10,
       12, 14, 16, 18])
```

- **Création d'un tableau à partir d'une liste**

```
>>> l=range(1,10)
>>> a=N.array(l)
>>> a
array([1, 2, 3, 4, 5, 6, 7, 8,
       9])
```

- **Addition de tableaux**

```
>>> t=N.array([1,2,3,4.])
>>> t2=N.array([1,2,3,4.])
>>> t+t2
array([ 2.,  4.,  6.,  8.])
```

- **Utilisation de fonctions mathématiques**

```
>>> t=N.arange(0,10,2)
>>> t
array([0, 2, 4, 6, 8])
>>> t*N.pi
array([0., 6.28318531, 12.56637,
       18.84955592, 25.13274123])
```



# Type tableau (array)

- Tableaux à plusieurs dimensions

```
>>> t=N.arange(0,15)
>>> t.shape
(15,)
>>> tableau2d=N.reshape(t,(3,5))
>>> tableau2d
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

- Accès aux données

```
>>> tableau2d[0,3]
3
```

**Attention tableau[y,x] !!!**

- Parties

```
>>> tableau2d[:,3]
array([ 3,  8, 13])
```

# Fonctions array

## Functions

- array - NumPy Array construction
- zeros - Return an array of all zeros
- shape - Return shape of sequence or array
- rank - Return number of dimensions
- size - Return number of elements in entire array or a certain dimension
- fromstring - Construct array from (byte) string
- take - Select sub-arrays using sequence of indices
- put - Set sub-arrays using sequence of 1-D indices
- putmask - Set portion of arrays using a mask
- reshape - Return array with new shape
- repeat - Repeat elements of array
- choose - Construct new array from indexed array tuple
- cross\_correlate - Correlate two 1-d arrays
- searchsorted - Search for element in 1-d array
- sum - Total sum over a specified dimension
- average - Average, possibly weighted, over axis or array.
- cumsum - Cumulative sum over a specified dimension
- product - Total product over a specified dimension
- cumproduct - Cumulative product over a specified dimension
- alltrue - Logical and over an entire axis
- sometrue - Logical or over an entire axis
- allclose - Tests if sequences are essentially equal

help(Numeric)

help(function)

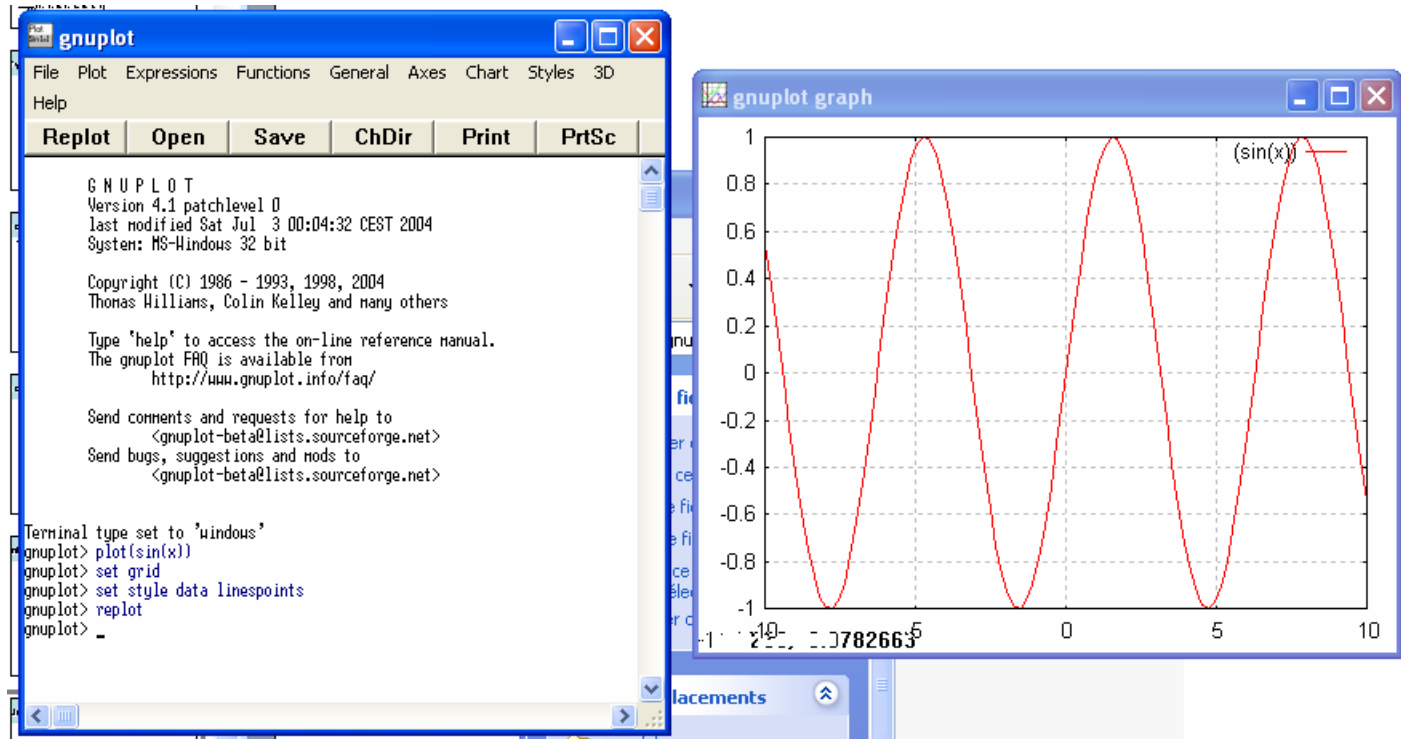
## More Functions:

- arange (arange) - Return regularly spaced array
- asarray - Guarantee NumPy array
- sarray - Guarantee a NumPy array that keeps precision
- convolve - Convolve two 1-d arrays
- swapaxes - Exchange axes
- concatenate - Join arrays together
- transpose - Permute axes
- sort - Sort elements of array
- argsort - Indices of sorted array
- argmax - Index of largest value
- argmin - Index of smallest value
- innerproduct - Innerproduct of two arrays
- dot - Dot product (matrix multiplication)
- outerproduct - Outerproduct of two arrays
- resize - Return array with arbitrary new shape
- indices - Tuple of indices
- fromfunction - Construct array from universal function
- diagonal - Return diagonal array
- trace - Trace of array
- dump - Dump array to file object (pickle)
- dumps - Return pickled string representing data
- load - Return array stored in file object
- loads - Return array from pickled string
- ravel - Return array as 1-D
- nonzero - Indices of nonzero elements for 1-D array
- shape - Shape of array
- where - Construct array from binary result
- compress - Elements of array where condition is true
- clip - Clip array between two values
- zeros - Array of all zeros
- ones - Array of all ones
- identity - 2-D identity array (matrix)

# Introduction à Gnuplot

- Gnuplot n'est pas un programme Python
- Gnuplot est un logiciel permettant de tracer des courbes (correctement !)
- En ligne de commande → pas très convivial
- Charge des données issues de fichiers et permet certains calculs
- Permet d'automatiser le tracé des figures
- Dispose d'un grand nombre de types de figures
- Gratuit et OpenSource
- Multiplateforme (windows, linux, mac)
- → Même philosophie que Python
- Python peut interfacer Gnuplot → le tracé est dédié à Gnuplot

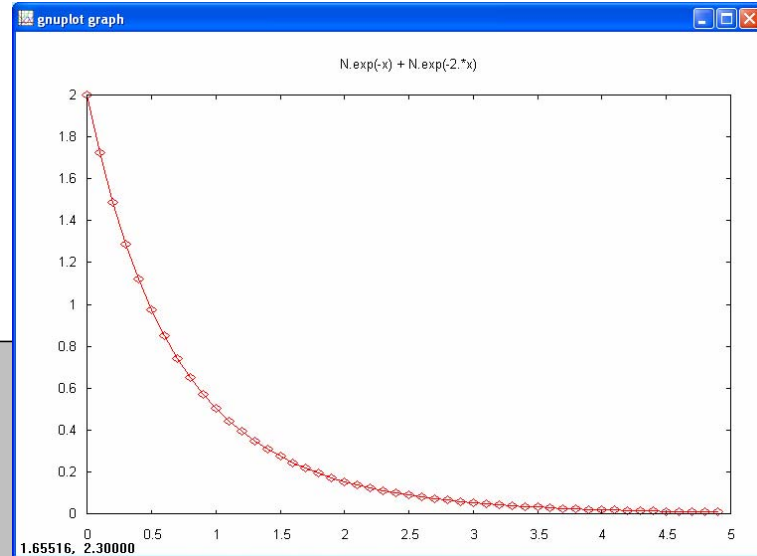
# Une courbe avec Gnuplot



# Interfaçage avec Python

Utilisation du module  
Gnuplot.py

```
import Gnuplot
import Numeric as N
# Initialize Gnuplot
g = Gnuplot.Gnuplot()
# Set up data
x = N.arange(0., 5., 0.1)
y = N.exp(-x) + N.exp(-2.*x)
curve=Gnuplot.Data(x, y,with='linespoints')
g.title('N.exp(-x) + N.exp(-2.*x)')
g.plot(curve)
```



# Références

- [www.python.org](http://www.python.org)
- Numerical Python (<http://numeric.scipy.org/>)
- Scientific Python (Konrad Hinsén)  
<http://starship.python.net/~hinsén/ScientificPython/>
- [www.SciPy.org](http://www.SciPy.org)
- [www.gnuplot.info](http://www.gnuplot.info)
- Transparents inspirés de “Introduction to Scientific Computing with Python”
- [www-drecami.cea.fr/scm/lions/python](http://www-drecami.cea.fr/scm/lions/python)