Les tableaux numpy

Dans le cadre du programme du lycée et par souci de simplification, nous ne créeront que des tableaux numpy à une dimension (une ligne).

Ces tableaux ressemblent grandement à des listes mais permettent de faire des opérations dites vectorisées bien plus intuitives et d'éviter les boucles. Comme pour une liste, un tableau numpy est un objet mutable qui permet de lister différents éléments, par contre ces derniers seront obligatoirement tous du même type (entiers, décimaux, chaînes de caractères, listes, tuples…) et la longueur du tableau est non-modifiable.

ATTENTION : comme pour une liste, l'indice du premier élément d'un tableau numpy a pour valeur 0 !

Remarque sur l'organisation de ce fichier noteboook : très souvent, les cellules fonctionnent par paire; elles portent alors le même titre. La première cellule explique le cours et la suivante est une cellule de codes illustrant le cours.

Créer des tableaux numpy à une dimension

- *T = np.array([5,2,8,17,6,14])*
  - Convertit une liste contenant les éléments 5, 2, 8, 17, 6, 14 en un tableau numpy contenant les mêmes éléments et dans le même ordre.
- *T1 = numpy.arange(15)*
  - Crée un tableau contenant 15 valeurs entières allant de 0 à 14.
- *T2 = numpy.arange(0.9,8.1,0.5)*
  - Crée un tableau contenant des valeurs séparées de 0.5 comprises dans l'intervalle [0.9;8.1].
  - Le premier paramètre précise la valeur initiale du tableau.
  - Les valeurs du tableau seront comprises dans l'intervalle [premier paramètre, second paramètre].
  - Le dernier paramètre indique l'intervalle entre deux valeurs successives du tableau.
- *T3 = numpy.linspace(0,1/4,16)*
  - Crée un tableau contenant 16 valeurs (de 0 à 1/4=0,25).
  - Le premier paramètre précise la valeur initiale du tableau.
  - Le second paramètre précise la valeur finale du tableau.
  - Le dernier paramètre indique le nombre total de valeurs.

In [1]:

```
# Plusieurs manières de créer des tableaux numpy à une dimension
import numpy as np

T = np.array([5,2,8,17,6,14])
print(T)
T1 = np.arange(15)
print(T1)
T2 = np.arange(0.9,8.1,0.5)
print(T2)
T3 = np.linspace(0,1/4,16)
print(T3)
```

```
[ 5  2  8 17  6 14]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
[0.9 1.4 1.9 2.4 2.9 3.4 3.9 4.4 4.9 5.4 5.9 6.4 6.9 7.4 7.9]
[0.         0.01666667 0.03333333 0.05       0.06666667 0.08333333
 0.1        0.11666667 0.13333333 0.15       0.16666667 0.18333333
 0.2        0.21666667 0.23333333 0.25      ]
```

Récupérer les éléments d'un tableau numpy

Nous allons travailler sur un exemple :

- T[0] renvoie le premier élément, ici 5
- T[2] renvoie l'élément d'indice 2 (en 3ème position donc), ici 8
- T[-1] renvoie le dernier élément, ici 14
- T[-2] renvoie l'avant-dernier élément, ici 6
- T[1:3] renvoie les éléments d'indice 1 et 2 (ATTENTION : indice 3 non inclus)
- T[3:] renvoie les éléments à partir de l'indice 3.
- len(T) renvoie la longueur du tableau, ici 6

- T4 = np.array([]) crée un tableau vide

In [2]:

```
# Plusieurs manières de récupérer les éléments d'un tableau numpy :
# applications

T=np.array([5,2,8,17,6,14])
print(T)
print(T[0])
print(T[2])
print(T[-1])
print(T[-2])
print(T[1:3])
print(T[3:])
print(len(T))
T4 = np.array([])
print(T4)
```

```
[ 5  2  8 17  6 14]
5
8
14
6
[2 8]
[17  6 14]
6
[]
```

Ajouter une valeur ou supprimer une valeur d'un tableau numpy

Les fonctions ne modifient pas les tableaux d'origine mais retournent de nouveaux tableaux en ajoutant ou en supprimant une valeur du tableau d'origine. Leur syntaxe est toujours similaire : np.fonction(T)

In [3]:

```
# PLusieurs fonctions pour ajouter une valeur ou supprimer des valeurs
# d'un tableau numpy (cours et applications)

T=np.array([5,2,8,17,6,14])

print(T)
T1=np.append(T,18)      # Crée un tableau T1 à partir du tableau T
print(T1)               # en ajoutant lélément 18 à la fin du
print(T)                # tableau T. Ne modifie pas le tableau T.

T=np.append(T,18)       # Ajoute lélément 18 à la fin du tableau T.
print(T)

T=np.insert(T,2,20)     # Insère lélément 20 à la position d'indice 2
print(T)                # (donc en troisième position) du tableau T.

T=np.delete(T,2)        # Supprime l'élément d'indice 2 (ici lélément 20)
print(T)                # du tableau T.
T=np.delete(T,-1)       # Supprime lélément dindice -1 (donc le dernier
print(T)                # élément) du tableau T.
```

```
[ 5  2  8 17  6 14]
[ 5  2  8 17  6 14 18]
[ 5  2  8 17  6 14]
[ 5  2  8 17  6 14 18]
[ 5  2 20  8 17  6 14 18]
[ 5  2  8 17  6 14 18]
[ 5  2  8 17  6 14]
```

Analyser le contenu d'un tableau numpy

Les fonctions ne modifient pas les tableaux numpy et leur syntaxe est toujours similaire : fonction(T)

Les méthodes modifient les tableaux numpy et leur syntaxe est toujours similaire : T.méthode()

- min(T) ou np.min(T) : Renvoie le plus petit élément du tableau T.
- max(T) ou np.max(T) : Renvoie le plus grand élément du tableau T.
- sorted(T) : Renvoie une copie triée du tableau contenant les éléments du tableau T rangés par ordre croissant ou alphabétique. MAIS, le tableau T n'est pas modifié !
- sorted(T,reverse=True) : Renvoie une copie triée du tableau contenant les éléments du tableau T rangés par ordre décroissant ou inverse du sens alphabétique. MAIS, le tableau T n'est pas modifié !
- T.sort() : Modifie le tableau T qui dorénavant contiendra les éléments triés (mais ne le renvoie pas).

Remarque : sort() est une méthode et non une fonction… D'où sa syntaxe différente.

- choice(T) : Renvoie en choisissant au hasard un élément du tableau T. Cette fonction appartient au module random.

In [4]:

```
# Quelques fonctions permettant d'analyser le contenu d'une liste :
# applications

# La fonction choice appartient à la bibliothèque random
from random import choice

T=np.array([5,2,8,17,6,14])
print("T = ",T,'\n')

print(min(T),'et',np.min(T))
print(max(T), 'et',np.max(T),'\n')

Ttrie_endroit=sorted(T)
print("Le tableau Ttrie_endroit est une copie triée de T :\n",
      "Ttrie_endroit = ",Ttrie_endroit)
print("Comme vous pouvez le constater, le tableau T n'est pas modifié :\n",
      "T = : ",T,'\n')

Ttrie_envers=sorted(T,reverse=True)
print("Le tableau Ttrie_envers est une copie triée de T :\n",
      "Ttrie_envers = ",Ttrie_envers)
print("Comme vous pouvez le constater, le tableau T n'est pas modifié :\n",
      "T =",T,'\n')

T.sort()
print("Comme vous pouvez le constater, le tableau T est modifié :\n",
      "T =",T,'\n')

print(choice(T))
```

```
T =  [ 5  2  8 17  6 14]

2 et 2
17 et 17

Le tableau Ttrie_endroit est une copie triée de T :
 Ttrie_endroit =  [2, 5, 6, 8, 14, 17]
Comme vous pouvez le constater, le tableau T n'est pas modifié :
 T = :  [ 5  2  8 17  6 14]

Le tableau Ttrie_envers est une copie triée de T :
 Ttrie_envers =  [17, 14, 8, 6, 5, 2]
Comme vous pouvez le constater, le tableau T n'est pas modifié :
 T = [ 5  2  8 17  6 14]

Comme vous pouvez le constater, le tableau T est modifié :
 T = [ 2  5  6  8 14 17]

5
```

Parcourir le contenu d'un tableau numpy

Comme pour les listes , la boucle ń for ż est particulièrement bien adaptée aux tableaux de valeurs.

Soit T un tableau numpy de longueur n :

→ Si on a besoin de parcourir un tableau élément par élément grâce à leur indice pour ensuite utiliser une instruction faisant intervenir cet indice, on utilise l'instruction : **for i in range(0, len(T)) :**

la variable i (qui représente l'indice d'un élément) prendra les valeurs de 0 à len(T)-1 soit de 0 à n-1

→ Si on a besoin de parcourir un tableau, élément par élément, en nous intéressant uniquement à leur valeur pour ensuite utiliser une instruction permettant de travailler sur ces valeurs, on peut utiliser l'instruction : **for x in T :**

la variable x prendra l'une après l'autre toutes les valeurs du tableau T.

Remarque 1: On peut toujours utiliser la première instruction à la place de la deuxième, mais pas l'inverse !

Remarque 2: les lettres i, j et k sont traditionnellement utilisées pour désigner les indices alors que les autres lettres désignent des variables. Par exemple, ici, la lettre x parcourt les valeurs de T.

In [5]:

```
# Comment parcourir le contenu d'un tableau numpy :
# applications de base

T=np.array([5,2,8,17,6,14])

print("premier exemple :")

# i balaye (sera égal à) tous les indices des cellules du tableau
# un par un (ici de 0 à 5)

for i in range(0,len(T)):

# test permettant de sélectionner les indices i pairs
# (reste de la division de i par 2 vaut 0)

    if i%2==0 :
        print(T[i])

print("second exemple :")

# x balaye toutes les valeurs du tableau (ici 5, puis 2, puis 8...)
for x in T :
    if x > 7:
        print(x)
```

```
premier exemple :
5
8
6
second exemple :
8
17
```

14

In [6]:

```
# Comment parcourir le contenu d'un tableau :
# applications plus poussées

print("premier exemple :")
T=np.array([5,2,8,17,6,14])
for i in range (0,len(T)):
    T[i] = T[i] + 1          # on additionne 1 à chaque valeur du tableau
print(T)

print("second exemple :")
T=np.array([5,2,8,17,6,14])

# création d'une nouvelle variable s qui est initialisée à 0
s = 0

for x in T :

# après le premier passage dans la boucle, s sera égal à son
# ancienne valeur (0) additionné à x
    s = s + x

    print (" s intermédiaire : ", s)
print (" Somme finale : ", s)

# test pour savoir si la valeur 8 est dans le tableau

if 8 in T : print("le nombre 8 est présent dans le tableau")
else : print("le nombre 8 n'est pas présent dans le tableau")
```

```
premier exemple :
[ 6  3  9 18  7 15]
second exemple :
 s intermédiaire :  5
 s intermédiaire :  7
 s intermédiaire :  15
 s intermédiaire :  32
 s intermédiaire :  38
 s intermédiaire :  52
 Somme finale :  52
le nombre 8 est présent dans le tableau
```

Opérations mathématiques sur les tableaux numpy (vectorisation)

Numpy permet de réaliser des opérations directement sur les éléments d'un tableau sans être obligé de créer une boucle for comme pour les listes, d'où la grande utilité d'utiliser des tableaux numpy au lieu de listes dans les programmes liés aux sciences-physiques.

Prenons un exemple: Soit un dipôle ohmique de résistance R = 3 Ω. On veut déterminer les valeurs que prend la tension U pour une intensité I dont les valeurs sont regroupées dans la liste: I = [0, 0.10, 0.20, 0.30]

In [7]:

```
# Création de la liste U

# Si l'on fait :
R = 3         # R est forcément un entier sinon une exception
              # (message d'erreur) sera renvoyée lors de
              # l'exécution du code.

I = [0.10, 0.20, 0.30]
U = R*I
print (U)
print ("On obtient la concaténation de 3 fois la liste I \n",
       "donc MAUVAISE METHODE")
```

```
[0.1, 0.2, 0.3, 0.1, 0.2, 0.3, 0.1, 0.2, 0.3]
On obtient la concaténation de 3 fois la liste I
 donc MAUVAISE METHODE
```

In [8]:

```
# On doit donc faire :
I = [0.10, 0.20, 0.30]
R = 3
U = []
for i in range (len (I)):
    Ui=R*I[i]
    U.append(Ui)
print (U)
print ("On obtient bien une liste contenant les valeurs de la tension U \n",
       "donc bonne méthode mais demandant 4 lignes de code")
```

```
[0.30000000000000004, 0.6000000000000001, 0.8999999999999999]
On obtient bien une liste contenant les valeurs de la tension U
 donc bonne méthode mais demandant 4 lignes de code
```

In [9]:

```
# Regardons maintenant ce que cela donne avec des tableaux numpy.
I=np.array([0.10, 0.20, 0.30])
R=3
U=R*I
print (U)
print ("On obtient un tableau contenant les valeurs de la tension U \n",
       "donc BONNE METHODE ne demandant qu'une seule ligne de code!")
```

```
[0.3 0.6 0.9]
On obtient un tableau contenant les valeurs de la tension U
 donc BONNE METHODE ne demandant qu'une seule ligne de code!
```

De nombreuses opérations mathématiques de base peuvent être réalisées avec des tableaux numpy. De plus, un grand nombre de fonctions de la bibliothèque math sont aussi définies dans la bibliothèque numpy et peuvent être utilisées lors de calculs utilisant les tableaux numpy.

In [10]:

```
import math

print (math.pi, math.log(1),math.sqrt(25))

print (np.pi, np.log(1),np.sqrt(25))
```

```
3.141592653589793 0.0 5.0
3.141592653589793 0.0 5.0
```

Conversion d'un tableau numpy de caractères vers une chaîne de caractères

Attention : ici, le tableau numpy doit être forcément constitué de caractères

T = np.array(['I','S','N']) : crée un tableau numpy contenant 3 éléments de type caractère

'sep'.join(T) : renvoie une chaîne de caractères obtenue en concaténant les éléments du tableau T séparés par le séparateur sep.

In [11]:

```
T=np.array(['I','S','N'])
mot='-'.join(T)      # ici le séparateur est un tiret -
print(T)
print(mot)
```

```
['I' 'S' 'N']
I-S-N
```

Création de tableau numpy à deux dimensions

Nous allons travailler sur un nouvel exemple

tableau = np.array ([['Anne','Tom','Léo','Eva'], [6,7,8,9],[10,20,30,40]] :
crée un tableau contenant 3 lignes de 4 éléments chacune donc un tableau de 3 lignes x 4 colonnes

- Tableau[0] : renvoie la première ligne du tableau
- Tableau[i][j] : renvoie le jème élément de la ième ligne du tableau

In [12]:

```
# On peut créer des tableaux à plusieurs dimensions

tableau = np.array([['Anne','Tom','Léo','Eva'], [6,7,8,9],[10,20,30,40]])
print("Tableau\n", tableau,'\n')
print("Ligne\n", tableau[0],'\n')
print("Elément d'une ligne\n", tableau[0][0],'\n')
print("Elément d'une ligne\n", tableau[1][2],'\n')
print("Elément d'une ligne, à partir de la fin\n", tableau[-1][-1],'\n')
print("Elément d'une ligne, à partir de la fin\n",tableau[-1][0], "\n")
print("Transposition\n", tableau.T, "\n")
print("Partie de tableau\n", tableau[0:3,1:3], "\n")
print("Colonne\n", tableau[:,0], "\n")
```

```
Tableau
 [['Anne' 'Tom' 'Léo' 'Eva']
 ['6' '7' '8' '9']
 ['10' '20' '30' '40']]

Ligne
 ['Anne' 'Tom' 'Léo' 'Eva']

Elément d'une ligne
 Anne

Elément d'une ligne
 8

Elément d'une ligne, à partir de la fin
 40

Elément d'une ligne, à partir de la fin
 10

Transposition
 [['Anne' '6' '10']
 ['Tom' '7' '20']
 ['Léo' '8' '30']
 ['Eva' '9' '40']]

Partie de tableau
 [['Tom' 'Léo']
 ['7' '8']
 ['20' '30']]

Colonne
 ['Anne' '6' '10']
```