TRAVAUX PRATIQUES n° 7
Constructions de listes

I. Modifier une liste

1) Concaténation de listes

On peut « coller » deux listes l'une après l'autre pour former une liste plus grande : on appelle cela **la concaténation**. La syntaxe est la suivante :

```
nom_liste_1 + nom_liste_2
```

Testez les exemples suivants **dans le shell** en vérifiant après chaque ligne ce qui se trouve dans la liste (en tapant son nom) et **n'hésitez pas à modifier ces exemples pour voir les effets des modifications !** :

```
>>> liste_1 = [1, 3, 5]
>>> liste_2 = [2, 4, 6, 8]
>>> liste_1 + liste_2
>>> liste_3 = liste_1 + liste_2
>>> liste_4 = 2 * liste_1 + 3 * liste_2 + liste_3
```

2) Méthodes sur les listes

Les listes sont un type de données particulier : il existe des fonctions un peu spéciales, appelées méthodes, qu'on ne peut appliquer qu'aux listes. La différence entre une **fonction** et une **méthode** se situe dans la syntaxe :

- ⋆ Application d'une fonction à une liste :

```
nom_fonction(nom_liste)
```

- ⋆ Application d'une méthode à une liste :

```
nom_liste.nom_methode(arg_1, arg_2, ..., arg_n)
```

La liste des méthodes à connaître se trouvent à la fin de ce TP.

Testez les exemples suivants **dans le shell** en vérifiant après chaque ligne ce qui se trouve dans la liste (en tapant son nom) et **n'hésitez pas à modifier ces exemples pour voir les effets des modifications !** :

```
>>> liste = []                      # Création d'une liste vide
>>> liste.append(5)                 # Ajout de la valeur 5 à la fin de la liste
>>> liste.append(-3)                # Ajout de la valeur -3 à la fin de la liste
>>> liste.append(12)                # Ajout de la valeur 12 à la fin de la liste
>>> liste.insert(1, 12)             # Insertion de la valeur 12 en indice 1
>>> liste.insert(2, 17)             # Insertion de la valeur 17 en indice 2
>>> liste.insert(len(liste), 40)    # Insertion de la valeur 40 à la fin de la liste
>>> x = liste.pop(2)                # Suppression de l'élément d'indice 2
                                    # et récupération de sa valeur dans la variable x
>>> liste.pop()                     # Suppression du dernier élément de la liste
>>> liste.remove(12)                # Suppression du premier élément qui vaut 12
>>> liste.reverse()                 # Inversion de l'ordre des éléments
>>> liste.sort()                    # Trie dans l'ordre croissant
>>> liste.sort(reverse=True)        # Trie dans l'ordre décroissant
>>> liste.index(5)                  # Renvoie l'indice de l'élément qui vaut 5
                                    # ValueError si cet élément n'existe pas
>>> liste.count(5)                  # Renvoie le nombre d'éléments qui valent 5
```

II. Exercices

Sauf mention contraire, il est interdit d'utiliser les syntaxes suivantes :

- ⋆ `sum(liste)`
- ⋆ `min(liste)`
- ⋆ `liste.index(element)`
- ⋆ `liste.sort()`
- ⋆ `max(liste)`
- ⋆ `element in liste`
- ⋆ `liste.count(element)`

Exercice 1 (*Construire une liste de nombres aléatoires*) : Écrire une fonction `liste_aleatoire(n, a, b)` renvoyant une liste de `n` nombres choisis au hasard entre `a` et `b`.

On pourra utiliser la fonction `randint(a, b)` *du module* `random` *que l'on importera grâce à l'instruction* `from random import randint`.

Remarque : cette fonction peut être très utile pour tester vos prochaines fonctions sur des données variées afin de détecter d'éventuelles erreurs.

Exercice 2 (*Tester*) : Voici quatre fonctions dont le but est de déterminer si une liste est constituée uniquement de nombres positifs ou nuls ou non :

```
1  def tous_positifs_1(liste):
2      for i in range(len(liste)):
3          if liste[i] >= 0:
4              return True
5          else:
6              return False
7
8  def tous_positifs_2(liste):
9      for i in range(len(liste)):
10         if liste[i] >= 0:
11             return True
12     return False
13
14 def tous_positifs_3(liste):
15     for i in range(len(liste)):
16         if liste[i] < 0:
17             return False
18         else:
19             return True
20
21
```

```python
def tous_positifs_4(liste):
    for i in range(len(liste)):
        if liste[i] < 0:
            return False
    return True
```

1) Parmi ces fonctions, lesquelles sont correctes ? Pour celles qui ne le sont pas, donner un exemple de liste pour laquelle ces fonction renvoient un résultat incorrect.
2) Écrire une fonction `est_premier(n)` qui renvoie `True` si `n` est un nombre premier et `False` sinon.
3) Écrire une fonction `contient_pair(liste)` qui renvoie `True` s'il existe un élément de `liste` qui est un entier pair et `False` sinon.
4) Écrire une fonction `est_croissante(liste)` qui renvoie `True` si les éléments de `liste` sont dans l'ordre croissant et `False` sinon.

Exercice 3 (*Compter*) : Écrire une fonction `compte(liste, e)` qui calcule et renvoie le nombre de fois où l'élément `e` est présent dans la `liste`.

Exercice 4 (*Maximum et minimum*) :

1) Écrire une fonction `maximum(liste)` renvoyant le maximum d'une liste de nombres.
2) Écrire une fonction `indices_maximum(liste)` renvoyant la liste de tous les indices des éléments égaux au maximum de la liste.
3) Écrire une fonction `deuxieme_maximum(liste)` renvoyant le deuxième maximum d'une liste de nombres distincts de taille au moins 2, c'est-à-dire l'élément de la liste tel qu'il existe un unique autre élément plus grand que lui.

Exercice 5 (*Médiane*) : Écrire une fonction `mediane(liste)` renvoyant la médiane d'une liste de nombres.

On pourra utiliser la méthode `sort` *pour trier la liste et séparer les cas en fonctions de la parité du nombre d'éléments de la liste.*

Exercice 6 (*Équation de Pell-Fermat*) : On considère l'équation suivante :

$$n^2 - 3m^2 = 1$$

d'inconnue $(n, m) \in \mathbb{N}^2$.

Écrire une fonction `pell_fermat(N)` qui renvoie la liste des couples de solutions (n, m) tels que $0 \leqslant n \leqslant N$ et $0 \leqslant m \leqslant N$.

Exercice 7 (*Simulation*) : On considère l'expérience suivante :

> Soit $n \in \mathbb{N}^*$. On dispose d'un paquet de n cartes numérotées de 1 à n et rangées dans l'ordre croissant (la carte numéro 1 tout en bas de paquet). A chaque étape, on prend la carte du dessus du paquet et on la met en dessous du paquet, puis on prend une nouvelle carte au dessus du paquet que l'on met de côté. On recommence cette opération jusqu'à ce qu'il ne reste plus qu'une seule carte dans le paquet initial.

Vérifier, à l'aide d'un programme Python que si $n = 2023$, alors la carte restante est la carte numéro 25.

Vous ne devez pas essayer de résoudre ce problème mathématiquement ou de manière logique : il faut le résoudre informatiquement, c'est-à-dire qu'il faut demander à Python de faire l'expérience à votre place.

Exercice 8 (*Triangle de Pascal*) : Écrire une fonction `ligne_binome(n)` renvoyant la liste des $\binom{n}{k}$ pour $k \in [\![0, n]\!]$ en implémentant l'algorithme suivant :

- ⋆ Initialiser une liste `L = [1]`.
- ⋆ Pour `k` allant de `1` à `n` (inclus) :
  - Créer une liste `nouveau_L = [1]`.
  - Pour `i` allant de `0` à `len(L)-2` (inclus) :
    - ▶ Ajouter à `nouveau_L` l'élément `L[i] + L[i+1]`.
  - Ajouter finalement l'élément `1` à `nouveau_L`.
  - Actualiser la liste `L` : `L = nouveau_L`.
- ⋆ Renvoyer `L`.

Attention aux différents indices des boucles qui sont donnés avec inclusion à chaque fois alors que ce n'est pas le cas d'une boucle `for` *en Python.*

Exécuter alors ensuite le script suivant :

```
for i in range(10):
    print(ligne_binome(i))
```

Exercice 9 (*Crible d'Eratosthène*) : Soit $N \geqslant 2$. Pour déterminer la liste de tous les nombres premiers plus petits ou égaux à N, une méthode efficace est donnée par le crible d'Eratosthène. L'algorithme du crible d'Eratosthène est le suivant :

- ⋆ Initialiser deux listes :
  - une liste vide appelée `premiers`
  - une liste de taille `N+1`, appelée `est_premier`, dont les deux premiers éléments sont `False` et tous les autres sont `True`
- ⋆ Pour `i` allant de `0` à `N` (inclus) :

  Si `est_premier[i]` est égal à `True`, alors
  - on initialise une variable `k = 2`
  - tant que `k * i <= N` :
    - ▶ on modifie `est_premier[k * i] = False`
    - ▶ on augmente `k` de `1`
  - on ajoute `i` à la liste `premiers`
- ⋆ Enfin, on renvoie la liste `premiers`

Écrire une fonction `eratosthene(N)` prenant en argument un entier $N \geqslant 2$ et renvoyant la liste de tous les nombres premiers plus petits ou égaux à N à partir de l'algorithme d'Eratosthène.

III. Résumé des syntaxes

⋆ **Définition d'une liste :**

- `nom_liste = [e_1, e_2, ..., e_n]`
- `nom_liste = nombre * [valeur]`
- `nom_liste = [f(k) for k in range(a, b)]`
- `nom_liste = [f(k) for k in autre_liste]`

⋆ **Accès à un élément :** `nom_liste[indice]`

⋆ **Modification d'un élément :** `nom_liste[indice] = nouvelle_valeur`

⋆ **Fonctions sur les listes :**

- **Taille / longueur d'une liste :** `len(nom_liste)`
- **Somme d'une liste numérique :** `sum(nom_liste)`
- **Maximum d'une liste numérique :** `max(nom_liste)`
- **Minimum d'une liste numérique :** `min(nom_liste)`

⋆ **Parcours d'une liste :**

- **Parcours des éléments :**

```
for e in nom_liste:
    Instructions # e = élément de nom_liste
```

- **Parcours des indices :**

```
for i in range(len(nom_liste)):
    Instructions # i = indice de l'élément nom_liste[i] de nom_liste
```

⋆ **Concaténation de listes :** `nom_liste_1 + nom_liste_2`

⋆ **Méthodes sur les listes :**

Méthode	Syntaxe	Effet
`append`*	`liste.append(valeur)`	Ajoute l'élément `valeur` à la fin de `liste`
`insert`*	`liste.insert(i,valeur)`	Ajoute `valeur` en indice `i` dans `liste`
`pop`*	`liste.pop(i)`	Supprime et renvoie l'élément d'indice `i` dans `liste`
`remove`*	`liste.remove(valeur)`	Supprime le premier élément qui vaut `valeur` dans `liste`
`reverse`*	`liste.reverse()`	Inverse l'ordre des éléments dans `liste`
`sort`*	`liste.sort()`	Trie `liste` par ordre croissant (si cela a un sens)
`index`	`liste.index(valeur)`	Renvoie l'indice du premier élément qui vaut `valeur` dans `liste`
`count`	`liste.count(valeur)`	Renvoie le nombre d'éléments valant `valeur` dans `liste`

Les méthodes avec une étoile **modifient** la liste sur laquelle elles s'appliquent : ce sont des fonctions à effet de bord. Les autres ne la modifient pas.