Mémento Python

python3 program.py ipython3 jupyter notebook

exécuter le programme program.pv

console Python: utile pour tester des bouts de code, faire des calculs, ...

création de documents mélangeant code Python, images, graphiques et texte (markdown, html/css, latex)

Types usuels

```
Exemples: (Le type des variables est détecté automatiquement).
                                                                                                              objet modifiable
                                                             True False
      bool
                   booléen
                                                             k = 1075
     int
                   entier (sans limite de taille)
                                                             12.4 \quad x = 3.2e18
     float
                   réel en virgule flottante
      complex
                   nombre complexe
                                                             2 + 3i
     str
                   chaîne de caractères
                                                             s = 'bon' "C'est" + s texte = """ Je peux m'étendre
                                                                                                 sur plusieurs lignes et
     list
                   liste d'obiets
                                                             [1, 3, 3, 'plus', 5.1] + [8]
                                                                                                 utiliser des caractères
     tuple
                   liste non modifiable d'obiets
                                                             (1, 3, 3, 'plus', 5.1) (8, )
                                                                                                 "spéciaux" comme
                   ensemble
{ } :set
                                                                                                 " ' \n \t ...
                                                             {2, 4, 'ABC', 5.1}
{ } dict
                   dictionnaire (ensemble de clé → valeur)
                                                             {'A': 1, 'B': 2}
```

obiets itérables (obiet conteneur dont on peut parcourir les éléments

Identifiants (noms de variables, fonctions, ...), affectations (=)

```
x, y, z = 4.5, 2.2, 'yes'
Affectation:
                                                                                                Exemple:
                 x = 3e8/2e-5
                                             i = i = 0
                                                                affectation multiple (via tuple)
                                                                                                    L = [1, 2, 3]
Identifiant: • sensible min/Mai
                                                                                                   \Delta M = 1
                                             i += 1 \Leftrightarrow i = i + 1
                                                                                                    L.append(4)

    accents accentés

           · est un identifiant possible
                                             i *= 2 \Leftrightarrow i = i * 2
                                                                                                M et L pointent vers le même obiet.
                                                                                                la liste [1,2,3], qui peut être modifiée.
```

```
Boucles
for variable in sequence: \to tout objet iterable
                                     (list, str. tuple, set, dict, ...)
        Bloc d'instructions ...
                                  bloc défini par l'indentation
▶ Boucle sur des entiers
   range (debut, fin, pas) crée un objet itérable
                                                   fin n'est
  pour parcourir une séquence d'entiers.
                                                  pas inclus.
   for i in range(1, 3):
                                     # pour i = 1, 2
   for i in range(4):
                                     # nour i = 0.1.2.3
   for i in range (4, 0, -1): # pour i = 4, 3, 2, 1
                                       Pour convertir range en liste
▶ Boucle sur des objets itérables
                                       list(range(debut, fin))
  for lettre in "Mamma mia"
  for element in ['k'. 7]
  for x in reversed(sequence)
  for index, element in enumerate(ma_liste)
      print("L'élément à l'index", index, "est", element)
   for v1. v2 in zip(sequence1.sequence2)
                        - Parcourir plusieurs séquences en parallèle
                                          Exemple:
```

x = 500

while x > 1.0:

x = x/2

print(x)

break Maths

```
La division / donne
                         toujours un float
//
        Division entière
**
        puissance . a^b signifie a XOR b
%
        modulo (reste de la division entière)
        (entre tableaux numpy)
        multiplication matricielle ou
        produit scalaire
abs(x)
                  valeur absolue
                                               etc..
```

round (x, n) arrondir à n décimales

continue passage immédiat au tour suivant

sortie immédiate de la boucle

while expression_logique:

Bloc d'instructions

from math import * / Définit pi et e et les fonctions: sin(pi/2)# vaut 1.0 sart(2) # √2

Appliquer une fonction à une liste / tableau: 1 utiliser le module *numpy* Exemple: 2*np.sin(T) T: tableau ndarray ② générateur de séguences log(e**3)# In(e3) Ex: [2*sin(x) for x in L] $log(100, 10) # log_{10}(100)$ ③ fonction map(): radians(90), degrees(pi) Ex: list(map(lambda x: 2*sin(x)), L) fonction $x \rightarrow 2 \sin(x)$ définit une fonction "en ligne" map() crée un itérateur

qui applique la fonction

aux éléments de la liste.

Tests if x > 0: **if** s == 'oui': print('positif') instructions

Optionnel: 1 ou plusieurs clauses sinon-si (elif signifie else if).

Logique booléenne:

True False

a **and** b

a or b

not a

print('négatif ou nul')

Blocs d'instructions définis par l'indentation du code. par exemple 4 espaces ou 1 tabulateur.

⇒ **obligation** d'indenter correctement le programme.

elif 2 < x < 5:

instructions

instructions

(fct anonyme, i.e. sans identificateur)

else:

Tout bloc est précédé par le caractère : à la ligne précédente.

égalité == On peut comparer non != différent de seulement des nombres, mais aussi des séquences > < sup./inf. (objets de type str, list, ...) sup./inf. ou = in

y = expression1 if x > 2 else expression2

L = [1,2,3,4]# [1,2,3,-4] L[3] = -4L[:3] L[-2:] # [3,-4] (avant-dernier → fin) L[:] = 1[1,1,1,1]M = L[:]

M = sorted(sea) crée une liste triée

Générateur de séquence

```
expression for élément in itérateur if condition
Exemple: [2*i for i in range(3)] # liste [0,2,4]
```

```
L.append (élément)
L.insert(index, valeur)
L. remove (valeur)
L.index(valeur)
```

Entrée/Sortie: écran, fichiers

```
Formatage de chaînes de caractères: '%format' % (variable) ou f'{variable:format}'
s = input('Votre réponse: ')
                                              Exemples: 'Variable %s = %8.3f' % ('x'. x)
                                                                                                        (codes de format du langage C)
    Chaîne de caractères
     Convertir avec int(s) ou float(s)
                                                          'Variable {} = {} et y = {:.4f}.'.format('x', x, y)
print('texte', variable, ...)
                                                                                                              alignement (optionnel):
                                                         f Variable x = \{x\} et y = \{y: ^10.4 f\}.
                                                                                                                         è à nauche
            Options suppl.: sep=
                                                                                                                        > à droite
^ centré
                                                        Python > 3 6
                                                                                (ant ) largeur
                                                                                              (opt.)
                                                                                                      type
                            end='\n'
                                                                             = nb caractères nb décimales d entier (b bin., x hex.)
                            file=f
                                                                                                          flottant
                                                                                                        e format scientifique
Fichiers: lecture
                                                        Écriture
                                                                                                        g général (sélection auto.)
                                                        f = open('fichier.ext', 'w')
                                                                                                            chaîne de caractères
f = open('fichier.txt')
                                                         f.write('Ligne à écrire\n') w write
f.readline() #retourne la ligne suivante
f.readlines() #retourne une liste des lignes
                                                        À la fin:
f.read()
                    #chaîne avec contenu du fichier
                                                        f.close()
                                                                                               r read
with open ('fichier, txt') as f: Avec un bloc with, le fichier est fermé automatiquement à la fin du bloc.
     for ligne in f:
                                            ← ligne contient le caractère "\n" à la fin. Remarque; Avec readline(). ligne = '' si fin du fichier.
         print(ligne.rstrip())
                                            ← rstrip() supprime les caractères blancs (\n, \t et les espaces) à la fin.
```

Fonctions

```
def ma fonction(arg1, arg2, arg opt=9):
        "Documentation opt. multi-ligne"
        Bloc d'instructions
                                       Valeur par défaut de
        return valeur
                                      l'argument optionnel
Appel: v = \text{ma fonction}(\text{arg1=3, arg2='texte'})
```

Exemple: **def** norme2(x, y): return x**2 + y**2 n = norme2(1.5, 3.0)

Pas de restriction sur le type d'obiet retourné (→ tuple pour retourner plusieurs valeurs). Pas de 'return' \in return None. Accès aux variables du prog. principal possible en lecture.

Séquences (seq = list, tuple, str, ...)

```
len(sea)
            nombre d'éléments
                                     seq + seq2
sea[i]
             élément d'index i
                                     seq * 3
del sea[i] supprime l'élément d'index i
min(seq), max(seq), sum(seq)
```

seq[0] 1er élément е seq[-1] dernier él. -2 -1Sous-liste: seq[debut:fin:pas] Exemples:

fin n'est pas inclus.

```
# [1,2,3] (début → élément 3 non inclus)
     crée une copie (superficielle) de la liste L
```

filter(x<0. sea) Itérateur retenant les éléments selon la condition Convertir, au besoin, avec list(),

```
Méthodes spécifiques aux listes
L.pop (index) ⇔ del L[index] mais retourne la valeur de l'élément
L. sort() trie la liste L en place (et retourne None)
```

Conversions de type

```
#retourne le type de variable
type(variable)
float → int
                           int(2.5), round(2.5), floor(-1.6), ceil(-1.6)
str → int, float int("5"), float("2.5")
int, float \rightarrow str str(2.5)
list → str
                           "-".join(['A'. 'B'. 'C']) → 'A-B-C'
                           list(map(str, [.1, .5])) \rightarrow ['.1', '.5']
                           \mathsf{list}(\mathsf{"}\mathsf{Ça}\;\;\mathsf{va"}) \to [\mathsf{'}\mathsf{C}',\,\mathsf{'a'},\,\mathsf{'}\;\,\mathsf{'},\,\mathsf{'v'},\,\mathsf{'a'}]
str → list
                           "Ça va".split(" ") → ['Ça', 'va']
 Dictionnaire
```

```
D.keys()
D = \{'A': 1. 'B': 2\}
                                 D.values()
D[new_key] = new_value
for key, value in D:
    print('Clé:', kev, 'Valeur:', value)
```

Bibliothèque standard

```
► Arguments du prog. import svs: liste args = svs.argv
```

```
▶ Opérations sur fichiers/dossiers chemin/acces/fichier.txt
 import os
                                    os.path.basename(path)
  import shutil
 import glob
                                    os.path.dirname (path)
```

liste fichiers = qlob.qlob("*.jpq")

```
Fichiers / dossiers
shutil.copv(src. dest)
                            os.getcwd()
os.rename(src. dest)
                            os.listdir(path)
os.remove(file path)
                            os.mkdir(path)
```

► Exécuter une commande shell ► Autres modules standard import subprocess as sproc random, time datetime. re sproc.run('ls -l'.split(' '), \ stdout=sproc.PIPE).stdout.decode()

Débogueur

```
Entrer dans le débogguer
import pdb; pdb.set trace()
pdb>>> 🔼 next
                               print (p) Affiche une expression
                               list (1) Affiche le code source
             step in
                       (S)
             return
                               !commande python à exécuter
             continue (C)
                               répéter la dernière commande
             quit
```

Aide

? module (ou type ou autre) raccourci invthon nour heln() dir (module) liste le contenu d'un module ou d'une classe d'objets. Cet aide-mémoire est un aperçu non exhaustif de commandes Python. Documentation complète: http://docs.python.org

Auteur: Vincent Ballenegger (décembre 2017)

Module numpy (numerical python)

import numpy as np

- tableaux d'éléments de même type
- opérations mathématiques rapides sur des tableaux
- algèbre linéaire, nombres aléatoires, transf. de Fourier

Création de tableaux (de type ndarray)

Tableaux à 1d : vecteur de n composantes

<pre>np.zeros(n)</pre>	vecteur de composantes 0
np.ones(n)	vecteur de composantes 1
np.empty(n)	vecteur non initialisé

Tableaux à 2d : matrice de n lignes et m colonnes

```
np.zeros((n.m))
                     matrice nulle
np.identitv(n)
```

matrice identité **np.eve(***n*, *m*, *k***)** matrice avec 1 sur kième diagonale

np.diag([valeurs], k) matrice diagonale (cas k=0)

Tableaux à d dimensions : taille N₁ × N₂ × ... × N_d (cas général)



nb de valeurs dans chaque dimension: tuple (N1, N2, N3, ...)

(optionnel) type des éléments: float réel (np.float16, ..., 64) complex nb complexe (np.complex64, 128) entier (np.int8, 16, 32, 64) np.uint entier positif (np.uint8, ..., 64)

T = np.random.rand(N1.N2...)

Crée un tableau de forme N₁, N₂, ... rempli de nombre aléatoires (loi de probabilité uniforme sur l'intervalle [0,1]).

T = np.fromfunction(function, shape)

Crée un tableau initialisé avec une fonction (voir help(np.fromfunction)).

Accès aux éléments

```
nombre de lianes
Exemple: for i in range(T.shape[0]):
            for j in range(T.shape[1]):
               T[i,j] = ...
          no de liane colonne
```

En général: T[i.i.k....]

Vue sur une portion de tableau: *T[début:fin:incrément]* (pas de copie)

T[i:i+h, j:j+l] T[k,:] ligne k du tableau sous-matrice $h \times I$ T[:.k] colonne k du tableau

Copier un tableau T2 = T1.copy()

Fichiers de données

Format texte

```
Fichier data.txt:
T = np.loadtxt('data.txt',options)
                                                          # x y z
1900 3e3 4e3
1920 2876 3e9
Options: skiprows=n
                           ignorer les n 1res lignes
         comments='#'
                           symbole des commentaires (# par défaut)
         delimiter=str séparateur entre les valeurs (esp./tab par défaut)
         unpack=bool si True: x, v, z = loadtxt(...)
         usecols=(0,2) lit les colonnes 0 et 2 uniquement
```

np.savetxt('data.out', fmt='%.3e', autres options)

Format binaire .npv

```
np.save('data.out', T)
np.load('data.out')
```

: Attributs d'un tableau T

```
T.ndim
                nb de dimensions (axes)
T. shape
                tuple avec nb d'éléments dans chaque dimension
T.size
                nh total d'éléments
T.dtype
                type des éléments (data-type)
T.itemsize taille d'un élément (octets)
```

Conversions

```
list→array: np.array([[1,2],[3,4]])
array→list: T.tolist()
                                     → Î[1.2].[3.4]]
```

T. astype (dtype) Copie du tableau converti au type spécifié

Opérations mathématiques

```
T.sum(), T.min(), T.max(), T.mean()
              somme de tous les éléments
T.sum(axis=0) \Leftrightarrow T[0,:] + T[1,:] + ... = somme des colonnes si 2d
T.sum(axis=1) \Leftrightarrow T[:, 0] + T[:, 1] + .... = somme des lignes si 2d
```

Les fonctions mathématiques du module numpy s'appliquent à chaque élément des tableaux.



Les opérations + - * / ** sont effectuées terme-à-terme.

Tableau + nombre — effectué sur chaque terme

1	1	1	 0	1	2	_	0	1	2
5	5	5	0		2	-	0	5	10

Calcul matriciel

```
    Produit scalaire entre 2 vecteurs (tableaux à 1d): vec1 @ vec2

Produit matriciel
                             (tableaux à 2d): mat1 @ mat2
Produit extérieur de 2 vecteurs: np.outer(vec1, vec2)
Trace: mat.trace()
Matrice transposée: mat.T
                                     La transposition est sans effet
```

sur un tableau à 1d. Il n'y a pas de distinction entre vecteur ligne ou colonne en 1d. On peut réaliser cette distinction avec des matrices en 2d composées de 1 ligne ou 1 colonne (distinction non recommandée car peu utile).

Autre notation (Python < 3.5) $m1 @ m2 \Leftrightarrow np.dot(m1, m2) \Leftrightarrow m1.dot(m2)$ $v1 @ v2 \Leftrightarrow np.dot(v1,v2) \Leftrightarrow v1.dot(v2)$ np.transpose(M) ⇔ m.transpose()

numpy.linalg: Méthodes d'algèbre linéaire

```
import numpy.linalg as la
la.det(M)
                           déterminant de la matrice M
la.inv(M)
                           matrice inverse
la.eig(M)
                           valeurs propres
la.solve(A.B)
                           renvoie X tel que A \cdot X = B
la.matrix rank(M)
                           rang de M
la.matrix_power(M, n)
```

numpy.random: nombres aléatoires

```
np.random.rand(N1, N2, ...)
  tableau de forme N1, N2, ... rempli de nombre aléatoires distribués
  selon la loi de probabilité uniforme sur l'intervalle [0,1[.
np.random.randn(N1, N2, ...) idem pour la loi normale
np.random.randint(min. max. nb)
 nb nombres entiers aléatoires dans l'intervalle [min. max[ (max est exclu).
np.random.random integers(min. max. nb)
 nb nombres entiers aléatoires dans l'intervalle [min, max] (max est inclus).
```

numpy.fft: transformées de Fourier

Module matplotlib: création de graphiques

```
import matplotlib.pvplot as plt
```

Notebook jupyter: - utiliser %matplotlib inline (ou notebook) pour l'importation - instructions plt.figure() et plt.show() non obligatoires

Création d'un graphique

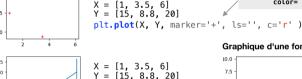
```
plt.figure(figsize=(8,5))
                                                création d'un nouveau araphiaue
plt.plot(X, Y, label='mes données')
                                               X, Y: listes ou tableaux numpy contenant les coordonnées x, resp. y, des points.
plt.leaend()
                                                affiche une léaende (cf. option 'label' de plot)
plt.xlabel('légende axe x')
                                                légendes des axes
plt.ylabel('légende axe y')
plt.show()
                                                afficher le graphique
```

```
plt.xscale('log')
                                   axe x en échelle log
plt.axis('equal')
                                   même échelle pour les axes x et y
plt.xlim(xmin, xmax)
                                   bornes de l'axe des x
plt.savefig('fichier.pdf')
                                   enreaistrer le araphique
```

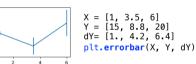
Création de tableaux de valeurs régulièrement espacées

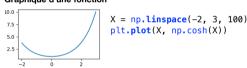
```
np.linspace(a, b, n)
                                   n valeurs dans l'intervalle [a,b]
np.logspace(a, b, n)
                                   n valeurs de 10-a à 10-b régulièrement espacées en échelle log
np.arange(a, b, dx)
                                   n valeurs de a (inclus) à b (exclus) par pas dx
```

Graphique de points ou d'une fonction

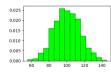








Histogramme

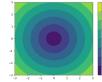


```
X = 100 + 15*np.random.randn(1000)
plt.hist(X, 15, normed=1, facecolor = 'lime', edgecolor = 'green')
               nombre de barres
```

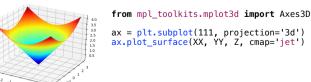
Options de plot():

marker=

Graphique en 3d: z = f(x,y)



```
X = np.linspace(-3, 3, 200)
                                   coordonnées selon axe x
                                    coordonnées selon axe v
Y = X
                                    coordonnées x et y de tous les points de la grille
XX, YY = np.meshgrid(X, Y)
Z = np.sart(XX**2 + YY**2)
                                    hauteurs: z = \sqrt{x^2 + v^2}
                                    araphiaue de courbes de niveaux (f = filled)
plt.contourf(XX, YY, Z)
                                    barre avec léaende des couleurs
plt.colorbar()
```



```
ax = plt.subplot(111, projection='3d')
ax.plot_surface(XX, YY, Z, cmap='jet')
```