SPUS201 - UE SCIENCES : Introduction à la programmation 2

Denis Dubruel - Cours Magistral N°6

UNIVERSITÉ CÔTE D'AZUR

Année 2024/2025

courriel : prenom.nom@univ-cotedazur.fr

Table des Matières.

Table des matières

Listes en compréhension

Situation : à partir de t=[5, 2, 0, 3, 7, 10], construire la liste L dont les éléments sont 10 fois les éléments de t, c'est-à-dire obtenir la liste : [50, 20, 0, 30, 70, 100] ? (bien sûr faisable avec ce script :)

```
1 t=[5, 2, 0, 3, 7, 10]
2 L : list[int]=[]
3 for k in range(len(t)):
4     L = L+ [10*t[k]]
```

A la place on utilise une liste en compréhension sous la forme

$$\boxed{\text{L} = \textbf{[} \text{expr(x) } \textbf{for } \text{x } \textbf{in } \text{t } \textbf{]}}$$

les crochets [ ] et mots clefs *for* et *in* sont obligatoires.

expr est une expression qui dépend de x et renvoie sa valeur dans L

Listes en compréhension

L'exemple précédent se traduit par :

```
>>> t=[5, 2, 0, 3, 7, 10]
    L = [ 10 * x  for x in t]
>>> L
[50, 20, 0, 30, 70, 100]
```

Si la liste de nombre t est remplacée par une chaîne (itérable également) :

```
>>w='Chou'
  L = [ 2*x for x in w]
>>>L
['CC','hh','oo','uu']
```

Listes en compréhension

Il est possible d'ajouter un test dans les listes :

```
>>>t=[14,25,68,69,63]
   L=[x for x in t if x%2==0]
>>> L
[14, 68]
```

```
>>>[x for x in t if x>60]
[68,69,63]
```

Listes en compréhension

Liste imbriquées ...comme les boucles !!

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

réponse à suivre

Listes en compréhension

Liste imbriquées ...comme les boucles !!

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

Ce code renvoie L qui contient [[0, 0, 0], [0, 0, 0]].

Traduction en liste en compréhension ....

```
1 [[0 for j in range(3)] for i in range(2)]
```

Listes en compréhension

On veut l'ensemble des couples d'entier (x, y) dans une liste tels que :

$0 \leqslant x$ puis $y \leqslant 3$ et $x + y \leqslant 3$

Recherche rapide à la main (0,0) (0,1) (0,2) (0,3) (1,0) (1,1) etc.

$y \leqslant 3$ se traduit par `y in range(4)` puis

comme $x + y \leqslant 3$ et $0 \leqslant x$ donc $x \leqslant 3$

on traduit par `x in range(4)`

Listes en compréhension

L'ensemble des couples d'entier (x, y) dans une liste tels que :

$0 \leqslant x$ puis $y \leqslant 3$ et $x + y \leqslant 3$

```
1 L=[ (x,y) for x in range(4) for  y in range (4)   if x + y <=3 ]
```

Ou en code ...plus long !

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

Listes en compréhension

Conclusion : les listes en compréhension sont une manière plus courte en ligne de code, plus compréhensible par le lecteur et aussi plus rapide pour l'ordinateur.

A utiliser dans le TP7.

Table des matières

Slicing

slice(anglais) se traduit par tranche en français.

Il s'agit de découper des itérables.

Exemple, ce code extrait la chaîne correspondant au mois.

```
>>>date='14/03/2024'
   mois : str =""
   for k in range(3,5):  # k vaudra 3 puis 4, pas 5 !
       mois=mois+date[k]
>>> mois
'03'
```

A la place on utilise le slicing

```
date='14/03/2024' :  # k vaudra 3 puis 4, pas 5 !
mois = date[3:5]
```

Avantage : permet d'extraire simplement des sous-chaines ou sous-listes.

Slicing avec un pas de découpages.

Avec un pas de découpage (exemple extraction d'un caractère sur 2 depuis le rang 0 jusqu'au rang 7 exclus soit les indices 0 2 4 6)

```
1 >>> chaine='abcdefghik'
2     extrait=chaine[0:7:2]
3 >>>extrait
4 'aceg'
```

A retenir : sequence[début :fin :pas]

- *début* indice de début (inclus)
- *fin* indice de fin (exclus)
- *pas* intervalle entre les indices.

Rappel : ordre des indices comme pour la fonction range(start, stop, step)

Slicing avec indices négatifs

Regardons les indices de la séquence ou liste L ici :

```
1 L=['G','I','R','A','F','E']
```

Les indices positifs : par rapport au début.

Les indice négatifs : par rapport à la fin (sans le 0 car déja utilisé)

0	1	2	3	4	5
G	I	R	A	F	E
-6	-5	-4	-3	-2	-1

```
1 >>> L=['G','I','R','A','F','E']
2     raf=L[-4:-1]        # toujours avec [début:fin]   fin étant exclus.
3 >>> raf
4 ['R', 'A', 'F']
```

Slicing avec indices négatifs et un pas

0	1	2	3	4	5
G	I	R	A	F	E
-6	-5	-4	-3	-2	-1

```
1 >>> L=['G','I','R','A','F','E']
2     grf=L[ -6 : -1 : 2 ]  # toujours avec [début:fin:pas]
  fin étant exclus.
3 >>> grf
4 ['G', 'R', 'F']
```

Slicing avec omission d'un paramètre.

Possibilité d'omettre un ou plusieurs paramètres.

- si *début* omis : commence au début
- si *fin* omis : va jusqu'à la fin
- si *pas* omis , le itère de 1

```
>>> chiffres=['0','1','2','3','4','5','6','7','8','9']
>>> chiffres[4::2]  # chiffres pairs depuis 4 jusqu'à la fin
['4', '6', '8']

>>> chiffres[::2]   # depuis le début jusqu'à la fin de 2 en 2
['0', '2', '4', '6', '8']

>>> chiffres[::]  # tous les éléments
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

>>> chiffres[1:7] # indices de 1 à 6 avec un pas par défaut de 1.
['1', '2', '3', '4', '5', '6']
```

Slicing avec omission d'un paramètre.

sequence[début :fin :pas]

Possibilité d'omettre un ou plusieurs paramètres.

```
>>> chiffres[-6:-9:-1] # lecture inversée entre le -6 et le -9 (exclus!)
['4', '3', '2']

>>> chiffres[-6:-9:-2] # lecture inversée pas de 2 du -6 au -9 (exclus!)
['4', '2']

>>> chiffres[4::-2]  # lecture depuis l'indice 4 inclus jusqu'au début
['4', '2', '0']

>>> chiffres[::-2]   # depuis la fin jusqu'au début de 2 en 2
['9', '7', '5', '3', '1']

>>> chiffres[::-1]  # inversion de la séquence.
['9', '8', '7', '6', '5', '4', '3', '2', '1', '0']
```

Licence

Ce document est publié sous licence Creative Commons :

- ©2024 — Denis Dubruel - Université Côte d'Azur
- Attribution
- Utilisation non commerciale
- Partage dans les mêmes conditions 4.0 International

https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.fr