Numpy Data Structures

- *Author* : **Ibrahima SY**
- *Email* : syibrahima31@gmail.com
- *Github* : Cliquez
- *Linkdin* : Cliquez
- *School* : Institut Supérieur Informatique (**ISI**)

OUTLINE

1. PRÉSENTATION GÉNERALE
2. BASICS IN NUMPY
3. ARRAY vs LIST
4. DOT PRODUCT
5. SPEED TEST ARRAY VS LIST
6. MULTIDIMENSIONNAL ARRAYS
7. ARRAY INDEXING /SLICING / BOOLEAN INDEXING
8. RESHAPE ARRAY
9. CONCATENATION
10. BROADCASTING

11. FUNCTIONS AXIS
12. DATATYPES
13. COPYING
14. GENERATING ARRAYS
15. RANDOM NUMBERS
16. LINEAR ALGEBRA
17. LOAD DATA FROM csv

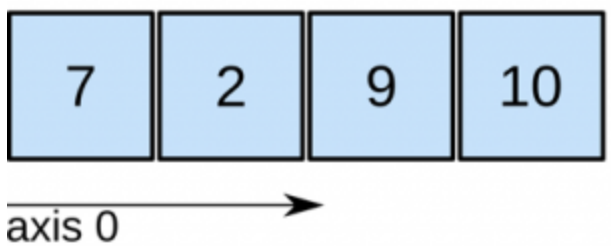
1. PRÉSENTATION GÉNÉRALE

NumPy est une bibliothèque pour langage de programmation Python, destinée àmanipuler des matrices ou tableaux multidimensionnels ainsi que des fonctionsmathématiques opérant sur ces tableaux.

Plus précisément, cette bibliothèque logicielle libre et open source fournit de multiplesfonctions permettant notamment de créer directement un tableau depuis un fichier ouau contraire de sauvegarder un tableau dans un fichier, et manipuler des vecteurs,matrices et polynômes.

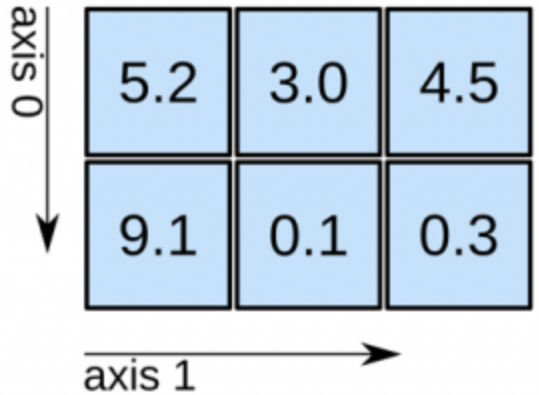
NumPy est la base de SciPy, regroupement de bibliothèques Python autour du calcul scientifique

1D array



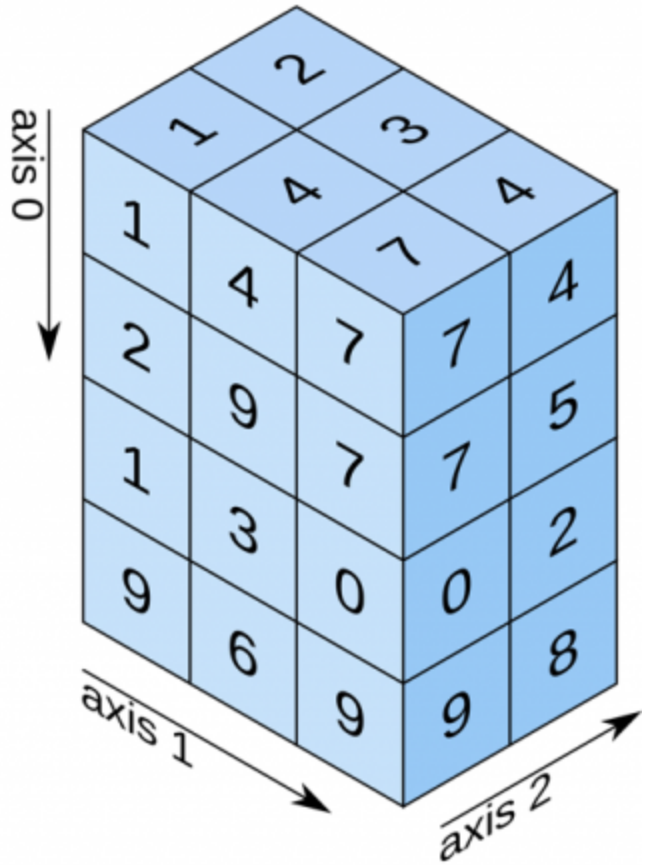
shape: (4,)

2D array



shape: (2, 3)

3D array



shape: (4, 3, 2)

2. BASICS IN NUMPY

Le type d'objet principal en Numpy est le tableau multidimensonnel: `numpy.array` alias `ndarray`. C'est une table d'éléments (habituellement des nombres), qui sont tous du même type et qui sont indexés par un tuple (plusieurs dimensions) d'entiers positifs.

```
# Importation de numpy
>>> import numpy as np

# On peut rergarder la version de numpy
>>> np.__version__
```

Il y a plusieurs façons de créer des tableaux.

La première consiste à créer un tableau à partir d'une `liste` ou d'un `tuple` en utilisant la fonction `array`.

```
#on commence toujours a importer le module numpy
>>> import numpy as np

#creation d'un vecteur 1D
>>> np.array([1,2,3])

#création d'une matrice 2D
>>> a =  np.array([[1,2,3], [4,5,6]])

# Voir la taille d'une matrice
>>> a.shape

# Voir la dimension de la matrice  a
>>> a.ndim

# retourne le nombre d'elements du tableau
>>> a.size
```

```
# acces au premier de la matrice
>>> a[0]

# multiplication de  deux tableau
>>> np.array([1,2,4]) * np.array([4,5,6])
```

3. ARRAY vs LIST

Numpy ajoute le type `array` qui est similaire à une liste (`list`) avec la condition supplémentaire que tous les éléments sont du même type.

Particularité sur les listes

```
# creation d'une liste
>>> L = [1,2,3]
print(L)

# creation d'un tableau numpy
>>> A = np.array([1,2,3])
print(A)

# diffence
>>> L + 2         # don't works
>>> L * 2         # pas le meme sens
>>> L + [4]       # meme role que append a peut prés
```

Particularités sur les arrays

```
# tableau numpy

>>> A + 2         #  il fait ce qu'on appelle le broadcasting
>>> A  * 2        #  meme chose
>>> A.append(3)  # pas possible de rallonger le tableau une fois créer  c'est fini

>>> A + np.array([2]) # meme idée que le broadcasting


# Notion de vectorisation
>>> np.sqrt(L)  # retourne un tableau de meme taille
```

- On ne stocke des données que d’un seul type dans un `ndarray`
- On peut avoir des objets `ndarray` avec autant de dimensions que nécessaire (une dimension pour un vecteur, deux dimensions pour une matrice...).
- Les objets `ndarray` constituent un format « minimal » pour stocker des données.
- Les objets `ndarray` possèdent des méthodes spécifiques optimisées permettantde faire des calculs de manière extrêmement rapide.
- Il est possible de stocker des `ndarray` dans des fichiers afin de réduire les ressources nécessaires.

4. DOT PRODUCT

Un tableau peut jouer le rôle d'une matrice si on lui applique une opération de calcul matriciel. Par exemple, la fonction `np.dot()` ou la méthode `.dot()` directement sur l'object permet de réaliser le produit matriciel.

```
>>> a = np.array([[1, 2, 3],
                  [4, 5, 6]])

>>> b = np.array([[4],
                  [2],
                  [1]])
# premier methode
>>> np.dot(a,b)

# Second methode
>>> a.dot(b)
```

NB : Le produit d'une matrice de taille $n \times m$ par une matrice $m \times p$ donne une matrice $n \times p$.

5. SPEED TEST ARRAY vs LIST

Un des avantages important des tableaux Numpy est leur rapidité. Faire des opérations à l'aide de cette structure de données est beaucoup plus efficace que sous une représentation basée sur des listes imbriquées.

Produit matricielle avec des listes

```
# création de deux liste L_1 et L_2
>>> L_1 = np.array([1,2,4,5])

>>> L_2 = np.array([4,5,6,7])

# Fonction effectant le  produit matricielle des deux listes

def prod_list( ):

    somme = 0
    for i in len(L_1):
        somme += L_1[i] * L_[2]

    return somme
```

Produit matricielle avec des arrays

```
# création de  deux arrays A_1 et A_2

>>> A_1 = np.array([1,2,4,5])
>>> A_2 = np.array([4,5,6,7])

# Fonction effectuant le prosuit matricielle de deux arrays
def prod_array():
    return A_1.dot(A_2)
```

Temps d'exécution de la fonction `prod_list()`

```
# importation du module time pour mesurer le temps d'exécution
>>> import time

# nombre d'ittération
>>> T = 1000

# temps d'execution de avec les listes
>>> begin_1  = time.time()
for i in range(T):
    prod_list()

>>> end_1  = time.time()

>>> t_1 = end_1 - begin_1
>>> print("Le temps d'exécution est {} : ".format(t_1))
```

Temps d'exécution de la fonction `prod_array()`

```
# importation du module time pour mesurer le temps d'exécution
>>> import time

# nombre d'ittération
>>> T = 1000

# temps d'execution de avec les listes
>>> begin_2  = time.time()
for i in range(T):
    prod_array()

>>> end_2  = time.time()

>>> t_2 = end_2 - begin_2
>>> print("Le temps d'exécution est {} : ".format(t_2))
```

6. MULTIDIMENSIONNAL ARRAYS

Dans NumPy, un tableau multidimensionnel est représenté par le type `ndarray` (N-**dimensional array**). Le terme `ndarray` est l'abréviation de tableau à n dimensions, ou en d'autres termes, de tableaux multidimensionnels. ndarray est un objet représentant un tableau homogène et multidimensionnel d'éléments de taille fixe.

Les dimensions et le nombre d'éléments sont définis par la forme, c'est-à-dire un tuple de N entiers qui représente le nombre d'éléments dans chaque dimension. Le type d'élément dans le tableau est défini par `dtype` - data-type object.

```
# creation d'un ndarray a 1 D
>>> a = np.array([1,2,4])
# affichage de la dimension
>>> a.shape
# création d'un ndarray a 2D
>>> b = np.array([[1,2,3]])
# affichage de la dimension
>>> b.shape
```

7. ARRAY INDEXNG -SLICING - BOOLEAN INDEXING

array indexing

On peut nauturellement (accessing) aux éléments d'un tableau np.ndarray, comme nous le faisons pour les listes Python.

```
# creation d'une matrice
>>> a = np.array([
    [1,2,3],
    [4,5,6]
])

# accéde au premier range
>>> a[0]

# accéde  au premier element
>>> a[0, 0]

# accéde au dernier element
>>> a[-1,-1]
```

slicing

On peut accéder un sous tableau numpy par tranche (slice)

```
# creation de la matrice
>>> a = np.array([[3,2,6,6],[4,5,9,0]])

# on extrait le sous tableau
>>> a[0:2,0:2]
```

boolean indexing

```
# tableau 2D
>>> t  = np.array([[1,2,4,0],[4,20,10,11]])

# On souhaite avoir lest elements supérieur á 5 dans t
>>> tab_bool = t > 5 # retourne un tableau boolean de meme imension que t

# ensuite on effectue l'operation
>>> t[tab_bool] # on recupére ca sur un tableau a 1 D


# autre possibilité
np.where(t>5, t, 0) # recupere sous forme de tableau 2D
```

8. RESHAPE ARRAY

Les dimensions (la `shape`) d'un tableau peuvent être modifiées de plusieurs façons.

Deux fonctions permettent de modifier les dimensions d'un tableau: `reshape` et `resize`. `resize` retourne une copie du tableau avec les dimensions spécifiées en argument, tandis que `reshape` retourne un tableau pointant vers le même espace mémoire mais ayant les dimensions spécifiées:

```
>>> a = np.array([1,2,3,4,5,6,7,8])

# la methode reshape permet de redimensionner la taille de  a
>>> a.reshape(2,4)

# la methode resize
>>> a.resize(2,3)

# les methode flatten() et ravel() permet d'applatir un tableau
>>> a.flatten() # a.ravel()
```

9. CONCATENATION

Il existe en général trois fonctions permettant de concaténer des tableaux numpy

`concatenate` , `hstack` et `vstack`

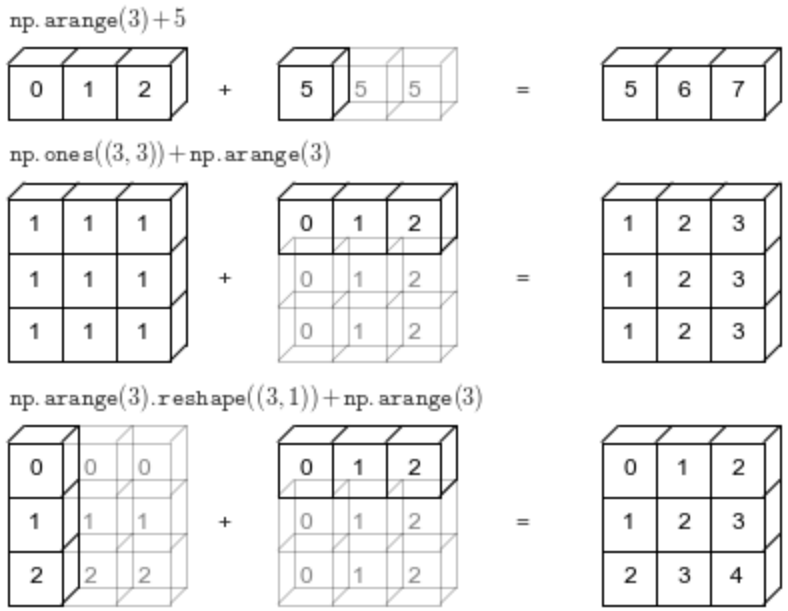
```
# on crée deux matrice / regarder le shape
>>> a = np.array([[1,2,3,4],[3,4,5,6]])
>>> b = np.array([[10,11,13,17]])    # on doit avoir tableau

# utilisation de concatenate
>>> np.concatenate([a,b ], axis=0) # concatene verticalement

# utilisation de vstack
>>> np.vstack([a,b])  # on peut aussi utiliser hstack
```

10. BROADCASTING

Lorsqu'une opération impliquant des tableaux avec différentes formes est effectuée, NumPy essaie de rendre leurs formes compatibles avant que l'opération n'ait lieu. Jetons un coup d'œil à quelques exemples visuels:



```
>>> x = np.array([[1,2,3],[4,5,6], [1,2,3], [4,5,6]])
>>> a = np.array([1,0,1])
# vous pouver regarder l'effet de broadcasting
>>> x + a
```

11. FUNCTIONS AXIS

Toutes les fonctions dans numpy utilisent le principe `axis` du type `ndarray`

```
>>> a = np.array([[1,2,4],[5,5,7]])

# si on applique la methode .sum()
>>> a.sum() # somme tous les elements

# si on precise axis= 0
>>> a.sum(axis=0)  # on la  somme par colone

# si on applique axis =1
>>> a.sum(axis=1)  # on applique la somme par ligne
```

12. DATATYPES

Les éléments d'un tableau multidimensionnel doivent tous être du même type. Il y a quatre types de données de base qui sont les booléens et les nombres entiers, flottants et complexes.

Quand on crée un tableau, le type de données de ses éléments est choisi selon le contexte. Pour connaitre ce type, on peut consulter l'attribut dtype des objets ndarray

```
# creation d'un tableau a
>>> a = np.array([1,2,3,4])

# pour voir le type de donnée stocker dans a
>>> a.dtype

# on peut controler le dtype dés la creation
>>> a = np.array([1,2,3,4], dtype= np.float)
```

13. COPYING

La méthode `copy` permet d'effectuer une copie complète en profondeur (deep copy) du tableau et de ses éléments.

```
# creation de a
>>> a = np.array([1,2,3,4])

# pour faire une copie profond on fais
>>> b = a.copy()
```

14. GENERATING ARRAYS

On peut citer quelques generateurs de tableaux numpy

```
# cree un tableau rempli de 1
>>> np.zeros(4)

# cree un tableau rempli de 1
>>> np.ones(2,2)

# cree un tableau numpy rempli de  d'un nombre choisi
>>> np.full((3,3), 5)

# cree une matrice identite
>>> np.eye(4)

# equivalent a range
>>> np.arange(10)

# cree un intervalle
>>> np.linspace(-100, 100, 100)
```

15. RANDOM NUMBERS

Générations aléatoires simple

- `np.random.randn(10)` : array 1d de 10 nombres d'une distribution gaussienne standard (moyenne 0, écart-type 1).
- `np.random.randn(10, 10)` : array 2d de 10 x 10 nombres d'une distribution gaussienne standard.
- `np.random.randint(1, 5, 10)` : une array 1d de 10 nombres entiers entre 1 et 5, 5 exclus.
- `np.random.random_integers(1, 5, 10)` : une array 1d de 10 nombres entiers entre 1 et 5, 5 inclus.
- `np.random.random_sample(7)` : renvoie 7 valeurs aléatoires dans l'intervalle [0,1[.

Choix

- `np.random.choice(np.array([4, 5, 6, 7, 8])` : renvoie une valeur choisie au hasard dans la liste.
- `np.random.choice(np.array([4, 5, 6, 7, 8], 3)` : renvoie un vecteur de 3 valeurs choisies au hasard dans la liste.
- `np.random.choice(np.array([4, 5, 6, 7, 8], (2, 3))` : renvoie une array 2 x 3 de valeurs choisies au hasard dans la liste.
- `np.random.choice(np.array([4, 5, 6, 7, 8], 3, replace = False)` : renvoie un vecteur de 3 valeurs choisies au hasard dans la liste, sans remise (valeur ne peut être tirée qu'une seule fois).

on peut partir d'une simple liste et/ou donner des probabilités pour chaque élément :

```
np.random.choice(['a', 'b', 'c'], 10, replace = True, p = [0.1, 0.4, 0.5])
```

Variables aléatoires de différentes distributions

`np.random.seed(5)` : pour donner la graine, afin d'avoir des valeurs reproductibles d'un lancement du programme à un autre.

`np.random.binomial(10, 0.3, 7)` : une array de 7 valeurs d'une loi binomiale de 10 tirages avec probabilité de succès de 0.3.

`np.random.binomial(10, 0.3)` : tire une seule valeur d'une loi binomiale à 10 tirages.

`np.random.binomial([10, 50, 100], 0.3)` : tire une seule valeur d'une loi binomiale pour 10, 50 et 100 tirages et renvoie donc une array de 3 valeurs.

`np.random.poisson(1, 7)` : une array de 7 valeurs issues d'une loi de Poisson de paramètre 1.

`np.random.standard_normal(7)` : une array de 7 valeurs issues d'une loi normale standard (moyenne 0, écart-type 1).

`np.random.normal(5, 2, 7)` : une array de 7 valeurs issues d'une loi normale de moyenne 5 et écart-type 2.

`numpy.random.uniform(0, 2, 7)` : une array de 7 valeurs issues d'une loi uniforme entre 0 et 2.

`numpy.random.standard_t(2, 7)` : une array de 7 valeurs issues d'une loi standard t de Student à 2 degrés de liberté.

`np.random.chisquare(2, 7)` : une array de 7 valeurs issues d'une loi du chi 2 à 2 degrés de liberté.

`np.random.f(2, 3, 7)` : une array de 7 valeurs issues d'une loi F de Fisher à 2 et 3 degrés de liberté.

16. LINEAR ALGEBRA

Il est possible de résoudre à l'aide de matrices et de fonctions de l'écosystème Numpy . On va aborder la résolution de systèmes d'équations linéaires.

Systèmes d'équations linéaires

Le premier type de problème qui nous intéresse est la résolution d'un système d'équations linéaires.

$$\begin{cases} x + y = 3 \\ z - 3x = 2 \\ 5y - z = 2 \end{cases}$$

- Pour résoudre le systéme ci-dessus on doit écrire sous forme matricielle

$A = \begin{pmatrix} 1 & 1 & 1 \\ -3 & 0 & 1 \\ 0 & 5 & -1 \end{pmatrix}$	$X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$	$B = \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$

- La forme matricielle de équation aprés transformation est :

$$AX = B$$

- La solution correspond :

$$X = BA^{-1}$$

Pour résoudre ce systéme il faut accéder dans le package `linalg` de numpy et importer la fonction `inv`

```
>>> from numpy.linalg import solve

#creation de des matrices A et b
>>> A =  np.array([
    [1,1,1],
    [-3,0,1],
    [0,5,-1]
])
>>> b = np.array([3,2,1])
# resolution de par la fonction solve
>>> solve(A,b)
```

17. LOAD DATA From csv

Pour importer un fichier `csv` dans un tableau numpy on a le choix entre deux fonctions

- `gentxtx`
- `loadtxt()`

```
# premiere possibilite
>>> tab_1 = np.genfromtxt("wine.csv", delimiter=",",skip_header= 1)

# deuxieme possibilite
>>> tab_2 =  np.loadtxt("wine.csv" , delimiter =",",  skiprows=1)
```