Les listes en Python

Résumé

Jusqu'ici, chaque variable contenait une seule valeur : un nombre, un texte, un booléen. Mais comment stocker les notes de trente élèves, les pixels d'une image ou les mots d'une phrase ? On a besoin d'une structure capable de regrouper plusieurs valeurs dans un même objet : c'est la **liste**. Une liste Python est une collection ordonnée d'éléments, que l'on peut parcourir, modifier, trier ou filtrer. C'est la structure de données la plus utilisée en Python : presque tous les algorithmes du programme de NSI manipulent des listes.

Prérequis : Les boucles, Les chaînes de caractères.

① Mémo

Créer et accéder

Création `L = [3, 1, 4, 1, 5]` ou `L = []` (liste vide).
Accès `L[0]` est le premier élément, `L[-1]` le dernier.
Longueur `len(L)` renvoie le nombre d'éléments.
Tranche `L[a:b]` renvoie les éléments d'indice `a` à `b-1`.
Les indices vont de `0` à `len(L) - 1`. Un accès hors limites provoque une erreur `IndexError`.

Modifier une liste

`L.append(x)` Ajoute `x` à la fin.
`L.insert(i, x)` Insère `x` à l'indice `i`.
`L.pop()` Retire et renvoie le dernier élément.
`L.pop(i)` Retire et renvoie l'élément d'indice `i`.
`L.remove(x)` Retire la première occurrence de `x`.
`L[i] = y` Remplace l'élément d'indice `i` par `y`.
Les listes sont **mutables** : on peut modifier leur contenu sans créer de nouvelle liste.

Construction par compréhension

```
[expression for x in sequence]
[expression for x in sequence if condition]
```

Exemples :

- `[i**2 for i in range(5)]` donne `[0, 1, 4, 9, 16]` ;
- `[x for x in L if x > 0]` filtre les éléments positifs de `L`.

⚠ Pièges fréquents

- `L2 = L` ne crée pas de copie : les deux variables désignent la même liste. Pour copier, utiliser `L2 = L[:]` ou `L2 = list(L)`.
- `L = [[0] * 3] * 3` crée trois références vers la même sous-liste ! Utiliser `[[0]*3 for _ in range(3)]`.
- Ne pas modifier une liste pendant qu'on la parcourt avec `for`.

⚠ Erreurs classiques

Code erroné	Code correct	Explication
`L[5]` quand `len(L) == 5`	`L[4]` (dernier indice)	Les indices vont de 0 à `len(L) - 1` : accéder à `L[len(L)]` lève un `IndexError`.
`L2 = L` (alias)	`L2 = L[:]` ou `L2 = list(L)`	L'affectation crée un alias : modifier `L2` modifie aussi `L`. Il faut copier.
`L.append([4, 5])` pour ajouter 4 et 5	`L.extend([4, 5])` ou `L += [4, 5]`	`append` ajoute la liste comme un seul élément ; `extend` ajoute chaque élément.
`L2 = L.sort()` puis utiliser `L2`	`L2 = sorted(L)`	`.sort()` trie en place et renvoie `None` ; `sorted()` renvoie une nouvelle liste triée.

② Exemples

Programme 1 · Parcours par valeur vs par indice

```
fruits = ["pomme", "banane", "cerise"]

for fruit in fruits:          # parcours par valeur
    print(fruit)

for i in range(len(fruits)):  # parcours par indice
    print(f"{i} : {fruits[i]}")
```

Programme 2 · Construction par compréhension

```
pairs = [x for x in range(20) if x % 2 == 0]
# [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Programme 3 · Piège de l'alias

```
A = [1, 2, 3]
B = A          # B est un alias de A, pas une copie
B[0] = 99
print(A)       # [99, 2, 3]  A est aussi modifié !

C = A[:]       # C est une vraie copie
C[0] = 0
print(A)       # [99, 2, 3]  A n'est pas modifié
```

Programme 4 · Méthodes utiles

```
notes = [12, 8, 15, 9, 14, 11]
notes.sort()           # trie en place : [8, 9, 11, 12, 14, 15]
triees = sorted(notes) # renvoie une NOUVELLE liste triée
notes.reverse()        # inverse en place
print(notes.count(12)) # nombre d'occurrences de 12
print(12 in notes)     # True (test d'appartenance)
```

Pour aller plus loin : une liste peut contenir d'autres listes (on parle de *liste de listes* ou de *tableau à deux dimensions*). Ce sujet fait l'objet d'une fiche dédiée (fiche 7, *Tableaux à deux dimensions*).

③ Exercices

Exercice 1 *Manipulations de base*

1. Écrire une fonction `moyenne(L)` qui renvoie la moyenne des éléments d'une liste de nombres.
2. Écrire une fonction `sans_doublons(L)` qui renvoie une nouvelle liste contenant les éléments de `L` sans répétition, en préservant l'ordre de première apparition.
3. Écrire une fonction `interleave(L1, L2)` qui entrecroise deux listes de même longueur. Par exemple, `interleave([1, 2, 3], [4, 5, 6])` renvoie `[1, 4, 2, 5, 3, 6]`.

Exercice 2 *Compréhension de listes* Réécrire chaque boucle sous forme de liste en compréhension :

```
# a)
resultat = []
for i in range(1, 11):
    resultat.append(i ** 3)

# b)
resultat = []
for mot in ["chat", "chien", "lapin", "hamster"]:
    if len(mot) > 4:
        resultat.append(mot.upper())
```

Exercice 3 *Analyse de notes* On dispose d'une liste de notes d'élèves (entières, entre 0 et 20).

1. Écrire une fonction `au_dessus(notes, seuil)` qui renvoie une **nouvelle** liste contenant uniquement les notes supérieures ou égales au seuil.
2. Écrire une fonction `remplacer(notes, ancien, nouveau)` qui renvoie une **copie** de la liste où toutes les occurrences de `ancien` sont remplacées par `nouveau` (la liste originale ne doit pas être modifiée).
3. Écrire une fonction `histogramme(notes)` qui renvoie une liste de 21 entiers, où l'élément d'indice `i` est le nombre d'élèves ayant obtenu la note `i`.

Tester avec `notes = [12, 8, 15, 8, 14, 11, 8, 17, 12, 9]`.

Solutions des exercices

Corrigé de l'exercice 1.

```python
def moyenne(L):
    return sum(L) / len(L)

def sans_doublons(L):
    resultat = []
    for x in L:
        if x not in resultat:
            resultat.append(x)
    return resultat

def interleave(L1, L2):
    resultat = []
    for i in range(len(L1)):
        resultat.append(L1[i])
        resultat.append(L2[i])
    return resultat
```

Vérification :

- `moyenne([10, 12, 14])` = 12,0 ;
- `sans_doublons([3, 1, 4, 1, 5, 3])` = `[3, 1, 4, 5]` ;
- `interleave([1, 2, 3], [4, 5, 6])` = `[1, 4, 2, 5, 3, 6]`.

Corrigé de l'exercice 2.

```python
# a)
resultat = [i ** 3 for i in range(1, 11)]
# [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]

# b)
resultat = [mot.upper() for mot in ["chat", "chien", "lapin", "hamster"]
            if len(mot) > 4]
# ["CHIEN", "LAPIN", "HAMSTER"]
```

Corrigé de l'exercice 3.

```python
def au_dessus(notes, seuil):
    return [n for n in notes if n >= seuil]

def remplacer(notes, ancien, nouveau):
    copie = notes[:]      # copie pour ne pas modifier l'originale
    for i in range(len(copie)):
        if copie[i] == ancien:
            copie[i] = nouveau
    return copie

def histogramme(notes):
    h = [0] * 21          # indices 0 à 20
    for n in notes:
        h[n] += 1
    return h
```

Vérification :

- `au_dessus([12, 8, 15, 8, 14, 11, 8, 17, 12, 9], 12)` renvoie `[12, 15, 14, 17, 12]` ;
- `remplacer([12, 8, 15, 8], 8, 10)` renvoie `[12, 10, 15, 10]` et la liste originale reste inchangée ;
- `histogramme([12, 8, 15, 8, 14, 11, 8, 17, 12, 9])` renvoie une liste où `h[8] = 3`, `h[12] = 2`, etc.

Remarque : la question 2 illustre le piège de l'alias. Si l'on écrivait `copie = notes` au lieu de `copie = notes[:]`, la modification de `copie` modifierait aussi `notes`.