Listes II

Les listes sont tellement utiles qu'il faut savoir les manipuler de façon simple et efficace. C'est le but de cette fiche !

Vidéo ■ Listes II - partie 1 - trancher des listes, trouver le rang

Vidéo ■ Listes II - partie 2 - liste par compréhension

Vidéo ■ Listes II - partie 3 - liste de listes

Cours 1 (Manipuler efficacement les listes)**.**

- **Trancher des listes.**
  - Tu connais déjà `maliste[a:b]` qui renvoie la sous-liste des éléments du rang a au rang $b-1$.
  - `maliste[a:]` renvoie la liste des éléments du rang a jusqu'à la fin.
  - `maliste[:b]` renvoie la liste des éléments du début jusqu'au rang $b-1$.
  - `maliste[-1]` renvoie le dernier élément, `maliste[-2]` renvoie l'avant-dernier élément,...
  - **Exercice.**

	7	2	4	5	3	10	9	8	3
rang :	0	1	2	3	4	5	6	7	8

    Avec `maliste = [7,2,4,5,3,10,9,8,3]`, que renvoient les instructions suivantes ?
    - `maliste[3:5]`
    - `maliste[4:]`
    - `maliste[:6]`
    - `maliste[-1]`
- **Trouver le rang d'un élément.**
  - `liste.index(element)` renvoie la première position à laquelle l'élément a été trouvé. Exemple : avec `liste = [12, 30, 5, 9, 5, 21]`, `liste.index(5)` renvoie 2.
  - Si on souhaite juste savoir si un élément appartient à une liste, alors l'instruction :

    `element in liste`

    renvoie `True` ou `False`. Exemple : avec `liste = [12, 30, 5, 9, 5, 21]`, « `9 in liste` » est vrai, alors que « `8 in liste` » est faux.
- **Liste par compréhension.**

  On peut définir un ensemble en donnant la liste de tous ses éléments, par exemple $E = \{0, 2, 4, 6, 8, 10\}$. Une autre façon est de dire que les éléments de l'ensemble doivent vérifier une certaine propriété. Par exemple le même ensemble E peut se définir par :

  $$E = \{x \in \mathbb{N} \mid x \leqslant 10 \text{ et } x \text{ est pair}\}.$$

  Avec `Python` il existe un tel moyen de définir des listes. C'est une syntaxe extrêmement puissante et efficace. Voyons des exemples :
  - Partons d'une liste, par exemple `maliste = [1,2,3,4,5,6,7,6,5,4,3,2,1]`.

— La commande `liste_doubles = [ 2*x for x in maliste ]` renvoie une liste qui contient les doubles des éléments de la liste `maliste`. C'est donc la liste `[2,4,6,8,...]`.
— La commande `liste_carres = [ x**2 for x in maliste ]` renvoie la liste des carrés des éléments de la liste initiale. C'est donc la liste `[1,4,9,16,...]`.
— La commande `liste_partielle = [x for x in maliste if x > 2]` extrait la liste composée des seuls éléments strictement supérieurs à 2. C'est donc la liste `[3,4,5,6,7,6,5,4,3]`.

- **Liste de listes.**
Une liste peut contenir d'autres listes, par exemple :
`maliste = [ ["Harry", "Hermione", "Ron"], [101,103] ]`
contient deux listes. Nous allons nous intéresser à des listes qui contiennent des listes d'entiers, que nous appellerons des ***tableaux***. Par exemple :
`tableau = [ [2,14,5], [3,5,7], [15,19,4], [8,6,5] ]`
Alors `tableau[i]` renvoie la sous-liste de rang i, alors que `tableau[i][j]` renvoie l'entier situé au rang j dans la sous-liste de rang i. Par exemple :
— `tableau[0]` renvoie la liste `[2,14,5]`,
— `tableau[1]` renvoie la liste `[3,5,7]`,
— `tableau[0][0]` renvoie l'entier 2,
— `tableau[0][1]` renvoie l'entier 14,
— `tableau[2][1]` renvoie l'entier 19.

Activité 1 (Listes par compréhension)**.**

Objectifs : mettre en pratique les listes par compréhension. Dans cette activité les listes sont des listes d'entiers.

1. Programme une fonction `multiplier(liste,k)` qui multiplie chaque élément de la liste par k. Par exemple `multiplier([1,2,3,4,5],2)` renvoie `[2,4,6,8,10]`.
2. Programme une fonction `puissance(liste,k)` qui élève chaque élément de la liste à la puissance k. Par exemple `puissance([1,2,3,4,5],3)` renvoie `[1,8,27,64,125]`.
3. Programme une fonction `addition(liste1,liste2)` qui additionne terme à terme les éléments de deux listes de même longueur. Par exemple `addition([1,2,3],[4,5,6])` renvoie `[5,7,9]`.
Indication. C'est un exemple de tâche où on n'utilise pas les listes par compréhension !
4. Programme une fonction `non_zero(liste)` qui renvoie la liste de tous les éléments non nuls. Par exemple `non_zero([1,0,2,3,0,4,5,0])` renvoie `[1,2,3,4,5]`.
5. Programme une fonction `pairs(liste)` qui renvoie la liste de tous les éléments pairs. Par exemple `pairs([1,0,2,3,0,4,5,0])` renvoie `[0,2,0,4,0]`.

Activité 2 (Atteindre une somme fixée)**.**

Objectifs : chercher à atteindre le total de 100 *dans une liste de nombres.*

On considère une liste de n entiers compris entre 1 et 99 (inclus). Par exemple la liste de $n = 20$ entiers :
`[16,2,85,27,9,45,98,73,12,26,46,25,26,49,18,99,10,86,7,42]`
qui a été obtenue au hasard par la commande :
`liste_20 = [randint(1,99) for i in range(20)]`
On cherche différentes manières de trouver des nombres de la liste dont la somme fait exactement 100.

1. Programme une fonction `somme_deux_consecutifs_100(liste)` qui teste s'il existe deux élé-

ments consécutifs de la liste dont la somme vaut 100. La fonction renvoie « vrai » ou « faux » (mais elle peut aussi afficher les nombres et leur position pour vérification). Pour l'exemple donné la fonction renvoie `False`.

2. Programme une fonction `somme_deux_100(liste)` qui teste s'il existe deux éléments de la liste, situés à des positions différentes, dont la somme vaut 100. Pour l'exemple donné la fonction renvoie `True` et peut afficher les entiers 2 et 98 (aux rangs 1 et 6 de la liste).
3. Programme une fonction `somme_suite_100(liste)` qui teste s'il existe des éléments consécutifs de la liste dont la somme vaut 100. Pour l'exemple donné la fonction renvoie `True` et peut afficher les entiers à suivre 25, 26, 49 (aux rangs 11, 12 et13).
4. *(Facultatif.)* Plus la taille de la liste est grande plus il y a de chances d'obtenir des entiers dont la somme vaut 100. Pour chacune des trois situations précédentes, détermine à partir de quelle taille n de la liste, la probabilité d'obtenir une somme de 100 est plus grande que 1/2.

   Indications. Pour chaque cas, tu obtiens une estimation de cet entier n, en écrivant une fonction `proba(n,N)` qui effectue un grand nombre N de tirages aléatoires de listes à n éléments (avec par exemple $N = 10\,000$). La probabilité est approchée par le nombre de cas favorables (où la fonction renvoie vraie) divisé par le nombre total de cas (ici N).

Activité 3 (Tableau)**.**

Objectifs : travailler avec des listes de listes.

Dans cette activité nous travaillons avec des tableaux carrés de taille $n \times n$ contenant des entiers. L'élément `tableau` est donc une liste de n listes ayant chacune n éléments.
Par exemple (avec $n = 3$) :

```
tableau = [ [1,2,3], [4,5,6], [7,8,9] ]
```

représente le tableau :

1 2 3
4 5 6
7 8 9

1. Écris une fonction `somme_diagonale(tableau)` qui calcule la somme des éléments situés sur la diagonale principale. La diagonale principale de l'exemple donné est constituée de 1, 5, 9, la somme vaut donc 15.
2. Écris une fonction `somme_anti_diagonale(tableau)` qui calcule la somme des éléments situés sur l'autre diagonale. L'anti-diagonale de l'exemple donné est constituée de 3, 5, 7, la somme vaut encore 15.
3. Écris une fonction `somme_tout(tableau)` qui calcule la somme totale de tous les éléments. Pour l'exemple la somme totale vaut 45.
4. Écris une fonction `affiche_tableau(tableau)` qui affiche proprement à l'écran un tableau. Tu peux utiliser la commande :

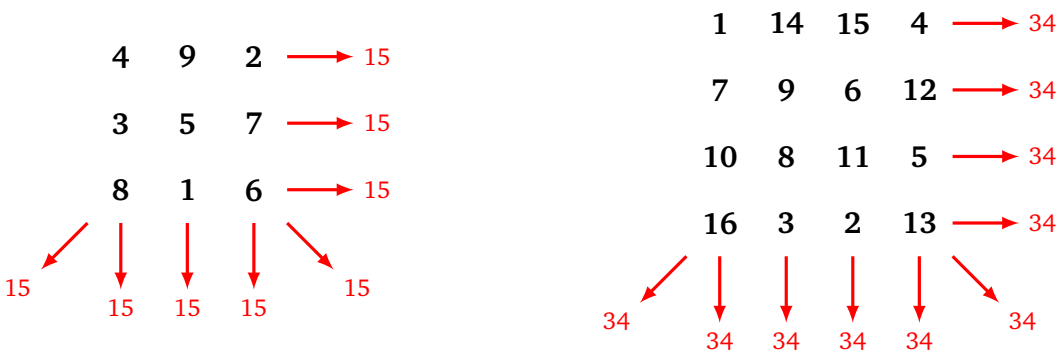
   ```
   print('{:>3d}'.format(tableau[i][j]), end="")
   ```

   Explications.
   - La commande `print(chaine,end="")` permet d'afficher une chaîne de caractères sans passer à la ligne.
   - La commande `'{:>3d}'.format(k)` affiche l'entier k sur trois cases (même s'il n'y a qu'un chiffre à afficher).

Activité 4 (Carrés magiques)**.**

Objectifs : construire des carrés magiques de taille aussi grande que l'on souhaite ! Il faut d'abord avoir fait l'activité précédente.

Un ***carré magique*** est un tableau carré de taille $n \times n$ qui contient tous les entiers de 1 à n^2 et qui vérifie que : la somme de chaque ligne, la somme de chaque colonne, la somme de la diagonale principale et la somme de l'anti-diagonale ont toutes la même valeur.
Voici un exemple de carré magique de taille 3×3 et un de taille 4×4.



Pour un carré magique de taille $n \times n$, la valeur de la somme est :

$$S_n = \frac{n(n^2+1)}{2}.$$

1. **Exemples.** Définis un tableau pour chacun des exemples 3×3 et 4×4 ci-dessus et affiche-les à l'écran (utilise l'activité précédente).
2. **Être ou ne pas être.** Définis une fonction `est_carre_magique(carre)` qui teste si un tableau donné est (ou pas) un carré magique (utilise l'activité précédente pour les diagonales).
3. **Carrés aléatoires.** *(Facultatif.)* Génère de façon aléatoire des carrés contenant les entiers de 1 à n^2 grâce à une fonction `carre_aleatoire(n)`. Vérifie expérimentalement qu'il est rare d'obtenir ainsi un carré magique !

   Indications. Pour une liste `maliste`, la commande `shuffle(maliste)` (issue du module `random`) mélange aléatoirement la liste (la liste est modifiée sur place).

   Le but des questions restantes est de créer des carrés magiques de grande taille.
4. **Addition.** Définis une fonction `addition_carre(carre,k)` qui ajoute un entier k à tous les éléments du carré. Avec l'exemple du carré 3×3, la commande `addition_carre(carre,-1)` soustrait 1 à tous les éléments et renvoie donc un tableau qui s'afficherait ainsi :

   $$\begin{matrix} 3 & 8 & 1 \\ 2 & 4 & 6 \\ 7 & 0 & 5 \end{matrix}$$

   Indications. Pour définir un nouveau carré, commence par le remplir avec des 0 :

   ```
   nouv_carre = [[0 for j in range(n)] for i in range(n)]
   ```

   puis remplis-le avec les bonnes valeurs par des commandes du type :

   ```
   nouv_carre[i][j] = ...
   ```
5. **Multiplication.** Définis une fonction `multiplication_carre(carre,k)` qui multiplie par k tous les éléments du carré. Avec l'exemple du carré 3×3, la commande

`multiplication_carre(carre,2)` multiplie tous les éléments par 2 et renvoie donc un tableau qui s'afficherait ainsi :

8	18	4
6	10	14
16	2	12

6. **Homothétie.** Définis une fonction `homothetie_carre(carre,k)` qui agrandit le carré d'un facteur k comme sur les exemples ci-dessous. Voici un exemple du carré 3×3 avec une homothétie de rapport $k = 3$.

4	9	2
3	5	7
8	1	6

$\longrightarrow$

4	4	4	9	9	9	2	2	2
4	4	4	9	9	9	2	2	2
4	4	4	9	9	9	2	2	2
3	3	3	5	5	5	7	7	7
3	3	3	5	5	5	7	7	7
3	3	3	5	5	5	7	7	7
8	8	8	1	1	1	6	6	6
8	8	8	1	1	1	6	6	6
8	8	8	1	1	1	6	6	6

Voici l'exemple d'un carré 4×4 avec une homothétie de rapport $k = 2$.

1	14	15	4
7	9	6	12
10	8	11	5
16	3	2	13

$\longrightarrow$

1	1	14	14	15	15	4	4
1	1	14	14	15	15	4	4
7	7	9	9	6	6	12	12
7	7	9	9	6	6	12	12
10	10	8	8	11	11	5	5
10	10	8	8	11	11	5	5
16	16	3	3	2	2	13	13
16	16	3	3	2	2	13	13

7. **Addition de blocs.** Définis une fonction `addition_bloc_carre(grand_carre,petit_carre)` qui ajoute par bloc un petit carré de taille $n \times n$ au grand carré de taille $nm \times nm$ comme sur l'exemple ci-dessous avec $n = 2$ et $m = 3$ (donc $nm = 6$). Le petit carré 2×2 à gauche est ajouté au grand carré au centre pour donner le résultat à droite. Pour cette addition le grand carré est décomposé en 9 blocs, il y a en tout 36 additions.

1	2
3	4

4	4	9	9	2	2
4	4	9	9	2	2
3	3	5	5	7	7
3	3	5	5	7	7
8	8	1	1	6	6
8	8	1	1	6	6

$\longrightarrow$

5	6	10	11	3	4
7	8	12	13	5	6
4	5	6	7	8	9
6	7	8	9	10	11
9	10	2	3	7	8
11	12	4	5	9	10

8. **Produits de carrés magiques.** Définis une fonction `produit_carres(carre1,carre2)` qui à partir de deux carrés magiques, calcule un grand carré magique appelé le produit des deux carrés. L'algorithme est le suivant :

Algorithme.

- — Entrées : un carré magique C_1 de taille $n \times n$ et un carré magique C_2 de taille $m \times m$.
  — Sortie : un carré magique C de taille $(nm) \times (nm)$.
- Définis le carré C_{3a} en retirant 1 à tous les éléments de C_2. (Utilise la commande `addition_carre(carre2,-1)`.)
- Définis le carré C_{3b} comme l'homothétie du carré C_{3a} de rapport n. (Utilise la commande `homothetie(carre3a,n)`.)
- Définis le carré C_{3c} en multipliant tous les termes du carré C_{3b} par n^2. (Utilise la commande `multiplication_carre(carre3b,n**2)`.)
- Définis le carré C_{3d} en ajoutant par bloc le carré C_1 au carré C_{3c}. (Utilise la commande `addition_bloc_carre(carre3c,carre1)`.)
- Renvoie le carré C_{3d}.

- Implémente cet algorithme.
- Teste-le sur des exemples, en vérifiant que le carré obtenu est bien un carré magique.
- Construis un carré magique de taille 36×36 !
- Vérifie aussi que l'ordre du produit est important ($C_1 \times C_2$ n'est pas le même carré que $C_2 \times C_1$).