Listes

Définition

Une **liste** est une structure de données qui contient une série de valeurs. 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 :

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

Lorsque l'on affiche une liste, *Python* la restitue telle qu'elle a été saisie.

Utilisation

Un des gros avantages d'une liste est que vous pouvez appeler 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 **attentifs** au fait que les indices d'une liste de `n` éléments commence à `0` et se termine à `n-1`. Voyez l' exemple suivant :

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

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

```
>>> animaux[4]
Traceback (innermost last):
  File "<stdin>", line 1, in ?
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 :

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

Dans l'exemple suivant nous allons créer une liste vide :

```
>>> a = []
>>> a
[]
```

puis lui ajouter deux éléments, l'un après l'autre, d'abord avec la concaténation :

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

puis avec la méthode `.append()` :

```
>>> a.append(13)
>>> a
[15, -5, 13]
>>> a.append(-3)
>>> a
[15, -5, 13, -3]
```

Dans l'exemple ci-dessus, nous ajoutons des éléments à une liste en utilisant l'opérateur de concaténation + ou la méthode `.append()`. Nous vous conseillons dans ce cas précis d'utiliser la méthode `.append()` dont la syntaxe est plus élégante.

Nous reverrons en détail la méthode `.append()` dans le chapitre 11 Plus sur les listes.

Indiçage négatif

La liste peut également être indexée avec des nombres négatifs selon le modèle suivant :

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

ou encore :

```
liste          : ["A", "B", "C", "D", "E", "F"]
indice positif :   0    1    2    3    4    5
indice négatif :  -6   -5   -4   -3   -2   -1
```

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 cette liste. L'avant-dernier élément a lui l'indice `-2`, l'avant-avant dernier l'indice `-3`, etc.

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

Pour accéder au premier élément de la liste avec un indice négatif, il faut par contre connaître le bon indice :

```
>>> animaux[-4]
'girafe'
```

Dans ce cas, on utilise plutôt `animaux[0]`.

Tranches

Un autre avantage des listes est la possibilité 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` inclus à l'élément `n+1` exclu). On dit alors qu'on récupère une **tranche** de la liste, par exemple :

```
>>> 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']
```

Notez que lorsqu'aucun indice n'est indiqué à gauche ou à droite du symbole deux-points, Python prend par défaut tous les éléments depuis le début ou tous les éléments jusqu'à la fin respectivement.

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

```
[1, 4]
```

Finalement, on se rend compte que l'accès au contenu d'une liste fonctionne sur le modèle `liste[début:fin:pas]`.

Fonction `len()`

L'instruction `len()` vous permet de connaître la longueur d'une liste, c'est-à-dire le nombre d'éléments que contient la liste. Voici un exemple d'utilisation :

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

Les fonctions `range()` et `list()`

L'instruction `range()` est une fonction spéciale en Python qui génère des nombres entiers compris dans un intervalle. Lorsqu'elle est utilisée en combinaison avec la fonction `list()`, on obtient une liste d'entiers. Par exemple :

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

La commande `list(range(10))` a généré une liste contenant tous les nombres entiers de `0` inclus à `10` **exclu**. Nous verrons l'utilisation de la fonction `range()` toute seule dans le chapitre 5 Boucles et comparaisons.

Dans l'exemple ci-dessus, la fonction `range()` a pris un argument, mais elle peut également prendre deux ou trois arguments, voyez plutôt :

```
>>> 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]
```

L'instruction `range()` fonctionne sur le modèle `range([début,] fin[, pas])`. Les arguments entre crochets sont optionnels. Pour obtenir une liste de nombres entiers, il faut l'utiliser systématiquement avec la fonction `list()`.

Enfin, prenez garde aux arguments optionnels par défaut (`0` pour début et `1` pour pas) :

```
>>> list(range(10,0))
[]
```

Ici la liste est vide car Python a pris la valeur du pas par défaut qui est de `1`. Ainsi, si on commence à `10` et qu'on avance par pas de `1`, on ne pourra jamais atteindre `0`. Python génère ainsi une liste vide. Pour éviter ça, il faudrait, par exemple, préciser un pas de `-1` pour obtenir une liste d'entiers décroissants :

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

Listes de listes

Pour finir, sachez qu'il est tout à fait possible de construire des listes de listes. Cette fonctionnalité peut parfois être très pratique. Par exemple :

```
>>> enclos1 = ["girafe", 4]
>>> enclos2 = ["tigre", 2]
>>> enclos3 = ["singe", 5]
>>> zoo = [enclos1, enclos2, enclos3]
>>> zoo
[['girafe', 4], ['tigre', 2], ['singe', 5]]
```

Dans cet exemple, chaque sous-liste contient une catégorie d'animal et le nombre d'animaux pour chaque catégorie.

Pour accéder à un élément de la liste, on utilise l'indiçage habituel :

```
>>> zoo[1]
['tigre', 2]
```

Pour accéder à un élément de la sous-liste, on utilise un double indiçage :

```
>>> zoo[1][0]
'tigre'
>>> zoo[1][1]
2
```

On verra un peu plus loin qu'il existe en Python des dictionnaires qui sont également très pratiques pour stocker de l' information structurée. On verra aussi qu'il existe un module nommé *NumPy* qui permet de créer des listes ou des tableaux de nombres (vecteurs et matrices) et de les manipuler.

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.

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

Nous avions déjà croisé `min()`, `max()` dans le chapitre 2 Variables. On avait vu que ces deux fonctions pouvaient prendre plusieurs arguments entiers et / ou *floats*, par exemple :

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

Soit on passe plusieurs entiers et/ou *floats* en argument, soit on passe une liste unique.