

# Listes

## Spé ISN

*À partir d'une version originale de M.Duclosson*



### Introduction

Pour traiter des données complexes comme des images on a besoin d'un type de variable plus élaboré qu'un simple entier (<code>class 'int'>). Il faudra, dans une unique variable, pouvoir stocker un nombre représentant le niveau de gris de chaque élément de l'image, c'est un tableau d'entiers dont on a besoin. En Python, les listes permettent de faire cela.

## I Définitions et opérations sur les listes



### Définition 1 (*Listes*)


Les listes de Python correspondent à une structure de données généralement appelée tableaux. Elles consistent en un ensemble d'objets de types quelconques (éventuellement dépareillés) ordonnés et accessibles par leur numéro de position appelé indice ou index. Attention le premier élément porte le numéro 0. On dit que c'est une structure de données de type composite.

Pour définir explicitement une liste à partir de ses éléments on utilise les délimiteur [ et ] et le séparateur , . Si il n'y a pas d'élément, on obtient la liste vide.

```
>>> L = [-3, 'ISN', 2.016]
>>> type(L)
<class 'list'>
>>> listeVide = []
```

On peut définir une liste de listes<sup>1</sup> ou même une liste de listes de listes, etc. La fonction `len` renvoie le nombre d'élément d'une liste.

```
>>> tables = [[0, 2, 4, 6], [0, 3, 6, 9], [0, 4, 8, 12]]
>>> len(tables)
3
```

Pour accéder à un élément, on utilise son indice.  Attention, les indices commencent à 0.

1. C'est une bonne façon de représenter un tableau à double entrée.

```
>>> L[0]
-3
>>> L[2]
2.016
>>> tables[2][1]
4
```

Le dépassement de la plage des indices génère une erreur classique :

```
>>> len(L)
3
>>> L[3]
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    L[3]
IndexError: list index out of range
```

Les indices négatifs permettent de parcourir la liste à l'envers :

```
>>> L[-1]
2.016
>>> L[-4]
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    L[-4]
IndexError: list index out of range
>>> L[-3]
-3
```

Pour supprimer un élément on utilise l'instruction `del` (delete signifie effacer en anglais).

```
>>> L
[-3, 'ISN', 2.016]
>>> del L[1]
>>> L
[-3, 2.016]
```

On peut modifier une liste en changeant la valeur d'un de ces éléments<sup>2</sup> :

```
>>> chiffresAnnee = [2, 0, 1, 6]
>>> chiffresAnnee[3] = 7
>>> chiffresAnnee
[2, 0, 1, 7]
```

## Intermède : notion d'objets et de méthode en Python

Python est un langage orienté objet ; tout ce qu'on manipule est objet : une variable, une liste, une fonction sont des objets. Un objet est muni de « méthodes », ce sont les actions que l'on peut lui appliquer.

Pour donner une idée de ce que c'est, on pourrait dire qu'un objet de type « chien » devrait disposer des méthodes : manger, aboyer, chercher le bâton (liste non exhaustive).

---

2. On dit qu'une liste est mutable ou modifiable.

Pour appliquer une méthode de sa classe à un objet on utilise la syntaxe :

`nom_objet.nom_methode`

Par exemple la méthode `.sort()` permet de trier une liste :

```
>>> listeEntiers = [4, 0, 7, 2]
>>> listeEntiers
[4, 0, 7, 2]
>>> listeEntiers.sort() # effet de bord
>>> listeEntiers
[0, 2, 4, 7]
```

Les principales méthodes applicables aux listes :

```
>>> L = [1, 3, 5, 7]
>>> L.append(9) # ajouter un élément en fin de liste
>>> L
[1, 3, 5, 7, 9]
>>> L.pop(1) # renvoie l'élément d'indice donné et le supprime
3
>>> L
[1, 5, 7, 9]
>>> L.insert(3, 5) # insère à la position donnée l'élément
>>> L
[1, 5, 7, 5, 9]
>>> L.count(5) # renvoie le nombre d'occurrences de la valeur
2
>>> L.index(5) # renvoie l'indice de la première occurrence
1
>>> L.remove(5) # élimine la première occurrence de la valeur
>>> L
[1, 7, 5, 9]
>>> L.extend([11, 13]) # ajoute les éléments d'une autre liste
>>> L
[1, 7, 5, 9, 11, 13]
```

La fonction `list()` renvoie une liste correspondant à son argument :

```
>>> list('abcbde')
['a', 'b', 'c', 'b', 'd', 'e']
>>> list(range(4, 10, 2))
[4, 6, 8]
```

Les listes sont *itérables*. On peut les parcourir avec une boucle `for` **sans utiliser les indices** :

```
>>> noms = ['Alice', 'Bernard', 'Cécile']
>>> for x in noms:
    print(x)

Alice
Bernard
Cécile
```

Mais le parcours par indice est parfois utile voire nécessaire :

```
>>> nbPremiers = [2, 3, 5, 7, 11]
>>> for i in range(len(nbPremiers)):
    print(nbPremiers[i])

2
3
5
7
11
```

On peut tester l'appartenance à une liste :

```
>>> 4 in nbPremiers
False
```

### Exercice 1

On suppose donnée une liste d'entiers L, un entier x. Écrire un code qui compte dans une variable n le nombre d'éléments de L supérieurs strictement à x.

.....

.....

.....

.....

.....

.....

.....

.....

### Exercice 2

On suppose donnée une liste d'entiers L et un entier x. Écrire un code qui renvoie la liste (éventuellement vide) des indices des occurrences de la valeur x dans la liste L.

.....

.....

.....

.....

.....

.....

## II Concaténation et tranchage

On peut mettre bout à bout deux listes pour en former une troisième.

```
>>> L1 = [1, 2, 3]
>>> L2 = [4, 5]
>>> L1_2 = L1 + L2
>>> L1_2
[1, 2, 3, 4, 5]
>>>
```

Chacun sait que la répétition d'une addition c'est une multiplication :

```
>>> L1*4
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

On peut découper une tranche dans une liste (la logique est celle de la fonction [range](#)) :

```
>>> L = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
>>> L[1:5]
[1, 4, 1, 5]
>>> L[4:]
[5, 9, 2, 6, 5, 3]
>>> L[:5]
[3, 1, 4, 1, 5]
>>> L[2:8:2]
[4, 5, 2]
>>> L[::-1]    #inversion de l'ordre d'apparition des éléments d'une
               liste
[3, 5, 6, 2, 9, 5, 1, 4, 1, 3]
```

On peut même remplacer une tranche par une autre liste de taille différente :

```
>>> L = [0, 4, 8, 12]
>>> L[1:3] = [4, 5, 6, 7, 8]
>>> L
[0, 4, 5, 6, 7, 8, 12]
```

Ou la supprimer :

```
>>> del L[1:3]
>>> L
[0, 6, 7, 8, 12]
```

### III Listes par compréhension



#### Exercice 3

Quelle liste produit ce code?

```
L = []  
for k in range(10):  
    L.append(k**2)
```

Vérifier le à l'aide de l'éditeur.

Une façon plus courte d'obtenir le même résultat est d'utiliser les listes décrites par compréhension :

```
>>> listeCarres = [k**2 for k in range(10)]  
>>> listeCarres  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

On peut y ajouter des conditions :

```
>>> premiers = [2, 3, 5, 7, 11, 13]  
>>> composes = [k for k in range(2, 14) if not(k in premiers)]  
>>> composes  
[4, 6, 8, 9, 10, 12]
```



#### Exercice 4

1. Écrire une instruction qui génère une liste de tous les carrés des entiers pairs entre 0 et 100.

.....

2. Compléter le code ci-dessous pour que la liste L contienne uniquement les images positives des entiers compris entre 0 et 100 par la fonction cosinus.

```
from math import cos
```

```
L = .....
```

3. Écrire un code qui produit une liste de listes tables qui contient les résultats des tables de multiplication entre 1 et 9 :

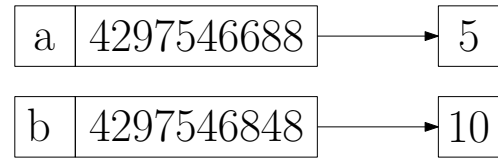
```
[[1, 2, 3, 4, 5, 6, 7, 8, 9],  
.....,  
[9, 18, 27, 36, 45, 54, 63, 72, 81]]
```

.....

.....

[illegible]

```
>>> b = 2*b
>>> id(b)
4297546848
>>> b
10
```



Lors de l'exécution de l'instruction `b = 2*b`, il n'y a pas modification de la valeur contenue dans la mémoire<sup>4</sup> à l'adresse 4297546688 mais Python crée une nouvelle valeur pour le résultat. Ensuite la référence de `b` est modifiée (4297546848) pour pointer vers cette valeur.

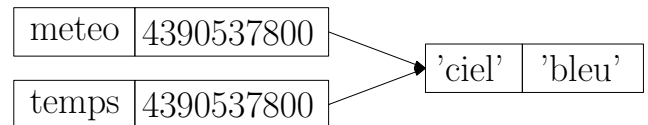
Noter que Python ne crée pas de nouvelles valeurs lorsque ce n'est pas nécessaire :

```
>>> c = b//2
>>> id(c)
4297546688      # On retrouve
                la référence de a
>>> c
5
```

**NB :** Les types : chaînes de caractères ('str'), flottants ('float'), booléen ('bool'), tuple ('tuple') et fonction ('fonction') sont tous non modifiables et se comporteront donc comme les entiers.

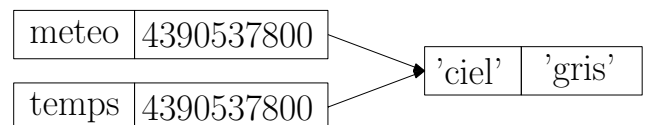
Les listes sont **modifiables**, leur valeur peut être changée après leur création. Les choses se passent donc très différemment :

```
>>> meteo = ['ciel', 'bleu']
>>> id(meteo)
4390537800
>>> temps = meteo
>>> id(temps)
4390537800
```



Le temps passe au gris, voyons ce qu'il en est de la météo :

```
>>> temps[1] = 'gris'
>>> meteo
['ciel', 'gris']
>>> id(temps)
4390537800
```



Les noms `meteo` et `temps` pointent vers la même liste. La modification de cette liste change aussi bien la valeur de `meteo` que de `temps`. Il faut bien être conscient de ce comportement car il peut être la source d'erreurs. On parle d'effet de bord<sup>5</sup> non désiré.

Certaines opérations modifient la liste et d'autres, qu'on pourrait croire équivalentes, créent une nouvelle liste.

4. Ce serait catastrophique car la valeur de `a` ne serait plus 5!

5. C'est une traduction de « side effect » qui signifie aussi effet secondaire.



```
>>> L = [2, 3, 5, 7]
>>> id(L)
4390537608
>>> L.append(11)      # .append() modifie la liste
>>> id(L)
4390537608
>>> L
[2, 3, 5, 7, 11]
>>> L = L + [13]      # La concaténation crée une nouvelle liste
>>> id(L)
4386666376
>>> L
[2, 3, 5, 7, 11, 13]
```

Pour créer une copie d'une liste on peut utiliser la fonction `list()` ou un tranchage.

```
>>> M = list(L)
>>> id(M)
4390545928
>>> M
[2, 3, 5, 7, 11, 13]
>>> N = L[:]
>>> N
[2, 3, 5, 7, 11, 13]
>>> id(N)
4386732168
```



### Exercice 5

Écrire un code pour permuter les éléments d'indices a et b de la liste L.

```
L = [2, 3, 5, 7, 11, 13, 17]
a = 2
b = 5
# commencer ici
```

.....

.....

.....

.....

.....



### Exercice 6

Compléter l'instruction évaluée dans la console ci-dessous pour que la liste L3 contienne alternativement les éléments des listes L1 et L2 de même taille.

```
In [4]: L1 =[2, 3, 5, 7]
...: L2 = [17, 24, 35, 81]

In [5]: L3 =
.....

In [6]: L3
Out[6]: [2, 17, 3, 24, 5, 35, 7, 81]
```

## Exercice 7

Le site <http://pythontutor.com/visualize.html> fournit un outil de visualisation de l'état courant d'un programme très pratique.

Exécutez le code ci-dessous dans <http://pythontutor.com/visualize.html#mode=edit> puis commentez.

```
M = [ [0, 0, 0] for i in range(3) ]
N = M
P = [e for e in M ]
Q = [ e[:] for e in M ]
M[2][1] = 3
```

Dans une liste de listes un effet de bord peut apparaître alors qu'on pensait l'avoir évité.

```
>>> L = [[2, 3], [5]]
>>> M = list(L)
>>> id(L)
4386734344
>>> id(M)
4390529160
>>> M[0][1] = 7
>>> L
[[2, 7], [5]]
>>> M
[[2, 7], [5]]
```

Pour copier en profondeur une liste on peut utiliser la fonction `deepcopy()` du module `copy`.

```
>>> from copy import *
>>> M = deepcopy(L)
>>> M[0][1] = 7
>>> L
[[2, 3], [5]]
>>> M
[[2, 7], [5]]
```

Une explication très claire du caractère modifiable des listes est donnée dans cette video :

<https://d381hmu4snvm3e.cloudfront.net/videos/MhgBaG50LRrH/SD.mp4>