Cours Introduction à la programmation en langage Python
Master 1 ENSPD

Dr jean Koudi

13 février 2025

Table des matières

0.1 Introduction

0.2 Présentation du Langage Python

Python est un langage de programmation (au même titre que le C, C++, fortran, java, $\cdots$), développé en 1989. Ses principales caractéristiques sont les suivantes :

i) "open-source" : son utilisation est gratuite et les fichiers sources sont disponibles et modifiables ;

ii) simple et très lisible ;

iii) doté d'une bibliothèque de base très fournie ;

iv) importante quantité de bibliothèques disponibles : pour le calcul scientifique, les statistiques, les bases de données, la visualisation $\cdots$;

v) grande portabilité : indépendant vis à vis du système d'exploitation (linux, windows, MacOS) ;

vi) orienté objet ;

vii) typage dynamique : le typage (association à une variable de son type et allocation zone mémoire en conséquence) est fait automatiquement lors de l'exécution du programme, ce qui permet une grande flexibilité et rapidité de programmation, mais qui se paye par une surconsommation de mémoire et une perte de performance ;

viii) présente un support pour l'intégration d'autres langages. Comment faire fonctionner le code source ?

Il existe deux techniques principales pour traduire un code source en langage machine :

1. la compilation : une application tierce, appelée compilateur, transforme les lignes de code en un fichier exécutable en langage machine. A chaque fois que l'on apporte une modification au programme, il faut recompiler avant de voir le résultat.
2. l'interprétation : un interpréteur s'occupe de traduire ligne par ligne le programme en langage machine.

Ce type de langage offre une plus grande commodité pour le développement, mais les exécutions sont souvent plus lentes. Dans le cas de Python, on peut admettre pour commencer qu'il s'agit d'un langage interprété, qui fait appel à des modules compilés.

Pour les opérations algorithmiques coûteuses, le langage Python peut s'interfacer à des bibliothèques écrites en langage de bas niveau comme le langage C.

Les différentes versions

Il existe plusieurs versions de Python : les versions 2. et les versions 3.

Les versions 3. ne sont pas une simple amélioration des versions 2. car toutes les librairies Python n'ont pas effectué la migration de 2.7 à 3.

L'interpréteur

Dans un terminal, taper python (interpréteur classique) ou ipython (interpréteur plus évolué) pour accéder à un interpréteur Python. Vous pouvez maintenant taper dans ce terminal des instructions Python qui seront exécutées.

0.3 Utilisations de Python et librairies.

— web : Django, Zope, Plone,. . .
— bases de données : MySQL, Oracle,. . .
— réseaux : TwistedMatrix, PyRO, VTK,. . .
— représentation graphique : matplotlib, VTK,. . .
— calcul scientifique : numpy, scipy, . . .

0.4 Le mode programmation

Il s'agit d'écrire dans un fichier une succession d'instructions qui ne seront effectuées que lorsque vous lancerez l'exécution du programme. Cela permet tout d'abord de sauvegarder les commandes qui pourront être utilisées ultérieurement, et d'autre part d'organiser un programme, sous-forme de fichier principal, modules, fonctions etc.

Le fichier à exécuter devra avoir l'extension .py, et devra contenir en première ligne le chemin pour accéder au compilateur Python, ainsi que l'encodage :

-*- coding : utf-8 -*-

Il pourra être exécuté en lançant dans un terminal la commande python nomdufichier.py.

0.4.1 L'environnement de programmation Spyder (sous Anaconda)

L'environnement Spyder permet de réaliser des programmes informatiques écrits avec le langage Python. Il est disponible avec la distribution Anaconda, qui présente de nombreux avantages, notamment celui d'être simple à installer. Son téléchargement se fait à l'adresse suivante : https ://store.continuum.io/cshop/anaconda/

On peut télécharger directement spyder pour :
windows 32bit sur https ://download.cnet.com/spyder-32-bit/3000-2212_4-75915014.html
windows 64bit sur https ://www.bing.com/search ?q=spyder%20download

Une fois la distribution Anaconda téléchargée et installée, on peut commencer à lancer Spyder en tapant Spyder dans un terminal.

Au lancement de Spyder, apparaît une fenêtre partagée en deux zones. La zone en-bas à droite, appelée console (shell en anglais), est celle où l'on peut travailler de façon interactive avec l'interpréteur Python. La zone à gauche est un éditeur de texte, spécialement conçu pour écrire des programmes dans le langage Python.

0.5 Types et opérations de base.

Le langage Python est orienté objet, c'est-à-dire qu'il permet de créer des objets, en définissant des attributs et des fonctions qui leur sont propres. Cependant, certains objets sont pré-définis dans le langage. Nous allons voir à présent les plus importants.

0.5.1 Les nombres et les booléens

— entiers (32 bits)	type : int	exemples : 0, 1, 20
— réels (64 bits)	type : float	exemples : 4., 5.1, $1.23 \cdot 10^{-6}$
— complexes	type : complex	exemples : 3+4j 3+4J
— booléens	type : bool	exemples : True, False

0.5.2 Opérations de base

Affectation

L'affectation se fait grâce à l'opérateur "=". On a comme exemple :

»> i = 3 (l'affectation de la valeur 3 à la variable i). Donc i vaut 3

»> a, k = 3.14159, True (l'affectation de la valeur 3.14159 à la variable a et la valeur True à la variable k)

»> k=r=2.15 (l'affectation de la valeur 2.15 à la variable k et à la variable r)

»> x=complex(3,4) (l'affectation de la valeur complexe 3+4j à la variable x)

Affichage

L'affichage se fait grâce à la fonction "print" de python.

On écrit print(information à afficher)

Pour afficher un texte écrit, il faut nécéssairement utiliser les guillemets " informantion "

1. »> print(" Bonjour les amis") Bonjour les amis
2. La valeur de i est 3 et on veut afficher i.
   »> print(i) 3

Opérateurs arithmétiques

Il s'agit ici des opérateurs intervenants dans les opérations arithmétiques.

Ils servent à agir sur les variables de types numériques (Entiers et réels). Le tableau ci-dessous présente une liste des opérateurs arithmétiques selon leur priorité et leurs symboles.

Priorité	Opérateur	Symbole Mathématiques	Symbole Algorithmique	Exemple Exemple
0	Parenthèse	()	()	3*2(9/3)=18
1	Puissance	$\wedge$	** ou pow	2^3 = $2^3 = pow(2,3) = 8$
	Multiplication	x	*	10*5 = 10
2	Division	$\div$	/	$9/2 = 4.5$
2	Modulo	Mod ou $\equiv$	%	11%3=2 ; 9%2=1
2	Division entière	//	//	11//3=3 ; 9 // 2=4
3	Addition	+	+	23+5=28
3	Soustraction	-	-	5-13=-8
	Valeur absolue	$\lvert \cdot \rvert$	abs	abs(-5) = $\lvert -5 \rvert = 5$

Opérateurs de comparaisons

Il s'agit des opérateurs intervenants beaucoup plus dans la réalisation des conditions (tests). Le résultat des opérations effectuées avec ces opérateurs est booléen (Vrai ou faux). Ces opérateurs peuvent être appliquer aux variables numériques et chaînes de caractères. Le tableau ci-dessous présente une liste de ces opérateurs :

Opérateur	Symbole Mathématiques	Symbole Algorithmique	Exemple
Inférieur	$<$	$<$	a=2 et b=2 ; a<b est fausse
Inférieur ou égale ou égale	$\leq$	<=	a=-2 et b=5 ; a<=b est vraie
Supérieur	$>$	$>$	a=2 et b=-5 ; a>b est vraie
Supérieur ou égale	$\geq$	>=	a=2 et b=-5 a>=b est vraie
Égalité	$=$	$=$	a=2, b=2 et c=-5 a=b est vraie a=c est fausse
Différent	$\neq$	$< >$ ou !=	a=2 et b=-5 ; a< >b est vraie

D'autres opérateurs de comparaison sont : "is", "is not"

Opérateurs logiques ou de liaisons.

Ces opérateurs servent essentiellement à faire la liaison entre deux opérations de comparaisons afin de réaliser des conditions plus complexes. Il s'agit des opérateurs «ET» en anglais «AND» (symbole mathématique $\wedge$), «OU» en anglais «OR» (symbole mathématique $\vee$) et «NON» en anglais «NOT» (symbole mathématique $\overline{P}$ ou $> P$) . Le résultat de ces opérations est booléen. C'est-à-dire un résultat susceptible seulement deux valeurs possibles : VRAI ou FAUX. On convient de représenter VRAI par «1» et FAUX par «0». Ainsi les tables de vérités de ces deux opérateurs se présentent comme suit :

A	B	(A $\vee$ B)
0	0	0
0	1	1
1	0	1
1	1	1

A	B	(A $\wedge$ B)
0	0	0
0	1	0
1	0	0
1	1	1

A	$\overline{A}$
0	1
1	0

0.5.3 Les chaînes de caractères

Une chaîne de caractères (string en anglais) est un objet de la classe (ou de type) str.

Définition et affichage

Une chaîne de caractères peut être définie de plusieurs façons :
»> "je suis une chaîne"
'je suis une chaîne'

»> 'je suis une chaine'
'je suis une chaine'
»> "pour prendre l'apostrophe"
'pour prendre l'apostrophe'
»> "pour prendre l'apostrophe" 'pour prendre l'apostrophe'
NB : La commande \n permet d'aller à la ligne après affichage.
» " " "écrire sur plusieurs lignes" " " 'écrire \n sur \n plusieurs \n lignes'

Concaténation

On peut mettre plusieurs chaînes de caractères bout à bout avec l'opérateur binaire de concaténation, noté +.

» s = 'i vaut'
» i = 1
»print(s+"i") ou
» print(s+str(i)) ou
» print(s + ' ' + str(i)) (pour créer de l'espace entre s et i)
i vaut 1
» print('*-' * 5)
--*-*-*-

accès aux caractères

Les caractères qui composent une chaîne sont numérotés à partir de zéro. On peut y accéder individuellement en faisant suivre le nom de la chaîne d'un entier encadré par une paire de crochets :
» "bonjour"[3] ; "bonjour"[-1]
'j'
'r'
» "bonjour"[2 :] (afficher à partir de la 3è lettre) ;
'njour'
"bonjour"[:3](afficher tout ce qui est avant la 4è lettre) ;
'bon'
"bonjour"[3 :5] (afficher de la 4è au 5è lettre)
'jo'
» "bonjour"[-1 : :-1] (afficher de la dernière à la première lettre ;
'ruojnob'

Autres fonctions : Soit s une chaîne de caractères.

- len(s) : renvoie la taille d'une chaîne(la longueur de la chaîne).
- s.find : recherche une sous-chaîne dans la chaîne (sa position dans le chaîne).
- s.rstrip : enlève les espaces de fin. s.rstrip() renvoie une copie de la chaîne dans laquelle tous les caractères ont été supprimés à partir de la fin de la chaîne. Si aucun paramètre n'est passé à la méthode, les caractères d'espacement qui se trouvent à la fin de la chaine, sont supprimés par défaut.
- s.replace : remplace une chaîne par une autre,

— ...

0.5.4 Les listes

Une liste consiste est une succession ordonnée d'objets, qui ne doivent pas nécessairement être du même type. Les termes d'une liste sont numérotés à partir de 0 (comme pour les chaînes de caractères).

Initialisation

Une liste vide est initialisée par : [ ] ; ou list() ;
Des exemples de listes :

[1, 2, 3, 4, 5] ; ['point', 'triangle', 'quad'] ; [1, 2, 3, 4, 5] ; ['point', 'triangle', 'quad'] ; [1,4,'mesh',4,'triangle',['point',6]] ; [1, 4, 'mesh', 4, 'triangle', ['point', 6]]
La fonction 'range' est une fonction qui renvoie une liste. Si n est un entier naturel non nul, rang(n) renvoie la liste des n premiers entiers naturels.
Exemple : range(5) donne [0,1,2,3,4].

modification

Contrairement aux chaînes de caractères, on peut modifier les éléments d'une liste :
» l=[1,2,3,4,5]
» l[2 :]=[2,2,2]
» l

[1, 2, 2, 2, 2]
On a remplacer les éléments de la 3eme à la 5ème position par 2.
On peut aussi faire :
» l[2]= 2, l[3] = 2, l[4] = 2 ;
» l

[1, 2, 2, 2, 2]

concaténation

» [0]*7

[0, 0, 0, 0, 0, 0, 0]
» L1, L2 = [1,2,3], [4,5]
» L1

[1, 2, 3]
» L2

[4, 5]
» L1+L2

[1, 2, 3, 4, 5]

copie d'un objet

» L = ['Dans','python','tout','est','objet']
» T = L
» T[4] = 'bon'
» T

['Dans', 'python', 'tout', 'est', 'bon']
» L

['Dans', 'python', 'tout', 'est', 'bon']
» L=T[:]
» L[4]='objet'
» T ;L

['Dans', 'python', 'tout', 'est', 'bon']

['Dans', 'python', 'tout', 'est', 'objet']

Quelques fonctions des listes

L est une liste.
— len(L) : renvoie la taille de la liste L,
— L.sort : trie la liste L,
— L.append : ajoute un élément à la fin de la liste L,
— L.reverse : inverse la liste L,
— L.index : recherche l'indice d'un élément dans la liste L,
— L.remove : retire un élément de la liste L,
— L.pop : retire le dernier élément de la liste L,

0.6 Les structures de contrôle

a) L'indentation

Les fonctions Python n'ont pas de begin ou end explicites, ni d'accolades qui pourraient marquer là où commence et où se termine le code de la fonction. Le seul délimiteur est les deux points (« : ») et l'indentation du code lui-même. Les blocs de code (fonctions, instructions if, boucles for ou while etc) sont définis par leur indentation. L'indentation démarre le bloc et la désindentation le termine. Il n'y a pas d'accolades, de crochets ou de mots clés spécifiques. Cela signifie que les espaces blancs sont significatifs et qu'ils doivent

être cohérents. Voici un exemple :

$a = -150$
$if\ a < 0 :$
$\quad print('a\ est\ negatif')$

En général on a :

```
ligne d'en-tête
   première instruction du bloc
   ...
   dernière instruction du bloc
```

b) Le test de conditions

Le test de condition se fait sous la forme générale suivante :
if < condition1 > :
< blocs d'instructions 1>
elif < condition2 > :
< blocs d'instructions 2 >
...
else :
< blocs d'instructions 3 >

Exemples

Exemple 1

```
a = 10.
if a > 0 :
    print('a est strictement positif')
    if a >= 10 :
        print (' a est un nombre ')
    else :
        print (' a est un chiffre ')
        a += 1
elif a is not 0 :
    print(' a est strictement negatif ')
else :
    print(' a est nul ')
```

Exemple 2

```
L = [1, 3, 6, 8]
if 9 in L :

    print( '9 est dans la liste L')
else :
    L.append(9)
```

c) La boucle conditionnelle

La forme générale d'une boucle conditionnelle est
while < test1 > :

< blocs d'instructions 1 >
if < test2 > : break
if < test3 > : continue
else : < blocs d'instructions 2 >

où l'on a utilisé les méthodes suivantes :
break : sort de la boucle sans passer par else,
continue : remonte au début de la boucle,
pass : ne fait rien.

La structure finale else est optionnelle. Elle signifie que l'instruction est lancée si et seulement si la boucle se termine normalement.

Exemples

Exemple 1 : Une boucle infinie

```
while 1 :
    print('Bonjour')
```

Exemple 2 Vérifier qu'un nombre est premier.

```
print('Entrer un entier y')
x = y/2
while x > 1 :
    if y%x ==0 :
        print(str(y)+'est facteur de '+str(x))
        break
    x = x - 1
else :
    print( str(y)+ ' est premier')
```

(d) La boucle inconditionnelle

La forme générale d'une boucle inconditionnelle est

```
for < cible > in < objet > :
    < blocs d'instructions 1 >
    if < test1 > : break
    if < test2 > : continue
    else :
        < blocs d'instructions 2 >
```

Exemples

Exemple 1 :

```
sum = 0
for i in [1, 2, 3, 4] :
    sum += 1
    prod = 1
```

```
    for p in range(1, 10) :
       prod *= p
    s = 'bonjour'
for c in s :
    print c,
L = [ x + 10 for x in range(10) ]
```

0.7 Les fonctions

La structure générale de définition d'une fonction est la suivante
def <nom fonction> (arg1, arg2,... argN) :
...
bloc d'instructions
...
return < valeur(s) >

0.7.1 Quelques exemples

0.7.2 Exemple 1

```
def table7() :
n=1
while n < 11 :
   print n*7
   n+=1
```

0.7.3 Exemple : Une fonction sans return

Une fonction qui n'a pas de return renvoie par défaut None. def table(base) :

```
n=1
while n < 11 :
   print n*base
n+=1
```

0.7.4 Exemple 3

Une fonction qui prend des arguments et return un résulta.

```
def table(base, debut=0, fin=11) :
    print ('Fragment de la table de multiplication par '+str(base)+' : ')
    n=debut
    l=[]
    while n < fin :
       print n*base
    l.append(n*base)
    n+=1
return l
```

0.7.5 Exemple 4 : Une fonction sans connaître ses paramètres

On peut également déclarer une fonction sans connaître ses paramètres :

```
def f(*args, **kwargs) :
    print(args)
    print(kwargs)
```

Il existe une autre façon de déclarer une fonction de plusieurs paramètres :
f = lambda argument 1, ... , argument N : expression utilisant les arguments

0.8 Les modules

Les modules sont des programmes Python qui contiennent des fonctions que l'on est amené à souvent réutiliser (on les appelle aussi bibliothèques, ou libraries en anglais). Ce sont des « boîtes à outils » qui nous seront très utiles.

Les développeurs de Python ont mis au point de nombreux modules qui effectuent différentes tâches. Pour cette raison, prenez toujours le réflexe de vérifier si une partie du code que vous souhaitez écrire n'existe pas déjà sous forme de module.

La plupart de ces modules sont déjà installés dans les versions standards de Python. Vous pouvez accéder à une documentation exhaustive sur le site de Python.

Voici tout d'abord un exemple de module permettant l'utilisation des nombres de Fibonacci.

Fichier fibo.py(Nom du fichier)

```
def print_fib(n) :
      a, b = 0, 1
      while b < n :
        print(b),
        a, b = b, a + b
def print_fib(n) :
      result, a, b = [ ], 0, 1
      while b < n :
        result.append(b),
        a, b = b, a + b
return result
```

0.8.1 Importation de modules

Il existe plusieurs façons d'importer un module :

— import fibo
— import fibo as f
— from fibo import print_fib, list_fib
— from fibo import* (importe tous les noms sauf les variables et les fonctions privées)

0.8.2 Les modules standards de python.

Il existe une série de modules que vous serez probablement amenés à utiliser si vous programmez en Python. En voici une liste non exhaustive (pour la liste complète, reportez-vous à la page des modules sur le site de Python) :

math : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).

sys : interaction avec l'interpréteur Python, notamment pour le passage d'arguments (voir plus bas).

pathlib : gestion des fichiers et des répertoires (voir plus bas).

random : génération de nombres aléatoires.

time : accès à l'heure de l'ordinateur et aux fonctions gérant le temps.

urllib : récupération de données sur internet depuis Python.

Tkinter : interface python avec Tk. Création d'objets graphiques

re : gestion des expressions régulières (voir chapitre 17 Expressions régulières et parsing).

NumPy : NumPy est un outil performant pour la manipulation de tableaux à plusieurs dimensions. Il ajoute en effet le type array, qui est similaire à une liste, mais dont tous les éléments sont du même type : des entiers, des flottants ou des booléens.

Le module NumPy possède des fonctions basiques en algèbre linéaire, ainsi que pour les transformées de Fourier.

matplotlib : Matplotlib est une bibliothèque en Python utilisée pour créer des visualisations de données sous forme de graphiques.

La fonction **dir()** permet de connaître toutes les fonctions et variables d'un module. Il suffit de taper dir(nom du module)

0.9 Gestion des tableaux

En Python, les tableaux sont souvent manipulés à l'aide de la bibliothèque NumPy, qui offre des fonctionnalités avancées pour le calcul numérique. Un tableau NumPy, également appelé "array", permet de créer et de manipuler des données en une seule ou plusieurs dimensions.

0.9.1 Création de tableaux.

Création d'un tableau dont on connaît la taille

Voici un exemple simple d'utilisation de NumPy pour créer et manipuler un tableau :

```
import numpy as np
tableau_1D = np.array([1, 2, 3, 4, 5])
print("Tableau 1D :", tableau_1D)
tableau_2D = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("Tableau 2D :\n", tableau_2D)
```

Un tableau peut être multidimensionnel, comme ici, de dimension 3 : »> a=np.zeros((nx,ny,3))

Mais nous nous limiterons dans ce cours aux tableaux uni et bi-dimensionnels.

Il existe également les fonctions np.ones, np.eye, np.identity, np.empty, etc. pour créer des tableaux spécifiques.

Par exemple :
np.identity(3) est la matrice identité d'ordre 3,
np.empty est un tableau vide,
np.ones(5) est le vecteur [1 1 1 1 1] et
np.eye(3,2) est la matrice à 3 lignes et 2 colonnes contenant des 1 sur la diagonale et des zéro partout ailleurs.

Par défaut les éléments d'un tableau sont des float (un réel en double précision) ; mais on peut donner un deuxième argument qui précise le type (int, complex, bool, etc.).
Exemple : »> np.eye(2, dtype=int)

Création d'un tableau avec une séquence de nombre

La fonction "**linspace(premier, dernier, taille n)** renvoie un tableau unidimensionnel commençant par le premier élément, se terminant par le dernier élément avec n éléments régulièrement espacés. Une variante est la fonction **arange** : Exemple
»> a = np.linspace(-4, 4, 9) ou
»> a = np.arange(-4, 4, 1)

On peut convertir une ou plusieurs listes en un tableau via la commande array :
»> a = np.array([1, 2, 3])
»> b = np.array(range(10))
»> L1, L2 = [1, 2, 3], [4, 5, 6]
»> a = np.array([L1, L2])

Création d'un tableau à partir d'une fonction

»> def f(x, y) :
.....return x**2 + np.sin(y)
...
»> a = np.fromfunction(f, (2, 3)) »> b = np.fromfunction(f, (4, 3))

0.9.2 Manipulations d'un tableau

Caractéristiques d'un tableau

- a.shape : retourne les dimensions du tableau
- a.dtype : retourne le type des éléments du tableau
- a.size : retourne le nombre total d'éléments du tableau
- a.ndim : retourne la dimension du tableau (1 pour un vecteur, 2 pour une matrice)

Indexation d'un tableau

Comme pour les listes et les chaînes de caractères, l'indexation d'un vecteur (ou tableau de dimension 1) commence à 0. Pour les matrices (ou tableaux de dimension 2), le premier index se réfère à la ligne, le deuxième à la colonne.

Quelques exemples :
»> L1, L2 = [1, -2, 3], [-4, 5, 6]
»> a = np.array([L1, L2])
»> a[1, 2] extrait l'élément en 2ème ligne et 3ème colonne

»> a[:, 1] extrait toute la deuxième colonne array([-2, 5])
»> a[1,0 :2 :2] extrait les éléments d'indice [dbut=0, pas=2, fin=2] de la première ligne, le dernier élément n'est pas inclus array([-4]) »> a[:, -1 :0 :-1] extrait les éléments d'indice [dbut=-1, pas=-1, fin=0] de la première colonne, le dernier élément n'est pas inclus »> a[a < 0] extrait les éléments négatifs

Copie d'un tableau

Un tableau est un objet. Si l'on affecte un tableau A à un autre tableau B, A et B font référence au même objet. En modifiant l'un on modifie donc automatiquement l'autre. Pour éviter cela on peut faire une copie du tableau A dans le tableau B moyennant la fonction "copy()". Ainsi, A et B seront deux tableaux identiques, mais ils ne feront pas référence au même objet.

Exemple :
»> a = np.array([1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12])
»> b = a
»> c = a.copy()
»> a[1][2] = -3

Une façon de faire une copie de tableau sans utiliser la méthode copy() est la suivante :
»> d = np.zeros(b.shape, a.dtype)
»> d[:] = b

Redimensionnement d'un tableau

Voici comment modifier les dimensions d'un tableau :
On considère la tableau à une dimension "a" suivant : »> a = np.linspace(1, 10, 10)
La commande "a.shape = (2, 5) " transforme le tableau "a" en un tableau 2x5.

La fonction "flatten" renvoie une vue d'un tableau bi-dimensionnel en tableau uni-dimensionnel.

Boucles sur les tableaux

Exemple :
»> a = np.zeros((2, 3))
»> for i in range(a.shape[0]) :
... for j in range(a.shape[1]) :
... a[i, j] = (i + 1)*(j + 1)
»> print a [[1. 2. 3.] [2. 4. 6.]]

0.9.3 Opérations sur les tableaux ou matrice.

Pour bien des calculs il est possible d'éviter d'utiliser des boucles (qui sont très consommatrices de temps de calcul). On peut faire directement les calculs sur des tableaux. Ceux-ci sont faits via des fonctions C, un langage de programmation bas niveau, et sont donc plus rapides. C'est ce qu'on appelle «vectoriser» un programme.

Opérations	Fonction numpy	Procédure
Produit de deux matrices	dot	dot(A, B)
Transposé de matrice	transpose	transpose(A) ou A.T
déterminant d'une matrice	det	linalg.det(A)
inverse d'une matrice	inv	linalg.inv(A)
valeurs propres	eig	eig(A)
Carré terme à terme	A**2	A**2
Produit scalaire	vdot	vdot(u, v)

Le sous-module **linalg** propose des méthodes numériques pour inverser une matrice, calculer son déterminant, résoudre un système linéaire . .

La fonction "sum" de numpy

Cette fonction fait la somme des éléments d'un tableau array (Contenant rien que des nombres)
somme = np.sum(tableau_2D)
print("Somme des éléments du tableau_2D :", somme)

la fonction "mean" pour la moyenne

La fonction "mean" de numpy permet de calculer la moyenne des éléments d'un tableau de type array.
moyenne = np.mean(tableau_2D)
print("Moyenne des éléments du tableau_2D :", moyenne)

0.9.4 Extraction d'un sous-tableau d'un tableau

sous_tableau = tableau_2D[1 :, 1 :]
print("Sous-tableau :\n", sous_tableau)

0.10 Importer et exporter des données avec Python

0.10.1 Méthode manuelle

Pour ouvrir et fermer un fichier nommé nomdufichier.txt on crée une variable fichier de type file à travers laquelle on pourra accéder au fichier. **fichier = open ('nomfichier.txt','r')**
Le paramètre **«r»** indique qu'on accède au fichier en mode lecture. Pour accéder au fichier

en mode écriture il faut utiliser le paramètre **«w»**. A la fin des opérations sur le fichier, on le ferme en appelant la fonction close : **fichier.close()**

Voici trois façons différentes de parcourir les lignes d'un fichier ouvert en mode écriture : **for ligne in fichier :**

La variable ligne est une chaîne de caractère qui prendra les valeurs successives des lignes du fichier de lecture (sous la forme d'une chaîne de caractères). ligne = fichier.readline() A chaque fois que la fonction readline de fichier est appelée, la ligne suivante du fichier est lue et constitue la valeur de retour de la fonction. lignes = fichier.readlines () Cette fonction lit toutes les lignes d'un coup et la valeur de retour est une liste de chaînes de caractères. Attention, si le fichier est gros cette méthode est à éviter car elle bloque instantannément une grande place mémoire. Il est dans ce cas préférable d'utiliser une méthode de lecture caractère par caractère (grâce à la fonction read()) ou ligne par ligne. Enfin, la fonction fichier.write(chaine) permet d'écrire la chaîne de caractères en argu- ment dans un fichier ouvert en mode écriture.

0.10.2 Méthode automatisée

La fonction numpy.loadtxt lit les lignes du fichier en argument et renvoie une matrice dont les lignes correspondent aux lignes du fichier et les colonnes aux différentes valeurs délimitées par un espace. Parmi les options les plus utiles, skiprows est le nombre de lignes à ne pas lire dans l'en–tête (par défaut 0) et delimiter remplace le délimiteur (espace) par la chaîne que l'on souhaite. Exemple : le fichier data.dat contient les lignes :

0.11 Représentation graphique avec Matplotlib

Le module Matplotlib est une bibliothèque très utile en Python pour créer des visualisations de données. Voici quelques-unes de ses principales fonctionnalités :

Création de graphiques : Il permet de créer une grande variété de graphiques, tels que des graphiques en courbes, en barres, en secteurs, et bien plus encore.

Personnalisation : Il offre une grande flexibilité pour personnaliser les éléments de vos graphiques (titres, étiquettes, couleurs, etc.).

Intégration : Il s'intègre bien avec d'autres bibliothèques Python comme NumPy et pandas, ce qui est utile pour l'analyse et la manipulation des données.

Publication : Il permet de sauvegarder vos graphiques dans différents formats (PNG, PDF, SVG, etc.) pour une utilisation ultérieure.

En gros, Matplotlib transforme vos données en visuels clairs et attrayants.

0.11.1 Représentation discrète d'une fonction avec Matplotlib

Pour tracer la courbe représentative d'une fonction f avec Matplotlib, il faut tout d'abord la "discrétiser" ; c'est-à-dire définir une liste de points $(x_i, f(x_i))$ qui va permettre d'approcher le graphe de la fonction par une ligne brisée. Bien sûr, plus on augmente le nombre de points, plus la ligne brisée est "proche" du graphe (en un certain sens).

Plus précisément, pour représenter le graphe d'une fonction réelle f définie sur un intervalle $[a, b]$, on commence par construire un vecteur X, discrétisant l'intervalle $[a, b]$, en

considérant un ensemble de points équidistants dans $[a, b]$. Pour ce faire, on se donne un entier naturel N grand et on considère **X=numpy.linspace(a,b,N)** (ou, ce qui revient au même, on pose $h = \dfrac{b-a}{N-1}$ et on considère **X=numpy.arange(a,b+h,h))**.

On construit ensuite le vecteur Y dont les composantes sont les images des X_i par f et on trace la ligne brise reliant tous les points (X_i, Y_i).

0.11.2 La fonction plot du module matplotlib.pyplot

Cette fonction permet de tracer des lignes brisées. Si X et Y sont deux vecteurs-lignes (ou vecteurs-colonnes) de même taille n, alors **plt.plot(X,Y)** permet de tracer la ligne brisée qui relie successivement les points de coordonnées (X_i, Y_i) pour $i = 1, \cdots, n$.

Pour connaître toutes les options, le mieux est de se référer à la documentation de Matplotlib.

Exemple : Représentons la fonction $f(x) = \exp\left(\dfrac{x}{2}\right) cos(5x)$

Le code

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0., 2*np.pi, 100)
plt.plot(x, np.exp(x/2)*np.cos(5*x), '-ro')
plt.title('Fonction f(x) = e^{x/2}cos(5x)')
plt.xlabel('x')
plt.text(1, 15, 'Courbe', fontsize=22)
plt.savefig('1D_ exemple.pdf')
plt.show()
```

0.11.3 Création de plusieurs graphiques dans une même fenêtre

La commande subplot permet de découper la fenêtre graphique en plusieurs sous-fenêtres.

Voici un exemple d'utilisation de cette commande.

```
x = np.linspace(-4., 4., 25)
plt.subplot(2, 2, 1)
plt.plot(x, -x**4 + x**3 + x**2 + 1, 'o-')
plt.title("f")
plt.subplot(2, 2, 2)
plt.plot(x, -4*x**3 + 3*x**2 + 2*x, '-')
plt.title("f^(1)")
plt.subplot(2, 2, 3)
plt.plot(x, -12*x**2 + 6*x + 2, '–')
plt.title("f^(2)")
plt.subplot(2, 2, 4)
plt.plot(x, -24*x + 6, ' :')
plt.title("f^(3)")
plt.tight_layout()
```

$plt.savefig("1D\_subplot2.pdf")$
$plt.show()$

0.12 Quelques fonction particulières de python.

https ://waytolearnx.com/2019/05/fonction-sum-python.html
Fonction input() :permet de lire les informations au clavier.
Fonction max() : – Python
Fonction min() – Python
Fonction len() – Python
Fonction abs() – Python
Fonction eval() – Python
Fonction exec() – Python
Fonction compile() – Python
Fonction ord() – Python
Fonction sum() – Python
Fonction all() – Python
Fonction any() – Python
Fonction str() – Python
Méthode super() – Python
Fonction print() – Python
Fonction round() – Python
Fonction complex() – Python
Fonction filter() – Python
Fonction set() – Python
Fonction tuple() – Python
Fonction list() – Python
Fonction dict() – Python
Fonction vars() – Python
Fonction type() – Python
Fonction sorted() – Python
Fonction slice() – Python
Fonction setattr() – Python
Fonction reversed() – Python
Fonction repr() – Python
Fonction range() – Python
Fonction property() – Python
Fonction open() – Python
Fonction oct() – Python
Fonction object() – Python
Fonction next() – Python
Fonction map() – Python
Fonction locals() – Python
Fonction iter() – Python