Python : Les listes

Définition

Une liste est une structure de données qui contient une série de valeurs.

Les valeurs contenus dans une liste s'appellent aussi les **éléments** de cette liste. Les éléments sont modifiables et pas forcémentdu même type. Pour créer une liste on écrit les valeurs entre **crochets** et séparées par des *virgules*.

Exemples :

```
# Création d'une liste vide
vide = []

# Création d'une liste avec des string
animaux = ["girafe", "tigre", "singe", "souris"]

# Création d'une liste avec de nombres (int et float)
tailles = [5, 2.5, 1.75, 0.15]

# Création d'une liste avec plusieurs types de données
mixte = ["girafe", 5, "souris", 0.15]
```

Afficher un élément d'une liste

Les éléments d'une liste peuvent être afficher en utilisant leur ***indice*** (ou index).

Les indices positifs :

```
liste  : ["girafe", "tigre", "singe", "souris"]
indice :        0        1        2         3
```

Exemples :

```
animaux = ["girafe", "tigre", "singe", "souris"]

print(animaux[0])
    > girafe

print(animaux[3])
    > souris

print(animaux[4])
    > Traceback (most recent call last):
       File "main.py", line 4, in <module>
         print(animaux[4])
      IndexError: list index out of range
```

Les indices négatifs :

```
liste          : ["girafe", "tigre", "singe", "souris"]
indice positif :        0        1        2         3
indice négatif :       -4       -3       -2        -1
```

Les indices négatifs reviennent à compter à partir de la fin. Leur principal avantage est que vous pouvez accéder au dernier élément d'une liste à l'aide de l'indice `-1` sans pour autant connaître la longueur de cette liste.

Exemples :

```
animaux = ["girafe", "tigre", "singe", "souris"]
print(animaux[-1])
    > "souris"
print(animaux[-2])
    > "singe"
```

Afficher une tranche d'éléments

Un autre avantage des listes est la possibilité de sélectionner une partie d'une liste en utilisant un indiçage construit sur le modèle `[m:n+1]` pour récupérer tous les éléments, du émième au énième (de l'élément `m` inclus à l'élément `n+1` exclu). On dit alors qu'on récupère une **tranche** de la liste

Exemples :

```
animaux = ["girafe", "tigre", "singe", "souris"]

print(animaux[0:2])
    > ['girafe', 'tigre']

print(animaux[1:])
    > ['tigre', 'singe', 'souris']

print(animaux[:])
    > ['girafe', 'tigre', 'singe', 'souris']

print(animaux[1:-1])
    > ['tigre', 'singe']
```

On peut aussi préciser le pas en ajoutant un symbole deux-points supplémentaire et en indiquant le pas par un entier.

Syntaxe : `liste[début:fin:pas]`

Exemples :

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

print(x[::2])
    > [0, 2, 4, 6, 8]

print(x[::3])
    > [0, 3, 6, 9]

print(x[1:6:3])
    > [1, 4]
```

Operations sur les listes

Tout comme les chaînes de caractères, les listes supportent l'opérateur **+** de concaténation, ainsi que l'opérateur ***** pour la duplication :

Exemples :

```
ani1 = ["girafe", "tigre"]
ani2 = ["singe", "souris"]

print(ani1 + ani2)
    > ['girafe', 'tigre', 'singe', 'souris']

print(ani1 * 3)
    > ['girafe', 'tigre', 'girafe', 'tigre', 'girafe', 'tigre']
```

Ajouter un élément dans une liste

Il existe deux méthodes pour ajouter un élément dans un liste :

- avec l'opérateur **+** de concaténation

```
mylist = [2,77,32]

mylist = mylist + [15]
print(mylist)
    > [2,77,32,15]
```

- avec la fonction **.append()**

  Syntaxe : `name_list.append(new_value)`

```
mylist.append(88)
print(mylist)
    > [2,77,32,15,88]
```

Supprimer un élément dans une liste

Il existe deux méthodes pour supprimer un élément dans un liste :

- avec **del** et l'indice

```
lettres = ["a", "b", "c","d"]

del lettres[1]
print(lettres)
    > ['a', 'c', 'd']
```

- avec la fonction **.remove()**

  Syntaxe : `name_list.remove(del_value)`

```
lettres.remove("a")
print(lettres)
    > ['c', 'd']
```

La fonction **.clear()** permet d'effacer tous les éléments d'une liste qui devient vide.

Syntaxe : `name_list.clear()`

```
lettres = ["a","b","c","d"]

lettres.clear()
print(lettres)
    > []
```

Modifier des éléments dans une liste

Les listes sont une structure de données dont les éléments peuvent être modifiés.

Elles sont **mutables**.

Exemples :

```
lettres = ["A","B","B","D","E"]

lettres[2] = "C"
print(lettres)
    > ['A', 'B', 'C', 'D', 'E']
```

Insérer un élément dans une liste

Pour insérer un élément d'une liste, on utilise la fonction **.insert()**

Syntaxe : `name_list.insert(indice, value)`

Exemple :

```
lettres = ["a","b","d","e"]

lettres.insert(2,"c")
print(lettres)
    > ['a', 'b', 'c', 'd', 'e']
```

Inverser les éléments d'une liste

Pour inverser les éléments d'une liste, on utilise la fonction **.reverse()**

Syntaxe : `name_list.reverse()`

Exemple :

```
lettres = ["a", "b", "c","d"]

lettres.reverse()
print(lettres)
    > ['d', 'c', 'b', 'a']
```

Trier les éléments d'une liste

Pour trier les éléments d'une liste il existe deux méthodes :

- la fonction **.sort()**

  Syntaxe : `name_list.sort()`

Exemple :

```
lettres = ["c","a","d","b"]

lettres.sort()
print(lettres)
    > ['a', 'b', 'c', 'd']
```

- la fonction **sorted()**

  Syntaxe : `sorted(name_list)`

Exemple :

```
lettres = ["c","a","d","b"]

sort_lettres = sorted(lettres)
print(sort_lettres)
    > ['a', 'b', 'c', 'd']
```

Compter le nombre d'élément d'une liste : la longueur de la liste

Pour trouver la longueur de la liste, on utilise la fonction **len()**

Syntaxe : `len(name_list)`

Exemple :

```
lettres = ["a", "b", "c","d"]

print(len(lettres))
    > 4
```

Compter le nombre d'occurence d'une valeur

Pour compter le nombre d'occurence d'un élément dans une liste, on utilise la fonction **.count()**

Syntaxe : `name_list.count(value)`

Exemple :

```
lettres = ["a","a","a","b","c","c"]

print(lettres.count("a"))
    > 3
print(lettres.count("c"))
    > 2
```

Trouver l'indice d'une valeur

Pour connaître la position et donc l'indice d'un élément dans une liste, on utilise la fonction **.index()**

Syntaxe : `name_list.index(value)`

Exemple :

```
chiffre = ["a","b","c","c","d"]

print(chiffre.index("b"))
    > 1
print(chiffre.index("c"))
    > 2
```

La fonction range() et la fonction list()

La fonction **range()** génère une liste composée d'une simple suite de nombres entiers.

Syntaxe : range(debut, fin, pas)

Par défault, la fonction range génère un tuple. Pour avoir une liste, on utilise la fonction **list()**

Exemples :

- **range(a)** → Suite de valeurs de 0 à a

```
print(list(range(12)))
    > [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

- **range(a, b)** → Suite de valeur de a à b-1

```
print(list(range(4,12)))
    > [4, 5, 6, 7, 8, 9, 10, 11]
```

- **range(a, b, c)** → Suite de valeur de a à b-1 par pas de c

```
print(list(range(4,12,2)))
    > [4, 6, 8, 10]

print(list(range(2,-2,-1)))
    > [2, 1, 0, -1]
```

Les fonctions max(), min() et sum()

Les fonctions **min()**, **max()** et **sum()** renvoient respectivement le minimum, le maximum et la somme d'une liste passée en argument.

Exemples :

```
my_list = list(range(10))
print(my_list)
    > [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(sum(my_list))
    > 45
print(min(my_list))
    > 0
print(max(my_list))
    > 9
```

Transformer un string en liste

Pour transformer une chaîne de caractère en liste, on utiliste la fonction **.split()**

Syntaxe : `name_list.split(sep)`

Exemple :

```
my_str = "Prenom:NOM:Ville"
print(my_str)
    > Prenom:NOM:Ville

print(my_str.split(":"))
    > ['Prenom', 'NOM', 'Ville']
```

Transformer une liste en string

Pour transformer une liste en chaîne de caractère, on utiliste la fonction **.join()**

Syntaxe : `sep.join(name_list)`

Exemple :

```
my_list = ["Prenom","NOM","Ville"]
print(my_list)
    > ['Prenom', 'NOM', 'Ville']

print(":".join(my_list))
    > Prenom:NOM:Ville
```

Test pour trouver un item dans une liste

Pour savoir si un élément est dans une liste, vous pouvez utiliser le mot clé **in**

Syntaxe : `value in name_list`

Exemples :

```
my_liste = [1,2,3,5,10]

print(3 in my_liste)
    > True

print(11 in my_liste)
    > False
```

Listes de listes

Les listes peuvent être utilisée comme élément dans une autre liste.

```
enclos1 = ["girafe", 4]
enclos2 = ["tigre", 2]
enclos3 = ["singe", 5]
zoo = [enclos1, enclos2, enclos3]

print(zoo)
    > [['girafe', 4], ['tigre', 2], ['singe', 5]]
```

Pour accéder à un élément de la liste, on utilise l'indiçage habituel :

```
print(zoo[1])
    > ['tigre', 2]
```

Pour accéder à un élément de la sous-liste, on utilise un double indiçage :

```
print(zoo[1][0])
    > tigre

print(zoo[1][1])
    > 2
```

Remarque : Il existe d'autre structure de données permettant de mieux gérer les données dans ce genre de cas. Par exemple les **dictionnaires**.