```
##########################
# Manipuler les ndarray
##########################

import numpy as np

#======================
# Indexing et slicing
#======================

#-------------------------------------
# Les index
#-------------------------------------

# Commençons par instancier un ndarray à 2 dimensions que nous allons manipuler avec les index.

matrice = np.array([[0,1,2,3,4], [5,6,7,8,9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]])
print(matrice)
print("-----------------------------------------------------------------------")
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

La position du premier indice est égal à 0. (On parle de 0-indexing.)
Si on veut récupérer le premier élément de cette matrice, qui est le ndarray [0 1 2 3 4]
Alors il faut utiliser l'indice 0.

```
matrice = np.array([[0,1,2,3,4], [5,6,7,8,9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]])
print(matrice[0])
print("-----------------------------------------------------------------------")
```

```
[0 1 2 3 4]
```

Par ailleurs, on peut tester le type de matrice[0] et voir que c'est bien un ndarray.

```
matrice = np.array([[0,1,2,3,4], [5,6,7,8,9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]])
print(type(matrice[0]))
print("-----------------------------------------------------------------------")
```

```
<class 'numpy.ndarray'>
```

```
# Comment obtenir le premier élément du premier ndarray de la matrice définie ci-dessus?

matrice = np.array([[0,1,2,3,4], [5,6,7,8,9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]])
print(matrice[0][0])
print("-----------------------------------------------------------------------")
```

```
0
```

Si matrice[0] est un ndarray, alors pour obtenir son premier élément, il faut utiliser une deuxième fois l'Indexing avec les crochets []. (Voir code ci-dessus).
Obtenir le troisième élément de la deuxième ligne de la matrice définie ci-dessus?
Si vous écrivez : matrice[1][2], cela renverra le 3ème élément de la 2ème ligne.
Alors on en conclue donc que les premiers crochets nous permettent d'obtenir une ligne spécifique, et les seconds crochets nous permettent d'obtenir une colonne spécifique.

```
# Obtenir le troisième élément de la deuxième ligne de la matrice définie ci-dessus?

matrice = np.array([[0,1,2,3,4], [5,6,7,8,9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]])
```

```
print(matrice)
print()
print(matrice[1][2])
print("------------------------------------------------------------------------")
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

7
```

7 A la 2ème ligne, 3ème colonne, on a le chiffre 7.
Que se passe-t-il si on indique un index qui ne correspond à rien dans la matrice?
La matrice ci-dessus possède 4 lignes. Regardons ce qui se passe si on essaie d'afficher la 5ème ligne (qui n'existe pas).

```
matrice = np.array([[0,1,2,3,4], [5,6,7,8,9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]])
print(matrice[5])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-18-810f89dcb2f6> in <module>
      1 matrice = np.array([[0,1,2,3,4], [5,6,7,8,9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]])
      2
----> 3 print(matrice[5])
```

IndexError: index 5 is out of bounds for axis 0 with size 4
On obtient une erreur nommée IndexError, qui nous indique que nous avons donné un indice hors-limite.
Il existe une autre façon d'accéder à un élément dans un ndarray
Au lieu d'utiliser des double crochets [][] pour accéder à un élément dans un ndarray à 2 dimensions, on peut utiliser cette écriture : [num_ligne, num_colone]

```
# Il existe une autre façon d'accéder à un élément dans un ndarray

matrice = np.array([[0,1,2,3,4], [5,6,7,8,9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]])
print(matrice[1,2])
print("------------------------------------------------------------------------")
```

```
7
```

matrice[1,2] affiche l'élément à la deuxième ligne et à la troisième colonne.

```
#-------------------------------------
# Les index négatifs
#-------------------------------------

#Créons un vecteur composé de 10 éléments.

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

```
[ 1  2  3  4  5  6  7  8  9 10]
```

Il est possible d'atteindre le dernier élément de ce vecteur en utilisant l'index -1.

```
# Il est possible d'atteindre le dernier élément de ce vecteur en utilisant l'index -1.

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

10
On obtient bien le nombre 10, qui est le dernier élément de la variable vecteur. Les indices négatifs commencent à l'index -1. Voici comment cela fonctionne : [-1] : dernier élément [-2] : avant-dernier élément [-3] : 3ème élément en partant de la fin etc... etc... Bien entendu, ces éléments doivent exister, sinon cela lèvera l'erreur IndexError.

```
# Obtenir le 4ème élément en partant de la fin

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

7
Le 4ème élément en partant de la fin est bien le chiffre 7.

```
# Cela marche aussi avec les matrices, bien entendu

matrice = np.array([[1,2], [3,4], [5,6]])
matrice[-1]
print("------------------------------------------------------------------")
```

array([5, 6])
Ici, on a récupéré la dernière ligne de la variable matrice.

```
#--------------------------------------
# Le slicing
#--------------------------------------
```

Jusque-là, nous nous sommes servis des index pour récupérer un élément particulier. Cet élément pouvant être un ndarray à 0 ou 1 dimension. Il est possible avec le slicing de récupérer des matrices. Prenons une matrice de taille (5x4) -> 5 lignes, 4 colonnes

```
# Il est possible avec le slicing de récupérer des matrices.

matrice = np.array([[0,1,2,3], [4,5,6,7], [8,9,10, 11], [12, 13, 14, 15], [16,17,18,19]])
print(matrice)
print("------------------------------------------------------------------")
```

[[0 1 2 3] [4 5 6 7] [8 9 10 11] [12 13 14 15] [16 17 18 19]]
Jusqu'ici, vous êtes capable de récupérer la n-ème ligne ou le m-ème élément de la n-ème ligne. Mais vous ne savez pas encore comment récupérer tous les éléments à la colonne 2 et 3 de chaque ligne. Pour répondre à cette problématique, on va utiliser une méthode que l'on appelle le slicing. On l'utilise comme ceci : [indice_debut: indice_fin]. Cela va récupérer tous les éléments allant de l'indice de début, jusqu'à l'indice de fin (exclus).

```
#Récupérer les colonnes 2 et 3 de toutes les lignes

matrice = np.array([[0,1,2,3], [4,5,6,7], [8,9,10, 11], [12, 13, 14, 15], [16,17,18,19]])
matrice[:,2:4]
print("------------------------------------------------------------------")
```

```
array([[ 2,  3],
       [ 6,  7],
       [10, 11],
       [14, 15],
       [18, 19]])
```

Dans l'exemple ci-dessus, on récupère les colonnes 2 et 3 de toutes les lignes.
Le : à gauche de la virgule, indique que l'on désire récupérer les 5 lignes.
Le 2:4 à droite de la virgule, indique que l'on désire récupérer les colonnes 2 à 4 (exclus) donc les colonnes 2 et 3.
Au final, cela nous donne un ndarray de taille : 5x2 (5 lignes et 2 colonnes)

```
# Récupérer les trois dernières colonnes des trois dernières lignes

matrice = np.array([[0,1,2,3], [4,5,6,7], [8,9,10, 11], [12, 13, 14, 15], [16,17,18,19]])
matrice[2:5, 1:4]
print("----------------------------------------------------------------------")
```

```
array([[ 9, 10, 11],
       [13, 14, 15],
       [17, 18, 19]])
```

Il n'est pas obligé d'indiquer l'index de fin quand on prend toute la fin d'un élément!

```
# Il n'est pas obligé d'indiquer l'index de fin quand on prend toute la fin d'un élément!

matrice = np.array([[0,1,2,3], [4,5,6,7], [8,9,10, 11], [12, 13, 14, 15], [16,17,18,19]])
matrice[2:, 1:]
print("----------------------------------------------------------------------")
```

```
array([[ 9, 10, 11],
       [13, 14, 15],
       [17, 18, 19]])
```

```
# Récupérer la première colonne de la deuxième, troisième et quatrième ligne

matrice = np.array([[0,1,2,3], [4,5,6,7], [8,9,10, 11], [12, 13, 14, 15], [16,17,18,19]])
matrice[1:4, 0]
print("----------------------------------------------------------------------")
```

```
array([ 4,  8, 12])
```

```
# Cela marche aussi avec les vecteurs. Admettons que l'on veut les 4 derniers éléments du vecteur suivant.

vecteur = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14])
vecteur[10:15]
print("----------------------------------------------------------------------")
```

```
array([11, 12, 13, 14])
```

```
# On aurait aussi pu utiliser les index négatifs, ce qui est dans ce cas, plus lisible.

vecteur = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14])
vecteur[-4:]
print("----------------------------------------------------------------------")
```

```
array([11, 12, 13, 14])
```

```
#-------------------------------------
# Le pas
#-------------------------------------
```

On peut indiquer un pas pour prendre un élément tous les x index.
Cela s'écrit : [indice_debut:indice_fin:pas]
Pour la démonstration, on va réutiliser le vecteur précédent et récupérer tous les éléments à des indices pairs.

```
# On peut indiquer un pas pour prendre un élément tous les x index.

vecteur = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14])
vecteur[::2]
print("------------------------------------------------------------------------")
```

```
array([ 1,  3,  5,  7,  9, 11, 13])
```

On récupère bien les éléments à l'indice : 0, 2, 4, 6, 8, 10, 12.
Le premier double-point : indique qu'on veut travailler sur l'entièreté du vecteur.
Le :2 indique qu'on veut récupérer les éléments tous les 2 indices. Comme on commence à l'indice 0, cela récupère tous les éléments aux indices pairs.

```
# Autre exemple : récupérer tous les éléments aux indices impairs

vecteur = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14])
vecteur[1::2]
print("------------------------------------------------------------------------")
```

```
array([ 2,  4,  6,  8, 10, 12, 14])
```

Cette fois-ci, on démarre à l'index 1 (donc au deuxième élément), et on parcourt le ndarray de 2 en 2, récupérant tous les éléments aux indices impairs.
A suivre avec le cours : Preprocessing Data with NumPy : 3.2

```
#-------------------------------------
# Assigner des valeurs
#-------------------------------------

# Pour assigner une valeur à un élément à un index spécifique d'un vecteur, on peut utiliser l'écriture suivante.

vecteur = np.array([1,2,3])
print(vecteur)
vecteur[-1] = 10
print(vecteur)
print("------------------------------------------------------------------------")
```

```
[1 2 3]
[ 1  2 10]
```

Dans l'exemple ci-dessus, nous avons assigné la valeur 10 au dernier élément du vecteur, qui contenait précédemment la valeur 3.

```
# On va prendre un deuxième exemple et changer toutes les valeurs d'un vecteur une par une grâce aux indices.

vecteur = np.array([1,2,3,4,5])
print(vecteur)
vecteur[0] = 5
vecteur[1] = 4
vecteur[2] = 3
vecteur[3] = 2
vecteur[4] = 1
```

```
print(vecteur)
print("------------------------------------------------------------------")
```

```
[1 2 3 4 5]
[5 4 3 2 1]
```

Nous avons changé toutes les valeurs du np.array à 1 dimension qui est passé de [1,2,3,4,5] à [5,4,3,2,1].

```
# Prenons pour exemple une matrice de dimension 4x3 (4 lignes et 3 colonnes), et changeons la valeur de
# l'élément à la deuxième ligne et à la troisième colonne.

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

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

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

La deuxième ligne, c'est la ligne à l'indice 1 et la troisième colonne, c'est la colonne à l'indice 2. Nous avons changé la valeur 6 par la valeur 10.

```
# Prenons un deuxième exemple et changeons toutes les valeurs d'une matrice de dimension 2x2.

matrice = np.array([[4,3], [2,1]])
print(matrice)
print()
matrice[0,0] = 1
matrice[0,1] = 2
matrice[1,0] = 3
matrice[1,1] = 4
print(matrice)
print("------------------------------------------------------------------")
```

```
[[4 3]
 [2 1]]

[[1 2]
 [3 4]]
```

```
# Affecter une même valeur à toute une ligne

matrice = np.array([[10,11,12], [-6,-5,-4], [1,2,3], [100, 108, 402], [121, 323, 454]])
print(matrice)
print()
matrice[0] = 0
print(matrice)
print("------------------------------------------------------------------")
```

```
[[ 10  11  12]
 [ -6  -5  -4]
```

```
[  1   2   3]
[100 108 402]
[121 323 454]]


[[  0   0   0]
 [ -6  -5  -4]
 [  1   2   3]
 [100 108 402]
 [121 323 454]]
```

Toutes les valeurs de la première ligne sont devenues des zéros.

```
# Affecter un vecteur ou une liste à une ligne de matrice

matrice = np.array([[10,11,12], [-6,-5,-4], [1,2,3], [100, 108, 402], [121, 323, 454]])
print(matrice)
print()
matrice[0] = matrice[1]
print(matrice)
print("------------------------------------------------------------------------")
```

```
[[ 10  11  12]
 [ -6  -5  -4]
 [  1   2   3]
 [100 108 402]
 [121 323 454]]

[[ -6  -5  -4]
 [ -6  -5  -4]
 [  1   2   3]
 [100 108 402]
 [121 323 454]]
```

Les valeurs de la première ligne sont égales aux valeurs de la deuxième ligne.

```
# On peut également assigner les valeurs contenues dans une liste à une ligne.

matrice = np.array([[10,11,12], [-6,-5,-4], [1,2,3], [100, 108, 402], [121, 323, 454]])
print(matrice)
print()
matrice[0] = [-3, -2, -1]
print(matrice)
print("------------------------------------------------------------------------")
```

```
[[ 10  11  12]
 [ -6  -5  -4]
 [  1   2   3]
 [100 108 402]
 [121 323 454]]

[[ -3  -2  -1]
 [ -6  -5  -4]
 [  1   2   3]
 [100 108 402]
 [121 323 454]]
```

Ici, nous avons affecté les valeurs de la liste [-3, -2, -1] à la première ligne de notre matrice.

```
# Changer les colonnes d'une matrice
```

```
matrice = np.array([[10,11,12], [-6,-5,-4], [1,2,3], [100, 108, 402], [121, 323, 454]])
print(matrice)
print()
matrice[:,-1] = 0
print(matrice)
print("----------------------------------------------------------------------")
```

```
[[ 10  11  12]
 [ -6  -5  -4]
 [  1   2   3]
 [100 108 402]
 [121 323 454]]

[[ 10  11   0]
 [ -6  -5   0]
 [  1   2   0]
 [100 108   0]
 [121 323   0]]
```

Remarquez bien l'écriture : on a indiqué entre crochets [:,-1], ce qui signifie que l'on veut récupérer toutes les lignes d'une colonne donnée. En l'occurence, la dernière colonne (Rappel : l'index -1 récupère le dernier élément)

```
# Affecter un vecteur ou une liste à une colonne d'une matrice

matrice = np.array([[10,11,12], [-6,-5,-4], [1,2,3], [100, 108, 402], [121, 323, 454]])
print(matrice)
print()
matrice[:,1] = [1,2,3,4,5]
print(matrice)
print("----------------------------------------------------------------------")
```

```
[[ 10  11  12]
 [ -6  -5  -4]
 [  1   2   3]
 [100 108 402]
 [121 323 454]]

[[ 10   1  12]
 [ -6   2  -4]
 [  1   3   3]
 [100   4 402]
 [121   5 454]]
```

```
# On peut également affecter les valeurs d'un vecteur à une colonne.

matrice = np.array([[10,11,12], [-6,-5,-4], [1,2,3], [100, 108, 402], [121, 323, 454]])
print(matrice)
print()
matrice[:,1] = np.array([1,2,3,4,5])
print(matrice)
print("----------------------------------------------------------------------")
```

```
[[ 10  11  12]
 [ -6  -5  -4]
 [  1   2   3]
 [100 108 402]
 [121 323 454]]
```

```
[[ 10   1  12]
 [ -6   2  -4]
 [  1   3   3]
 [100   4 402]
 [121   5 454]]
```

Dans ce cas, il est tout de même plus rapide à écrire et plus lisible d'affecter une liste plutôt qu'un vecteur ndarray.
INFO : la colonne ou la ligne qui se voit affecter une liste reste des ndarray.

```
# On peut le prouver en regardant le type de la ligne ou de la colonne.

matrice = np.array([[10,11,12], [-6,-5,-4], [1,2,3], [100, 108, 402], [121, 323, 454]])
print(matrice)
print()
matrice[:,1] = [1,2,3,4,5]
print(type(matrice[:,1]))
print("------------------------------------------------------------------------")
```

```
[[ 10  11  12]
 [ -6  -5  -4]
 [  1   2   3]
 [100 108 402]
 [121 323 454]]

<class 'numpy.ndarray'>
```

```
# Changer toutes les valeurs d'une matrice

matrice = np.array([[1,2,3], [4,5,6]])
print(matrice)
print()
matrice[:] = 404
print(matrice)
print("------------------------------------------------------------------------")
```

```
[[1 2 3]
 [4 5 6]]

[[404 404 404]
 [404 404 404]]
```

ATTENTION : les crochets avec les double-points [:] sont indispensables pour assigner toutes les valeurs à votre ndarray! Si vous essayez d'affecter une valeur à votre ndarray sans ceux-ci, alors votre variable va devenir un int (nombre entier).

```
matrice = np.array([[1,2,3], [4,5,6]])
print(matrice)
print()
matrice = 404
print(matrice)
print(type(matrice))
print("------------------------------------------------------------------------")
```

```
[[1 2 3]
 [4 5 6]]
```

```
404
<class 'int'>
```

```
# Affecter des valeurs avec le slicing

matrice = np.array([[1,2,3,4,5],
                    [6,7,8,9,10],
                    [11,12,13,14,15],
                    [16,17,18,19,20],
                    [21,22,23,24,25]])
matrice[2:4, 2:4] = 0
print(matrice)
print("---------------------------------------------------------------------")
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12  0  0 15]
 [16 17  0  0 20]
 [21 22 23 24 25]]
```

Comme vous pouvez le voir dans l'exemple ci-dessus, les éléments dont les indices des lignes et des colonnes sont 2 ou 3 se sont vus affecter la valeur 0.
matrice[2:4, 2:4] signifie : les éléments allant de la ligne 2 à la ligne 4 exclue, et les éléments de la colonne 2 à la colonne 4 exclue.
Rappel : les index commencent à zéro. Donc la ligne 2 = troisième ligne.
Prenons un autre exemple avec la même matrice, mais cette fois on veut affecter la valeur 0 aux deux dernières colonnes, première ligne exclue.

```
matrice = np.array([[1,2,3,4,5],
                    [6,7,8,9,10],
                    [11,12,13,14,15],
                    [16,17,18,19,20],
                    [21,22,23,24,25]])
matrice[1:, -2:] = 0
print(matrice)
print("---------------------------------------------------------------------")
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  0  0]
 [11 12 13  0  0]
 [16 17 18  0  0]
 [21 22 23  0  0]]
```

matrice[1:, -2:] = les éléments allant de la ligne 1 (deuxième ligne) jusqu'à la fin, et les éléments allant de l'avant-dernière colonne jusqu'à la fin.
Il est également possible de procéder au slicing et à l'affectation de valeur sur les vecteurs.

```
# Dans l'exemple suivant, on veut affecter la valeur 0 aux éléments allant de l'indice 2 à l'indice 6 (inclus).
vecteur = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])
vecteur[2:7] = 0
print(vecteur)
print("---------------------------------------------------------------------")
```

```
[ 1  2  0  0  0  0  0  8  9 10 11 12 13 14 15]
```

```
#======================
#Les propriétés éléments par éléments, sont les fonctionnalités de Numpy qui permettent de procéder à des
# modifications sur les ndarray élément par élément.
#======================


#-------------------------------------
#L'addition

#-------------------------------------
```

Si on utilise l'opérateur somme (+) entre un ndarray et un entier, cela va ajouter la valeur de l'entier à tous les éléments du ndarray.
Les deux cellules suivantes vous présentent un exemple, tout d'abord avec un vecteur (ndarray de dimension 1) et une matrice (un ndarray de dimension 2).

```
vecteur = np.array([0,1,2])
vecteur + 1
```

```
array([1, 2, 3])
```

```
matrice = np.array([[1,2,3], [3,4,5], [5,6,7]])
matrice + 1
print("------------------------------------------------------------------------")
```

```
array([[2, 3, 4],
       [4, 5, 6],
       [6, 7, 8]])
```

Tous les éléments du vecteur et de la matrice se sont vus incrémentés de 1.
RAPPEL : l'opérateur somme permet de concaténer des listes entre elles, et il permet de faire des manipulations mathématiques sur des ndarray. Ces deux objets servent à répondre à des besoins différents.

```
liste = [1,2,3,4]
liste + [5]
print("------------------------------------------------------------------------")
```

```
[1, 2, 3, 4, 5]
```

L'addition d'une liste et d'un entier ne fonctionne pas. Cela soulève un TypeError indiquant qu'on ne peut pas concaténer un nombre entier et une liste.

```
In [30]:
liste = [1,2,3,4]
liste + 5
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-55d7873d48f4> in <module>
      1 liste = [1,2,3,4]
----> 2 liste + 5
TypeError: can only concatenate list (not "int") to list
```

```
# On peut additionner la ligne d'une matrice et un vecteur si leurs dimensions correspondent.
# Dans l'exemple suivant, on va additionner le vecteur à la première ligne de notre matrice.

vecteur = np.array([0,1,2])
matrice = np.array([[1,2,3], [3,4,5], [5,6,7]])
```

```
matrice[0] + vecteur
print("------------------------------------------------------------------------")
```

array([1, 3, 5])
Cela nous affiche la nouvelle valeur de la première ligne : [1,3,5] = [0+1, 1+2, 2+3] Pour ce type d'addition : on parle d'addition élément par élément (elementwise addition).

```
# On peut additionner le vecteur à toutes les lignes de la matrice.

vecteur = np.array([0,1,2])
matrice = np.array([[1,2,3], [3,4,5], [5,6,7]])
matrice + vecteur
print("------------------------------------------------------------------------")
```

array([[1, 3, 5], [3, 5, 7], [5, 7, 9]])
Toutes les lignes se sont vues incrémentées par les valeurs présentes dans le ndarray vecteur. Cette opération a pu être possible car ils avaient des dimensions correspondantes (le même nombre de colonnes).

```
#---------------------------------------
#La soustraction
#---------------------------------------
```

La soustraction fonctionne comme l'addition à un seul détail, l'ordre dans lequel vous écrivez vos variables pour la soustraction est important.

```
# Prenons un exemple avec deux ndarray de dimension 1.

vecteur_1 = np.array([1,2,3,4])
vecteur_2 = np.array([-1, -2, -3, -4])
print(vecteur_1 - vecteur_2)
print()
print(vecteur_2 - vecteur_1)
print("------------------------------------------------------------------------")
```

[2 4 6 8] [-2 -4 -6 -8]
Comme vous pouvez le voir, le résultat n'est pas le même, faites donc très attention.

```
#---------------------------------------
#La multiplication
#---------------------------------------
```

Tout ce que l'on a vu précédemment pour l'addition marche pour la multiplication.

```
# Dans l'exemple ci-dessous, on va multiplier un vecteur et une matrice par un nombre entier.

vecteur = np.array([0,1,2])
vecteur * 2
print("------------------------------------------------------------------------")
```

array([0, 2, 4])

```
matrice = np.array([[1,2,3], [3,4,5], [5,6,7]])
matrice * 2
print("------------------------------------------------------------------------")
```

```
array([[ 2,  4,  6],
       [ 6,  8, 10],
       [10, 12, 14]])
```

```
# On peut multiplier la ligne d'une matrice avec les valeurs d'un vecteur.

vecteur = np.array([0,1,2])
matrice = np.array([[1,2,3], [3,4,5], [5,6,7]])
matrice[0] * vecteur
print("------------------------------------------------------------------------")
```

```
array([0, 2, 6])
```

C'est une multiplication élément par élément. Chaque élément de la première ligne de la matrice se voit multiplier par l'élément au même indice dans le vecteur. On obtient donc :
[0 * 1, 1 * 2, 2 * 3] = [0, 2, 6]

```
# On peut également multiplier toutes les lignes d'une matrice par un vecteur.

vecteur = np.array([0,1,2])
matrice = np.array([[1,2,3], [3,4,5], [5,6,7]])
matrice * vecteur
print("------------------------------------------------------------------------")
```

```
array([[ 0,  2,  6],
       [ 0,  4, 10],
       [ 0,  6, 14]])
```

Pour chaque ligne de la matrice, le premier élément a été multiplié par 0, le deuxième élément par 1, le troisième élément par 2. Soit, les éléments de la variable vecteur [0,1,2].

```
#--------------------------------------
#La division
#--------------------------------------
```

La division fonctionne comme la multiplication, mais comme pour la soustration, l'ordre est important. Prenons un exemple avec deux vecteurs.

```
vecteur_1 = np.array([2,4,6,8])
vecteur_2 = np.array([2,2,2,2])
print(vecteur_1 / vecteur_2)
print()
print(vecteur_2 / vecteur_1)
print("------------------------------------------------------------------------")
```

```
[1. 2. 3. 4.]

[1.         0.5        0.33333333 0.25      ]
```

Comme vous pouvez le constatez, les résultats diffèrent selon l'ordre dans lequel les variables sont positionnées dans l'opération.