Les listes Python

Les bases des listes

Exercice 1 : Tester les réponses touvées dans l'exercice 1 de la feuille papier.

Génération des listes

Exercice 2 : Tester les réponses touvées dans l'exercice 2 de la feuille papier.

Parcours de listes

Les listes sont des itérables et il est donc possible de les parcourir très simplement :

```
>>> for v in [2, 5, 7, 9, 10]:
        print(v)

2
5
7
9
10
```

Exercice 3 : Tester la fonction trouvée dans l'exercice 3 de la feuille papier.

Exercice 4 : Tester la fonction trouvée dans l'exercice 4 de la feuille papier.

Les listes peuvent aussi être parcourues indice par indice :

```
>>> liste = [4, 1, 3]
>>> for i in range(len(liste)):
        print(i, liste[i])

0, 4
1, 1
2, 3
```

Exercice 5 : Tester la fonction trouvée dans l'exercice 5 de la feuille papier.

Exercice 6 : Tester la fonction trouvée dans l'exercice 6 de la feuille papier.

Exercice 7 : Tester la fonction trouvée dans l'exercice 7 de la feuille papier.

Exercice 8 : Tester la fonction trouvée dans l'exercice 8 de la feuille papier.

Exercice 9 : Écrire une fonction `position_max(liste)` qui renvoie l'indice du maximum des valeurs de `liste`. S'il apparaît plusieurs fois, c'est la position de la première occurence qui est renvoyée.

```
>>> position_max([5, 1, 8, 14, 2])
3
>>> position_max([15])
0
```

```
>>> position_max([-9, -27, -2, -48])
2
>>> position_max([9, 2, -57, 1, 8])
0
```

Parcourir plusieurs listes en même temps

Exercice 10 : Compléter la fonction `addition(liste1, liste2)` qui prend deux listes de même taille et qui renvoie une nouvelle liste où à chaque indice se trouve la somme des éléments de `liste1` et de `liste2` de même indice. L'assertion en début de la fonction permet de garantir que les deux listes sont bien de même taille.

```
def addition(liste1, liste2):
    assert len(liste1) == len(liste2)
    resultat = ...
    for ... in ...:
        resultat.append(...)
    return ...
```

```
>>> addition([1, 2], [10, 30])
[11, 32]
>>> addition([6], [1])
[7]
>>> addition([9, 4, 3, 7, 1, 2, 7], [3, 5, 1, 7, 8, 2, 0])
[12, 9, 4, 14, 9, 4, 7]
```

`liste1:`	`[9, 4, 3, 7, 1, 2, 7]`
`liste2:`	`[3, 5, 1, 7, 8, 2, 0]`
`addition(liste1, liste2):`	`[12, 9, 4, 14, 9, 4, 7]`

Exercice 11 : Écrire une fonction `alterner(liste1, liste2)` qui renvoie une nouvelle liste où les éléments de `liste1` et `liste2` sont alternés : le premier de `liste1`, puis celui de `liste2`, le deuxième de `liste1`, celui de `liste2` et ainsi de suite.

```
>>> alterner([1, 2, 3], [4, 5, 6])
[1, 4, 2, 5, 3, 6]
>>> alterner([9, 5, 1, 7], [-6, -2, -5, -9])
[9, -6, 5, -2, 1, -5, 7, -9]
```

Exercice 12 : Écrire une fonction `egales(liste1, liste2)` qui renvoie un booléen indiquant si les deux listes sont égales. C'est-à-dire qu'elles contiennent les mêmes éléments dans le même ordre. Vous pouvez tester l'égalité des longueurs et des éléments, mais vous ne devez pas écrire `liste1 == liste2`.

```
>>> egales([], [])
True
>>> egales([], [7])
False
>>> egales([1, 2], [2, 1])
False
>>> egales([1, 2], [1, 2])
True
>>> egales([5, 9, 3], [5, 9])
False
```

Exercice 13 : Écrire une fonction `selection(liste1, liste2, liste3)` qui prend 3 listes de même longueur et qui renvoie une liste telle que pour chaque indice `i`, l'élément d'indice `i` est `liste1[i]` si `liste3[i]` > 0, `liste2` si `liste3[i]` < 0 et `0` sinon.

```
>>> selection([4, 5, 6], [1, 2, 3], [8, 7, 6])
[4, 5, 6]
>>> selection([4, 5, 6], [1, 2, 3], [-8, -7, -6])
[1, 2, 3]
>>> selection([4, 5, 6], [1, 2, 3], [2, -8, 0])
[4, 2, 0]
>>> selection([4, 5, 6, -9, -1], [1, -2, 3, -8, -7], [4, -8, 1, 0, -11])
[4, -2, 6, 0, -7]
```

Pour aller plus loin

Exercice 14 : Écrire une fonction `liste_speciale1(n)` qui renvoie la liste :

`[1,1, 1,2, 1,3, 1,4, 1,5, 1,..., 1,n]`

Les éléments ont été regroupés pour aider à trouver la façon de construire cette liste.

Exercice 15 : Écrire une fonction `liste_speciale2(n)` qui renvoie la liste :

`[1, 1,2, 1,2,3, 1,2,3,4, 1,..., 1,2,3,...,n]`

Les éléments ont été regroupés pour aider à trouver la façon de construire cette liste.

Exercice 16 : Écrire une fonction `premiers_premiers(n)` qui renvoie la liste des nombres premiers inférieurs à `n`.
Combien est-ce qu'il y a de nombres premiers inférieurs à 100 000 ?