

Listes

Fiche de synthèse

① DÉFINITION

Définition 1 (Liste) Une **liste** regroupe plusieurs valeurs dans une seule variable, entre crochets et séparées par des virgules. Ses éléments sont ordonnés : chacun a un **indice**. Une liste est **mutable** : on peut ajouter, supprimer ou remplacer des éléments.

```
notes = [15, 12, 18, 14, 16]
panier = [] # liste vide
lettres = list("NSI") # ['N', 'S', 'I']
```

② INDICES

```
print(notes[0]) # 15, premier élément
print(notes[-1]) # 16, dernier élément
print(len(notes)) # 5 éléments
notes[1] = 14 # remplace le 2e élément
```

Comme pour une chaîne, les indices commencent à 0 et vont jusqu'à $\text{len}(L) - 1$; `notes[5]` provoque l'erreur `IndexError`. Contrairement à une chaîne, on peut affecter une nouvelle valeur à un indice.

③ MÉTHODES

Méthode	Effet
<code>L.append(x)</code>	ajoute <code>x</code> à la fin
<code>L.insert(i, x)</code>	insère <code>x</code> à l'indice <code>i</code>
<code>L.pop()</code>	retire et renvoie le dernier élément
<code>L.pop(i)</code>	retire et renvoie l'élément d'indice <code>i</code>
<code>L.remove(x)</code>	retire la première occurrence de <code>x</code>
<code>L.sort()</code>	trie la liste
<code>L.reverse()</code>	inverse l'ordre
<code>L.index(x)</code>	indice de la première occurrence de <code>x</code>
<code>L.count(x)</code>	nombre d'occurrences de <code>x</code>

⚠ Attention. `append`, `sort`, `reverse` et `remove` modifient la liste **en place** et renvoient `None` : on n'écrit jamais `L = L.sort()`.

④ PARCOURIR UNE LISTE

```
for note in notes: # les valeurs
    print(note)

for i in range(len(notes)): # les indices
    notes[i] = notes[i] + 1

for i, note in enumerate(notes): # les deux
    print(f"Note {i} : {note}")
```

- Le parcours par les **valeurs** suffit pour afficher ou additionner les éléments.
- Le parcours par les **indices** donne la position : il permet de modifier la liste. Écrire `note = note + 1` dans une boucle sur les valeurs ne change pas la liste.
- On ne modifie pas la longueur d'une liste pendant qu'on la parcourt.

⑤ CONSTRUIRE UNE LISTE

```
admis = []
for note in [15, 8, 12, 6, 18]:
    if note >= 10:
        admis.append(note)
print(admis) # [15, 12, 18]
```

On part d'une liste vide et on ajoute un élément à chaque tour avec `append`, éventuellement sous condition, pour **filtrer**.

Une **liste en compréhension** fait la même chose en une ligne : l'élément à ajouter, puis la boucle, puis la condition éventuelle.

```
carres = [x ** 2 for x in range(5)]
# [0, 1, 4, 9, 16]
pairs = [x for x in range(10) if x % 2 == 0]
```

⑥ OPÉRATIONS ET FONCTIONS

- `[1, 2] + [3]` vaut `[1, 2, 3]`; `[0] * 3` vaut `[0, 0, 0]`; `x in L` teste la présence de `x`.
- `len`, `min`, `max` et `sum` donnent la longueur, le minimum, le maximum et la somme : la moyenne est `sum(L) / len(L)`.

⑦ CHÂÎNES ET LISTES : SPLIT ET JOIN

```
"28-03-2026".split("-")
# ['28', '03', '2026']
" Python est génial ".split()
```

```
# ['Python', 'est', 'génial']
" ".join(["Python", "est", "génial"])
# 'Python est génial'
```

`split` découpe une chaîne en liste, sur le séparateur donné ou, sans argument, sur les espaces. `join` s'écrit sur le séparateur et assemble une liste de chaînes.

⑧ COPIE ET EFFETS DE BORD

```
a = [1, 2, 3]
b = a          # b est un autre nom pour a
b[0] = 99
print(a)       # [99, 2, 3]
c = a.copy()   # vraie copie, comme list(a)
```

- `b = a` ne copie pas la liste : `b` est un **alias**, un autre nom de la même liste. `a.copy()` ou `list(a)` créent une nouvelle liste.
- Une fonction qui reçoit une liste reçoit un alias : si elle la modifie, avec `L.append()` ou `L[i] = ...`, la modification est visible après l'appel. C'est un **effet de bord**. Réaffecter le paramètre, `L = []`, ne touche pas la liste de l'appelant.
- Une fonction modifie la liste en place et ne renvoie rien, comme `L.sort()`, ou renvoie une nouvelle liste sans toucher à son argument, comme `sorted(L)` : sa docstring le précise.

⑨ LISTES DE LISTES

```
matrice = [[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]]
print(matrice[1])      # [4, 5, 6] : ligne 1
print(matrice[1][2])   # 6 : ligne 1, colonne 2
```

`M[i][j]` est l'élément de la ligne `i` et de la colonne `j`. On parcourt tous les éléments avec deux boucles imbriquées. `a.copy()` ne copie pas les listes intérieures : `deepcopy(a)`, du module `copy`, les copie aussi.

📌 À retenir.

- Indices de 0 à `len(L) - 1` ; `L[-1]` est le dernier élément.
- `append`, `insert`, `pop`, `remove` modifient la liste.
- `for x in L` parcourt les valeurs, `for i in range(len(L))` les indices.
- Liste vide, boucle et `append` : la façon courante de construire une liste.
- `b = a` crée un alias ; `a.copy()` crée une copie.