

LISTES

I. Notions de base sur les listes

1) Création d'une liste

En Python, les *listes* sont des structures modifiables, ordonnées, contenant une collection d'éléments pouvant être de types distincts ; on parle parfois aussi de *tableaux*.

On peut créer une liste en donnant tous ces éléments, séparés par des virgules et encadrés par une paire de crochets.

```
>>> L1 = [9, 4, 6, 2, 0]
>>> type(L1)
<class 'list'>
>>> L2 = [1, 5.21, 'toto', L1]
>>> L2
[1, 5.21, 'toto', [9, 4, 6, 2, 0]]
```

Remarque : Ici, la liste `[9, 4, 6, 2, 0]` est **un** élément de la liste `L2`.

2) Longueur d'une liste

La *longueur* d'une liste est son nombre d'éléments. On l'obtient grâce à la fonction `len`.

```
>>> L1 = [9, 4, 6, 2, 0]
>>> len(L1)
5
>>> L2 = [1, 5.21, 'toto', L1]
>>> len(L2)
4
```

3) Accès aux éléments d'une liste

Les éléments d'une liste `L` de longueur ℓ sont indexés de 0 à $\ell - 1$.

- L'instruction `L[k]` renvoie l'élément d'indice k de `L` si k est compris entre 0 et $\ell - 1$.
- Si $k \geq \ell$, l'instruction `L[k]` renvoie une erreur.
- Si k est compris entre $-\ell$ et -1 , l'instruction `L[k]` renvoie l'élément d'indice $k + \ell$ de `L` ; si $k < -\ell$, l'instruction `L[k]` renvoie une erreur.

```
>>> L = [9, 4, 6, 2, 0]
>>> len(L)
5
>>> L[0], L[1], L[2], L[3], L[4]
(9, 4, 6, 2, 0)
>>> L[-5], L[-4], L[-3], L[-2], L[-1]
(9, 4, 6, 2, 0)
>>> L[-6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> L[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

4) Modification d'un élément

Les listes sont *mutables*, c'est-à-dire modifiables ; on peut affecter une nouvelle valeur à l'élément d'indice k de la liste `L` grâce à l'instruction `L[k] = <valeur>`.

```
>>> L = [0, 9, 4, 6, 2, 0]
>>> L[0] = 42
>>> L
[42, 9, 4, 6, 2, 0]
```

5) Définition en compréhension

On peut définir les éléments d'une liste *en compréhension* c'est-à-dire en donnant une formule définissant les éléments de la liste.

```
>>> L1 = [i for i in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> L2 = [i**2 for i in range(6)]
[0, 1, 4, 9, 16, 25]
>>> L3 = [2*i for i in L2]
[0, 2, 8, 18, 32, 50]
>>> L4 = [[i*j for j in range(i)] for i in range(4)]
[[], [0], [0, 2], [0, 3, 6]]
```

II. Opérations sur les listes

1) Ajout d'un élément

a) Ajout d'un élément en fin de liste

La fonction (ou plutôt, méthode) *append* permet d'ajouter un élément à la fin d'une liste :

```
>>> L1 = [0, 9, 4, 6, 2, 0]
>>> L1.append(2016)
>>> L1
[0, 9, 4, 6, 2, 0, 2016]
>>> L2 = []
>>> L2.append(5)
>>> L2
[5]
```

Cette méthode modifie la liste mais ne renvoie rien.

b) Ajout d'un élément à un indice donné

La méthode *insert* permet d'ajouter un élément à un indice donné. Elle prend deux arguments : le premier correspond à l'indice auquel insérer le nouvel élément, le deuxième à l'élément à ajouter :

```
>>> L = [1, 2, 3, 4, 5, 6]
>>> L.insert(2, 1515)
>>> L
[1, 2, 1515, 3, 4, 5, 6]
>>> L.insert(7, 1789)
>>> L
[1, 2, 1515, 3, 4, 5, 6, 1789]
```

Cette méthode modifie la liste mais ne renvoie rien.

2) Suppression d'un élément

a) Suppression du dernier élément d'une liste

La méthode *pop* renvoie le dernier élément d'une liste non vide, et le supprime de la liste :

```
>>> L = [1, 2, 3, 2]
>>> L.pop()
2
>>> L
[1, 2, 3]
```

b) Suppression d'un élément d'indice donné

La méthode `pop` peut aussi s'utiliser avec un argument correspondant à l'indice de l'élément à supprimer (et à renvoyer) :

```
>>> L = [1, 2, 3, 4, 3, 2, 1]
>>> L.pop(4)
3
>>> L
[1, 2, 3, 4, 2, 1]
```

c) Suppression d'une occurrence

Enfin, la méthode `remove` permet de supprimer la première occurrence d'une valeur passée en paramètre dans la liste, à condition que cette valeur soit bien présente dans la liste ; dans le cas contraire, la méthode renvoie une erreur.

```
>>> L = [1, 0, 2, 0, 3, 0, 4]
>>> L.remove(0)
>>> L
[1, 2, 0, 3, 0, 4]
>>> L.remove(25)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

3) Découpage en tranches

Le découpage en tranche (*slicing*) permet d'obtenir une sous-liste d'une liste L constitué d'éléments consécutifs de L .

L'instruction `L[i:j]` renvoie la liste des éléments de L situés entre les indices i (inclus) et j (exclu).

```
>>> L = [9, 4, 6, 2, 0]
>>> L[1:4]
[4, 6, 2]
```

Si l'indice de début est omis, l'indice de départ sera l'indice 0.

```
>>> L = [9, 4, 6, 2, 0]
>>> L[:3]
[9, 4, 6]
```

Si l'indice de fin est omis, la dernière valeur prise en compte est la dernière valeur de la liste.

```
>>> L[2:]
[6, 2, 0]
```

4) Listes et affectation

⚠ ATTENTION ! L'instruction `L2 = L1` ne crée pas une copie de la liste $L1$; les deux variables $L1$ et $L2$ désignent alors un même objet !

```
>>> L1 = [8, 1, -2, 9, 4]
>>> L2 = L1
>>> L1[0] = 2016
>>> L2
[2016, 1, -2, 9, 4]
```

Si on souhaite créer une copie d'une liste, on pourra :

— construire la nouvelle liste par compréhension :

```
>>> L2 = [L1[i] for i in range(len(L1))]
```

ou encore :

```
>>> L2 = [e for e in L1]
```

— utiliser un découpage en tranche (qui crée dans tous les cas une nouvelle liste) :

```
>>> L2 = L1[:]
```

5) Concaténation

Il est possible de concaténer deux listes (les mettre bout à bout) avec l'opérateur `+` ; cela crée une nouvelle liste :

```
>>> L1 = [1, 2, 3]
>>> L2 = [4, 5]
>>> L3 = L1 + L2
>>> L3
[1, 2, 3, 4, 5]
```

L'opérateur `*` permet de concaténer plusieurs fois une même liste (utile pour créer une longue liste dont on modifiera ensuite les éléments) :

```
>>> L = [0]*10
>>> L
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

III. Parcours d'une liste

Pour parcourir tous les éléments d'une liste, on peut accéder à chacun d'entre eux par son indice à l'aide d'une boucle `for`. Par exemple, si on veut afficher tous les éléments d'une liste `L`, on pourra écrire :

```
for i in range(len(L)):
    print(L[i])
```

En pratique, la boucle `for` de Python sait parcourir directement tous les éléments d'une liste ; on pourra donc écrire :

```
for e in L:
    print(e)
```

Remarque : Pour tester si un élément `e` est dans une liste `liste`, on pourra utiliser le mot-clé `in`

```
>>> L = [2, 4, 6, [8,10]]
>>> 3 in L
False
>>> 2 in L
True
>>> 8 in L
False
```

Exercices

EXERCICE 1

Écrire une fonction prenant en argument une liste de nombres et renvoyant la somme de ces nombres.

EXERCICE 2

Écrire une fonction prenant en argument une liste de nombres et qui renvoie la moyenne et la variance de cette liste.

EXERCICE 3

Écrire une fonction qui prend en argument une liste `L` et qui renvoie une liste obtenue à partir de `L` en supprimant toutes les occurrences de 0.

On n'utilisera ni la méthode `pop` ni la méthode `remove`. La liste `L` ne devra pas être modifiée lors de l'appel à la fonction.

EXERCICE 4

1. Écrire une fonction qui prend en argument une liste d'entiers et qui renvoie le plus grand de ses éléments.
2. Modifier la fonction précédente pour renvoyer la liste des indices où se trouve le plus grand élément.
3. Écrire une fonction qui prend en argument une liste d'entiers et qui renvoie le couples constitués des deux plus grands de ses éléments (éventuellement égaux).