TP 5 Python – Manipuler des listes

Dans des deux derniers TPs, nous avons codé "à la main" un certain nombre de fonctions sur les listes : créer une liste de taille fixée remplie de zéros, connaître la longueur d'une liste ou son maximum, concaténer deux listes.
L'objectif de ce TP est d'apprendre à manipuler les listes en Python de manière **efficace**, à l'aide des opérations, fonctions et méthodes déjà existantes dans le langage.

1. Longueur d'une liste

Définition

La **longueur** d'une liste est le nombre d'éléments de cette liste.
En Python, la *fonction* `len` prend en paramètre une liste et renvoie sa **longueur**.

Exemple

```
>>> len([-1, 3, 0, 0, 3])
5
>>> len([])
0
>>> len([0, [1, 2], [3, 4, 5, 6], 7]
4
>>> ma_liste
[0, -1, "test", 2, "blah"]
>>> len(ma_liste)
5
>>> ma_liste[5]
Traceback (most recent call last):
  File "<console>", line 1, in <module>
IndexError: list index out of range
```

⚠ `ma_liste` est de longueur 5, mais ses éléments sont numérotés de 0 à 4 !
Ainsi, l'instruction `L[len(L)]` renverra systématiquement une erreur "list index out of range".

À retenir : Pour **parcourir une liste**, on a globalement deux options :

- `for elem in L :` Avec cette instruction, la variable `elem` prend directement les **valeurs** de la liste, mais on n'a pas moyen d'accéder facilement à leur position dans la liste.
- `for i in range(len(L)) :` Avec cette instruction, la variable `i` correspond aux **indices** (positions) dans la liste, et pour accéder aux valeurs associées, il suffit d'écrire `L[i]`.

Exercice 1. Les fonctions de cet exercice sont à connaître parfaitement.

1. Compléter les fonctions suivantes, qui prennent en argument une liste (non vide), et qui doivent renvoyer respectivement la **valeur** du maximum et la **position** du maximum.

```
def valeur_maximum(L):
    max = L[0]
    for ..................:
        if ...... > max :
            max = .....
    return max
```

```
def position_maximum(L):
    pos_max = 0
    for ..................:
        if ...... > L[pos_max] :
            pos_max = ....
    return pos_max
```

2. Écrire sur le même modèle deux fonctions qui renvoient respectivement la valeur et la position de l'élément minimal d'une liste.

2. Concaténations et ajout d'éléments

Définition

La **concaténation** de deux liste L_1 et L_2 est la liste contenant les éléments de L_1 puis ceux de L_2. En Python, on écrit simplement `L1 + L2`.

Exemple

- Avec `liste1 = [1, 2]` et `liste2 = [6, 5]`, alors `liste1 + liste2` vaut `[1, 2, 6, 5]`.
- `liste2 + liste1` est-il égal à `liste1 + liste2` ?

Exercice 2. On considère les listes $L1$ et $L2$ données par `L1 = [0, "bonjour", 3]` et `L2 = [7.5, 1]`. Prédire puis vérifier ce que contiennent les listes suivantes :

a) `L1 + L2` b) `L1 + [L2]` c) `L1 + 0` d) `L1 + [0]` e) `L2 + L2 + L2`

On peut également concaténer une liste avec elle-même plusieurs fois de manière directe, avec l'instruction `L * k`, ou `k * L`, qui crée la liste `L + L + ... + L` (k fois).

Exemple

```
>>> [1, 4, 2] * 5
[1, 4, 2, 1, 4, 2, 1, 4, 2, 1, 4, 2, 1, 4, 2]
>>> [] * 25
[]
>>> [True] * 12
[True, True, True, True, True, True, True, True, True, True, True, True]
```

À retenir : La plupart du temps, cette instruction est utilisée pour *initialiser* une liste dans un programme, en écrivant `ma_liste = [0] * n`, pour créer une liste de longueur n, qu'on modifiera par la suite pour qu'elle contienne les valeurs qui nous intéressent.

Remarque. Attention, cette opération n'a de sens qu'entre *une liste et un entier*. On ne peut pas écrire `L1*L2` où $L1$ et $L2$ sont deux listes.

Exercice 3. Créer, avec les opérations $+$ et $*$, une liste `nb_chiffres` de taille 1500 telle que, pour tout i entre 0 et 1499, `nb_chiffres[i]` indique le nombre de chiffres dans l'écriture décimale de i.

Définition

La méthode `append` en Python permet d'ajouter un élément à la fin d'une liste existante, à l'aide de l'instruction : `L.append(elem)`.
Cette instruction seule est *strictement équivalente* à `L = L + [elem]` !

Remarque. ⚠ La méthode `append` constitue donc en soi une affectation de variable : elle modifie le contenu de la liste. Ainsi, on prendra garde à ne **jamais** écrire d'instruction du type `L = L.append(elem)`, ou encore `return L.append(elem)` : ceci donnera toujours une erreur !

Exemple

```
>>> ma_liste = [0, -1, "test"]
>>> ma_liste.append(2)
>>> ma_liste
[0, -1, "test", 2]
>>> ma_liste.append("blah")
>>> ma_liste
[0, -1, "test", 2, "blah"]
```

```
1 nouvelle_liste = []
2 for i in range(7):
3     if i%2 == 0:
4         nouvelle_liste.append("oui")
5     else:
6         nouvelle_liste.append("non")
```

Prédire puis vérifier : que contient la liste `nouvelle_liste` à la fin du programme de droite ?

Exercice 4.
1. Écrire une fonction Python `zéros(k)`, qui prend en paramètre un entier naturel k, et qui renvoie la liste contenant exactement k fois l'élément 0. On demande de ne pas utiliser les opérations `+` et `*`, mais d'utiliser la méthode `append` à la place.
2. Écrire une fonction Python `concatène(L1, L2)`, qui prend en paramètre deux listes, et qui renvoie la concaténation de $L1$ et $L2$. On demande de ne pas utiliser l'opération `+`, et d'utiliser la méthode `append` à la place.

3. Création d'une liste par compréhension

Une manière souvent efficace de créer une liste est de le faire à partir d'une autre liste ou d'un autre *itérable* (par exemple un `range`, une chaîne de caractères, …).
La syntaxe est alors la suivante : `L = [f(i) for i in itérable]`.

Exemple

- Pour créer la liste des 20 premiers multiples de 3, je peux écrire :
  `multiples = [3*i for i in range(20)]`.
- Pour créer une liste de 42 zéros, une autre option est donc d'écrire :
  `Liste0 = [0 for i in range(42)]`
- ```
  >>> liste_lettres = [lettre for lettre in "bonjour"]
  >>> liste_lettres
  ['b', 'o', 'n', 'j', 'o', 'u', 'r']
  >>> liste_test = [[1]*i for i in range(6)]
  >>> liste_test
  [[], [1], [1, 1], [1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1, 1]]
  ```

Exercice 5.
1. Écrire une instruction (en une seule ligne !) qui crée la liste des 31 premiers carrés parfaits.
2. Écrire une instruction qui crée la liste de tous les multiples de 3 entre 15 et 99.
3. Écrire une fonction Python `premiers_éléments(k, L)`, qui prend en paramètre un entier k et une liste L (supposée assez longue), et qui renvoie une liste contenant les k premiers éléments de la liste L.
   Le programme écrit doit faire *2 ou 3 lignes*.

4. Extraction d'une sous-liste, ou "slicing"

Pour rappel, on peut accéder à l'élément d'indice i d'une liste L grâce à l'instruction `L[i]`. Cette instruction peut aussi permettre de modifier l'élément d'indice i.
Remarque : on peut également indexer par des nombres négatifs : `L[-1]` correspond au dernier élément, puis `L[-2]` à l'avant dernier, etc. Ainsi, tout élément est numéroté de deux façons, l'une avec un nombre positif, l'autre avec un nombre négatif.

Exemple

```
>>> liste_lettres
['b', 'o', 'n', 'j', 'o', 'u', 'r']
>>> liste_lettres[3]
'j'
>>> liste_lettres[-1]
'r'
>>> liste_lettres[-5]
'n'
```

Donner les deux manières d'accéder à l'élément `'b'` avec cette syntaxe.
À votre avis, que renvoie `liste_lettres[-8]` ? Vérifiez.
```

Définition

On peut accéder à une **sous-liste**, extraite d'une liste donnée, avec le principe de **slicing** (ou *extraction*, ou *saucissonnage*…). La syntaxe en Python est la suivante : `sous_liste = liste[n:m:p]`
La nouvelle liste `sous_liste` contient alors tous les éléments de `liste` dont les indices sont compris entre `n` (inclus) et `m` **exclu**, avec un pas (= écart) de `p`.

Exemple

```
>>> liste = [8 + i for i in range(9)]
>>> liste
[8, 9, 10, 11, 12, 13, 14, 15, 16]
>>> liste[2:5]
[10, 11, 12]
>>> liste[1:7:2]
[9, 11, 13]
>>> liste[3:-2]
[10, 11, 12, 13, 14]
>>> liste[3:0]
[]
>>> liste[5:2:-1]
[13, 12, 11]
```

À votre avis, que renverra `liste[2:1000:3]` ? Vérifiez !

5. C'est tout ?

D'autres fonctions existent dans Python pour manipuler des listes. Cependant, il ne vous est demandé d'en connaître aucune ! Lorsque vous serez amenés à en utiliser (par exemple dans les exercices qui suivent, mais aussi dans un sujet de concours), elles seront *nécessairement rappelées dans l'énoncé* !

6. Exercices bilan

Exercice 6. Prévoir ce qu'affiche chacun des programmes suivant, puis vérifier votre prédiction :

a)

```
L = [1, 2, 3, "coucou", 4, 5, 6, "fin"]
print(L[3:8])
```

b)

```
liste = [[4, 5, 6], [1, 10, 14], [5, 19, 23]]
a = []
for i in liste:
    for j in i:
        a = a + [j]
print(a)
```

c)

```
Liste = []
A = ["test"]
B = [1, 3, -7, 2, 8, 10]
for i in range(len(A)):
    Liste.append(B[i])
print(Liste)
```

Exercice 7. Écrire une fonction `éléments_pairs(L)`, qui prend en paramètre une liste d'entiers, et qui renvoie une liste contenant uniquement les éléments pairs de la liste.
Par exemple, `éléments_pairs([3, 2, 14, -2, 5, 2, 7, 9])` devra renvoyer `[2, 14, -2, 2]`, et `éléments_pairs([1, 5, 3, 11])` devra renvoyer `[]`.

Exercice 8. On considère la suite (u_n) définie par $u_0 = -1$, et pour tout $n \in \mathbb{N}$, $u_{n+1} = nu_n - 3$. Écrire une fonction Python `premières_valeurs(n)` qui prend en argument un rang $n \in \mathbb{N}$, et qui renvoie la liste contenant toutes les valeurs de la suite de u_0 à u_n.

Exercice 9.

1. Écrire une fonction `position_minimum(L)`, qui prend en paramètre une liste de nombres, et qui renvoie son plus petit élément.
2. La méthode `pop` permet de supprimer un élément d'une liste, avec l'instruction : `L.pop(i)`.
   Elle renvoie l'élément `L[i]`, et elle supprime directement cet élément de la liste. Par exemple :

   ```
   >>> liste_ex = [1, 2, 3, 4, 5, 6, 7]
   >>> liste_ex.pop(4)
   5
   >>> liste_ex
   [1, 2, 3, 4, 6, 7]
   ```

   Écrire une fonction Python `ajoute_minimum(L1, L2)` qui prend en argument deux listes L1 et L2, et qui renvoie les deux listes, où l'élément minimal de L1 a été retiré de la liste L1 et ajouté à la fin de la liste L2.
3. En déduire une fonction `tri_selection(L)` qui prend en paramètre une liste L, et qui renvoie une liste contenant les éléments de L triés dans l'ordre croissant.

Exercice 10. Écrire une fonction `palindrome(L)` qui prend en argument une liste, et qui vérifie si cette liste est un palindrome, c'est-à-dire si ses éléments sont les mêmes si on les lit dans l'autre sens.
Par exemple, la liste `[-3, "ok", 2, "ok", -3]` est un palindrome, et la liste `[-1,0, 5, 5, 3, -1]` n'en est pas un.
Votre fonction devra renvoyer une réponse sous forme d'un booléen.

Exercice 11. Écrire une fonction `découpe(L)`, qui prend en paramètre une liste L, et qui la découpe en deux liste de taille égale (à 1 près, dans le cas des listes de longueur impaire).

Exercice 12 (**)**.** Écrire une fonction `fusion(L1, L2)`, qui prend en paramètre deux listes, qu'on suppose triées dans l'ordre croissant, et qui les fusionne : elle doit renvoyer une liste contenant tous les éléments de `L1` et de `L2`, triés dans l'ordre croissant.

Exercice 13 (**)**.** *Pour aller plus loin :* On peut utiliser les deux exercices précédents pour créer un algorithme de tri, appelé le *tri fusion*. Essayez de *décrire* un tel algorithme, qui prend en entrée une liste non triée, et qui renvoie une liste contenant les mêmes éléments, triés dans l'ordre croissant.