Représentation des données - Les listes

Les bases des listes

Une liste est : Une séquence d'éléments modifiables.
Pour créer une liste on utilise des **crochets**
L'accès aux éléments se fait avec leurs indices

```
>>> l = [4, 1, 2, 6, 7]
```

La liste vide est notée `[]`. Il est possible de rajouter un élément à une liste :

```
>>> l.append(15)
>>> l.append(7)
>>> l
[4, 1, 2, 6, 7, 15, 7]
```

Il est possible d'accéder à n'importe quel élément d'une liste, comme pour une chaîne de caractères :

```
>>> l[0]
4
>>> l[1]
1
>>> l[-1]
7
>>> l[-2]
15
```

Il est également possible d'obtenir des sous-listes d'une liste, comme pour les chaînes de caractères :

```
>>> l[2:]
[2, 6, 7, 15, 7]
>>> l[3:6]
[6, 7, 15]
>>> l[:3]
[4, 1, 2]
```

Pour concaténer 2 listes, il suffit d'utiliser l'opérateur + ou `extend` :

```
>>> [6,3,1] + [4,8]
[6, 3, 1, 4, 8]
>>> [6,3,1].extend([4,8])
```

Supprimer un élément d'une liste en indiquant soit sa valeur, soit son index :

```
l=[4, 1, 2, 6, 7, 15, 7]
>>> del l[0]
>>> l
[1, 2, 6, 7, 15, 7]
>>> l.remove(15)
>>> l
[1, 2, 6, 7, 7]
```

Génération des listes

Une liste peut être construite de plusieurs façons possibles :

```
>>> [0]*5
[0, 0, 0, 0, 0]
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list('bonjour') # fonction list()
['b', 'o', 'n', 'j', 'o', 'u', 'r']
```

Python permet également de générer une liste en appliquant une fonction à un ensemble de valeurs :

```
[f(x) for x in ITERABLE]
```

Cela permet de créer une liste en une seule ligne au lieu de passer par une boucle :

```
>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> [x//2 for x in [3,1,7]]
[1, 0, 3]
```

Il est même possible de filtrer la liste obtenue à l'aide d'une condition :

```
[f(x) for x in ITERABLE if TEST]
```

On peut ainsi obtenir la liste des entiers inférieurs à 50 qui sont des multiples de 3 mais pas de 4 :

```
>>> [x for x in range(50) if x%3==0 and x%4!=0]
[3, 6, 9, 15, 18, 21, 27, 30, 33, 39, 42, 45]
```

Remarque :
En Python, les listes sont polymorphiques. C'est à dire qu'elles peuvent contenir plusieurs types de données différentes dans une même liste, y compris d'autres listes, comme `["mot",4,5.1,True,[1,2]]`.

Parcours de listes

Les listes sont des itérables et il est donc possible de les parcourir très simplement :

```
>>> for i in [2, 5, 7, 9, 10]:
        print(i)

2
5
7
9
10
```

Ou bien en utilisant une boucle sur les indices:

```
l=[2, 5, 7, 9, 10]
>>> for i in range(len(l)):
        print(i)
2
5
7
9
10
```

Autres méthodes

`liste.insert(i, x)`: Insère un élément (x) à la position indiquée (i).

```
>>> L=[ 3 , 6 , 2 , 7 , 1 , 9 , 5]
>>>L.insert(2,0)
>>>L
[3, 6, 0, 2, 7, 1, 9, 5]
```

- Donc `L.insert(0,x)` permet d'ajouter x au début de la liste
- Donc `L.insert(-1,x)` permet d'ajouter x à la fin de la liste

`liste.pop(i)` : Enlève de la liste l'élément situé à la position indiquée et retourne l'élément supprimé.
Si aucune position n'est indiqué, `liste.pop()` enlève et retourne le dernier élément de la liste.

```
>>> L=[ 3 , 6 , 2 , 7 , 1 , 2 , 5]
>>> L.pop(3)
Retourne 7
>>> L
[ 3 , 6 , 2, 1 , 2 , 5]
```

`liste.index(x)`: Retourne la position du premier élément de la liste ayant la valeur x.
`len(liste)` : Retourne la longueur de la liste
`liste.count(x)`: Retourne le nombre d'éléments ayant la valeur x dans la liste.
`liste.sort()`: Trier les éléments sur place
`sorted(L)` : Retourne la liste L triée
`liste.reverse()`: Inverse l'ordre des éléments de la liste.

Copie d'une liste

Ce programme résume la situation :

```
>>> L1=[2,4,1,6,5,7]
>>> L2=L1
>>> L1.append(3)
>>> L2
[2, 4, 1, 6, 5, 7, 3]
```

Pour créer une copie de liste indépendente, il faut utiliser la fonction : `L2=list(L1)`