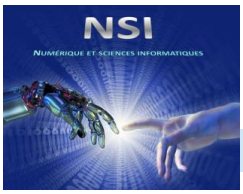

CHAPITRE 6 : LISTES

Nous avons étudié au chapitre précédent les chaines, qui sont composées de caractères. La chaine est une séquence **non-modifiable** car il n'est pas possible de changer les caractères au sein d'une chaine existante. Nous allons étudier un autre type de donnée, les listes, qui sont des séquences **modifiables**. Ensuite nous étudierons dans un prochain chapitre les tuples et les dictionnaires.

LISTES

Définition

Les listes sont des séquences, comme les chaines de caractères, mais au lieu de contenir des caractères, elles peuvent contenir n'importe quel objet. A la différence des chaines, les listes sont **modifiables** (on parle d'objets « mutables »). .

Remarque

On délimite une liste à l'aide de **crochets**.
On peut avoir une liste contenant des nombres entiers, des flottants, des chaines de caractères... et une liste mélangeant des objets de différents types. On peut même avoir une liste contenant d'autres listes.
Les termes de la liste sont ordonnés. On peut les compter et les repérer par leur rang **(ou indice) qui commence à 0**.

Premières fonctions

- **len(L)** renvoie le nombre d'éléments de la *liste* L.
- **L[i]** renvoie l'élément d'indice i de la liste L.
  Attention **le premier indice est 0**, donc **le dernier indice est len(L) - 1**
- **del(L[i])** supprime l'élément d'indice i de la liste L.

Exemples:

```
>>> nombres = [3,5,7,9,11,13,15]
>>> type(nombres)
<class 'list'>
>>> len(nombres)
7
>>> nombres[4]
11
>>> nombres[0]
3
>>> nombres[1]= 4
>>> nombres
[3,4,7,9,11,13,15]
```

Opérations

L[*deb* : *fin*] donne une liste de l'élément d'indice ***deb*** à celui d'indice ***fin-1***.

L[::-*1*] donne la liste en inversant l'ordre des éléments.

L*1* + L*2* fusionne les listes L*1* et L*2*.

L∗ *n* répète n fois la liste L

Exemples:

```
>>> lettres = ['A','Z','E','R','T','Y']
>>> lettres[2:4]
['E','R']
>>> lettres[::-1]
['Y','T','R','E','Z','A']
```

Parcourir et rechercher

- for *element* **in** L **:** Boucle qui parcourt directement les éléments d'une liste L

- *x* **in** L **:** Booléen qui vaut True si x est dans la liste L et False sinon.

Exemples:

Les deux scipts ci-dessous permettent de parcourir tous les éléments d'une liste, le premier en parcourant directement les éléments et le deuxième en passant par les indices.

```
tab = [5, 8, 6, 9]
for element in tab:
    print(element)
```

```
tab = [5, 8, 6, 9]
for i in range(len(tab)):
    print(tab[i])
```

Attention

Danger avec la copie de listes !!!!

Si vous utilisez une simple affectation pour copier une liste telle que : *L2 = L1*
À la suite de cette instruction, il n'existera toujours qu'une seule liste dans la mémoire de l'ordinateur.
Ce que vous avez créé est seulement une nouvelle référence vers cette liste.
Ainsi, **les modifications de l'une sont répercutées dans l'autre, et vice-versa.**

```
>>> L1 = [1,2,3]
>>> L2=L1
>>> L2|0]=4
>>> L1
[4,2,3]
```

Solution Pour dupliquer une liste indépendante

```
L1=[1,2,3]
L2=[]
for i in L1:
    L2.append(i)
```

Astuce Pour avoir la même chose

```
L1=[1,2,3]
L2=L1[:]
```

Fonctions

Sous Python, les listes sont des objets à part entière, et vous pouvez donc leur appliquer un certain nombre de ***méthodes*** particulièrement efficaces :

append, **insert,, remove,, index,, count,, reverse,, sort**

append : .....

insert : .....

remove : .....

index : .....

count : .....

reverse : .....

sort : .....

Listes en compréhension :

La compréhension de liste est un outils très puissant permettant de créer des listes à partir de listes existantes. En Python, les compréhensions de liste sont construites comme suit:

nouvelle_liste = [fonction(x) for x in liste]

Exemple 1:
Pour créer classiquement une liste contenant tous les doubles d'une liste L, on utilise 3 lignes **:**

```
resultat = []
for i in L:
    resultat.append(i*2)
```

En utilisant la compréhension de liste, on obtient la même chose en une seule ligne :

```
resultat = [i*2 for i in L]
```

Remarque:
On peut aussi ajouter des conditions... **nouvelle_liste = [fonction(x) for x in liste if condition]**

Exemple 2:
Pour créer une liste contenant tous les nombres supérieurs à 50 d'une liste L, on utilise 4 lignes :

```
resultat = []
for x in L:
    if x>50:
resultat.append(x)
```

En utilisant la compréhension de liste : *resultat = [x for x in L if x >50]*

Tableaux : listes à deux dimensions (liste de listes)

Un tableau comme le tableau ci-dessous sera représenté en python par une liste de listes :

1	2	3
4	5	6
7	8	9

mon_tableau = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

Accès aux éléments du tableau :

- Accès à un élément : *mon_tableau[1][2]* donne ..... ……
- Accès à une ligne : *mon_tableau[1]* donne .....……

Ajout d'une ligne : on ajoute une liste à la fin de la liste de listes

mon_tableau.append([10, 11, 12])

Exemple :

Créer un tableau de 4 lignes et 3 colonnes de nombres aléatoires entre 1 et 9 **:**

```
tab = []
for i in range(4):
    ligne=[]
    for j in range(3):
        ligne.append(randint(1,9))
    tab.append(ligne)
```

En utilisant la compréhension de liste : *L = [[randint(1,9) for j in range(3)] for i in range(4)]*