MODULES PYTHON

I- INTRODUCTION

En python il existe de nombreuses bibliothèques de programmes, appelées **modules**.

Pour pouvoir utiliser un module il faut l'importer par la commande **`import`** :

```
import nom_module
```

Si le nom du module est un peu long il est préférable de créer un **`alias`**.

```
import nom_module as alias
```

Une fonction présente dans un module peut être appelée sous la forme suivante.

```
alias.nom_fonction
```

Exemple :

```
>>> import random
>>> random.randint(1,100)
56
```

Le module **random** peut être remplacé par un alias :

```
>>> import random as rd
>>> rd.randint(1,100)
80
```

Il est possible d'importer une ou plusieurs fonctions d'une bibliothèque et de les utiliser sans mettre comme préfixe ni le nom ni l'alias de la bibliothèque.

```
>>> from random import randint
>>> randint(1,100)
35
```

Pour importer toutes les fonctions du module **random** :

```
>>> from random import *
```

Exemples de modules en python :

`time`	Diverses fonctions relatives au temps
`random`	Pour générer aléatoirement des nombres
`os`	Pour dialoguer avec le système d'exploitation
`Tkinter`	Pour créer de fenêtres graphiques et gérer la souris
`math`	Pour accéder aux fonctions mathématiques usuelles
`matlibplot`	Pour de multiples de fonctions de tracé de courbes en deux dimensions
`numpy`	Bibliothèque de fonctions pour faire des calculs scientifiques efficaces
`scipy`	Bibliothèque complémentaire de **`numpy`** pour le traitement de données
`skimage`	Pour le traitement d'images
`io`	Pour lire et créer des fichiers

II- Bibliothèque numpy

Le module **numpy** permet de créer de vrais tableaux et de faire du calcul matriciel. Un tableau **numpy** aura une dimension prédéfinie et tous les éléments qui le composent seront du même type.

Dans toute la suite on supposera qu'on a effectué :

```
>>>import numpy as np
```

A- Fonctions utiles de la bibliothèque numpy

1- Fonction array()

La fonction **array** prend en argument une liste et renvoie un tableau **numpy** ayant les mêmes éléments. En cas de liste de listes, l'opérateur s'applique récursivement à chaque sous liste : la valeur de retour est donc un tableau de tableaux.

```
>>> np.array([1, 2, 3])         # Un tableau d'entiers
array([1, 2, 3])
>>> np.array([1, 2, 3.0])       # On impose implicitement les flottants
array([ 1., 2., 3.])
>>> np.array([[1, 2], [3, 4]])  # Plus d'une dimensions
array([[1, 2],
       [3, 4]])
```

2- Fonction arange()

La fonction **arange** a le même comportement que range, mais renvoie un **array**.

```
>>> np.arange(1,6,2)
array([1, 3, 5])
>>> a = np.arange(15) # Tableau 1D à 15 éléments
>>> a
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
```

3- Fonction linspace()

La fonction **linspace crée un tableau de valeurs régulièrement espacées entre des nombres.**

```
>>>x=np.linspace(1,5,20)  #construire un tableau de 20 nombres régulièrement
                           espacés entre 1 et 5
>>> x
array([ 1. , 1.21052632, 1.42105263, 1.63157895, 1.84210526,
2.05263158, 2.26315789, 2.47368421, 2.68421053, 2.89473684,
3.10526316, 3.31578947, 3.52631579, 3.73684211, 3.94736842,
4.15789474, 4.36842105, 4.57894737, 4.78947368, 5. ])
>>> len(x)
20
```

4- **Fonction reshape()**

La fonction **reshape** retourne un tableau 2D à partir d'un tableau 1D.

```
>>> a = np.arange(15) # Tableau 1D à 55 éléments
>>> a.reshape(5,3)    # Changé en tableau 2D à 5*3 éléments
array([[ 0, 1, 2],
       [ 3, 4, 5],
       [ 6, 7, 8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

5- **Fonction matrix()**

La fonction **matrix** permet construire une matrice de faire du calcul matriciel

```
>>> a = np.arange(9)
>>> a
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8])

>>> A = a.reshape(3,3)
>>> A
array([[ 0, 1, 2],
       [ 3, 4, 5],
       [ 6, 7, 8]])

>>> A = np.matrix(A)
>>> A     # La matrice A
matrix ([[ 0, 1, 2],
       [ 3, 4, 5],
       [ 6, 7, 8]])
```

Par la suite on peut faire des calculs sur la matrice A.

```
>>> A**2    # Son carré matriciel,
matrix([[ 15, 18, 21],
             [ 42, 54, 66],
             [ 69, 90, 111]])
>>> A.transpose()    # Sa transposée
matrix([[0, 3, 6],
        [1, 4, 7],
        [2, 5, 8]])
>>> A*A.transpose()   # Le produit matriciel avec la transposée
matrix([[ 5, 14, 23],
        [ 14, 50, 86],
        [ 23, 86, 149]])
```

Remarque :

Pour une matrice **numpy m**, il est possible d'accéder directement à l'élément de ligne **i** et de colonne **j** avec la syntaxe **m[i,j]** (alors qu'avec une matrice python "classique" il faudrait écrire **m[i][j]**).

Exemple :

```
>>> a = [[2,3],[1,8]]           # a est une liste de listes
>>> b = np.array(a)            # b est un tableau 2D
>>> b[1]                                  # Ligne d'indice 1
array([1, 8])
>>> b[:,0]                                # Colonne d'indice 0
array([2, 1])
>>> b[:,-1]                               # Dernière colonne
array([3, 8])
>>> b[1,0]                               # Elément ligne 1 colonne 0
1
```

6- **Fonction zeros()**

- **ones(n)** : crée un **tableau** de longueur **n** contenant uniquement des **0**
- **ones((p,q))** : crée une **matrice numpy** avec **p** lignes et **q** colonnes composée uniquement de **0**.

7- **Fonction ones()**

- **ones(n)** : crée un **tableau** de longueur **n** contenant uniquement des **1**
- **ones((p,q))** : crée une **matrice numpy** avec **p** lignes et **q** colonnes composée uniquement de **1**.

8- **Fonction eye()**

- **eye(n)** : renvoie la matrice identité de taille n

Exemples

```
>>> np.zeros(2)
array([ 0., 0.])
>>> np.zeros((2,3))
array([[ 0., 0., 0.],
       [ 0., 0., 0.]])
>>> np.ones(2)
array([ 1., 1.])
>>> np.ones((2,3))
array([[ 1., 1., 1.],
       [ 1., 1., 1.]])
>>> np.eye(2)
array([[ 1., 0.],
       [ 0., 1.]])
```

B- **Opérations sur les tableaux numpy**

1- **addition, soustraction, multiplication, division.**

Les opérations usuelles (addition, soustraction, multiplication, division) s'appliquent aux tableaux Numpy en opérant coefficient par coefficient.

```
>>> a=np.array([[1,4],[1,2]])
>>> b=np.ones((2,2))
>>> a*b
array([[ 1., 4.],
       [ 1., 2.]])
>>> a/b                    # Au sens matriciel usuel b n'est pas inversible
array([[ 1., 4.],
       [ 1., 2.]])
>>> a**2
array([[ 1, 16],
       [ 1, 4]])
```

Remarque :

Chaque fonction mathématique usuelle possède une fonction **numpy** qui lui est homonyme et qui calcule la même chose sur les objets de type **float**. Par exemple il existe **`math.sin`** et **`np.sin`** L'avantage des fonctions **`numpy f`**, c'est que si **`a`** est un tableau numpy **`a = array[a0, . . . , an-1]`**, l'appel **`f(a)`** renvoie le tableau **`array[f(a0), . . . , f(an-1)]`**. Ce mécanisme s'applique aussi aux matrices.

Exemple :

```
>>> import numpy as np
>>> x=np.array([3,4])
>>> np.exp(x)
array([ 20.08553692, 54.59815003])
>>> import math
>>> math.exp(x)
TypeError: only length-1 arrays can be converted to Python scalars
```

2- **Produit matriciel.**

La fonction **`dot(A,B)`** retourne le produit matriciel A.B

La fonction **`dot`** fait est définie dans le sous module **`linalg`** du module **`numpy`**.

Exemple :

```
>>> from numpy.linalg import dot
>>> A=[[1,2],[3,4]]
>>> B= np.eye(2)
>>> dot (A,B)              # Syntaxe correcte du produit matriciel
array([[ 1., 2.],
       [ 3., 4.]])
>>> A*B                    # Faux (produit terme a terme)
array([[ 1., 0.],
       [ 0., 4.]])
```

3- **Produit scalaire.**

La fonction v**dot(A,B)** retourne le produit scalaire de deus vecteurs A et B

```
>>> from numpy import vdot
>>> A=[1,2,3]
>>> B=[4,5,6]
>>> vdot(A,B)
32
```

1- **Résolution d'un système de Cramer :**

La fonction **solve(A,B)** retourne le vecteur **X** solution de **A.X = B**

Exemple : Résolution du système : $\begin{cases} x+2y=2 \\ 3x+4y=8 \end{cases}$

```
>>> from numpy.linalg import solve
>>> solve ([[1,2],[3,4]], [2,8])
array([ 4., -1.])
```

2- **Calculer le déterminant :**

La fonction **det(A)** retourne le déterminant de A

```
>>> from numpy.linalg import
>>> det ([[1,2],[3,4]])
-2.0000000000000004
```

3- **Calculer la trace d'une matrice :**

La fonction **trace(A)** retourne la trace de la matrice A

```
>>> from numpy import trace
>>> trace([[1,2],[3,4]])
5
```

4- **Calculer l'inverse d'une matrice :**

- La fonction **inv(A)** retourne l'inverse de A (A^{-1})

```
>>> from numpy.linalg import inv
>>> inv ([[1,2],[3,4]])
array([[-2. , 1. ],
       [ 1.5, -0.5]])
```

5- **Calculer la puissance d'une matrice :**

- La fonction **matrix_power(A,n)** retourne la matrice **A** à la puissance **n** (A^n).

```
>>> from numpy.linalg import matrix_power
>>> matrix_power ([[1,2],[3,4]], -2)
array([[ 5.5 , -2.5 ],
       [-3.75, 1.75]])
```

6- **Transposée d'une matrice :**

- La fonction **transpose (A)** : retourne la matrice transposée de A (tA)

```
>>> from numpy import transpose
>>> transpose([[1,2],[3,4]])
array([[1, 3],
       [2, 4]])
```

7- **Autres fonctions. :**

- **rad2deg(), deg2rad() :** conversion de radians en dégrées et vice versa. Notez bien, que partout en Python les angles sont donnés en radians ;

```
>>>import numpy as np
>>>np.deg2rad(180)
   3.1415926535897931
```

- **abs() : la valeur absolue ;**

```
>>>np.abs(-180)
180
```

- **cos(), sin(), tan() : les fonctions trigonométriques de base ;**
- **arccos(), arcsin(), arctan() : les fonctions inverses de cos, sin et tan ;**
- **ceil(), floor(), round() : arrondir une valeur ;**

```
>>>np.ceil(3.01)            # arrondi au plus petit entier supérieur
>>> np.floor(3.99)          # arrondi au plus grand entier inférieur
>>> np.round(3.49)          # arrondi au plus proche
```

- **sort() : Trier les éléments;**

```
>>>import numpy as np
>>> x = np.array([10, 2, 72, 35, 4, 123, 8])
>>> x.sort()
```

- **sum() : Somme des éléments de x ;**

```
>>> x.sum()
```

- **var() : La variance des éléments ;**

```
>>> x.var()
```

- **std() : L'écart-type des éléments ;**

```
>>> x.std()
```

- **max() : La valeur maximale ;**

```
>>> x. max()
```

- **min() : La valeur minimale ;**

```
>>> x. min()
```

- **mean() : La moyenne ;**

```
>>>x.mean()
```

III- BIBLIOTHEQUE matplotlib

La bibliothèque **matplotlib** fournit des outils pour tracer des courbes et afficher des images.

Fonctions du sous-module **pyplot** de **matplotlib**

Plot	Pour tracer une courbe à partir d'un tableau de valeurs
Show	Pour ouvrir un tableau et afficher l'image créée
Axis	Pour préciser la nature des axes
Xlabel,ylabel	Pour donner un nom aux axes
Grid	Pour ajouter une grille
Legend	Pour ajouter une légende
Savefig	Pour enregistrer la figure dans un fichier image

Voici une illustration des possibilités des fonctions du sous-module **pyplot** de **matplotlib**

```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(-np.pi,np.pi,256)
f=np.cos(x)
g=np.sin(x)
plt.plot(x,f,'b') #tracé du cosinus
plt.plot(x,g,'g') #tracé du sinus

plt.grid(True) #Affichage d'une grille
plt.legend(('cos','sin'),'upper right', shadow=True) #Affichage de la légende
plt.xlabel('axe des x') #Etiquette sur l'axe x
plt.ylabel('axe des y') #Etiquette sur l'axe y

plt.text(-3,0.5,r'$f(x)=cos(x)$',
horizontalalignment='left',fontsize=18,color='b')
plt.text(2.5, 0.5,r"$g(x)=sin(x)$",
horizontalalignment='left',fontsize=18,color='g')

plt.title('Fonctions sinus et cosinus') #titre
plt.savefig('ExempleTrace') #sauvegarder l'image
plt.show() #Affichage de courbes
```