CH 4 : LES LISTES

1.1 Les chaînes de caractères

Dans une chaîne de caractères, chaque « lettre » est repérée par sa position dans la chaîne, le premier caractère ayant la position 0.

```
>>> mot="bonjour"
>>> print (mot[0])
b
```

Faire afficher dans la console la troisième lettre de mot.

On peut facilement décomposer une chaîne de caractère :

```
>>> for lettre in mot:
...     print (lettre)
...
b
o
n
j
o
u
r
```

On peut également procéder ainsi :

```
>>> for i in range(len(mot)):
...     print (mot[i])
...
b
o
n
j
o
u
r
```

En fait, Python considère toute chaîne de caractère comme une liste. Nous allons voir maintenant ce que sont les listes.

1.2 Généralités et notation

Les listes (souvent appelées tableaux dans les autres langages) vont nous permettre de stocker plusieurs valeurs (chaîne, nombre) dans une structure unique.

Pour lire le contenu d'une liste, il suffit d'utiliser un « indice de position » (le 1er élément de la liste à l'indice 0, le 2ème élément à l'indice 1, etc...). On note : **vecteur[indice de position]**

Quel est le résultat attendu après l'exécution du programme ci-contre ? Vérifier.

```
liste=[1 , 7 , 9 , -4 , 3 , 2]
print(liste[0])
print(liste[1])
print(liste[0]+liste[4])
liste[2]=liste[4]*liste[5]
print(liste)
```

ATTENTION : Oublier que l'indice de position du 1er élément d'une liste est 0 et pas 1 est une erreur « classique ».

Une liste peut contenir toute sorte d'objets :

```
liste1=[1 , 2 , 3] #liste d'entiers
liste2=[11.2 , 2.5 , 5.0 , 3.2] #liste de flottants
liste3=[[1 , 2] , [1] , [2 , 5 , 7]] #liste de listes
```

Quel est le résultat après exécution des deux programmes ci-dessous ?

```
liste=["pomme" , "poire" , "banane" , "peche"]
liste[0]=liste[1] + " " + liste[2]
print(liste)
```

Et

```
liste_mixte=[2 , 1.45 , 'un torchon' , 'une serviette']
print (liste_mixte[2])
liste_mixte[2]=liste_mixte[2]+liste_mixte[3]
print(liste_mixte)
```

On peut également utiliser des indices négatifs : -1 est alors l'indice du dernier élément de la liste et on « part vers la gauche ».

Quel est le résultat après exécution du programme ci-dessous ?

```
liste=[1 , 2 , 3 , 4 , 5]
print(liste[-1])
print(liste[-4])
```

1.3 Créer une liste par compréhension

Il est possible de créer une liste de manière plus efficace qu'en écrivant tous ses éléments (écriture en extension), c'est la méthode par compréhension :

L'instruction suivante :

```
liste=[i for i in range(10)]
```

crée une liste contenant tous les entiers de 0 à 9 :

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Que contiennent liste1, liste2, et liste3 après exécution des instructions suivantes ?

```
n=5
liste1=[i**2 for i in range(n)]

liste2=[i/2 for i in range(n)]

liste3=[i for i in range(1 , 10 , 2)]
```

Créer une liste par compréhension contenant les cubes des 4 premiers entiers naturels.

On peut également utiliser une fonction pour créer une liste comme ci-dessous:

```
>>> def f(x):
...     return x**2-3*x+2
...
>>> liste1=list(range(-6 , 10))
>>> liste2=[f(u) for u in liste1]
>>>
>>> print (liste2)
[56, 42, 30, 20, 12, 6, 2, 0, 0, 2, 6, 12, 20, 30, 42, 56]
```

1.4 Opérations sur les listes

Concaténation et duplication

La concaténation de listes consiste à mettre bout à bout ces deux listes. Elle se note « + ».

La duplication consiste à écrire plusieurs fois à la suite la liste. Elle se note « * ».

```
>>> [2 , 4 , 6 , 8] + [10 , 12 , 14]
[2, 4, 6, 8, 10, 12, 14]
>>>
>>> [1 , 2 , 3]*3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>>
```

Ajout d'un élément en fin de liste

Pour ajouter un élément à la fin d'une liste, on utilise la **méthode** .append().

Tester le programme suivant :

```
liste=[]
liste.append(4)
print(liste)
```

Recherche de la présence d'un élément dans une liste

Pour savoir si un élément fait partie d'une liste, on peut utiliser « in » :

```
liste=[2 , 3 , 4]
>>> 5 in liste
False
>>> 2 in liste
False
```

Et bien d'autres choses …

Le tableau suivant donne une liste non exhaustive des opérations que l'on peut faire sur les listes …

Type	Commande	Affichage ou résultat
Créer une liste vide	`L = []`	`[]`
Créer une liste de `n` zéros	`L = [0]*n`	`[0,.,0]`
Créer une liste en compréhension : obtenir un ensemble d'entiers	`L=[i for i in range(n)]`	`[0,1,.,n-1]`
Créer une liste en compréhension : obtenir un ensemble de réels	`L=[i*0.1 for i in range(n)]`	`[0.0,.,(n-1)*0.1]`
Appel de l'élément `i` de la liste	`liste[i]`	élément d'indice `i` de la liste
Appel de l'élément `(i,j)` d'une liste de listes	`liste[i][j]`	élément d'indices `(i,j)` de la liste
Appel du 1er élément d'une liste	`liste[0]`	1er élément de la liste
Appel du dernier élément d'une liste	`liste[-1]`	dernier élément de la liste
Affecter la valeur `n` à l'élément `i` de la liste	`liste[i] = n`	`liste[i]` vaut `n` : `[.,n,.]`
Afficher le nombre d'éléments de la liste	`len(liste)`	longueur de la liste
Concaténation de deux listes avec l'opérateur +	`liste1 + liste2`	une seule liste contenant `liste1` et `liste2` mises bout à bout

Type	Commande	Affichage ou résultat
Duplication d'une liste avec l'opérateur *	`liste*n`	une seule liste contenant `n` fois `liste`
Ajout de `n` en fin de liste par la méthode append	`liste.append(n)`	`liste` vaut `[.,.,.,n]`
Ajout de `n` en fin de liste	`liste + [n]`	`liste` vaut `[.,.,.,n]`
Suppression d'un élément liste[i]	`del liste[i]`	Supprime `liste[i]`