

Python : NumPy

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

`elmouelhi.achref@gmail.com`



- 1 Introduction
- 2 Création d'un `ndarray`
 - `size`
 - `ndim` **et** `shape`
 - `ndmin`
- 3 Types de données
- 4 Génération de valeurs
 - `arange`
 - `reshape`
 - `zeros`
 - `ones`
 - `identity`
 - `eye`
 - `fromfunction`

- 5 Accès aux éléments d'un `ndarray`
- 6 Extraction d'un sous-tableau
- 7 Itération sur un `ndarray`
 - `for ... in`
 - `nditer`
 - `ndenumerate`
- 8 Opérations de base sur les `ndarray`
 - Opérations arithmétiques
 - Fonctions universelles
 - Opérations algébriques

9 Affectation, copie et vue

- Affectation
- Vue
- Copie

10 Fusion de `ndarray`

- `concatenate`
- `stack`
- `hstack`
- `vstack`
- `dstack`
- `block`

11 Décomposition

- `array_split`
- `split`

12 Filtres

Python

NumPy : Numerical Python

- librairie **Python** dédiée aux tableaux (très utilisés en science de données : Data Science)
- open-source
- créée en 2005 par **Travis Oliphant**
- écrite partiellement en **Python** (les parties nécessitant des calculs rapides sont écrites en **C** ou **C++**)

© Achre

Python

NumPy : Numerical Python

- librairie **Python** dédiée aux tableaux (très utilisés en science de données : Data Science)
- open-source
- créée en 2005 par **Travis Oliphant**
- écrite partiellement en **Python** (les parties nécessitant des calculs rapides sont écrites en **C** ou **C++**)

NumPy, pourquoi ?

un accès jusqu'à 50 fois plus rapide que les listes, tuples... de **Python** : **NumPy** stocke les valeurs dans un espace mémoire continu

Python

Documentation officielle

<https://numpy.org/>

Démarche

- Créez un répertoire `cours-numpy` dans votre espace de travail
- Lancez **VSC** et allez dans `File > Open Folder...` et choisissez `cours-numpy`
- Dans `cours-numpy`, créez un fichier `main.py`

Python

Pour installer **NumPy**, lancez la commande

```
pip install numpy
```

© Achref EL MOUELHI ©

Python

Pour installer **NumPy**, lancez la commande

```
pip install numpy
```

Pour utiliser NumPy, il faut l'importer (ici sous l'alias `np`)

```
import numpy as np
```

Python

Pour installer **NumPy**, lancez la commande

```
pip install numpy
```

Pour utiliser **NumPy**, il faut l'importer (ici sous l'alias **np**)

```
import numpy as np
```

Pour afficher la version de **NumPy**

```
import numpy as np

print(np.__version__)
# affiche 1.19.1
```

Python

Pour créer un tableau

```
import numpy as np

tab = np.array([1, 2, 3, 4, 5])

print(tab)
# affiche [1 2 3 4 5]
```

© Achref EL MOUËL

Python

Pour créer un tableau

```
import numpy as np

tab = np.array([1, 2, 3, 4, 5])

print(tab)
# affiche [1 2 3 4 5]
```

Pour déterminer la taille (le nombre d'éléments)

```
print(len(tab))
# affiche 5
```

Python

Pour créer un tableau

```
import numpy as np

tab = np.array([1, 2, 3, 4, 5])

print(tab)
# affiche [1 2 3 4 5]
```

Pour déterminer la taille (le nombre d'éléments)

```
print(len(tab))
# affiche 5
```

Pour déterminer le type

```
print(type(tab))
# affiche <class 'numpy.ndarray'>
```

Explication

- `ndarray` : type de tableau défini par **NumPy**
- La méthode `array()` accepte en paramètre une liste **Python** (à plusieurs dimensions) ou un tuple et retourne un `ndarray`.
- Dimension d'un tableau = nombre d'imbrication

Python

Pour créer un tableau à deux dimensions

```
tab2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

© Achref EL MOUELHI ©

Python

Pour créer un tableau à deux dimensions

```
tab2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

La fonction `len` retourne le nombre d'éléments : ici 2 car le premier tableau contient deux sous-tableaux

```
print(len(tab2))  
# affiche 2
```

Python

Pour créer un tableau à deux dimensions

```
tab2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

La fonction `len` retourne le nombre d'éléments : ici 2 car le premier tableau contient deux sous-tableaux

```
print(len(tab2))  
# affiche 2
```

La propriété `size` retourne le nombre total d'éléments

```
print(tab2.size)  
# affiche 10
```

Python

Pour connaître le nombre de dimensions, on utilise la propriété `ndim`

```
print(tab2.ndim)  
# affiche 2
```

© Achref EL MOUËL

Python

Pour connaître le nombre de dimensions, on utilise la propriété `ndim`

```
print (tab2.ndim)
# affiche 2
```

La propriété `shape` retourne un tuple contenant le nombre d'éléments par dimension

```
print (tab2.shape)
# affiche (2, 5)
```

Python

Pour créer un tableau à trois dimensions (3 niveaux d'imbrications)

```
tab3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

© Achref EL MOUELHI ©

Python

Pour créer un tableau à trois dimensions (3 niveaux d'imbrications)

```
tab3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

La fonction `len` retourne le nombre d'éléments : ici 2 car le premier tableau contient deux sous-tableaux

```
print(len(tab3))  
# affiche 2
```

Python

Pour créer un tableau à trois dimensions (3 niveaux d'imbrications)

```
tab3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

La fonction `len` retourne le nombre d'éléments : ici 2 car le premier tableau contient deux sous-tableaux

```
print(len(tab3))  
# affiche 2
```

Pour connaître le nombre de dimensions, on utilise `ndim`

```
print(tab3.ndim)  
# affiche 3
```

Python

Attention, le tableau suivant a 3 éléments et 2 dimensions

```
array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
print(len(array))  
# affiche 3
```

```
print(array.ndim)  
# affiche 2
```

© Achret L

Python

Attention, le tableau suivant a 3 éléments et 2 dimensions

```
array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(len(array))
# affiche 3

print(array.ndim)
# affiche 2
```

Pour créer un ndarray à partir d'un tuple

```
tup = np.array((1, 2, 3))

print(len(tup))
# affiche 3
```

Python

Remarques

- Un ndarray contenant un seul élément (`np.array(4)` par exemple) est de dimension 0
- Chaque élément d'un ndarray est de dimension 0

Python

Pour créer un tableau à 4 dimensions, on peut faire

```
tab4 = np.array([1, 2, 3, 4], ndmin=4)
```

```
print(tab4)  
# affiche [[[1 2 3 4]]]
```

```
print(tab4.ndim)  
# affiche 4
```

Python

Types de données autorisés par NumPy

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string : chaîne + codage (**utf-8** par exemple)
- U - unicode string : chaîne non-codée
- ...

Python

Pour connaître le type d'un tableau, on utilise la propriété `dtype`

```
tab = np.array([1, 2, 3, 4, 5])
```

```
print(tab.dtype)
```

```
# affiche int32 (32 bits = 4 octets)
```

© Achref EL MOU...

Python

Pour connaître le type d'un tableau, on utilise la propriété `dtype`

```
tab = np.array([1, 2, 3, 4, 5])

print(tab.dtype)
# affiche int32 (32 bits = 4 octets)
```

Pour préciser le nombre d'octets

```
tab = np.array([1, 2, 3, 4, 5], dtype='i8')
print(tab.dtype)

# affiche int64
```

On peut aussi choisir un autre type

```
tab = np.array([1, 2, 3, 4, 5], dtype='S')
```

```
print(tab.dtype)
```

```
# affiche |S1
```

© Achref EL MOUELHI ©

On peut aussi choisir un autre type

```
tab = np.array([1, 2, 3, 4, 5], dtype='S')  
  
print(tab.dtype)  
# affiche |S1
```

Ou en utilisant la méthode `astype`

```
tab = np.array([1, 2, 3, 4, 5])  
strings = tab.astype('S')  
  
print(strings.dtype)  
# affiche |S11
```

On peut aussi choisir un autre type

```
tab = np.array([1, 2, 3, 4, 5], dtype='S')  
  
print(tab.dtype)  
# affiche |S1
```

Ou en utilisant la méthode `astype`

```
tab = np.array([1, 2, 3, 4, 5])  
strings = tab.astype('S')  
  
print(strings.dtype)  
# affiche |S11
```

Ou

```
tab = np.array([1, 2, 3, 4, 5])  
strings = tab.astype(str)  
  
print(strings.dtype)  
# affiche |U11
```

Remarques

- Une exception sera levée si une valeur ne peut être convertie.
- Les types pour lesquels on peut préciser la taille sont `i`, `u`, `f`, `S` et `U`.

Pour générer 15 valeurs incrémentales pour un tableau à une dimension

```
t = np.arange(15)
```

```
print(t)
```

```
# affiche [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

© Achref EL MOUELHI ©

Pour générer 15 valeurs incrémentales pour un tableau à une dimension

```
t = np.arange(15)
```

```
print(t)
```

```
# affiche [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

```
arange([start, ]stop, [step, ]dtype=None)
```

- Valeur par défaut pour `start` est 0
- Valeur par défaut pour `step` est 1

Pour générer 15 valeurs incrémentales pour un tableau à une dimension

```
t = np.arange(15)

print(t)
# affiche [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

`arange([start,]stop, [step,]dtype=None)`

- Valeur par défaut pour `start` est 0
- Valeur par défaut pour `step` est 1

Pour générer 15 valeurs incrémentales commençant de 1

```
t = np.arange(1, 16, 1)

print(t)
# affiche [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
```

Pour générer 15 valeurs incrémentales pour un tableau à plusieurs dimensions (ici 2)

```
t2 = np.arange(1, 16, 1).reshape(3, 5)
```

```
print(t2)
```

```
# affiche [[ 1  2  3  4  5] [ 6  7  8  9 10] [11 12 13 14 15]]
```

© Achref EL MOUELHI ©

Pour générer 15 valeurs incrémentales pour un tableau à plusieurs dimensions (ici 2)

```
t2 = np.arange(1, 16, 1).reshape(3, 5)
```

```
print(t2)
```

```
# affiche [[ 1  2  3  4  5] [ 6  7  8  9 10] [11 12 13 14 15]]
```

Pour générer 12 valeurs incrémentales pour un tableau à trois dimensions

```
t3 = np.arange(1, 13, 1).reshape(3, 2, 2)
```

```
print(t3)
```

```
# affiche [[[ 1  2] [ 3  4]] [[ 5  6] [ 7  8]] [[ 9 10] [11 12]]]
```

Pour générer 15 valeurs incrémentales pour un tableau à plusieurs dimensions (ici 2)

```
t2 = np.arange(1, 16, 1).reshape(3, 5)

print(t2)
# affiche [[ 1  2  3  4  5] [ 6  7  8  9 10] [11 12 13 14 15]]
```

Pour générer 12 valeurs incrémentales pour un tableau à trois dimensions

```
t3 = np.arange(1, 13, 1).reshape(3, 2, 2)

print(t3)
# affiche [[[ 1  2] [ 3  4]] [[ 5  6] [ 7  8]] [[ 9 10] [11 12]]]
```

Si les valeurs passées pour les paramètres ne permettent pas de créer le tableau, une exception sera levée

```
t3 = np.arange(1, 13, 1).reshape(3, 2, 3)

print(t3)
# affiche ValueError: cannot reshape array of size 12 into shape
(3,2,3)
```

Python

La méthode `reshape()` accepte aussi comme paramètre un tableau de dimension différente

```
t = np.arange(1, 13, 1)
t3 = t.reshape(3, 2, -1)

print(t3)
# affiche [[[ 1  2] [ 3  4]] [[ 5  6] [ 7  8]] [[ 9 10] [11 12]]]
```

© Achref EL M...

Python

La méthode `reshape()` accepte aussi comme paramètre un tableau de dimension différente

```
t = np.arange(1, 13, 1)
t3 = t.reshape(3, 2, -1)

print(t3)
# affiche [[[ 1  2] [ 3  4]] [[ 5  6] [ 7  8]] [[ 9 10] [11 12]]]
```

On peut aussi utiliser la valeur `-1` (une seule fois) pour que NumPy calcule le nombre manquant

```
t3 = np.arange(1, 13, 1).reshape(3, 2, -1)

print(t3)
# affiche [[[ 1  2] [ 3  4]] [[ 5  6] [ 7  8]] [[ 9 10] [11 12]]]
```

Python

`reshape(-1)` permet de convertir un tableau quelle que soit sa dimension en un tableau d'une seule dimension

```
t3 = np.arange(1, 13, 1).reshape(3, 2, -1)
t = t3.reshape(-1)
```

```
print(t)
# affiche [ 1  2  3  4  5  6  7  8  9 10 11 12]
```

Python

Pour créer un tableau et l'initialiser avec des zéros (réels)

```
zeros = np.zeros(3)

print(zeros)
# affiche [ 0.  0.  0.]
```

© Achref EL MOUELHI ©

Python

Pour créer un tableau et l'initialiser avec des zéros (réels)

```
zeros = np.zeros(3)

print(zeros)
# affiche [ 0.  0.  0.]
```

Pour créer un tableau et l'initialiser avec des zéros (entiers)

```
zeros = np.zeros(3, dtype=np.int32)

print(zeros)
# affiche [ 0  0  0]
```

Python

Pour créer un tableau et l'initialiser avec des zéros (réels)

```
zeros = np.zeros(3)

print(zeros)
# affiche [ 0.  0.  0.]
```

Pour créer un tableau et l'initialiser avec des zéros (entiers)

```
zeros = np.zeros(3, dtype=np.int32)

print(zeros)
# affiche [ 0  0  0]
```

Pour créer un tableau de zéros à deux dimensions (ou plus)

```
zeros = np.zeros((3,2), dtype=np.int32)

print(zeros)
# affiche [[0 0] [0 0] [0 0]]
```

Python

ones

- fonctionne exactement comme `zeros`
- retourne un tableau de 1

Python

Pour généré un tableau identité (au sens matrice carrée)

```
iden = np.identity(2)
```

```
print(iden)
```

```
# affiche
```

```
# [
```

```
# [1. 0.]
```

```
# [0. 1.]
```

```
# ]
```

Python

`identity` est un cas particulier de `eye`

```
oeil = np.eye(2)
```

```
print(oeil)
```

```
# affiche
```

```
# [
```

```
# [1. 0.]
```

```
# [0. 1.]
```

```
# ]
```

Python

`eye` permet de générer aussi des matrices non-carrées

```
oeil = np.eye(2, 3)
```

```
print(oeil)
```

```
# affiche
```

```
# [
```

```
# [1. 0. 0.]
```

```
# [0. 1. 0.]
```

```
# ]
```

Python

`eye` permet aussi de faire une translation de la diagonale

```
oeil = np.eye(3, 3, 1)
```

```
print(oeil)
```

```
# affiche
```

```
# [
```

```
# [0. 1. 0.]
```

```
# [0. 0. 1.]
```

```
# [0. 0. 0.]
```

```
# ]
```

Python

Pour générer les valeurs d'un tableau à partir d'une fonction lambda

```
fonction = np.fromfunction(lambda i, j: i + j, (3, 2))

print(fonction)
# affiche
# [
#  [0. 1.]
#  [1. 2.]
#  [2. 3.]
# ]
```

Pour accéder au premier élément d'un tableau d'une dimension

```
tab = np.array([1, 2, 3, 4, 5])
```

```
print(tab[0])  
# affiche 1
```

© Achref EL MOUELHI ©

Pour accéder au premier élément d'un tableau d'une dimension

```
tab = np.array([1, 2, 3, 4, 5])
```

```
print(tab[0])  
# affiche 1
```

Pour accéder au deuxième élément de la deuxième dimension d'un tableau à deux dimensions

```
tab2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

```
print(tab2[0, 1])  
# affiche 2
```

Pour accéder au premier élément d'un tableau d'une dimension

```
tab = np.array([1, 2, 3, 4, 5])
```

```
print(tab[0])  
# affiche 1
```

Pour accéder au deuxième élément de la deuxième dimension d'un tableau à deux dimensions

```
tab2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

```
print(tab2[0, 1])  
# affiche 2
```

Pour accéder au deuxième élément du premier tableau défini dans le deuxième tableau

```
tab3 = np.array([[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]])
```

```
print(tab3[1, 0, 1])  
# affiche 8
```

Python

Exemples avec des valeurs négatives

```
tab = np.array([1, 2, 3, 4, 5])
```

```
print(tab[-1])
```

```
# affiche 5
```

```
tab2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

```
print(tab2[-1, -3])
```

```
# affiche 8
```

```
tab3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

```
print(tab3[1, 0, -2])
```

```
# affiche 8
```

Démarche

- Pour extraire un sous tableau, on utilise l'opérateur `[deb:fin:pas]` ou `[deb:fin]`
- `deb`, `fin` et `pas` sont optionnels
- La valeur par défaut de `deb` est 0
- La valeur par défaut de `fin` est la taille du tableau
- La valeur par défaut de `pas` est 1

Python

Exemples

```
tab = np.array([1, 2, 3, 4, 5])
```

```
print(tab[1])  
# affiche 2
```

```
print(tab[1:])  
# affiche [2 3 4 5]
```

```
print(tab[1:4])  
# affiche [2 3 4]
```

```
print(tab[1:4:2])  
# affiche [2 4]
```

```
print(tab[1::2])  
# affiche [2 4]
```

Python

Exemples avec un tableau à deux dimensions

```
tab2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

```
print(tab2[0:2, 2:5])  
# affiche [[ 3  4  5]  [ 8  9 10]]
```

```
print(tab2[0:2, 2])  
# affiche [3 8]
```

```
print(tab2[1, 2:5])  
# affiche [ 8  9 10]
```

```
print(tab2[0:2, 2:5:2])  
# affiche [[ 3  5]  [ 8 10]]
```

```
print(tab2[0:1, 1::2])  
# affiche [[2 4]]
```

Python

Pour itérer sur un tableau à une dimension

```
tab = np.array([1, 2, 3, 4, 5])
```

```
for elt in tab:  
    print(elt)
```

```
# affiche
```

```
# 1
```

```
# 2
```

```
# 3
```

```
# 4
```

```
# 5
```

Python

Itérer sur un tableau à deux dimensions avec une seule boucle affiche

```
tab2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])  
  
for elt in tab2:  
    print(elt)  
  
# affiche  
# [1 2 3 4 5]  
# [ 6  7  8  9 10]
```

Python

Pour accéder à tous les éléments

```
tab2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

```
for elt in tab2:  
    for sub_elt in elt:  
        print(sub_elt)
```

```
# affiche
```

```
# 1
```

```
# 2
```

```
# 3
```

```
# 4
```

```
# 5
```

```
# 6
```

```
# 7
```

```
# 8
```

```
# 9
```

```
# 10
```

Python

Par analogie, pour un tableau à 3 dimensions, il faut 3 boucles `for`

```
tab3 = np.array([[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]])
```

```
for x in tab3:
    for y in x:
        for z in y:
            print(z)
```

```
# affiche
```

```
# 1
# 2
# 3
# 4
# 5
# 6
# 7
# 8
# 9
# 10
# 11
# 12
```

Python

NumPy nous offre une syntaxe plus simple pour simplifier le code précédent quelle que soit la dimension du tableau

```
tab3 = np.array([[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]])
```

```
for x in np.nditer(tab3):  
    print(x)
```

```
# affiche
```

```
# 1
```

```
# 2
```

```
# 3
```

```
# 4
```

```
# 5
```

```
# 6
```

```
# 7
```

```
# 8
```

```
# 9
```

```
# 10
```

```
# 11
```

```
# 12
```

Python

Pour avoir l'indice ainsi que la valeur, on utilise `ndenumerate`

```
tab = np.array([1, 2, 3, 4, 5])

for index, elt in np.ndenumerate(tab):
    print(index, elt)

# affiche
# (0,) 1
# (1,) 2
# (2,) 3
# (3,) 4
# (4,) 5
```

Python

Exemple avec un tableau à deux dimensions

```
tab2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

```
for index, elt in np.ndenumerate(tab2):  
    print(index, elt)
```

```
# affiche  
# (0, 0) 1  
# (0, 1) 2  
# (0, 2) 3  
# (0, 3) 4  
# (0, 4) 5  
# (1, 0) 6  
# (1, 1) 7  
# (1, 2) 8  
# (1, 3) 9  
# (1, 4) 10
```

Vectorisation

- Approche utilisée pour remplacer les boucles explicites (comme `for` ou `while`).
- Permet d'effectuer des opérations sur des vecteurs entiers sans avoir besoin d'itérer sur chaque élément.
- Conçue pour fonctionner efficacement avec des structures optimisées comme les tableaux **NumPy**.
- Les opérations sont exécutées en interne avec du code optimisé en langage **C**.

Opérations vectorisées

- **Opérations arithmétiques** : Addition (+), soustraction (-), multiplication (*), division (/), puissance (**)...
- **Opérations logiques** : comme ==, !=, <, >, <=, >=
- **Fonctions trigonométriques** : `np.sin()`, `np.cos()`, `np.tan()` ...
- **Opérations mathématiques** : `np.exp()`, `np.log()`, `np.sqrt()` ...
- **Opérations d'agrégation** : `np.sum()`, `np.mean()`, `np.median()`, `np.min()`, `np.max()`
- ...

Python

Considérons les deux tableaux (à une dimension) suivants

```
a = np.arange(1, 6)
b = np.arange(1, 10, 2)
```

© Achref EL MOUELHI ©

Python

Considérons les deux tableaux (à une dimension) suivants

```
a = np.arange(1, 6)
b = np.arange(1, 10, 2)
```

Il est possible d'effectuer les opérations arithmétiques suivantes

```
r = a + b
print(a, b, r)
# affiche [1 2 3 4 5] [1 3 5 7 9] [ 2  5  8 11 14]

r = a - b
print(a, b, r)
# affiche [1 2 3 4 5] [1 3 5 7 9] [ 0 -1 -2 -3 -4]

r = a * b
print(a, b, r)
# affiche [1 2 3 4 5] [1 3 5 7 9] [ 1  6 15 28 45]

r = a / b
print(a, b, r)
# affiche [1 2 3 4 5] [1 3 5 7 9] [1.  0.66666667 0.6 0.57142857 0.55555556]

r = a * 2
print(a, r)
# affiche [1 2 3 4 5] [ 2  4  6  8 10]

r = a ** 2
print(a, r)
# affiche [1 2 3 4 5] [ 1  4  9 16 25]
```

Python

Les opérateurs suivants sont également applicables

```
a += 2
print(a)
# affiche [3 4 5 6 7]

a *= 2
print(a)
# affiche [ 6  8 10 12 14]
```

Python

Attention pour les tableaux à deux dimensions ou plus, le produit ne correspond pas au produit de deux matrices

```
a2 = np.arange(1,5).reshape(2, 2)
b2 = np.arange(3,7).reshape(2, 2)
```

```
print (a2)
# affiche [[1 2] [3 4]]
```

```
print (b2)
# affiche [[3 4] [5 6]]
```

```
print (a2 * b2)
# affiche [[ 3  8] [15 24]]
```

© Achref EL

Python

Attention pour les tableaux à deux dimensions ou plus, le produit ne correspond pas au produit de deux matrices

```
a2 = np.arange(1,5).reshape(2, 2)
b2 = np.arange(3,7).reshape(2, 2)
```

```
print (a2)
# affiche [[1 2] [3 4]]
```

```
print (b2)
# affiche [[3 4] [5 6]]
```

```
print (a2 * b2)
# affiche [[ 3  8] [15 24]]
```

Pour faire le produit de deux matrices, on utilise l'opérateur @ (valable depuis Python 3.5)

```
print (a2)
# affiche [[1 2] [3 4]]
```

```
print (b2)
# affiche [[3 4] [5 6]]
```

```
print (a2 @ b2)
# affiche [[13 16] [29 36]]
```

Python

Ou en appelant la méthode `dot()`

```
print (a2)
# affiche [[1 2] [3 4]]

print (b2)
# affiche [[3 4] [5 6]]

print (a2.dot(b2))
# affiche [[13 16] [29 36]]
```

Python

Il est possible d'appliquer une fonction sur tous les éléments d'un tableau

```
print(np.sqrt((a)))  
# affiche [1.          1.41421356 1.73205081 2.  
          2.23606798]
```

Python

Autres fonctions

- `exp` : retourne un tableau d'exponentielle
- `average` : retourne la moyenne de tous les éléments d'un tableau
- `sum` : retourne la somme de tous les éléments d'un tableau
- `prod` : retourne le produit de tous les éléments d'un tableau
- `diff` : retourne le tableau de différence entre chaque élément et son successeur
- `max` : retourne le max d'un tableau
- `maximum` : retourne un tableau de tous les max en comparant les éléments ayant le même indice
- `all` : vérifie si tous les éléments d'un tableau respectent une condition
- `any` : vérifie s'il existe un élément d'un tableau respectant une condition
- `where` : retourne un tableau transformé selon la condition indiquée
- `sort` : retourne un tableau trié quel que soit le type de ses données
- ...

Python

Pour récupérer les indices des éléments pairs

```
print(np.where(a % 2 == 0))  
# affiche (array([1, 3], dtype=int32),)
```

© Achref EL MOU

Python

Pour récupérer les indices des éléments pairs

```
print(np.where(a % 2 == 0))  
# affiche (array([1, 3], dtype=int32),)
```

Pour multiplier les éléments pairs par 2 et les impairs par 3

```
print(np.where(a % 2 == 0, a * 2, a * 3))  
# affiche [ 3  4  9  8 15]
```

Python

Pour récupérer le max d'un tableau)

```
print(np.max(a))  
# affiche 5
```

© Achref EL MOUËZ

Python

Pour récupérer le max d'un tableau)

```
print(np.max(a))  
# affiche 5
```

Pour récupérer le max de chaque couple d'éléments ayant le même indice de deux tableaux différents

```
print(np.maximum(a, b))  
# affiche [1 3 5 7 9]
```

Python

Pour vérifier si tous les éléments d'un tableau vérifient une condition

```
print(np.all(a % 2 == 0))  
# affiche False
```

© Achref EL MOUELHI ©

Python

Pour vérifier si tous les éléments d'un tableau vérifient une condition

```
print(np.all(a % 2 == 0))  
# affiche False
```

Pour vérifier s'il existe un élément du tableau vérifiant une condition

```
print(np.any(a % 2 == 0))  
# affiche True
```

Python

Pour vérifier si tous les éléments d'un tableau vérifient une condition

```
print(np.all(a % 2 == 0))  
# affiche False
```

Pour vérifier s'il existe un élément du tableau vérifiant une condition

```
print(np.any(a % 2 == 0))  
# affiche True
```

Pour retourner le tableau de différence entre chaque élément et son successeur

```
print(np.diff(b))  
# affiche [2 2 2 2]
```

Python

Considérons la matrice suivante

```
a2 = np.arange(1, 5).reshape(2, 2)
```

```
print (a2)
```

```
# affiche [[1 2] [3 4]]
```

© Achref EL MOUELHI

Python

Considérons la matrice suivante

```
a2 = np.arange(1,5).reshape(2, 2)
```

```
print (a2)  
# affiche [[1 2] [3 4]]
```

Pour déterminer le transposé de a2

```
print (a2.transpose())  
# affiche [[1 3] [2 4]]
```

Python

Considérons la matrice suivante

```
a2 = np.arange(1, 5).reshape(2, 2)
```

```
print (a2)  
# affiche  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ 
```

Pour déterminer le transposé de a_2

```
print (a2.transpose())  
# affiche  $\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$ 
```

Pour calculer la trace de a_2

```
print (a2.trace())  
# affiche 5
```

Explication

- Affectation : permet d'avoir un deuxième nom pour le même tableau
- Vue : permet de créer un deuxième tableau dont les valeurs pointent sur les mêmes valeurs du tableau précédent
- Copie : permet de créer un tableau avec un espace mémoire différent pour les valeurs

Python

Exemple avec l'affectation

```
a = np.arange(1, 4)

b = a

b[2] = 10

print(a)
# affiche [ 1  2 10]

print(b is a)
# affiche True

print(id(a), id(b))
# affiche 58092304 58092304
```

Python

Exemple avec `view()`

```
a = np.arange(1, 4)

c = a.view()

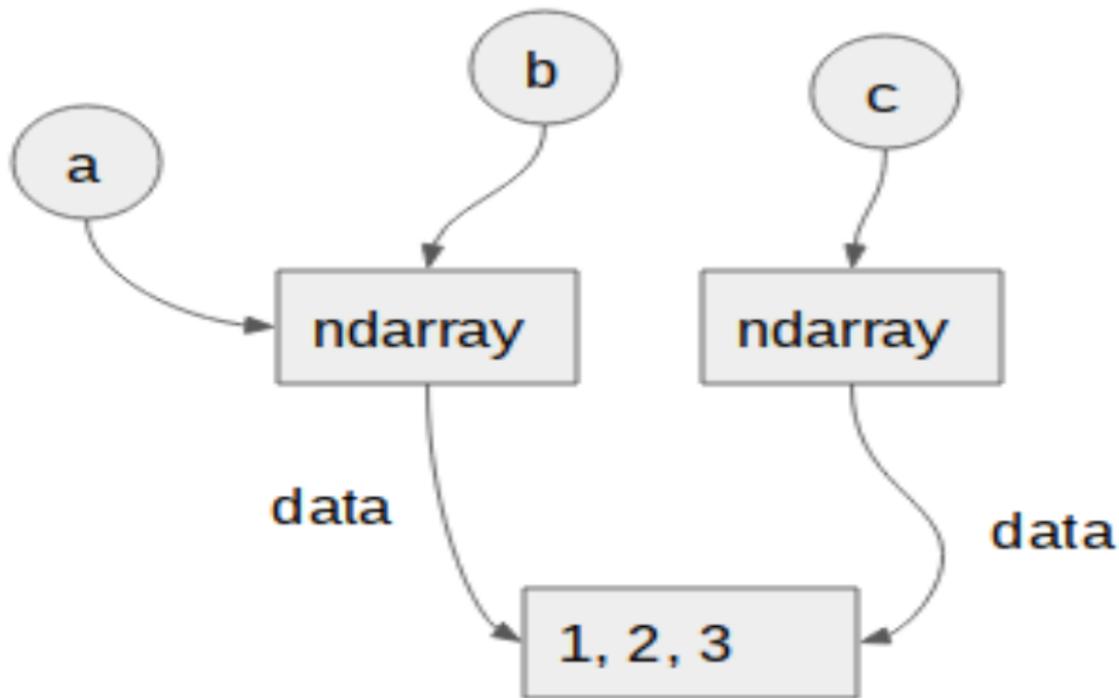
c[2] = 10

print(a)
# affiche [ 1  2 10]

print(c is a)
# affiche False

print(id(a), id(c))
# affiche 51800928 51801488
```

Python



(Image de **StackOverFlow**)

Python

Exemple avec `copie()`

```
a = np.arange(1, 4)

d = a.copy()

d[2] = 10

print(a)
# affiche [ 1  2  3]

print(d is a)
# affiche False

print(id(a), id(d))
# affiche 17984272 17984832
```

Python

Considérons les deux tableaux (à deux dimensions) suivants

```
m1 = np.arange(1, 5).reshape(2, 2)
```

```
print(m1)
```

```
# affiche [[1 2] [3 4]]
```

```
m2 = np.arange(5, 9).reshape(2, 2)
```

```
print(m2)
```

```
# affiche [[5 6] [7 8]]
```

Python

Pour fusionner les deux tableaux et obtenir un tableau à deux dimensions contenant les éléments du premier tableau ensuite le second

```
m = np.concatenate((m1, m2))  
  
print(m)  
# affiche [[1 2] [3 4] [5 6] [7 8]]
```

© Achref EL

Python

Pour fusionner les deux tableaux et obtenir un tableau à deux dimensions contenant les éléments du premier tableau ensuite le second

```
m = np.concatenate((m1, m2))  
  
print(m)  
# affiche [[1 2] [3 4] [5 6] [7 8]]
```

Le code précédent est un raccourci du code suivant

```
m = np.concatenate((m1, m2), axis=0)  
  
print(m)  
# affiche [[1 2] [3 4] [5 6] [7 8]]
```

Python

Pour fusionner les éléments de même indice ensemble

```
m = np.concatenate((m1, m2), axis=1)
```

```
print(m)
```

```
# affiche [[1 2 5 6]  [3 4 7 8]]
```

© Achref EL MOU

Python

Pour fusionner les éléments de même indice ensemble

```
m = np.concatenate((m1, m2), axis=1)
```

```
print(m)
```

```
# affiche [[1 2 5 6] [3 4 7 8]]
```

Pour fusionner et obtenir un tableau à une seule dimension

```
m = np.concatenate((m1, m2), axis=None)
```

```
print(m)
```

```
# affiche [1 2 3 4 5 6 7 8]
```

Python

Pour tout fusionner dans un nouveau tableau (de dimension 3)

```
m = np.stack((m1, m2), axis=0)
```

```
print(m)
```

```
# affiche [[[1 2] [3 4]] [[5 6] [7 8]]]
```

© Achref EL MOU

Python

Pour tout fusionner dans un nouveau tableau (de dimension 3)

```
m = np.stack((m1, m2), axis=0)
```

```
print(m)
```

```
# affiche [[[1 2] [3 4]] [[5 6] [7 8]]]
```

Pour fusionner les éléments de de même indice dans un tableau ensemble

```
m = np.stack((m1, m2), axis=1)
```

```
print(m)
```

```
# affiche [[[1 2] [5 6]] [[3 4] [7 8]]]
```

Python

Pour fusionner (horizontalement) les éléments de même indice ensemble (comme `concatenate` avec `axis=1`)

```
m = np.hstack((m1, m2))  
  
print(m)  
# affiche [[1 2 5 6]  [3 4 7 8]]
```

Python

Pour fusionner (verticalement) tous les éléments dans un seul tableau (comme concatenate avec `axis=0`)

```
m = np.vstack((m1, m2))
```

```
print(m)
```

```
# affiche [[1 2] [3 4] [5 6] [7 8]]
```

Python

Pour fusionner selon la profondeur

```
m = np.dstack((m1, m2))
```

```
print(m)
```

```
# affiche [[[1 5] [2 6]] [[3 7] [4 8]]]
```

Python

Pour construire un nouveau tableau par bloc

```
m = np.block([m1, m2])  
  
print(m)  
# affiche [[1 2 5 6] [3 4 7 8]]
```

Python

Ou sous forme d'un tableau à plusieurs dimensions

```
m = np.block(  
    [  
        [m1, np.zeros((2, 2), dtype=np.int32)],  
        [np.ones(4, dtype=np.int32)],  
        [np.zeros((2, 2), dtype=np.int32), m2]  
    ]  
)
```

```
print(m)
```

```
# affiche [[1 2 0 0]  
#  [3 4 0 0]  
#  [1 1 1 1]  
#  [0 0 5 6]  
#  [0 0 7 8]]
```

Python

Pour décomposer un tableau en plusieurs sous-tableaux, on peut utiliser la méthode `array_split`

```
t = np.arange(1, 11)

print (np.array_split(t, 2))
# affiche [
#     array([1, 2, 3, 4, 5]),
#     array([ 6,  7,  8,  9, 10])
# ]

print (np.array_split(t, 3))
# affiche [
#     array([1, 2, 3, 4]),
#     array([5, 6, 7]),
#     array([ 8,  9, 10])
# ]
```

Python

Exemple avec un tableau à deux dimensions

```
t2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])

print(np.array_split(t2, 3))
# affiche [array([[1, 2, 3],
#                [4, 5, 6]]), array([[ 7,  8,  9],
#                [10, 11, 12]]), array([[13, 14, 15],
#                [16, 17, 18]])]

print(np.array_split(t2, 2))
# affiche [array([[1, 2, 3],
#                [4, 5, 6],
#                [7, 8, 9]]), array([[10, 11, 12],
#                [13, 14, 15],
#                [16, 17, 18]])]
```

Python

On peut aussi utiliser la méthode `split` qui lève une exception si la taille du tableau n'est pas divisible par le paramètre

```
t = np.arange(1, 11)

print (np.array_split(t, 2))
# affiche [
#     array([1, 2, 3, 4, 5]),
#     array([ 6,  7,  8,  9, 10])
# ]

print (np.array_split(t, 3))
# affiche ValueError: array split does not result in an
#     equal division
```

Python

Remarque

- Comme pour `stack` et ses variantes `hstack`, `vstack` et `dstack`
- Il existe `hsplit`, `vsplit` et `dsplit`

Python

Pour filtrer les éléments impairs, on peut utiliser `where` (qui retourne un tableau d'indices)

```
tab = np.array([1, 2, 3, 4, 5])  
  
print(np.where(tab % 2 != 0))  
# (array([0, 2, 4], dtype=int32),)
```

© Achref EL M...

Python

Pour filtrer les éléments impairs, on peut utiliser `where` (qui retourne un tableau d'indices)

```
tab = np.array([1, 2, 3, 4, 5])  
  
print(np.where(tab % 2 != 0))  
# (array([0, 2, 4], dtype=int32),)
```

Si on a besoin de valeurs, on peut définir un tableau de booléens

```
filtre = [True, False, True, False, True]  
  
print(tab[filtre])  
# affiche [1 3 5]
```

Python

On peut aussi utiliser `where` pour construire le filtre (tableau de booléens)

```
filtre = np.where(tab % 2 != 0, True, False)

print(tab[filtre])
# affiche [1 3 5]
# (array([0, 2, 4], dtype=int32),)
```

© Achref EL

Python

On peut aussi utiliser `where` pour construire le filtre (tableau de booléens)

```
filtre = np.where(tab % 2 != 0, True, False)

print(tab[filtre])
# affiche [1 3 5]
# (array([0, 2, 4], dtype=int32),)
```

Ou sans le `where`

```
filtre = tab % 2 != 0

print(tab[filtre])
# affiche [1 3 5]
```