Pascal ORTIZ

Les listes en compréhension

Version du 9 août 2019

Table des matières

Listes en compréhension

Étant donné une liste d'entiers telle que `t=[5, 2, 0, 3, 7, 10]`, on cherche à construire la liste `L` dont les éléments sont 10 fois les éléments de `t`, c'est-à-dire `[50, 20, 0, 30, 70, 100]`.

Voici un code Python répondant au problème :

```
1 t = [5, 2, 0, 3, 7, 10]
2 print(t)
3 L = [10 * x for x in t]
4 print(L)
```

```
5 [5, 2, 0, 3, 7, 10]
6 [50, 20, 0, 30, 70, 100]
```

A la ligne 3, est définie une liste `L` dite *liste en compréhension* (le terme officiel de « compréhension de liste » n'est pas repris dans ce cours, pas plus que le terme utilisé parfois de *liste en intention* censé s'opposer à l'expression *liste en extension*).

Une liste en compréhension `L` a la syntaxe minimale suivante

```
[expr for x in t]
```

où

- la paire de crochets, les mots-clefs `for` et `in` sont obligatoires
- `t` est, assez souvent, une liste
- `x` est l'élément courant qui parcourt la liste `t` ; `x` est appelé *variable de contrôle* de la liste en compréhension
- `expr` est une expression qui dépend en général de `x` et dont la valeur est placée dans `L`

La liste en compréhension `L` avec la syntaxe ci-dessus a toujours même nombre d'éléments que la liste `L`.

Intérêt d'une liste en compréhension : générer une liste en une seule *expression* et non en une ou plusieurs *instructions*. L'intérêt des listes en compréhension est avant tout leur compacité d'édition dans le code.

Liste en compréhension et boucle `for`

Une liste en compréhension admet un équivalent créé avec une boucle `for` mais nécessitant un code plus long.

Soit la liste en compréhension `L` suivante :

```
1 t = [5, 2, 0, 3, 7, 10]
2 L = [10 * x for x in t]
3 print(L)
```

```
4 [50, 20, 0, 30, 70, 100]
```

Elle pourrait être obtenue avec une boucle `for` :

```
1 t = [5, 2, 0, 3, 7, 10]
2 L = []
3 for x in t:
4     L.append(10*x)
5 print(L)
```

```
6 [50, 20, 0, 30, 70, 100]
```

— La liste L est obtenue en trois lignes de code au lieu d'une.

Listes en compréhension à partir d'un itérable

Une liste en compréhension est souvent construite sur la base d'une liste mais on peut construire une liste en compréhension sur la base de n'importe quel itérable.

Par exemple, on peut construire une liste en compréhension à partir d'une chaîne

```
1 w = "SERPENT"
2 L = [2 * c for c in w]
3 print(L)
```

```
4 ['SS', 'EE', 'RR', 'PP', 'EE', 'NN', 'TT']
```

Listes en compréhension imbriquées

Des listes en compréhension peuvent être imbriquées. Néanmoins la liste n'est pas forcément créée dans l'ordre où on s'y attendrait.

Soit à créer une liste formée de tous « mots » commençant par une lettre A ou B et suivie d'un chiffre parmi 1, 2 ou 3. On peut utiliser une liste en compréhension :

```
1 L = [x+y for x in "AB" for y in "123"]
2 print(L)
```

```
3 ['A1', 'A2', 'A3', 'B1', 'B2', 'B3']
```

Le résultat ligne 3 montre que l'on fixe d'abord la lettre, c'est-à-dire les éléments du `for` lexicalement le plus interne (ici `for x`) puis que y varie. Ce n'est pas forcément très intuitif.

Le résultat est plus facilement compréhensible si la liste en compréhension est interprétée par deux boucles `for` imbriquées **exactement** dans l'ordre où on lit les apparitions des `for` dans la liste en compréhension, ce qui donne ici :

```
1 L = []
2 for x in "AB":
3     for y in "123":
4         L.append(x+y)
5 print(L)
```

```
6 ['A1', 'A2', 'A3', 'B1', 'B2', 'B3']
```

— la boucle `for x` apparaît avant la boucle `for y`, dans le même ordre que la liste en compréhension `[x+y for x in "AB" for y in "123"]`.

Création d'un tableau 2D initialisé

Une liste en compréhension est un moyen très simple de créer un conteneur pour un tableau ayant n lignes et p colonnes. Le code ci-dessous crée un tableau à 2 lignes et 3 colonnes initialisé avec des 0 :

```
L = [[0 for j in range(3)] for i in range(2)]
print(L)
L[1][2]=42
print(L)
```

```
[[0, 0, 0], [0, 0, 0]]
[[0, 0, 0], [0, 0, 42]]
```

— On crée une liste de 2 lignes, chaque ligne étant une liste de 3 éléments. L'indice `i` parcourt les lignes et l'indice `j` parcourt chaque colonne de chaque ligne.

Au passage, comme l'élément le plus interne dans les listes est immuable (il s'agit de l'entier 0), on peut même simplifier la syntaxe :

```
L = [[0]*3 for i in range(2)]
print(L)
L[1][2]=42
print(L)
```

```
[[0, 0, 0], [0, 0, 0]]
[[0, 0, 0], [0, 0, 42]]
```

Attention toutefois que le code suivant lui ne donne pas le résultat attendu :

```
L = [[0]*3]*2
print(L)
L[1][2]=42
print(L)
```

```
[[0, 0, 0], [0, 0, 0]]
[[0, 0, 42], [0, 0, 42]]
```

Voici l'équivalent du code ci-dessus sans utiliser de liste en compréhension et en utilisant deux boucles for imbriquées. On notera que le code est plus long :

```
L=[]
for i in range(2):
    lig = []
    for j in range(3):
        lig.append(0)
    L.append(lig)
print(L)
```

```
[[0, 0, 0], [0, 0, 0]]
```

Listes en compréhension et la clause `if`

Il existe une variante dans la syntaxe des listes de compréhension utilisant une clause `if`.

Par exemple, étant donné une liste `L` d'entiers, soit à construire la liste `t` obtenue en ne gardant que les entiers x de `L` tel que $x > 42$:

```
1 t = [65, 31, 9, 32, 81, 82, 46, 12]
2 L = [x for x in t if x >= 42]
3 print(L)
```

```
4 [65, 81, 82, 46]
```

— `x` varie dans `t` et est inséré dans la liste `L` en construction si $x \geq 42$.

L'équivalent avec une boucle `for` serait le suivant :

```
1 t = [65, 31, 9, 32, 81, 82, 46, 12]
2 L= []
3 for x in t:
4     if x >= 42:
5         L.append(x)
6 print(L)
```

Listes en compréhension imbriquées et clause `if`

Soit à placer dans une liste `L` tous les couples d'entiers (x, y) tel que

$$0 \leq x, y \leq 3 \text{ et } x + y \leq 3$$

Voici un code utilisant une liste en compréhension :

```
1 a = 3
2 L =[[x,y] for x in range(0,a+1) for y in range(0,a+1) if x+y <= a]
3 print(L)
```

```
4 [[0, 0], [0, 1], [0, 2], [0, 3], [1, 0], [1, 1], [1, 2], [2, 0], [2, 1], [3, 0]]
```

La syntaxe de la liste en compréhension a été prévue pour que la traduction depuis des boucles for imbriquées soit immédiate : on place les `for` et `if` dans l'ordre de gauche à droite où ils apparaissent dans la liste en compréhension :

```
1 a = 3
2 L =[]
3 for x in range(0,a+1):
4     for y in range(0,a+1):
5         if x+y <= a:
6             L.append([x,y])
7 print(L)
```

```
8 [[0, 0], [0, 1], [0, 2], [0, 3], [1, 0], [1, 1], [1, 2], [2, 0], [2, 1], [3, 0]]
```

Listes en compréhension et itérables

Soit L une liste d'entiers ; on cherche à calculer la somme S des carrés des éléments de L. L'usage d'une liste en compréhension se prête bien à la recherche de S :

```
1 L = [10, 16, 100, 9, 5]
2 M = [x**2 for x in L]
3 S = sum(M)
4 print(S)
```

```
5 10462
```

Avant de calculer S, on a stocké la liste des carrés des éléments de L. Mais en fait le calcul de S ne nécessite pas de stocker les éléments de M mais juste d'itérer sur ces éléments.

Il suffit donc de créer un itérable qui permette de parcourir les éléments de M sans avoir à stocker en permanence ses éléments. Un moyen simple est d'utiliser un *générateur* sous forme d'expression génératrice dont la syntaxe est proche des listes en compréhension :

```
1 L = [10, 16, 100, 9, 5]
2 it = (x**2 for x in L)
3 print(it)
4 S = sum(it)
5 print(S)
```

```
6 <generator object <genexpr> at 0xb728125c>
7 10462
```

— it est un itérateur, il permet d'itérer sur des éléments sans les stocker

— sum accepte tout itérable et donc en particulier un itérateur comme it.

Une liste en compréhension est d'abord une liste et donc une structure de données supposant un stockage en mémoire. En fonction de l'opération que l'on souhaite faire, il se peut que le stockage ne soit pas utile et se pose alors la question de la pertinence du choix d'une structure de liste.

Efficacité des listes en compréhension

Les listes en compréhension seraient plus rapides que l'équivalent avec une boucle `for`. Voici un exemple de comparaison de performances :

```
1 import time
2
3 N=10**7
4
5 # ===== Boucle for =================
6 start=time.clock()
7
8 L=[]
9 for i in range(N):
10     L.append(i**2)
11
```

```
12 t1=time.clock() - start
13 print("{:2f}".format(t1))
14
15 # ===== Liste en compréhension ================
16 start=time.clock()
17
18 L=[i**2 for i in range(N)]
19 t2=time.clock() - start
20 print("{:2f}".format(t2))
21 # ---------------------------------
22 t=(t1-t2)/t1*100
23 print("{:2f}".format(t))
```

```
24 5.140000
25 4.090000
26 20.428016
```

On constate une meilleure performance de 20

Toutefois, cette meilleure performance est à relativiser, cf. Using a List Comprehensions instead of loop in order to improve the performance

Portée des variables de contrôle d'une liste en compréhension

La variable de contrôle d'une boucle `for` est visible après la fin de la boucle `for` :

```
1 L=[]
2
3 for i in range(5):
4     L.append(5*i)
5 print(i)
```

```
6 4
```

— Ligne 5 : `i` est accessible après la fin de l'exécution de la boucle `for`.

Il n'en est pas de même d'une variable de contrôle d'une liste en compréhension :

```
1 L = [10 * i for i in range(5)]
2 print(i)
```

```
3 NameError: name 'i' is not defined
```

— Le nom `i` a une portée limitée à la liste en compréhension : à la ligne suivante, `i` est inconnu.