Chap 3 Listes

Types d'objets en Python - Classes

- **Classe str (String) : chaînes de caractères.**
- **Classe int (Integer) : nombres entiers.**
- **Classe float : nombres flottants (ou nombres réels).**
- **Remarque : Toute variable est un objet.**
- **Remarque : Toute fonction/méthode est un objet.**
- **Remarque : Un objet est une instance d'une classe.**
- **Donc, en Python, tout est objet.**

Définition d'une liste

Une liste est une structure de données qui contient une collection d'objets Python. Il s'agit d'un nouveau type par rapport aux entiers, float, booléens et chaînes de caractères. On parle aussi d'objet séquentiel en ce sens qu'il contient une séquence d'autres objets.

Python autorise la construction de liste contenant des valeurs de types différents (par exemple entier et chaîne de caractères), ce qui leur confère une grande

flexibilité. Une liste est déclarée par une série de valeurs (n'oubliez pas les guillemets, simples ou doubles, s'il s'agit de chaînes de caractères) séparées par des virgules, et le tout encadré par des crochets. En voici quelques exemples :

```
1 >>> animaux = ["girafe", "tigre", "singe", "souris"]
2 >>> tailles = [5, 2.5, 1.75, 0.15]
3 >>> mixte = ["girafe", 5, "souris", 0.15]
4 >>> animaux
5 ['girafe', 'tigre', 'singe', 'souris']
6 >>> tailles
7 [5, 2.5, 1.75, 0.15]
8 >>> mixte
9 ['girafe', 5, 'souris', 0.15]
```

Utilisation

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
```

Soyez très attentif au fait que les indices d'une liste de n éléments commencent à 0 et se terminent à n−1. Voyez l'exemple suivant :

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

```
4 >>> animaux[1]
5 'tigre'
6 >>> animaux[3]
7 'souris'
```

Par conséquent, si on appelle l'élément d'indice 4 de notre liste, Python renverra un message d'erreur :

```
1 >>> animaux[4]
2 Traceback (innermost last):
3 File "<stdin>", line 1, in ?
4 IndexError: list index out of range
```

N'oubliez pas ceci ou vous risquez d'obtenir des bugs inattendus !

Opération sur les listes

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

```
1 >>> ani1 = ["girafe", "tigre"]
2 >>> ani2 = ["singe", "souris"]
3 >>> ani1 + ani2
4 ['girafe', 'tigre', 'singe', 'souris']
5 >>> ani1 * 3
6 ['girafe', 'tigre', 'girafe', 'tigre', 'girafe', 'tigre']
```

L'opérateur + est très pratique pour concaténer deux listes.

Vous pouvez aussi utiliser la **méthode .append()** lorsque vous souhaitez ajouter un seul élément à la fin d'une liste.

Remarque

Revenons quelques instants sur la notion de méthode abordée dans ce chapitre avec .append(). En Python, on peut considérer chaque variable comme un objet sur lequel on peut appliquer des méthodes. Une méthode est simplement une fonction qui utilise et agit sur l'objet lui-même, les deux étant connectés par un point. La syntaxe générale est de la forme objet.méthode().

Dans l'exemple suivant :

```
1 >>> liste1 = [1, 2]
2 >>> liste1.append(3)
3 >>> liste1
4 [1, 2, 3]
```

la méthode .append() est liée à liste1 qui est un objet de type liste. La méthode modifie l'objet liste en lui ajoutant un élément. Nous aurons de nombreuses occasions de revoir cette notation objet.méthode().

Minimum, maximum et somme d'une liste

Les fonctions min(), max() et sum() renvoient respectivement le minimum, le maximum et la somme d'une liste passée en argument :

```
1 >>> liste1 = list(range(10))
2 >>> liste1
3 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4 >>> sum(liste1)
5 45
6 >>> min(liste1)
7 0
8 >>> max(liste1)
9 9
```

Même si en théorie ces fonctions peuvent prendre en argument une liste de *strings*, on les utilisera la plupart du temps avec des types numériques (liste d'entiers et / ou de *floats*).
Ces deux fonctions pouvaient prendre plusieurs arguments entiers et / ou *floats*, par exemple :

```
1 >>> min(3, 4)
2 3
```

Attention toutefois à ne pas mélanger entiers et *floats* d'une part avec une liste d'autre part, car cela renvoie une erreur :

```
>>> min(liste1, 3, 4)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'int' and 'list'
```