

## Chapitre 1 : Parcours des listes

### I- Vocabulaire

- Une liste est itérable ce qui signifie que chaque élément de cette dernière peut être lue.
- Une liste est indicable ce qui signifie que chaque élément possède un indice.

### II- Présentation des listes (AZE7KQ)

Création d'une liste <i>L</i> vide	$L = []$						
Création d'une liste <i>L</i> par extension (avec des éléments souhaités)	$L = [elt_1, elt_2, elt_3, \dots, elt_n]$ Il s'agit d'une liste de $n$ éléments et donc de longueur $n$ ( $len(L) = n$ ) <table border="1" style="margin-left: 20px;"> <tr> <td>Liste</td> <td><math>L = [elt_1, elt_2, \dots, elt_{n-1}, elt_n]</math></td> </tr> <tr> <td>Index</td> <td>0 1 ..... <math>n-2</math> <math>n-1</math></td> </tr> <tr> <td>Index</td> <td><math>-len(L)</math> <math>-len(L)+1</math> -2 -1</td> </tr> </table>	Liste	$L = [elt_1, elt_2, \dots, elt_{n-1}, elt_n]$	Index	0 1 ..... $n-2$ $n-1$	Index	$-len(L)$ $-len(L)+1$ -2 -1
Liste	$L = [elt_1, elt_2, \dots, elt_{n-1}, elt_n]$						
Index	0 1 ..... $n-2$ $n-1$						
Index	$-len(L)$ $-len(L)+1$ -2 -1						
Obtenir un élément d'une liste	<ul style="list-style-type: none"> <li>- <math>L[0]</math> spécifie la valeur du 1<sup>e</sup> élément</li> <li>- <math>L[1]</math> spécifie la valeur du 2<sup>e</sup> élément</li> <li>- Dernier élément <math>L[-1]</math> ou <math>L[len(L) - 1]</math></li> <li>- <math>elt</math> in <math>L</math> renvoie True si <math>elt</math> est dans <math>L</math></li> </ul>						
Obtenir une sous liste $L1$ à partir d'une liste $L$	$L1 = L[i:j:pas]$ <ul style="list-style-type: none"> <li>- <math>i</math> est l'indice du 1<sup>e</sup> élément inclus</li> <li>- <math>j</math> est l'indice du dernier élément exclu</li> <li>- <math>pas</math> est l'intervalle entre deux valeurs d'indice</li> </ul> Rq : par défaut $pas = 1$ si on écrit $L1 = L[i:j]$						
Modifier un ou des éléments d'une liste	<ul style="list-style-type: none"> <li>- <math>L[i] = a</math> affecte la valeur <math>a</math> à l'indice <math>i</math></li> <li>- <math>L[i:j+1] = L_2</math> affecte la liste <math>L_2</math> dans <math>L</math> entre l'indice <math>i</math> et l'indice <math>j</math> (inclus)</li> </ul>						
Opérations sur les listes	<ul style="list-style-type: none"> <li>- La méthode <code>append</code> : <math>L.append(elt)</math> permet de rajouter <math>elt</math> à la fin de la liste</li> <li>- Concaténation de deux listes <math>L_1</math> et <math>L_2</math> : <math>L_1 + L_2</math></li> <li>- <math>L_1 * 3 \equiv L_1 + L_1 + L_1</math></li> </ul>						

### III- Parcours de listes à l'aide de boucles (MZQ4Q8)

On distingue deux types de boucles :

- Les boucles non conditionnelles « for » : ces boucles (bornées) permettent d'itérer une liste un nombre fini de fois.

```
somme=0#initialisation
L=[0,1,2,3,4,5]#création de la liste
for i in L :#itération
    somme=somme+i# typage dynamique =>on obtient la somme des éléments de la liste
```

On utilise souvent la fonction range (début, fin (exclue), pas), on peut aussi écrire range(fin) qui impose un pas de 1 et un début à 0 par défaut.

```
L=[]
for i in range(0,5,1):
    L.append(i)
```

```
L=[]
for i in range(5):
    L.append(i)
```

- Les boucles conditionnelles « while » : La boucle while est utilisée pour répéter une séquence d'instructions tant qu'une condition est vérifiée. Il est donc essentiel d'atteindre cette condition pour stopper cette boucle.

```
somme=0#initialisation
L=[0,1,2,3,4,5]#création de la liste
i=0#initialisation
while i<len(L):
    somme=somme+L[i]
    i=i+1#essentiel pour stopper la boucle!
```

Rq : ici  $i$  est un compteur initialisé à 0, qui permet par exemple de connaître le nombre de passages dans une boucle while.

**Commenté [AM1]:** Une liste est un exemple de séquence ou structure de données. Les chaînes, les tuples, les dictionnaires, les tableaux numpy sont aussi itérables

**Commenté [AM7]:** Attention les instructions conditionnelles `if`, `elif`, `else` ne sont pas de boucles.

**Commenté [AM2]:** Typiquement avec une boucle for

**Commenté [AM3]:** On peut aussi créer une liste de liste : `liste[i][j]` est l'élément d'indice  $j$  dans la liste d'indice  $i$

**Commenté [AM4]:** On peut donc générer des listes :  
 - Par extension `L=[0,1,2,3,4]`  
 - Par conversion `L=list(range(5))`  
 - par compréhension :  
`liste=[i**2 for i in range(5) if i%2==0]`  
 - Par ajout successif avec une boucle for et la méthode `append`

**Commenté [AM8]:** `L.append(elt)` rajoute un `elt` à la fin de `L`  
`L.pop()` a pour valeur `L[-1]` et enlève le dernier élément de `L`

**Commenté [AM9]:** Instruction `=` Morceau de code minimal qui produit un effet

**Commenté [AM10]:** Il est possible d'utiliser l'instruction `break`

```
somme=0#initialisation
L=[0,1,2,3,4,5,6,7,8,9,10]#création de la liste
i=0#initialisation
while i<len(L):
    somme=somme+L[i]
    i=i+1#essentiel pour stopper la boucle!
    if i>4:
        break
print(somme)#10
```

**Commenté [AM5]:** Une liste est donc mutable

**Commenté [AM6]:** Ce remplacement nécessite `len(L2) = len(L[i:j+1])`

**Commenté [AM11]:** Un accumulateur est semblable à un compteur mais il peut être incrémenté d'une valeur différente de 1 ou décrémenté.