

CHAPITRE 5 : LES LISTES ET LES CHAÎNES DE CARACTÈRES**I. Les listes.****1.1 Découpage en tranches d'une liste (slicing).**

Le découpage d'une liste permet de récupérer une portion de la liste à partir de l'indice des éléments. Le découpage basique d'une liste consiste à indexer une liste avec deux entiers séparés par deux points. Cela renvoie une **nouvelle liste** contenant toutes les valeurs de l'ancienne liste entre les indices.

```
>>> carres=[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> carres[2:6] #Attention le premier entier est inclus et le second exclus
[4, 9, 16, 25]
>>> carres[0:1]
[0]
>>> #si le premier entier est omis, on commence au début de la liste
>>> carres[:7] #le second indice est exclu !
[0, 1, 4, 9, 16, 25, 36]
>>> carres[7:] #si le deuxième entier est omis, on termine à la fin de la liste
[49, 64, 81]
>>> carres[:2] #le troisième entier correspond au pas
[0, 4, 16, 36, 64]
>>> carres[2:8:3] #le huitième élément est exclu !
[4, 25]
>>> carres[1:-1] #du deuxième à l'avant dernier élément
[1, 4, 9, 16, 25, 36, 49, 64]
>>> carres[::-1] #inversion de la liste !!
[81, 64, 49, 36, 25, 16, 9, 4, 1, 0]
```

1.2 Méthodes associées aux listes.

➤ **nom_list.append(x)** : ajoute **un seul élément x** à la fin de la liste.

```
>>> li=[1,2,3]
>>> li.append(4)
>>> li
[1, 2, 3, 4]
>>> id(li)
2989451425544
>>>
```

Ce qui est équivalent à

```
>>> li+=[5]
>>> li
[1, 2, 3, 4, 5]
>>> id(li)
2989451425544
>>>
```

➤ **nom_list.extend(L)** : étend la liste en y ajoutant **tous les éléments de la liste L** fournie.

```
>>> li.extend([6,7])
>>> li
```

```
[1, 2, 3, 4, 5, 6, 7]
>>> id(li)
2989451425544
>>>
```

```
>>> li=li+[8]
>>> li
[1, 2, 3, 4, 5, 6, 7, 8]
>>> id(li)
2989450544840
>>>
```

Les méthodes **append** et **extend** modifient la liste sur laquelle elles travaillent, alors que l'addition `li=li+[x]` crée un nouvel objet.

➤ **nom_list.insert(i,x)** : insère un seul élément `x` dans une liste avec un indice `i` déterminé.

```
>>> li.insert(-1,10) #insère 10 avant le dernier élément de la liste
>>> li
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>>
```

Remarques :

- **nom_list.insert(0, x)** insère l'élément en tête de la liste.
- **nom_list.insert(len(a),x)** est équivalent à **nom_list.append(x)**.

nom_list.remove(x) : pour supprimer un élément `x` d'une liste à partir de sa valeur.

```
>>> li.remove(5) #une erreur s'affiche si l'élément n'existe pas dans la liste
>>> li
[1, 2, 3, 4, 6, 7, 8, 9, 10, 11]
>>>
```

del nom_list[i] : pour supprimer un élément d'une liste à partir de son indice `i`.

```
>>> del li[3] #supprime le 4ème élément de la liste
>>> li
[1, 2, 3, 6, 7, 8, 9, 10, 11]
>>> del li[4:] #supprime les éléments à partir du 5ème élément de la liste ou li[4:]=[]
>>> li
[1, 2, 3, 6]
>>> del li[:] #supprime tous les éléments de la liste ou li[:]=[]
>>> li
[]
>>>
```

nom_list.pop(i) : enlève de la liste l'élément situé à la position indiquée, et le retourne.

```
>>> li
[1, 2, 3, 6]
>>> li.pop(1)
>>> 2
>>> li
[1, 3, 6]
>>>
```

Si aucune position n'est indiquée, `list.pop()` enlève et retourne le dernier élément de la liste.

nom_list.sort() : trie une liste par ordre croissant par défaut

```
>>> li=[4,5,4,1,3,2,5,7,5]
>>> li2=sorted(li) #la liste initiale n'est pas modifiée
>>> li
[4, 5, 4, 1, 3, 2, 5, 7, 5]
>>> li2
[1, 2, 3, 4, 4, 5, 5, 5, 7]
>>> li.sort() #la liste initiale est modifiée
>>> li
[1, 2, 3, 4, 4, 5, 5, 5, 7]
>>> li.sort(reverse=True) #trie la liste par ordre décroissant
>>> li
[7, 5, 5, 5, 4, 4, 3, 2, 1]
>>>
```

nom_list.reverse() : pour inverser une liste.

```
>>> li.reverse()
>>> li
[1, 2, 3, 4, 4, 5, 5, 5, 7]
>>>
```

nom_list.index(x) : donne l'indice d'un élément x.

```
>>> li.index(4)
3
>>>
```

Remarques :

- Si l'élément n'est pas trouvé, alors une erreur s'affiche.
- En cas de doublon, c'est le rang du premier élément qui est renvoyé.

nom_list.count(x) : compte le nombre d'éléments x dans une liste.

```
>>> li.count(5)
3
>>> li.count(9)
0
>>>
```

choice(nom_list) : choisit un élément aléatoire d'une liste non vide.

```
>>> import random
>>> random.choice(li)
2
>>>
```

shuffle(nom_list) : mélange les éléments d'une liste. (Importer le module random)

```
>>> random.shuffle(li)
>>> li
[7, 4, 5, 3, 4, 5, 2, 1, 5]
```

sample(nom_list,k) : génère un échantillon de taille k d'une liste.

```
>>> import random
>>> random.sample(li,3)
```

```
[2, 3, 7]
>>>
```

La méthode `append()` est particulièrement pratique car elle permet de construire une liste au fur et à mesure des itérations d'une boucle. Pour cela, il est commode de définir préalablement une liste vide de la forme `li=[]`. Voici un exemple où une chaîne de caractères est convertie en liste :

```
>>> seq= 'CAAAGGTAACGC'
>>> li=[]
>>> li
[]
>>> for el in seq:
    li.append(el) #copie de la liste seq dans la liste li

>>> li
['C', 'A', 'A', 'A', 'G', 'G', 'T', 'A', 'A', 'C', 'G', 'C']
>>>
```

Remarquez que vous pouvez directement utiliser la fonction `list()` qui prend n'importe quel objet séquentiel (liste, chaîne de caractères, etc.) et qui renvoie une liste :

```
>>> seq = 'CAAAGGTAACGC'
>>> list(seq) #permet de convertir une chaîne de caractères en liste de caractères
['C', 'A', 'A', 'A', 'G', 'G', 'T', 'A', 'A', 'C', 'G', 'C']
>>>
```

Cette méthode est certes plus simple, mais il arrive parfois que l'on doive utiliser les boucles tout de même, comme lorsqu'on lit dans un fichier.

1.3 Test d'appartenance.

L'inscription `in` permet de tester si un élément fait partie d'une liste.

```
>>> li = [1, 3, 5, 7, 9]
>>> 3 in li
True
>>> 4 in li
False
>>> 3 not in li
False
>>> 4 not in li
True
```

1.4 Copie de listes.

L'affectation d'une liste à partir d'une liste préexistante crée en réalité une copie de la liste initiale **par référence** et non une copie **par recopie** :

```
>>> x = [1,2,3]
>>> y = x
>>> y
[1, 2, 3]
>>> x[1] = -15
```

```
>>> y
[1, -15, 3]
```

Vous voyez que la modification de x modifie y aussi. Rappelez-vous de ceci dans vos futurs programmes car cela pourrait avoir des effets désastreux ! Techniquement, Python utilise des pointeurs (comme dans le langage C) vers les mêmes objets et ne crée pas de copie à moins que vous n'en ayez fait la demande explicitement. Exemple :

```
>>> x = [1,2,3]
>>> y = x[:]
>>> x[1] = -15
>>> y
[1, 2, 3]
```

Dans l'exemple précédent, x[:] a créé une copie **par recopie** de la liste x. Vous pouvez utiliser aussi la fonction list() qui renvoie explicitement une liste :

```
>>> x = [1,2,3]
>>> y = list(x)
>>> x[1] = -15
>>> y
[1, 2, 3]
```

Attention, les deux techniques précédentes ne fonctionnent que pour les listes à une dimension, autrement dit les listes qui ne contiennent pas elles-mêmes d'autres listes.

```
>>> x = [[1,2],[3,4]]
>>> x
[[1, 2], [3, 4]]
>>> y = x[:]
>>> y[1][1] = 55
>>> y
[[1, 2], [3, 55]]
>>> x
[[1, 2], [3, 55]]
>>> y = list(x)
>>> y[1][1] = 77
>>> y
[[1, 2], [3, 77]]
>>> x
[[1, 2], [3, 77]]
```

La méthode de copie qui **marche à tous les coups** consiste à appeler la fonction **deepcopy()** du module copy.

```
>>> import copy
>>> x = [[1,2],[3,4]]
>>> x
[[1, 2], [3, 4]]
>>> y = copy.deepcopy(x)
>>> x[1][1] = 99
>>> x
[[1, 2], [3, 99]]
>>> y
[[1, 2], [3, 4]]
```

1.5 Liste de compréhension.

Une des caractéristiques les plus puissantes de Python est la liste de compréhension qui fournit un moyen concis d'appliquer une fonction sur chaque élément d'une liste afin d'en produire une nouvelle.

Exemples :

➤ Liste des 15 premiers carrés parfaits.

```
>>> carres=[i**2 for i in range(15)]
>>> print(carres)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196]
```

La compréhension de listes évite donc d'écrire le code "classique" suivant :

```
carres = []
for i in range(11):
    carres.append(i*i)
```

➤ Liste des 10 premiers carrés pairs parfaits.

```
>>> li=[i**2 for i in range(20) if i%2==0]
>>> print(li)
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

➤ Simulation de 10 lancers d'un dé à 6 faces.

```
de = [random.randint(1,6) for i in range(10)]
>>> print(de)
[3, 1, 5, 6, 4, 2, 1, 1, 3, 1]
```

➤ Liste des nombres impairs :

```
[n for n in range(1,40,2)]
[2*n+1 for n in range(20)]
[n for n in range(40) if n%2==1]
```

II. Les chaînes de caractères.

2.1 Exemples.

Les chaînes de caractères peuvent être considérées comme des listes.

```
>>> ch='Stephane KELLER'
>>> ch
'Stephane KELLER'
>>> len(ch)
15
>>> ch[3]
'p'
>>>
```

Nous pouvons utiliser les mêmes propriétés des listes concernant le découpage (slicing) :

```
>>> ch[0:4]
'Step'
>>> ch[9:]
'KELLER'
>>> ch[:5]
```

```
'Stephane K'
>>> ch[-1]
>>> R
>>>
```

A contrario des listes, les chaînes de caractères présentent toutefois une différence notable, ce sont des listes non modifiables. Une fois définie, vous ne pouvez plus modifier un de ses éléments. Le cas échéant, Python renvoie un message d'erreur.

```
>>> ch[9]='M'
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    nom[9]='M'
TypeError: 'str' object does not support item assignment
>>>
```

Par conséquent, si vous voulez modifier une chaîne, vous êtes obligés d'en construire une nouvelle. Pour cela, n'oubliez pas que les opérateurs de concaténation (+) et de duplication (×) peuvent vous aider. Vous pouvez également générer une liste, qui elle est modifiable, puis revenir à une chaîne.

2.2 Caractères spéciaux.

Il existe certains caractères spéciaux comme :

- Le retour à la ligne : \n (newline) ;
- La tabulation : \t (tabulation) ;
- Le caractère d'échappement : \ qui permet d'écrire un guillemet simple ou double (et que celui-ci ne soit pas confondus avec les guillemets de déclaration de la chaîne de caractères), vous pouvez utiliser \' ou \" ou utiliser respectivement des guillemets doubles ou simple pour déclarer votre chaîne de caractères.

```
>>> ch='Stephane\nKELLER\tEnseignant d'informatique'
>>> ch
"Stephane\nKELLER\tEnseignant d'informatique"
>>> print(ch)
Stephane
KELLER      Enseignant d'informatique
>>>
```

2.3 Méthodes associées aux chaînes de caractères.

lower() et **upper()** : passe un texte en minuscule et en majuscule respectivement. On remarque que l'utilisation de ces fonctions n'altère pas la chaîne de départ mais renvoie la chaîne transformée.

```
>>> ch='Stephane KELLER'
>>> ch.lower()
'stephane keller'
>>> ch.upper()
'STEPHANE KELLER'
>>>
```

Pour mettre en minuscule la première lettre seulement, vous pouvez faire :

```
>>> ch[0].lower()+ch[1:]
'stephane KELLER'
>>>
```

split() : pour fendre une chaîne de caractères en champs, en utilisant comme séparateur les espaces ou les tabulations. Il est possible de modifier le séparateur de champs :

```
>>> ch='Stephane KELLER'
>>> ch.split()
['Stephane', 'KELLER']
>>> ch.split('e') #Attention à la casse
['St', 'phan', ' KELLER']
>>>
```

find() ou **index()** : pour avoir le rang d'un élément.

```
>>> ch='Stephane KELLER'
>>> ch.find('K')
9
>>>
>>> ch.index('L')
11
>>>
```

Si l'élément recherché est trouvé, alors l'indice du début de l'élément dans la chaîne de caractères est renvoyé. Si l'élément n'est pas trouvé, alors la valeur -1 est renvoyée. En cas de doublon, c'est le rang du premier élément qui est renvoyé.

replace() : pour substituer des éléments d'une chaîne de caractère.

```
>>> ch.replace('e','o') #Attention à la casse
'Stophano KELLER'
>>>
```

count() : pour **compter** le nombre d'éléments (passé en argument) dans une chaîne de caractère.

```
>>> ch='Stephane KELLER'
>>> ch.count('e')
2
>>> ch.count('z')
0
>>>
```

join() : Jointure de liste.

```
>>> li=['C', 'A', 'A', 'A', 'G', 'G', 'T', 'A', 'A', 'C', 'G', 'C']
>>> ''.join(li) #permet de convertir une liste de caractères en chaîne de caractères
'CAAAGGTAACGC'
>>>
```

Programme équivalent :

```
>>> li=['C', 'A', 'A', 'A', 'G', 'G', 'T', 'A', 'A', 'C', 'G', 'C']
>>> ch=""
>>> for el in li:
>>>     ch+=el
>>> ch
```



```
'CAAAGGTAACGC'
```

```
>>>
```

III. Liste non exhaustive des avantages et inconvénients des listes et des chaînes de caractères.

Avantages	Inconvénients
liste	
<ul style="list-style-type: none"> ➤ Les éléments sont modifiables. ➤ On peut ajouter facilement un élément avec la méthode append(). ➤ On peut avoir des éléments de plusieurs types. ➤ On peut créer une liste de listes, qui s'apparente à un tableau à 2 dimensions. ➤ Tri immédiat des éléments d'une liste avec la méthode sort(). ➤ Inversion des éléments d'une liste avec la méthode reverse() ; permet de tester l'existence d'un palindrome. ➤ ... 	<ul style="list-style-type: none"> ➤ Nécessité de l'utilisation d'une boucle For pour l'affichage propre du contenu de la liste.
Chaîne de caractère	
<ul style="list-style-type: none"> ➤ L'affichage propre du contenu avec la méthode print(). ➤ Des méthodes comme lower(), upper() directement utilisables avec une chaîne de caractères. ➤ Possibilité d'utiliser certaines des opérations mathématiques avec des chaînes de caractères pour dupliquer une chaîne (×) ou concaténer (+) deux chaînes. 	<ul style="list-style-type: none"> ➤ Un élément n'est pas modifiable. ➤ Nécessité d'utiliser la concaténation pour ajouter un élément. ➤ Nécessité d'utiliser les techniques de "slicing" pour travailler sur une partie de la chaîne de caractères. ➤ Pas de tri des éléments d'une chaîne de caractères.