TP 6 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. Concaténation et variantes

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, l'opération de concaténation est simplement le « `+` »

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 1. Soient $L1$ et $L2$ deux listes données par `L1 = [0, "bonjour", 3]` et `L2 = [7.5, 1]`. Pour chacune des listes suivantes, dire quelle opération on a écrit pour la créer à partir $L1$ et $L2$.

a) `[0, "bonjour", 3, 7.5, 1]`
b) `[0, "bonjour", 3, [7.5, 1]]`
c) `[[0, "bonjour", 3], [7.5, 1]]`
d) `[1, 7.5, 0, "bonjour", 3]`
e) `[[0, "bonjour", 3, 7.5, 1]]`
f) `[0, "bonjour", 3, 2, 7.5, 1]`

On peut également concaténer une liste avec elle-même plusieurs fois de manière directe, avec l'instruction `L*k`, 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. L'instruction `k*L` fonctionne aussi, et renvoie la même valeur que `k*L`.
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 2. Créer une liste de taille 1500, appelée `nombre_de_chiffres`, telle que, pour tout i entre 0 et 1499, `nombre_de_chiffres[i]` indique le nombre de chiffres dans l'écriture décimale de i.

La méthode `append` : La méthode `append` 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]`. Elle modifie **directement** la liste L ! On prendra garde à ne **jamais** écrire de formule du type : `L = L.append(elem)`, qui renvoie 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")
```

Que contient la liste `nouvelle_liste` à la fin du programme de droite ?

Exercice 3.

1. 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.
2. 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. Longueur d'une liste

Python dispose d'une fonction `len`, qui prend en paramètre une liste (ou un autre itérable, comme une chaîne de caractère… nous verrons ça plus tard), et renvoie la **longueur** de la liste.

Exemple

```
>>> 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[4]
"blah"
```

⚠ `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".

3. Création d'une liste par compréhension

Une manière souvent efficace de créer une liste est de le faire 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 4.

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 entiers impairs entre 15 et 99.
3. Écrire une fonction Python `premiers(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 écrire doit faire *exactement 2 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[0]
'b'
>>> liste_lettres[-1]
'r'
>>> liste_lettres[-5]
'n'
```

À votre avis, que renvoie `liste_lettres[-8]` ? Vérifiez.

On peut accéder à une **sous-liste**, extraite d'une liste donnée, avec le principe de **slicing** (ou *extraction*, ou parfois *saucissonnage*...), dont la syntaxe est la suivante : `sous_liste = liste[début:fin:pas]`
`sous_liste` contient alors tous les éléments de `liste` dont les indices sont compris entre `début` (inclus) et `fin` (**exclu !**), avec un pas de `pas` (le pas correspond à l'écart entre deux indices successifs).

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 5. Prévoir la valeur de toutes les variables à la fin de chacun des programmes suivant :
TODO : todo…

Exercice 6. 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 7. 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 8. 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.
Pour aller plus loin : On peut utiliser cet exercice et l'exercice précédent 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.

Exercice 9.

1. Écrire une fonction `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 `tri_insertion(L)` qui prend en paramètre une liste L, et qui renvoie une liste contenant les éléments de L triés dans l'ordre croissant. On utilisera la fonction `minimum`, ainsi que les méthodes `pop` et `append`.