Les listes

I. Définitions

Une *liste* est un ensemble d'éléments séparés par des virgules et limité par des crochets : L = [7,5,3,1,5,9]. Le nombre d'éléments d'une liste est la longueur de la liste. Cette longueur est donnée par len(L).

Pour une liste de longueur N, les indices **valides** varient entre $-N$ et $(N-1)$ au sens large : $-N \leqslant k \leqslant N-1$.
L'utilisation d'un indice non valide provoque une erreur :

```
IndexError: list index out of range
```

La liste vide est []. Elle ne contient aucun élément et sa longueur est nulle.

Exemples

Les éléments qui constituent une liste peuvent être de toute nature.

```
[1,2,3,4,5,6]                 # entiers
[3.14, 0.693, 2.78]           # flottants
[True, True, False]           # booléens
['a', 'bra', 'ca', 'da', 'bra']   # chaînes de caractères
[1, 1.609, 'bonjour', None]   # plusieurs types de données
```

Création d'une liste

On peut convertir un *itérable* en liste avec l'opérateur list :

```
list(range(10))
list(range(1,11))
list(range(0,30,5))
list(range(0,-10,-1))
```

On peut énumérer les éléments d'une liste si on dispose d'une formule globale :

```
[2*n+1 for n in range(10)]        # méthode par compréhension
[sin(3*n) for n in range(10) if sin(3*n)>0]
```

II. Indexation

Accès à un élément

On considère une liste L de longueur N.

- L'indice du premier élément est nul.
- L'indice du dernier élément est égal à (N − 1).
- L'élément d'indice k est L[k].
- Si l'entier k est strictement négatif,
  alors $0 \leqslant N + k < N$ et la valeur de L[k] est égale à L[N+k].

Une liste est une variable *mutable*. On peut modifier la valeur de chaque élément en lui affectant une nouvelle valeur : L[0] = 2016

Assignation et référence
Lorsque l'on exécute, en Python, l'instruction A=[1,2,3], il n'est pas assigné la liste [1,2,3] à la variable A mais simplement une référence, c'est-à-dire l'adresse où se trouve la liste en mémoire, à la variable A.
Si on continue en tapant B=A, on ne copie pas la liste mais la référence. Toute modification sur B sera effective sur A et vice-versa.
Si on souhaite faire une véritable copie des données, il faut créer une nouvelle liste et la méthode .copy() des listes le permet :

```
A=[1,2,3]
B=A
B[0]=5
print(B)        # affiche [5,2,3]
print(A)        # affiche [5,2,3]
C=A.copy()
C[0]=0
print(C)        # affiche [0,2,3]
print(A)        # affiche [5,2,3]
```

Pour les mêmes raisons, si une liste L est passée en argument à une fonction f, les modifications de la liste L effectuées à l'intérieur de la fonction f sont encore effectives après l'exécution de f :

```
def f(L):
    L[0] = 2016
    return None
L = [0,1,2,3,4,5,6]
f(L)
print(L)        #affiche [2016,1,2,3,4,5,6]
```

III. Les sous-listes

On considère une liste L de longueur N et deux indices valides $i \leqslant j$.
On suppose que i et j sont positifs.

- La tranche L[i:j] est la sous-liste $(L_k)_{i \leqslant k < j}$
- La tranche L[i:] est la sous-liste $(L_k)_{i \leqslant k} = (L_k)_{i \leqslant k < N}$
- La tranche L[:j] est la sous-liste $(L_k)_{k < j} = (L_k)_{0 \leqslant k < j}$

La tranche L[:] est une copie de la liste L : elles ont la même taille, les mêmes éléments dans le même ordre, mais on peut modifier l'une sans changer l'autre.

```
L = [0,1,2,3,4,5,6]
M = L[:]        # tout comme la méthode M=L.copy()
L == M          # True
L[3] = 0
L == M          # False
```

III. Opérations et méthodes sur les listes

• Recherche d'un élément : L'objet x est un élément de la liste L si, et seulement si, le booléen **x in L** est égal à True.

• Le nombre d'occurrences de l'objet x dans la liste L est donné par L.count(x).
Ainsi, l'élément x appartient à la liste L si, et seulement si, l'entier L.count(x) est strictement positif.

• L'instruction L.index(x) retourne l'indice de la première occurrence de l'objet x dans la liste L ou une erreur ValueError si x n'est pas dans la liste L.

• Insertion et suppression d'un élément

Une liste L peut être modifiée par les méthodes *insert*, *remove* et *pop* :

- On insère l'élément x en position i dans la liste L avec l'instruction L.insert(i, x). La longueur de L augmente d'une unité.
- On enlève de L la première occurrence de x avec l'instruction L.remove(x). La longueur de L diminue d'une unité. Si x n'est pas dans la liste L, il se produit une erreur :

  ValueError: list.remove(x): x not in list
- L'instruction L.pop(i) retourne et supprime l'élément L[i] de la liste L. La longueur de L diminue d'une unité.

```
from random import randint
L = [randint(1,6) for i in range(5)] # On lance un dé 5 fois de suite
while (L!=[]):
    x = L.pop(0)
    print(x, L)
```

aura pour effet de produire

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

Concaténation

On ajoute un élément à la fin de la liste L avec la méthode *append* :

```
La = [1,2,3]
La.append(7)
print(La)       # affiche [1,2,3,7]
```

On concatène deux listes avec l'opérateur + :

```
La, Lb = [1,2,3], [4,5,6]
print(La + Lb) # affiche [1, 2, 3, 4, 5, 6]
Lb = Lb + [8]   # et Lb devient [4,5,6,8]    # comme la méthode append(8)
```

L'opérateur * effectue des concaténations répétées.

```
Lb = [4,5,6]
print(3*Lb)     # affiche [4, 5, 6, 4, 5, 6, 4, 5, 6]
print(Lb*3)     # affiche [4, 5, 6, 4, 5, 6, 4, 5, 6]
```

Classement

Une liste peut être modifiée par la méthode *reverse*, qui inverse l'ordre dans lequel les éléments sont écrits.

Si les éléments de la liste L sont comparables, cette liste peut être triée par ordre croissant avec la méthode L.sort().
Elle peut aussi être triée par ordre décroissant avec la méthode L.sort(reverse=True).

Opérations algébriques

On suppose que les éléments de la liste L sont comparables : nombres ou chaînes de caractères.

- Le plus petit élément de la liste L est donné par min(L).
- Le plus grand élément de la liste L est donné par max(L).
- Si les éléments de la liste L sont des nombres, on obtient la somme de ces nombres avec l'instruction sum(L).