Nous avons vu dans le TP sur les listes que nous pouvions définir une matrice mais les manipulations sur les listes ne permettent pas facilement de faire les opérations habituelles sur les matrices (somme, produit, inverse...). Pour définir des matrices, ou tableaux, nous importerons la librairie **numpy** et utiliserons **numpy.array()**.

Pour des questions pratiques, nous importons la librairie **numpy** classiquement avec **import numpy as np**.

Pour définir un tableau d'une ligne contenant les valeurs (1, 2, 3, 4), par exemple

```
import numpy as np
A = np. array ([1 ,2 ,3 ,4])
```

Pour une matrice de deux lignes par exemple :

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

et à trois lignes :

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

À retenir

Pour créer une matrice, on importe la librairie **numpy** et on utilise la commande **numpy.array**

Il s'agit d'un autre type d'objet que la liste.

Exercice 1. Directement dans la console

```
import numpy as np
A = np. array ([1 ,2 ,3 ,4])
type(A)
A=[1,2,3,4]
type(A)
```

Exercice 2. Directement dans la console

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

À retenir

On peut faire appel à un coefficient de la matrice, une ligne ou une colonne.

Exercice 3. Directement dans la console

```
B = np.array([[1, 2, 3], [4, 5, 6]])
np.size(B)
np.shape(B)
C = np.array([1, 2, 3])
np.size(C)
np.shape(C)
```

À retenir

La fonction **numpy.size()** renvoie le nombre d'éléments du tableau et la fonction **numpy.shape()** renvoie la dimension du tableau.

Exercice 4. Directement dans la console

```
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([[2, 1, 3], [3, 2, 1]])
print(A*B)
```

À retenir

Il est possible de réaliser un produit terme à terme grâce à l'opérateur *****. Il faut dans ce cas que les deux tableaux aient la même taille.

Exercice 5. Directement dans la console

```
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([[2, 1, 3], [3, 2, 1],[1,0,1]])
print(np.dot(A,B))
```

À retenir

La fonction **numpy.dot()** permet de réaliser le produit matriciel.

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

Exercice 6. Directement dans la console

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

À retenir

La fonction **numpy.transpose()** permet de transposer la matrice.

Exercice 7. Directement dans la console

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

Exercice 8. Directement dans la console

```
A = np.array([[1, 2, 3],[4, 5, 6]])
print(A)
print(A[0,1])
print(A[:,1:3])
print(A[:,1])
print(A[0,:])
```

À retenir

On peut extraire une partie d'un tableau, on indique entre crochets des indices pour définir le début et la fin de la tranche et à les séparer par deux-points. Dans la tranche [n :m], l'élément d'indice n est inclus, mais pas celui d'indice m. A[:,n] donne un tableau 1D correspondant à la colonne d'indice n de A. Si on veut obtenir un tableau 2D correspondant à la colonne d'indice n, il faut utiliser A[:,n :n+1].

Exercice 9. Directement dans la console

```
A = np.arange(1,16)
np.sum(A)
```

À retenir

⋆ **np.arange()** permet de créer une progression arithmétique

⋆ **np.sum()** calcule les sommes des termes d'un tableau

Exercice 10. Directement dans la console

```
np.zeros(3)
np.zeros((2,3)
np.ones(3)
np.ones((2,3))
np.eye(3)
```

À retenir

np.zeros(n) renvoie un tableau 1D de n zéros. **np.zeros((n,p))** renvoie tableau 2D de taille m x n, c'est-à-dire de shape (m,n). **np.eye(n)** renvoie tableau 2D carré de taille n x n, avec des uns sur la diagonale et des zéros partout ailleurs.

Exercice 11. Directement dans la console

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

np.det(A)
```

À retenir

np.det() revoie le déterminant d'un matrice

Exercice 12. Directement dans la console

```
A = np.array([[1, 3, 3],
              [1, 4, 3],
              [1, 3, 4]])
np.inv(A)
np.diag(A)
```

À retenir

np.inv() renvoi l'inverse d'un matrice carrée.
np.diag() permet d'afficher les éléments diagonaux d'un tableaux

Exercice 13. Directement dans la console

```
A = np.array([[3,1], [1,2]])

B = np.array([9,8])

X = np.linalg.solve(A, B)
print(X)
```

À retenir

On peut résoudre un système dont la forme matricielle est $AX = B$ avec **np.linalg.solve(A,B)**

Exercice 14. Directement dans la console

```
U = np.arange(1, 16)
np.shape(U)
U.shape = (3, 5)
np.shape(U)
A = np.arange(1, 6)
A.shape = (1, np.size(A))
A.shape = (np.size(A), 1)
```

À retenir

La commande **shape** permet de redimensionner un tableau

Exercice 15. On considère la matrice $A \in \mathcal{A}_{3,4}(\mathrm{R})$ définie par :

$$\begin{pmatrix} 4 & 6 & -2 & 3 \\ 2 & -1 & 0 & 1 \\ -7 & 0 & 1 & 12 \end{pmatrix}$$

1. Définir la matrice A comme un **np.array ()** .
2. Modifier la matrice A pour que ses deux premières lignes soient multipliées par 2 et que sa dernière colonne soit divisée par 3.
3. Créer une nouvelle matrice B définie par

$$\begin{pmatrix} 4 & 5 & 6 \\ 5 & 10 & 15 \\ 1 & 1 & 1 \end{pmatrix}$$

   en utilisant le fait que les lignes 1 et 2 sont composées des éléments successifs de deux suites arithmétiques et en utilisant np.arange et np.ones().
4. Réaliser le produit matriciel $D = B \times A$ avec **np.dot()**.
5. Que donne A*B ?
6. Calculer la somme des éléments de la matrice A
7. Calculer la somme des éléments la ligne 1 de la matrice A

Dans la librairie **numpy** on peut retrouver des fonctions d'algèbre linéaire qui sont appelées classiquement en important **import np.linalg as al**.

Exercice 16. Soit $A \in \mathcal{M}_3(\mathbb{R})$ et $B \in \mathcal{M}_{3,1}(\mathbb{R})$.

$$A = \begin{pmatrix} 1 & -1 & 2 \\ -1 & 2 & 3 \\ 0 & -1 & 1 \end{pmatrix}, \text{ et } B = \begin{pmatrix} 3 \\ -7 \\ 1 \end{pmatrix}$$

Résoudre à l'aide de la fonction **al.solve()** le système $AX = B$.

Exercice 17. Résoudre dans $\mathbb{R}^3$, les systèmes suivants : $\begin{cases} -x + 2y - z = -5 \\ x - 4y + 2z = 7 \\ -2x - 2y + 2z = 0 \end{cases}$ $\begin{cases} 3x + y - z = 0 \\ x + y - z = 0 \\ -x - y + z = 0 \end{cases}$

Exercice 18. On considère les matrices suivantes :

$$A = \begin{pmatrix} 5 & 1 & 2 \\ -1 & 7 & 2 \\ 1 & 1 & 6 \end{pmatrix}, P = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \end{pmatrix}, Q = \begin{pmatrix} \frac{1}{2} & 0 & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}, D = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 8 \end{pmatrix}$$

1. Calculer PQ, à l'aide de la commande **np.dot**. Que peut-on en déduire ?

2. Vérifier que $A = PDQ$.
3. Calculer A^4 et PD^4Q à l'aide de la commande **al.matrix_power()**.

Exercice 19. On considère la matrice A définie par :

$$A = \begin{pmatrix} 1 & 3 & 2 \\ 1 & 2 & 2 \\ 1 & 3 & 1 \end{pmatrix}$$

1. Calculer $A^3 - 4A^2 - 6A$. En déduire une expression de A^{-1} en fonction de A et de I_3.
2. Calculer l'inverse de A avec la commande **al.inv()** puis vérifier que l'expression correspond avec 1.

À retenir

al.matrix_power() pour calculer la puissance d'une matrice