Listes en compréhension

NSI1

6 janvier 2022

Principe

Jusqu'à présent

Jusqu'à présent

Pour construire des listes on fait souvent ceci :

Jusqu'à présent

Pour construire des listes on fait souvent ceci :

- on crée une liste `lst` vide ;

Jusqu'à présent

Pour construire des listes on fait souvent ceci :

- on crée une liste `lst` vide ;
- on construit une boucle `for` ou `while` ;

Jusqu'à présent

Pour construire des listes on fait souvent ceci :

- on crée une liste `lst` vide ;
- on construit une boucle `for` ou `while` ;
- on peuple la liste avec `lst.append` .

Exemple

Exemple

```
lst = []
for i in range(10):
    lst.append(i*i)
print(lst)
```

Exemple

```
lst = []
for i in range(10):
    lst.append(i*i)
print(lst)
```

Ce programme affiche :

Exemple

```
lst = []
for i in range(10):
    lst.append(i*i)
print(lst)
```

Ce programme affiche :

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Exemple

```
lst = []
for i in range(10):
    lst.append(i*i)
print(lst)
```

Ce programme affiche :

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

C'est la liste des carrés des 10 premiers entiers naturels.

L’écriture en compréhension

L’écriture en compréhension

En mathématiques, l’ensemble des carrés des 10 premiers entiers naturels se note

L'écriture en compréhension

En mathématiques, l'ensemble des carrés des 10 premiers entiers naturels se note

$$\left\{i^2 \mid i \in \mathbf{N},\, i < 10\right\}$$

L'écriture en compréhension

En mathématiques, l'ensemble des carrés des 10 premiers entiers naturels se note

$$\left\{i^2 \mid i \in \mathbf{N},\, i < 10\right\}$$

C'est une écriture en *compréhension*.

L'écriture en compréhension

En mathématiques, l'ensemble des carrés des 10 premiers entiers naturels se note

$$\left\{i^2 \mid i \in \mathbf{N},\, i < 10\right\}$$

C'est une écriture en *compréhension*.
On peut faire la même chose en PYTHON :

L'écriture en compréhension

En mathématiques, l'ensemble des carrés des 10 premiers entiers naturels se note

$$\left\{i^2 \mid i \in \mathbf{N},\, i < 10\right\}$$

C'est une écriture en *compréhension*.
On peut faire la même chose en PYTHON :

```
lst = [i*i for i in range(10)]
```

L'écriture en compréhension

En mathématiques, l'ensemble des carrés des 10 premiers entiers naturels se note

$$\left\{i^2 \mid i \in \mathbf{N},\, i < 10\right\}$$

C'est une écriture en *compréhension*.
On peut faire la même chose en PYTHON :

```
lst = [i*i for i in range(10)]
```

Évidemment, c'est plus rapide que la méthode précédente...

L'écriture en compréhension

En mathématiques, l'ensemble des carrés des 10 premiers entiers naturels se note

$$\left\{i^2 \mid i \in \mathbf{N},\, i < 10\right\}$$

C'est une écriture en *compréhension*.
On peut faire la même chose en PYTHON :

```
lst = [i*i for i in range(10)]
```

Évidemment, c'est plus rapide que la méthode précédente...
Et on peut faire bien plus !

Des applications

Des applications

On peut utiliser une liste pour en construire une autre :

Des applications

On peut utiliser une liste pour en construire une autre :

```
lst1 = [2, -1, 3, 4, 7]
lst2 = [x + 1 for x in lst1]
print(lst2)
```

Des applications

On peut utiliser une liste pour en construire une autre :

```
lst1 = [2, -1, 3, 4, 7]
lst2 = [x + 1 for x in lst1]
print(lst2)
```

Ce programme affiche :

Des applications

On peut utiliser une liste pour en construire une autre :

```
lst1 = [2, -1, 3, 4, 7]
lst2 = [x + 1 for x in lst1]
print(lst2)
```

Ce programme affiche :

```
[3, 0, 4, 5, 8]
```

Des applications

Des applications

Dans le même esprit :

Des applications

Dans le même esprit :

```
lst1 = ['2', '0', '13']
lst2 = [int(x) for x in lst1]
print(lst2)
```

Des applications

Dans le même esprit :

```
lst1 = ['2', '0', '13']
lst2 = [int(x) for x in lst1]
print(lst2)
```

Ce programme affiche :

Des applications

Dans le même esprit :

```
lst1 = ['2', '0', '13']
lst2 = [int(x) for x in lst1]
print(lst2)
```

Ce programme affiche :

```
[2, 0, 13]
```

Des applications

Des applications

Ou encore :

Des applications

Ou encore :

```
lst1 = ['Fred', 'Titouan', 'Tinaïg']
lst2 = [prenom[0] for prenom in lst1]
print(lst2)
```

Des applications

Ou encore :

```
lst1 = ['Fred', 'Titouan', 'Tinaïg']
lst2 = [prenom[0] for prenom in lst1]
print(lst2)
```

Ce programme affiche :

Des applications

Ou encore :

```
lst1 = ['Fred', 'Titouan', 'Tinaïg']
lst2 = [prenom[0] for prenom in lst1]
print(lst2)
```

Ce programme affiche :

```
['F', 'T', 'T']
```

Écriture en compréhension avec conditions

Encore mieux

Encore mieux

Il est possible d'utiliser `if` en compréhension :

Encore mieux

Il est possible d'utiliser `if` en compréhension :

```
lst1 = [8, 0, 11, 10, 3, 15]
lst2 = [2 * x for x in lst1 if x > 10]
print(lst2)
```

Encore mieux

Il est possible d'utiliser `if` en compréhension :

```
lst1 = [8, 0, 11, 10, 3, 15]
lst2 = [2 * x for x in lst1 if x > 10]
print(lst2)
```

Ce programme affiche :

Encore mieux

Il est possible d'utiliser `if` en compréhension :

```
lst1 = [8, 0, 11, 10, 3, 15]
lst2 = [2 * x for x in lst1 if x > 10]
print(lst2)
```

Ce programme affiche :

```
[22, 30]
```

Encore mieux

Il est possible d'utiliser `if` en compréhension :

```
lst1 = [8, 0, 11, 10, 3, 15]
lst2 = [2 * x for x in lst1 if x > 10]
print(lst2)
```

Ce programme affiche :

```
[22, 30]
```

On met dans `lst2` le double de chaque élément de `lst1` qui est supérieur à 10 (dans l'ordre de parcours).

Encore plus fort

Encore plus fort

Il est possible d'utiliser `if ... else ...` en compréhension, mais à ce moment là il faut écrire les conditions au début :

Encore plus fort

Il est possible d'utiliser `if ... else ...` en compréhension, mais à ce moment là il faut écrire les conditions au début :

```
lst1 = [8, -10, 11, -4, -3, 15]
lst2 = [(x if x > 0 else 0) for x in lst1]
# parenthèses facultatives
print(lst2)
```

Encore plus fort

Il est possible d'utiliser `if ... else ...` en compréhension, mais à ce moment là il faut écrire les conditions au début :

```
lst1 = [8, -10, 11, -4, -3, 15]
lst2 = [(x if x > 0 else 0) for x in lst1]
# parenthèses facultatives
print(lst2)
```

Ce programme affiche :

Encore plus fort

Il est possible d'utiliser `if ... else ...` en compréhension, mais à ce moment là il faut écrire les conditions au début :

```
lst1 = [8, -10, 11, -4, -3, 15]
lst2 = [(x if x > 0 else 0) for x in lst1]
# parenthèses facultatives
print(lst2)
```

Ce programme affiche :

```
[8, 0, 11, 0, 0, 15]
```

Encore plus fort

Il est possible d'utiliser `if ... else ...` en compréhension, mais à ce moment là il faut écrire les conditions au début :

```
lst1 = [8, -10, 11, -4, -3, 15]
lst2 = [(x if x > 0 else 0) for x in lst1]
# parenthèses facultatives
print(lst2)
```

Ce programme affiche :

```
[8, 0, 11, 0, 0, 15]
```

On crée une nouvelle liste en remplaçant tous les nombres négatifs de `lst1` par zéro.

Un dernier pour la route

Un dernier pour la route

Que fait le programme suivant ?

Un dernier pour la route

Que fait le programme suivant ?

```
lst1 = [8, -10, 11, -4, -3, 15]
lst2 = [i for i in range(len(lst1)) if lst1[i] > 0]
print(lst2)
```

Un dernier pour la route

Que fait le programme suivant ?

```
lst1 = [8, -10, 11, -4, -3, 15]
lst2 = [i for i in range(len(lst1)) if lst1[i] > 0]
print(lst2)
```

Ce programme affiche :

Un dernier pour la route

Que fait le programme suivant ?

```
lst1 = [8, -10, 11, -4, -3, 15]
lst2 = [i for i in range(len(lst1)) if lst1[i] > 0]
print(lst2)
```

Ce programme affiche :

```
[0, 2, 5]
```

Un dernier pour la route

Que fait le programme suivant ?

```
lst1 = [8, -10, 11, -4, -3, 15]
lst2 = [i for i in range(len(lst1)) if lst1[i] > 0]
print(lst2)
```

Ce programme affiche :

```
[0, 2, 5]
```

On crée une liste contenant les indices des éléments de `lst1` qui sont strictement positifs.

Les écritures en compréhension imbriquées

C’est un peu dur au départ

C’est un peu dur au départ

0 0 0 0
0 0 0 0
0 0 0 0

C’est un peu dur au départ

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

Si on veut représenter ce « tableau de nombres » par une liste, on peut écrire

C’est un peu dur au départ

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

Si on veut représenter ce « tableau de nombres » par une liste, on peut écrire

```
lst = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

pause

(3 lignes de chacune 4 colonnes)

Mais c’est plus pratique d’écrire

C’est un peu dur au départ

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

Si on veut représenter ce « tableau de nombres » par une liste, on peut écrire

```
lst = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

pause

(3 lignes de chacune 4 colonnes)

Mais c’est plus pratique d’écrire

```
lst=[[0 for j in range(4)] for i in range(3)]
```

Pour conclure

Super cocktail

Super cocktail

On peut combiner toutes les techniques que nous venons de voir...

Super cocktail

On peut combiner toutes les techniques que nous venons de voir...

Par exemple on peut créer une liste de listes de listes avec des conditions, *et cætera*.

Super cocktail

On peut combiner toutes les techniques que nous venons de voir...

Par exemple on peut créer une liste de listes de listes avec des conditions, *et cætera*.

La seule limite, c'est l'imagination et la capacité à écrire en PYTHON !