Les listes

Comme son nom l'indique, une liste permet de lister différents éléments.

Les éléments d'une liste :

- s'écrivent entre crochets,
- sont séparés par une virgule,
- peuvent être de nature différente (chaine de caractères, nombre entier, nombre réel,...)
- sont repérés par leur position dans la chaîne, appelée indice.

ATTENTION : le premier indice d'une liste 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.

Remarque supplémentaire :

Il existe des listes non modifiables appelées n-uplets (ou tuple en anglais).

Toutes les méthodes ou fonctions décrites ci-dessous et permettant de modifier les éléments d'une liste ne sont bien sûr pas applicables à un n-uplet, puisque, par définition, un n-uplet n'est pas modifiable.

Créer des listes

L = [5,2,8,17,6,14]

Crée une liste nommée L contenant les éléments 5, 2, 8, 17, 6 et 14.

L'initialisation d'un n-uplet s'effectue différemment. Il faut écrire les éléments entre parenthèses et non pas entre crochets comme pour les listes.

upl=(1,'a',3)

Le n-uplet nommé upl contient les éléments 1, a et 3.

In [1]:

```
# Plusieurs manières de créer des listes

L = [5,2,8,17,6,14]
print(L)
upl=(1,'a',3)
print(upl)
```

```
[5, 2, 8, 17, 6, 14]
(1, 'a', 3)
```

Récupérer l'indice d'un élément d'une liste

La méthode L.index() permet de récupérer l'indice d'un élément d'une liste. Attention, dans le cas où plusieurs éléments ont la même valeur, cette méthode renvoie l'indice de l'élément d'indice le plus petit.

In [2]:

```
L = [5,2,8,17,8,14]
a=L.index(5)
print(a)
b=L.index(8)
print(b)
```

```
0
2
```

Récupérer les éléments d'une liste

ATTENTION : l'accès à un élément d'un n-uplet par indice s'effectue grâce à des crochets comme pour les listes.

Nous allons travailler sur l'exemple de la liste créée ci-dessus :

```
L[0]          renvoie le premier élément, ici 5
L[2]          renvoie l'élément d'indice 2 (en 3ème position donc), ici 8
L[-1]         renvoie le dernier élément, ici 14
L[-2]         renvoie l'avant-dernier élément, ici 6
L[1:3]        renvoie les éléments d'indice 1 et 2  (ATTENTION : indice 3 non inclus)
L[3:]         renvoie les éléments dont les indices sont supérieurs à 3.
L[:4]         renvoie les éléments dont les indices ont pour valeur : 0, 1, 2 et 3.
len(L)        renvoie la longueur de la liste, ici 6
L1 = []       crée une liste vide
```

In [3]:

```
# Plusieurs manières de récupérer les éléments d'une liste
# Applications

L=[5,2,8,17,6,14]            # création d'une liste
upl=(1,'a',3)                #  création d'un n-uplet
print(L)
print(L[0])
print(L[2])
print(L[-1])
print(L[-2])
print(L[1:3])
print(L[3:])
print(L[:4])
print(len(L))
print(len(upl))
print(upl[1])
L1=[]
print(L1)
```

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

```
[5, 2, 8, 17]
6
3
a
[]
```

Ajouter une valeur ou supprimer une valeur d'une liste

Les méthodes modifient les listes et leur syntaxe est toujours similaire : L.méthode()

In [4]:

```
# PLusieurs méthodes pour ajouter une valeur ou supprimer des valeurs d'une liste
#(cours et applications)

L=[5,2,8,17,6,14]
print(L)
L.append(20)                  # Ajoute lélément 20 à la fin de la liste L.
print(L)
L.insert(2,20)                # Insère lélément 20 à la position d'indice 2
print(L)                      #... (donc en troisième position) de la liste L.
L.remove(20)                  # Supprime la première occurrence (apparition)
print(L)                      #...de lélément 20 dans la liste L.
L.pop(-1)                     # Supprime lélément dindice -1
print(L)                      #... (donc le dernier élément) de la liste L.
```

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

Analyser le contenu d'une liste

Les fonctions ne modifient pas les listes et leur syntaxe est toujours similaire : fonction(L)

Les méthodes modifient les listes et leur syntaxe est toujours similaire : L.méthode()

- min(L) : Renvoie le plus petit élément de la liste L.
- max(L) : Renvoie le plus grand élément de la liste L.
- sorted(L) : Renvoie une copie triée de la liste contenant les éléments de la liste L rangés par ordre croissant ou alphabétique. MAIS, la liste L n'est pas modifiée !
- sorted(L,reverse=True) : Renvoie une copie triée de la liste contenant les éléments de la liste L rangés par ordre décroissant ou inverse du sens alphabétique. MAIS, la liste L n'est pas modifiée !
- L.sort() : Modifie la liste L qui dorénavant contiendra les éléments triés (mais ne la renvoie pas).

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

- choice(L) : Renvoie en choisissant au hasard un élément de la liste L. Cette fonction appartient au module random.

- sample(L,3) : Retourne une liste de 3 éléments choisis aléatoirement et sans remise dans la liste L. Cette fonction appartient au module random.

In [5]:

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

from random import choice, sample
# Les fonctions sample et choice appartiennent à la bibliothèque random

L=[5,2,8,17,6,14]
print("L = ",L)
print(min(L))
print(max(L))
Ltrie_endroit=sorted(L)
print("la liste Ltrie_endroit est une copie triée de L :  ",Ltrie_endroit)
print("Comme vous pouvez le constater, la liste L n'est pas modifiée : L = : ",L)
Ltrie_envers=sorted(L,reverse=True)
print("la liste Ltrie_envers est une copie triée de L :  ",Ltrie_envers)
print("Comme vous pouvez le constater, la liste L n'est pas modifiée : L =",L)
L.sort()
print("Comme vous pouvez le constater, la liste L est modifiée : L =",L)
print(choice(L))
print(sample(L,3))
```

```
L =  [5, 2, 8, 17, 6, 14]
2
17
la liste Ltrie_endroit est une copie triée de L :   [2, 5, 6, 8, 14, 17]
Comme vous pouvez le constater, la liste L n'est pas modifiée : L = :  [5, 2, 8, 17, 6,
14]
la liste Ltrie_envers est une copie triée de L :   [17, 14, 8, 6, 5, 2]
Comme vous pouvez le constater, la liste L n'est pas modifiée : L = [5, 2, 8, 17, 6, 14]
Comme vous pouvez le constater, la liste L est modifiée : L = [2, 5, 6, 8, 14, 17]
6
[5, 2, 17]
```

Parcourir le contenu d'une liste

La boucle ń for ż est particulièrement bien adaptée aux listes de valeurs.

Soit L une liste de longueur n :

→ Si on a besoin de parcourir une liste 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(L)) :**

la variable i (qui représente l'indice d'un élément) prendra les valeurs de 0 à len(L)-1 soit de 0 à n-1

→ Si on a besoin de parcourir une liste, 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 L :**

la variable x prendra l'une après l'autre toutes les valeurs de la liste L.

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 L.

In [6]:

```
#Comment parcourir le contenu d'une liste : applications de base

L=[5,2,8,17,6,14]
print("premier exemple :")
for i in range(0,len(L)):
# i sera égal à tous les indices de la liste (ici de 0 à 5)
    if i%2==0 :  print(L[i])
# test permettant de sélectionner les indices i pairs
#...(reste de la division de i par 2 vaut 0)

print("second exemple :")
for x in L :
# x balaye toutes les valeurs de la liste (ici 5, puis 2,...)
    if x > 7:
        print(x)
```

```
premier exemple :
5
8
6
second exemple :
8
17
14
```

In [7]:

```
#Comment parcourir le contenu d'une liste : applications plus poussées

print("premier exemple :")
L=[5,2,8,17,6,14]
for i in range (0,len(L)):
    L[i] = L[i] + 1   # on additionne 1 à chaque valeur de la liste
print(L)

print("second exemple :")
L=[5,2,8,17,6,14]
# création d'une nouvelle variable s qui est initialisée à 0
s = 0
for x in L :
    s = s + x
# après le premier passage dans la boucle, s sera égal à
# ...son ancienne valeur (0) additionné à x
    print (" s intermédiaire : ", s)
print (" Somme finale : ", s)

# test pour savoir si la valeur 8 est dans la liste
if 8 in L : print("le nombre 8 est présent dans la liste")
else : print("le nombre 8 n'est pas présent dans la liste")
```

```
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 la liste
```

Concaténation de listes

On peut concaténer (mettre bout à bout) des listes, et ce, de plusieurs manières différentes.

En voici quelques exemples : (L1, L2 et L sont des listes)

L = 3 * L1 : L sera une liste contenant les éléments de L1, répétés 3 fois. Elle sera donc trois fois plus longue.

L = L1 + L2 : L sera la concaténation de L1 et L2. elle contiendra d'abord les éléments de L1, puis ceux de L2

In [8]:

```
# Opérations mathématiques sur les listes : applications
L1=[1,2,3]
L2=[4,5,6]
L=3*L1
print(L1)
print(L)
L=L1+L2
print(L)
```

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

Conversion d'une chaîne de caractères vers une liste de caractères ou l'inverse

→ D'une chaîne de caractères vers une liste de caractères

```
ch='ISN'  :     Création d'une chaîne de caractères nommée ch
                et assignation de cette chaîne avec les caractères ISN

list(ch)  :     Convertit la chaîne de caractères ch en liste de caractères
```

→ D'une liste de caractères vers une chaîne de caractères

Attention : la liste doit être forcément constituée de caractères

```
L = ['I','S','N']  :   crée une liste contenant 3 éléments de type caractère
'sep'.join(L)  :  renvoie une chaîne de caractères obtenue en concaténant les
                  éléments de la liste L séparés par le séparateur sep.
```

In [9]:

```
ch='ISN'
chbis=list(ch)
print(ch)
print(chbis)

L=['I','S','N']
chter='-'.join(L)       # ici le séparateur est un tiret -
print(L)
print(chter)
```

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

Listes de listes (tableaux)

Nous allons travailler sur un nouvel exemple

```
tableau = [['Anne','Tom','Léo','Eva'], [6,7,8,9],[10,20,30,40]]  :
crée un tableau contenant 3 listes de 4 éléments chacune

Tableau[0]                          # renvoie la première liste
Tableau[i][j]                       # renvoie le jème élément de la ième liste
```

In [10]:

```
# On peut créer des listes de listes (donc un tableau !)

tableau = [['Anne','Tom','Léo','Eva'], [6,7,8,9],[10,20,30,40]]
print(tableau)
print(tableau[0])
print(tableau[0][0])
print(tableau[1][2])
print(tableau[-1][-1])
print(tableau[-1][0])
```

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