Python les listes

liste[], objets, len(), append, Del, remote, …

En résumé

- Une **liste** en Python désigne une collection finie et ordonnée d'objets. Elle est la représentation d'un ensemble fini d'objets que l'on distingue par leur ordre d'apparition.
- On crée une liste à l'aide des crochets [ ] et on sépare les objets de cette liste à l'aide d'une virgule. On parle alors de listes **définies en extension**.

```
animaux = ["girafe", "tigre", "singe", "souris"]
tailles = [5, 2.5, 1.75, 0.15]
mixte = ["girafe", 5, "souris", 0.15]
```

- On peut également créer des listes par une commande du type [valeur boucle]. On parle alors de listes **définies par compréhension**.
- La méthode **len()** permet de renvoyer la longueur d'une liste, c'est-à-dire le nombre d'éléments qui la composent (il s'agit du **cardinal** de la liste).
  Une liste vide a pour longueur 0.

```
L1 liste1 = [1, 2, 3]
L2 liste2 = ["a","b","c","d"]
L3 liste3 = [ ]
```

- Les éléments d'une liste sont repérés à l'aide de leur **indice**, c'est-à-dire leur position à l'intérieur de la liste. Le premier élément a pour indice 0, le second 1, le dernier len(liste) – 1.
- La commande liste[ ] permet d'accéder facilement à un élément d'une liste grâce à son indice.

```
liste  : ["girafe", "tigre", "singe", "souris"]
indice :        0        1        2         3
```

- La commande del() permet de supprimer un élément d'une liste grâce à l'indice de cet élément.
- La commande remove() permet de supprimer un élément repéré par sa valeur dans une liste.
- Elle supprime le premier élément de la liste ayant cette valeur.

Création d'une liste

listes définies en extension.

On crée une liste à l'aide des crochets [ ] et on sépare les objets de cette liste à l'aide d'une virgule.

```
>>> liste = [1,2,3]
>>> liste
[1, 2, 3]
```

listes définies concaténation et duplication.

*Tout comme les chaînes de caractères, les listes supportent l'opérateur + de concaténation, ainsi que l'opérateur * pour la duplication.*

```
>>> ani1 = ["girafe", "tigre"]
>>> ani2 = ["singe", "souris"]
>>> ani1 + ani2
['girafe', 'tigre', 'singe', 'souris']
>>> ani1 * 3
['girafe', 'tigre', 'girafe', 'tigre', 'girafe', 'tigre']
```

On peut partir d'une liste vide et ajouter des éléments l'un après l'autre.

```
>>> liste1 = []
>>> liste1
[]
```

```
>>> liste1 = liste1 + [15]
>>> liste1
[15]
>>> liste1 = liste1 + [-5]
>>> liste1
[15, -5]
```

listes définies en ajoutant des objets avec la méthode 'append'

On peut créer une liste en ajoutant des objets avec la méthode append (ajouter en anglais).

Cette méthode est plus élégante

```
>>> liste = []
>>> liste
[]
>>> liste.append(1)
>>> liste
[1]
>>> liste.append("ok")
>>> liste
[1, 'ok']
```

listes définies avec l'instruction range()

On peut utiliser l'instruction `range()` *pour générer une liste d'entiers. On peut jouer avec les différents paramètres de la fonction range(m,n, p).*

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> list(range(0, 5))
[0, 1, 2, 3, 4]
>>> list(range(15, 20))
[15, 16, 17, 18, 19]
>>> list(range(0, 1000, 200))
[0, 200, 400, 600, 800]
>>> list(range(2, -2, -1))
[2, 1, 0, -1]
```

listes définies par compréhension

Une liste en compréhension (comprehension list) permet de créer une liste à partir d'une itération.

Cela permet de simplifier le code pour le rendre plus lisible et donc plus rapide à écrire et plus simple à maintenir.

```
liste = [x for x in range(5)]
print(liste)
# affiche [0, 1, 2, 3, 4]
```

```
liste = [1, 2, 3, 4]
nouvelle_liste = [2 * x for x in liste]
print(nouvelle_liste)
# affiche [2, 4, 6, 8]
```

```
liste=[1,2,3,4,5,6,7,8]
liste1=[x for x in liste if x > 5]
print(liste1)
[6,7,8]
```

Il est également possible d'appliquer un filtre lors de la création d'une liste en compréhension afin de ne pas prendre un compte certains éléments de la liste ou de l'itération de départ.

```
liste = ["hello", "the", "world"]
nouvelle_liste = [x for x in liste if len(x) < 4]
print(nouvelle_liste)
# affiche ['the']
```

```
liste = ["hello", "the", "world"]
nouvelle_liste = [len(x) for x in liste]
print(nouvelle_liste)
# affiche [5, 3, 5]
```

Il est également possible de combiner deux itérations dans une liste en compréhension. Cela permet de réaliser une itération sur chaque élément de la liste ou de l'itération de départ

Code équivalent

```
liste = ["hello", "the", "world"]
new_liste=[]
for mot in liste:
    for c in mot:
        new_liste.append(c)
print(new_liste)
```

```
liste = ["hello", "the", "world"]
nouvelle_liste = [c for mot in liste for c in mot]
print(nouvelle_liste)
# affiche ['h', 'e', 'l', 'l', 'o', 't', 'h', 'e', 'w',
```

Longueur d'une liste

L'instruction `len()` vous permet de connaître la longueur d'une liste, c'est-à-dire le nombre d'éléments que contient la liste.
Une liste vide a pour longueur 0

```
>>> animaux = ["girafe", "tigre", "singe", "souris"]
>>> len(animaux)
4
>>> len([1, 2, 3, 4, 5, 6, 7, 8])
8
```

Afficher les objets d'une liste

Un des gros avantages d'une liste est que vous accédez à ses éléments par leur position. Ce numéro est appelé **indice** (ou *index*) de la liste.

```
liste  : ["girafe", "tigre", "singe", "souris"]
indice :        0        1        2         3
```

Remarque** : Il faut faire attention car les indices d'une liste de n éléments commencent à **0** et se terminent à **n-1 .

```
liste  : ["girafe", "tigre", "singe", "souris"]
indice :        0        1        2         3
```

*La liste peut également être indexée avec des nombres négatifs.Les indices négatifs reviennent à compter à partir de la fin. Leur principal avantage est que vous pouvez accéder au dernier élément d'une liste à l'aide de l'indice **–1** sans pour autant connaître la longueur de*

```
liste           : ["girafe", "tigre", "singe", "souris"]
indice positif :        0        1        2         3
indice négatif :       -4       -3       -2        -1
```

***Remarque** : Il faut faire attention à ne pas dépasser le dernier indice négatif au risque d'avoir un message d'erreur.*

```
Traceback (innermost last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
```

*Il est possible de sélectionner une partie d'une liste en utilisant un indiçage construit sur le modèle `[m:n+1]` pour récupérer tous les éléments, du émième au énième (de l'élément `m` inclu à l'élément `n+1` exclu). On dit alors qu'on récupère une **tranche** de la liste.*

```
>>> animaux = ["girafe", "tigre", "singe", "souris"]
>>> animaux[0:2]
['girafe', 'tigre']
>>> animaux[0:3]
['girafe', 'tigre', 'singe']
>>> animaux[0:]
['girafe', 'tigre', 'singe', 'souris']
>>> animaux[:]
['girafe', 'tigre', 'singe', 'souris']
>>> animaux[1:]
['tigre', 'singe', 'souris']
>>> animaux[1:-1]
['tigre', 'singe']
```

On peut aussi préciser le pas en ajoutant un symbole deux-points supplémentaire et en indiquant le pas par un entier.

```
>>> animaux = ["girafe", "tigre", "singe", "souris"]
>>> animaux[0:3:2]
['girafe', 'singe']
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x[::2]
[0, 2, 4, 6, 8]
```

Trouver l'index d'une valeur

La méthode `index` vous permet de connaitre la position de l'item cherché.

```
>>> liste = ["a","a","a","b","c","c"]
>>> liste.index("b")
3
```

Pour récupérer l'index et l'objet d'une liste on utilise la fonction `enumerate`

```
>>> for lettre in enumerate(liste):
...     print lettre
...
(0, 'a')
(1, 'd')
(2, 'm')
```

Transformer une string en liste

il peut être utile de transformer une chaine de caractère en liste. Cela est possible avec la méthode `split`

```
>>> ma_chaine = "Olivier:ENGEL:Strasbourg"
>>> ma_chaine.split(":")
['Olivier', 'ENGEL', 'Strasbourg']
```