Tableau Numpy

In [1]:
```
import numpy as np
#Lien vers la doc numpy : https://numpy.org/doc/stable/contents.html
from PIL import Image
from math import *
```

Créations de matrices

In [2]:
```
# Créer un tableau numpy 2-D: np.array([[]])

identité_2 = np.array([[1,0],[0,1]])

print(type(identité_2))
print(identité_2)
```

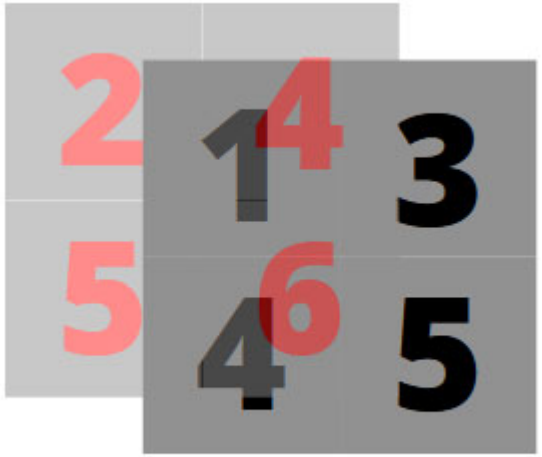
```
<class 'numpy.ndarray'>
[[1 0]
 [0 1]]
```

In [3]:
```
# Tableau 3-D :

three_dim = np.array([ [[1,2],[3,4]] , [[4,5],[5,6]] ])
print(three_dim)
print(three_dim.ndim)
```

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

 [[4 5]
  [5 6]]]
3
```

In [5]:
```
Image.open("./3D.png")
```

Out[5]:



In [6]:
```
# Créer la matrice identité : np.eye(nb)

identité_2_np = np.eye(2 , dtype = int)

print(type(identité_2_np))
print(identité_2_np)
```

```
<class 'numpy.ndarray'>
[[1 0]
 [0 1]]
```

In [8]:
```
# Créer la matrice nulle : np.zeros((nb lignes, nb colonnes))

zeros_2_np = np.zeros((2,2), dtype = int)

print(type(zeros_2_np))
print(zeros_2_np)
```

```
<class 'numpy.ndarray'>
[[0 0]
 [0 0]]
```

In [9]:
```
# Créer la matrice avec que des 1 : np.ones((nb lignes, nb colonnes))

ones_2_np = np.ones((2,2))

print(type(ones_2_np))
print(ones_2_np)
```

```
<class 'numpy.ndarray'>
[[1. 1.]
 [1. 1.]]
```

In [12]:
```
# Créer des tableaux 1-D
# 1ére solution : np.linspace(start, end, nb_elements), le end est pris en compte ici
# 2éme solution : np.arange(start, end, pas), le end est exclu ici
A = np.linspace(5,15,9)
B = np.arange(5,16.25,1.25)
print(A)
print(B)
```

```
[ 5.    6.25  7.5   8.75 10.   11.25 12.5  13.75 15.  ]
[ 5.    6.25  7.5   8.75 10.   11.25 12.5  13.75 15.  ]
```

In [17]:
```
# Créer des tableaux 2-D à partir de linspace ou arange
# On utilise la fonction reshape
#
A = np.linspace(5,15,9)
B = np.arange(5,17.5,1.25)
print("taille A : ", A.shape)
print("taille B : ", B.shape)

new_A = A.reshape((5,2))
new_B = B.reshape((2,5))
print("new_A taille: ",new_A.shape)
print("new_B taille: ",new_B.shape)
```

```
taille A :  (9,)
taille B :  (10,)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-d1b85fda476d> in <module>
      7 print("taille B : ", B.shape)
      8
----> 9 new_A = A.reshape((5,2))
     10 new_B = B.reshape((2,5))
     11 print("new_A taille: ",new_A.shape)

ValueError: cannot reshape array of size 9 into shape (5,2)
```

In [18]:
```
new_A_flat = new_A.flatten() #Décrémente de 1 la dimension, on passe de 2-D à 1-D ici
print(new_A_flat.shape)
```

```
(10,)
```

Quelques opérations élémentaires

In [20]:
```
# Addition, soustraction, multiplication

A = np.array([[1,2],[3,4]])
B = np.eye(2)

somme = A + B
diff = A - B
ele_par_ele = A * B #elles doivent avoir la même taille ! multiplication élément par élément
ele_par_ele_2 = np.multiply(A,B) #elles doivent avoir la même taille ! multiplication élément par é
lément
produit = np.dot(A,B) # vrai produit de matrice

print(A)
print(B)
print("somme :", somme)
print("diff :", diff)
print("multiplication élément par élément :", ele_par_ele)
print("multiplication élément par élément 2 :", ele_par_ele_2)
print((ele_par_ele == ele_par_ele_2))
print("produit : ", produit)
```

```
[[1 2]
 [3 4]]
[[1. 0.]
 [0. 1.]]
somme : [[2. 2.]
 [3. 5.]]
diff : [[0. 2.]
 [3. 3.]]
multiplication élément par élément : [[1. 0.]
 [0. 4.]]
multiplication élément par élément 2 : [[1. 0.]
 [0. 4.]]
[[ True  True]
 [ True  True]]
produit :  [[1. 2.]
 [3. 4.]]
```

In [21]:
```
# Taille d'une matrice : array.shape
# np.empty((nb lignes, nb cols), dtype = ...) = retourne un tableau d'une taille et d'un type donné
e,
# sans que l'on ait à initialiser le tableau.
# on peut utiliser len pour avoir le nb de lignes

A = np.empty((4, 19))

taille_A = A.shape
nb_lignes = A.shape[0]
nb_lignes_len = len(A)
nb_cols = A.shape[1]
A_transpose = np.transpose(A)

print("taille A : ", taille_A)
print("nb_lignes : ", nb_lignes)
print("nb_lignes_len : ", nb_lignes_len)
print("nb_cols : ", nb_cols)
print("taille A transpose : ",A_transpose.shape)
```

```
taille A :  (4, 19)
nb_lignes :  4
nb_lignes_len :  4
nb_cols :  19
taille A transpose :  (19, 4)
```

In [22]:
```
# On peut récupérer le nombre de dimensions de notre tableau

A = np.empty((4, 19))
nb_dims = A.ndim
print(nb_dims)
```

```
2
```

Extraire des valeurs d'un tableau numpy

In [25]:
```
# syntaxe : array[i,j]
# Indiçage des colonnes et des lignes commence toujours à 0 !!

np.random.seed(1)
A = np.random.randn(3,3)
print(A)

pos_1_2 = A[0,1]
print("Ligne 1 colonne 2 : ", pos_0_1)

pos_3_3 = A[2,2]
print("Ligne 3 colonne 3 : ", pos_2_2)
```

```
[[ 1.62434536 -0.61175641 -0.52817175]
 [-1.07296862  0.86540763 -2.3015387 ]
 [ 1.74481176 -0.7612069   0.3190391 ]]
Ligne 1 colonne 2 :  -0.6117564136500754
Ligne 3 colonne 3 :  0.31903909605709857
```

In [26]:
```
# On peut extraire des colonnes ou des lignes
# syntaxe :
# récupération des lignes de A de l'indice start à end : A[start:end, :]
# récupération des colonnes de A de l'indice start à end : A[:, start:end]
# on peut récupérer des morceaux aussi : A[startl:endl, startc:colc]

ligne_1_A = A[0, :]
colonne_2et3_A = A[:, 1:3] #3 est exlu !
A_l12_col23 = A[0:2,1:3]

print(A)
print("la première ligne de A", ligne_1_A)
print("les colonnes 2 et 3 de A", colonne_2et3_A)
print("la partie supérieur droite de A", A_l12_col23)
```

```
[[ 1.62434536 -0.61175641 -0.52817175]
 [-1.07296862  0.86540763 -2.3015387 ]
 [ 1.74481176 -0.7612069   0.3190391 ]]
la première ligne de A [ 1.62434536 -0.61175641 -0.52817175]
les colonnes 2 et 3 de A [[-0.61175641 -0.52817175]
 [ 0.86540763 -2.3015387 ]
 [-0.7612069   0.3190391 ]]
la partie supérieur droite de A [[-0.61175641 -0.52817175]
 [ 0.86540763 -2.3015387 ]]
```

Maximum et minimum

In [27]:
```
A = np.array([[1,2,3],[4,5,6]])
print(np.max(A))
```

```
6
```

In [28]:
```
minimum = np.min(B)
print(minimum)
```

```
0.0
```

Somme

In [29]:
```
# syntaxe : array.sum(A, axis = )
# si le tableau est 2-D alors axis = 0 (colonne) ou 1 (ligne)
A = np.array([[1,2,3],[4,5,6]])

somme = np.sum(A)
somme_ligne = np.sum(A, axis = 1)
somme_col = np.sum(A, 0)

print(A)
print("somme tous les élé de A : ", somme)
print("somme chaque ligne: ",somme_ligne)
print("somme chaque colonne: ",somme_col)
```

```
[[1 2 3]
 [4 5 6]]
somme tous les élé de A :  21
somme chaque ligne:  [ 6 15]
somme chaque colonne:  [5 7 9]
```

Moyenne

In [30]:
```
# syntaxe : array.mean(A, axis)

A = np.array([[1,2,3],[4,5,6]])
moy = np.mean(A)
moy_ligne = np.mean(A, axis = 1)
moy_col = np.mean(A, 0)

print(A)
print("moyenne tous les élé de A : ", moy)
print("moyenne chaque ligne: ",moy_ligne)
print("moyenne chaque colonne: ",moy_col)
```

```
[[1 2 3]
 [4 5 6]]
moyenne tous les élé de A :  3.5
moyenne chaque ligne:  [2. 5.]
moyenne chaque colonne:  [2.5 3.5 4.5]
```

In [34]:
```
# Petite parenthèse sur les float et int
# Plus le nb de bits augmente, plus la précision aussi
# Lien : https://numpy.org/doc/stable/user/basics.types.html
np.float64([pi])
```

Out[34]:
```
array([3.14159265])
```

In [31]:
```
print(type(moy_ligne[0]))
```

```
<class 'numpy.float64'>
```

Itérer avec un tableau numpy

In [35]:
```
# Simple boucle for
A = np.array([[1,2,3],[4,5,6]])
count_ele = 0
for i in A:
    count_ele += 1
    print(count_ele)
    print(i)
```

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

In [36]:
```
# On peut récupérer en même temps l'indice et l'élément avec
# la commande np.ndenumerate(A)

A = np.array([[1,2,3],[4,5,6]])
for indice, ele in np.ndenumerate(A):
    print(indice, ele)
print("matrice A : ")
print(A)
```

```
(0, 0) 1
(0, 1) 2
(0, 2) 3
(1, 0) 4
(1, 1) 5
(1, 2) 6
matrice A :
[[1 2 3]
 [4 5 6]]
```