

Mémo python

pour l'ingénierie numérique

B.A. BA

`a=` affecte le contenu de droite à `a`
`a,b=b,a` échange les variables (*swap*)
`type(a)` affiche le type de `a`
`id(a)` identifiant de `a` (place mémoire)
`del(a)` supprime `a` de la mémoire
`print(a)` affiche `a` dans le shell
`dir()` espace mémoire
`none` type vide

Tests, booléens

`True, False` vrai (≥ 1), faux (0)
`==, !=` égal, différent
`not(a)` NON ($\neg a$)
`a and b, a & b` ET logique ($a \wedge b$)
`a or b, a | b` OU logique ($a \vee b$)
`a xor b, a ^ b` ou exclusif ($a \oplus b$)

Branchements conditionnels

```
if condition:
    bloc instructions
elif autre_condition:
    bloc instructions
else:
    bloc instructions
```

Boucle

```
for e in s:
    bloc instructions
    if condition:
        ...
        break # pour quitter la boucle
```

Boucle conditionnelle

```
while condition:
    bloc instructions
```

Nombres

`int(x)` conversion en entier `int` (rep. exacte de \mathbb{Z})
`float(x)` conversion en flottant `float` (rep. approchée de \mathbb{R} limitée à $[-1,7 \cdot 10^{308}; 1,7 \cdot 10^{308}]$)
`complex(re,im)` conversion de 2 flottants en `complex` (rep. approchée de \mathbb{C} avec 2 flottants)
`round(x,a)` arrondit `x` à 10^{-a} près
`inf, nan` infini, *Not A Number*

Compareurs

`==, !=` égal, différent
`<, >` strictement inférieur, supérieur
`<=, >=` inférieur/supérieur ou égal

Opérations élémentaires

`x + y, x - y` somme, différence
`x * y, x / y` produit, division
`x**y` ou `pow(x,y)` puissance

`pow(x,y,z)` puissance modulaire ($x^y [z]$)
`x // y, x % y` quotient et reste de la division euclidienne
`divmod(x,y)` quotient et reste de la division euclidienne
`a+=` \iff `a=a+` fonctionne avec `-`, `*` et `/`

Séquences

`[1, 2, 3]` `list` objet dynamique
`(1, 2, 3)` `tuple` objet immuable
`"chaîne"` `str` objet immuable

`len(s)` nombre d'éléments de `s`
`s[i]` élément en position $i \in \llbracket 0, \text{len}(s) - 1 \rrbracket$
`s[i:j]` coupe des éléments d'indice i à $j-1$
`s[i:j:k]` coupe des éléments d'indice i à $j-1$ par pas de `k`
`Out Of Range` si indice $\geq \text{len}(s)$

Opérations élémentaires

`s + t` concaténation de 2 séquences
`n * s` recopie $n \in \mathbb{N}$ fois la séquence `s`
`x in s` test d'appartenance de `x` à `s`
`s.index(x)` position de `x` dans `s`
`s.count(x)` nombre d'occurrences de `x` dans `s`
`sorted(s)` copie de la séquence triée
`reversed(s)` copie de la séquence en ordre inverse

Parcours exhaustif

`for i in range(len(s)):` par indice, l'objet est `s[i]`
`for e in s:` par élément, l'objet est `e`
`for i,e in enumerate(s):` les 2, avec `e==s[i]`
`for x,y in zip(s,t):` 2 séquences de même longueur
`for i,x,y in izip(count(),s,t):` avec un compteur plus rapide que `for i,(x,y) in enumerate(zip(s,t)):`

Listes

`L=[e for e in s]` définition par compréhension
`L.append(x)` ajoute l'élément `x` à la fin de la liste
`L.insert(i, x)` insère l'élément `x` à l'indice `i`
`L.extend(s)` ajoute la séquence `s` à la fin de la liste
`L.remove(x)` supprime les éléments `x`
`L.pop(i)` supprime l'élément d'indice `i` (par défaut le dernier) et le renvoie
`L.sort()` trie la liste
`L.reverse()` inverse l'ordre des éléments de la liste
`list(s)` convertit la séquence `s` en liste

Tuples

`t=(1,)` définit un tuple de 1 élément
`tuple(s)` convertit la séquence `s` en tuple

Chaînes de caractères

`s=""` définit une chaîne vide
`str(num)` convertit un nombre en chaîne
`s.split(x)` découpe la chaîne `s` sur le caractère `x`
`s.strip()` supprime les caractères EOL, EOF
`s.replace(a,b)` remplace la sous-chaîne `a` par `b`
`\n, \t` retour à la ligne, tabulation
`chr, ord` code Unicode d'un caractère, réciproque

Séquences d'entiers

`range(n)` séquence d'entiers de $\llbracket 0, n - 1 \rrbracket$
`range(a,b)` séquence d'entiers de $\llbracket a, b - 1 \rrbracket$
`range(a,b,h)` //... de $\llbracket a, b - 1 \rrbracket$ par pas de h
`list(range(n))` liste des n entiers de $\llbracket 0, n - 1 \rrbracket$

Autres types

Ensembles

`s={1, 2, 3}` ensemble
`set()` ensemble vide
`set(s)` conversion de séquence en ensemble
`frozenset(s)` conversion en ensemble immuable

Opération élémentaires

`s.add(e)` ajoute l'élément e
`s.update(t)` ajoute les éléments de la séquence s
`s.discard(e)` supprime l'élément e
`s.clear()` supprime tous les éléments de s
`s & t` intersection de 2 ensembles ($s \cap t$)
`s | t` union de 2 ensembles ($s \cup t$)
`s - t` différence de 2 ensembles ($s \setminus t$)
`s ^ t` différence symétrique ($s \Delta t = t - s | s - t$)

Dictionnaires

`d={}` dictionnaire vide
`d={"cle1":val1, "cle2": val2}` dictionnaire
`d.keys(), d.values()` clés, valeurs
`d.items()` couples (clés, valeurs)
`d["cle"], d["cle"]=val` affiche, affecte la valeur
`d.update(e)` ajoute les couples de e à d
`d.pop(cle)` supprime le couple associé à cle
`d.clear()` vide le dictionnaire

Fonctions mathématiques

Nombres réels

`import math as ma` chargement du module `math`
`abs(x)` $|x|$
`ma.pi` π
`ma.sqrt(x)` \sqrt{x}
`ma.floor(x)` $\lfloor x \rfloor$
`ma.factorial(x)` $x!$
`ma.exp(x), ma.log(x)` $\exp^x, \ln(x)$
`10**x, ma.log10(x)` $10^x, \log(x)$
`ma.cos(x), ma.acos(x)` $\cos(x), \text{Arccos}(x)$
`ma.sin(x), ma.asin(x)` $\sin(x), \text{Arcsin}(x)$
`ma.tan(x), ma.atan(x)` $\tan(x), \text{Arctan}(x)$
`ma.gcd(x,y)` PGCD(x, y)

Nombres complexes

`z = 8+3j` ou `z = complex(8,3)` nombre complexe
`z.real` partie réelle
`z.imag` partie imaginaire
`abs(z)` module
`z.conjugate` conjugué

`import cmath as cm` chargement du module `cmath`
`cm.phase(z)` $\arg z$
`cm.exp(z), cm.log(z)` $\exp^z, \ln(z)$
`10**z, cm.log10(z)` $10^z, \log(z)$

`cm.cos(x), cm.acos(x)` $\cos(x), \text{Arccos}(x)$
`cm.sin(x), cm.asin(x)` $\sin(x), \text{Arcsin}(x)$
`cm.tan(x), cm.atan(x)` $\tan(x), \text{Arctan}(x)$
`cm.polar(z)` coordonnées cartésiennes \rightarrow polaires
`cm.rect(r, phi)` coordonnées polaires \rightarrow cartésiennes

Tableaux

`import numpy as np` chargement du module `numpy`

`T.dim` nombre de dimensions (1 vecteur, 2 matrices)
`T.shape` ou `np.shape(T)` nombre d'éléments par dim.
`T.size` nombre total d'éléments (produit des dim.)
`N = T.copy()` recopie des éléments \rightarrow nouveau tableau
`T.dtype()` type des éléments du tableau
(`'int8'`, `'int64'`, `'float64'`, `'complex128'`, ...)
`N = T.astype('float64')` changement de type
des éléments vers flottant
`T.tolist()` conversion en liste

Fonctions : $T \mapsto$ tableau de mêmes dimensions que T

`np.abs(T)` $\{|x| : x \in T\}$
`np.floor(T)` $\{\lfloor x \rfloor : x \in T\}$
`np.sqrt(T)` $\{\sqrt{x} : x \in T\}$
`np.power(T,n)` $\{x^n : x \in T\}$
`np.exp(T)` $\{\exp(x) : x \in T\}$
`np.log(T)` $\{\ln(x) : x \in T\}$
`np.log10(T)` $\{\log(x) : x \in T\}$
`np.cos(T)` $\{\cos(x) : x \in T\}$
`np.arccos(T)` $\{\text{Arccos}(x) : x \in T\}$
`np.sin(T)` $\{\sin(x) : x \in T\}$
`np.arcsin(T)` $\{\text{Arcsin}(x) : x \in T\}$
`np.tan(T)` $\{\tan(x) : x \in T\}$
`np.arctan(T)` $\{\text{Arctan}(x) : x \in T\}$

Analyses de données option : 0/colonne, 1/ligne

`T.min()` ou `np.amin(T)` minimum
`T.max()` ou `np.amax(T)` maximum
`T.mean()` ou `np.mean(T)` moyenne
`T.std()` ou `np.std(T)` écart-type
`T.sum()` ou `np.sum(T)` somme
`T.cumsum()` ou `np.cumsum(T)` somme cumulée
`T.cumprod()` ou `np.cumprod(T)` produit cumulé

Tableaux de nombres complexes

`T.real` partie réelle $\{\Re(z) : z \in T\}$
`T.imag` partie imaginaire $\{\Im(z) : z \in T\}$
`T.angle` argument $\{\arg(z) : z \in T\}$
`T.conjugate()` conjugué $\{\bar{z} : z \in T\}$

Vecteurs (tableaux 1D)

`u = np.array([1,2,3,4])` vecteur
`u.shape[0], len(u)` nombre d'éléments
`u[i:j]` coupe des éléments d'indice i à $j-1$
`u[i:j:k]` coupe des éléments d'indice i à $j-1$ par pas de k
`np.zeros(n)` vecteur nul de \mathbb{R}^n
`np.ones(n)` vecteur rempli de 1 de \mathbb{R}^n

Subdivisions régulières

`np.linspace(a,b,n)` n valeurs équiréparties dans $[a, b]$
`np.logspace(a,b,n)` n valeurs logarithmiquement
équiréparties dans $[10^a, 10^b]$

`np.arange(a,b,h)` $\left[\frac{b-a}{h} \right]$ valeurs équiréparties
par pas de h dans $[a, b[$

Calcul vectoriel

`2*u` multiplication par un réel
`u+v` somme de deux vecteurs u et v de \mathbb{R}^n
`np.vdot(u,v)` produit scalaire de deux vecteurs de \mathbb{R}^n
`np.cross(u,v)` ... produit vectoriel de deux vecteurs de \mathbb{R}^3

Matrices (tableaux 2D)

`A = np.array([[1, 2, 3], [4, 5, 6]])` matrice
`np.diag([1,2,3])` matrice diagonale
`np.zeros((p,q))` matrice nulle de $\mathcal{M}_{p,q}(\mathbb{R})$
`np.zeros((p,q), dtype='complex')` // de $\mathcal{M}_{p,q}(\mathbb{C})$
`np.eye(n)` matrice identité de $\mathcal{M}_n(\mathbb{R})$
`np.ones((p,q))` matrice de $\mathcal{M}_{p,q}(\mathbb{R})$ remplie de 1

`A[i,j]` élément de la ligne i et de la colonne j
`A[i,:]` ou `A[i]` vecteur correspondant à la ligne i
`A[:,j]` vecteur correspondant à la colonne j
`A[a:b,c:d]` sous-matrice formée des lignes a à $b-1$
et des colonnes c à $d-1$

Parcours exhaustif

```

p,q=A.shape
for i in range(p):
    for j in range(q):
        A[i,j]
for ligne in A:
    for e in ligne:
        e

```

Opérations

`2*A` multiplication par un réel
`A+B` somme de deux matrices de $\mathcal{M}_{p,q}(\mathbb{K})$
`np.dot(A,B)` produit matriciel (si possible)
`np.transpose(A)` ou `A.T` transposée

Opérations particulières

`A.reshape((u,v))` redimensionner $A \in \mathcal{M}_{p,q}(\mathbb{R})$
vers $\mathcal{M}_{u,v}(\mathbb{R})$ avec $pq = uv$
`np.concatenate((A,B), axis=0)` .. superposer 2 matrices
de même nombre de colonnes
`np.concatenate((A,B), axis=1)` .. juxtaposer 2 matrices
de même nombre de lignes

Algèbre linéaire

des matrices carrées de $\mathcal{M}_n(\mathbb{R})$
`import numpy.linalg as alg` chargement du module
`alg.det(A)` déterminant
`alg.matrix_rank(A)` rang
`np.trace(A)` trace
`alg.inv(A)` inverse
`alg.solve(A, b)` solution x de $Ax = b$
`np.poly(A)` polynôme caractéristique
`alg.eig(A)` vecteurs propres et valeurs propres
`alg.eigvals(A)` valeurs propres
`alg.eig(A)[1]` ... vecteurs propres (colonne de la matrice)

Tracés et graphiques

`import matplotlib.pyplot as plt`
`plt.figure()` définition d'une nouvelle figure
`plt.subplot(nlignes, ncolonnes, num)` sous-figure
parcours par ligne avec `num` ∈ $\llbracket 1, nlignes \times ncolonnes \rrbracket$

`plt.plot(X,Y)` graphe « classique »
`plt.semilogx(X,Y)` ... échelle log pour l'axe des abscisses
`plt.semilogy(X,Y)` ... échelle log pour l'axe des ordonnées
`plt.loglog(X,Y)` échelle log pour les axes
`plt.xlim(xmin,xmax)` limites de l'axe des abscisses
`plt.ylim(ymin,ymax)` limites de l'axe des ordonnées
`plt.axis("equal")` échelle des axes commune
`plt.xlabel("")` légende de l'axe des abscisses
`plt.ylabel("")` légende de l'axe des ordonnées
`plt.title("")` titre du graphe
`plt.legend()` affiche la légende
nom défini pour chaque courbe avec `label=""`
`plt.grid()` affiche une grille
`plt.show()` affichage (du buffer) de la figure
`plt.savefig(fichier)` sauve le tracé dans fichier

Options graphiques

- trait : '-' plein, ':' pointillés, '-.' alterné
- couleur : 'k' noir, 'r' rouge, 'g' vert, 'b' bleu 'c' cyan, 'm' magenta, 'y' jaune
- marque : '+' plus, 'x' croix, '*' étoile, 'o' rond, 'd' diamant, 'h' hexagone

Fonction

```
plt.plot(X, [f(x) for x in X], '-b')
```

Nuages de points

```
plt.plot(X, Y, '*r')
plt.show()
```

Arc paramétré dans l'espace

```
from mpl_toolkits.mplot3d import Axes3D
ax = Axes3D(plt.figure())
ax.plot(X, Y, Z)
plt.show()
```

Surfaces

```
X, Y = np.meshgrid(X, Y)
ax.plot_surface(X, Y, Z)
plt.show()
```

Lignes de niveaux

```
X, Y = np.meshgrid(X, Y)
plt.contour(X, Y, Z, [vect niveaux])
plt.show()
```

Histogramme

```
hist, bins = np.histogram(x, bins=50)
center = (bins[:-1] + bins[1:]) / 2
plt.bar(center, hist, align='center')
plt.show()
```

Analyse numérique

`import scipy` chargement du module `scipy`

Résolution approchée d'équations

`import scipy.optimize as resol`
Pour résoudre une équation du type $f(x) = 0$ où f est une fonction d'une variable réelle

```
sol = resol.fsolve(f,x0)
```

Dans le cas d'une fonction f à valeurs vectorielles

```
sol = resol.root(f,X0)
sol.success # vraie si trouvée
sol.x # vecteur solution
```

Calcul approché d'intégrales

```
import scipy.integrate as integr
integr.quad(f,a,b) ..... intégrale de  $f$  entre  $a$  et  $b$ 
Noter np.inf pour  $\infty$ 
```

Résolution approchée d'équations différentielles

Pour résoudre une équation différentielle $X' = f(X, t)$ pour $t \in [a, b]$ par pas de h avec $X(a) = X_0$ comme condition initiale

```
import scipy.integrate as integr
def f(X,t):
    return(...)
T = np.arange(a,b,h)
X = integr.odeint(f,X0,T)
```

Probabilités

```
import numpy.random as rd
rd.randint(a,b) ..... tirage aléatoire dans  $[a, b[$ 
rd.randint(a,b,n) .....  $n$  tirages aléatoires dans  $[a, b[$ 
rd.randint(a,b,(p,q)) matrice  $p \times q$  de tirages aléatoires
rd.random(a,b) ..... tirage aléatoire dans  $[a, b[$ 
rd.binomial(n,p) ..... tirage suivant une loi binomiale

$$\mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

rd.geometric(p) ..... tirage suivant une loi géométrique

$$\mathbb{P}(X = k) = (1-p)^{k-1} p$$

rd.poisson(lam) ..... tirage suivant une loi de Poisson

$$\mathbb{P}(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

```

IO

Répertoire de travail

```
import os
os.chdir("chemin") ..... changer de répertoire
si MS Windows, remplacer chaque backslash \ par une paire \\
os.listdir() ..... lister le contenu du répertoire courant
os.path.join(dir,file) ..... concaténer les chaînes dir
et file avec le séparateur de l'OS os.path.sep
```

Fichiers texte

```
f=open(fichier,'r') .... ouvrir un fichier texte en lecture
ou 'w' écriture, 'a' ajout
f.readline() ..... lit la ligne suivante
```

```
f.read() ... lit tout le fichier si nb de caractères non donné
f.write(chaine) ..... écrit dans le fichier
f.writeline(chaine) ..... écrit une nouvelle ligne
f.close() ..... fermer le fichier
```

Lire un fichier texte

de 3 colonnes t, x, y séparées par des tabulations :

```
t,x,y=[],[],[]
L=[t,x,y]
fichier = open('fichier.txt','r')
for ligne in fichier:
    champs=ligne.strip().split('\t')
    for i in range(3):
        L[i].append(float(champs[i]))
fichier.close()
```

Écrire dans un fichier texte

```
fichier = open('fichier.txt','w')
for e in s:
    fichier.write(str(nombre)+"chaine"+"\\n")
fichier.close()
```

Définir un tableau avec les colonnes d'un fichier

```
col_k = np.loadtxt("fichier.txt")[:,k]
```

Images

```
I = plt.imread("image.png") .charge une image dans un
tableau
plt.imshow(I) ..... affiche l'image de tableau I
plt.imsave(fichier,I) ..... sauve l'image de tableau I
```

Options graphiques

```
plt.axis('off') ..... supprimer l'affichage des axes
import matplotlib.cm as .. charger des cartes de couleurs
plt.imshow(I, cmap=cm.gray) .afficher une image en N&B
```

Outils

Durée d'exécution de processus

```
import time
debut = time.clock()
# ... processus
duree = (time.clock() - debut)
```

Retrouver une fonction dans un shell

```
import inspect
print(inspect.getsource(fonction))
```

Modifier la taille de pile de récursivité

```
import sys
sys.setrecursionlimit(n)
```

avec $n = O(1000)$ par défaut.

Ce mémo peut être distribué librement et gratuitement sous le terme de la licence de libre diffusion [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/). Copyright © 2019 Nicolas Mesnier [<nmesnier@free.fr>](mailto:nmesnier@free.fr) — v1.1 (19/12/2019)
Téléchargement et mise à jour : <http://nmesnier.free.fr/>