NSI Première S Les Listes

I. Définition

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

Python autorise la construction de liste contenant des **valeurs de types différents** (par exemple entier et chaîne de caractères), ce qui leur confère une grande flexibilité.

Une liste est déclarée par une **série de valeurs** (n'oubliez pas les guillemets, simples ou doubles, s'il s'agit de chaînes de caractères) séparées par des **virgules**, et le tout encadré par des **crochets**. En voici quelques exemples :

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

tailles = [5, 2.5, 1.75, 0.15]

fruits = ["pomme", "orange", "fraise"]

mixte = ['girafe', 5, 'souris', 0.15]

liste_nombres = [56, 76, 45, 89]
```

II. Première approche

1. Pour **lire** le contenu d'une liste, il suffit **d'utiliser** un **« indice de position »** (le **1er élément** de la liste a **l'indice 0**, le deuxième élément a l'indice 1...).
   On aura : liste [indice de position].

   a) Soit le programme suivant :

   ```
   fruits =["pomme", "orange", "fraise"]
   print(fruits[1])
   ```

   Quel est le résultat attendu après l'exécution de ce programme ?
   Vérifiez votre réponse.

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

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

   b) Écrire une procédure qui prend en entrée comme paramètre un numéro de mois (de 1 à 12), puis affiche : « Vous avez sélectionné le mois de : xxxxxx » (avec xxxxxx janvier si l'utilisateur a choisi 1, août si l'utilisateur a choisi 8...).

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

Soit le programme suivant :

```
ani1 = ['girafe', 'tigre']
ani2 = ['singe', 'souris']
ani1 + ani2

ani1 * 3
```

Quels sont les résultats attendus après l'exécution de ce programme ?
Vérifiez votre réponse.

d) Soit le programme suivant :

```
Semaine = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']
I = 0
while i < 5:
      print(semaine[i])
      i = i + 1
print("C'est terminé")
```

Quel est le résultat attendu après l'exécution de ce programme ?
Vérifiez votre réponse.

e) La **boucle for** va nous permettre de parcourir les éléments d'une liste plus facilement qu'avec une boucle while.

Re écrire le programme de la question d) en utilisant une boucle for.
(*Vous pouvez utiliser la variable « jour »*).

2. Il est **possible** de connaître **la taille d'une liste** en utilisant l'instruction **len(nom_de_la_iste).**
Soit le programme suivant :

```
semaine = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']
print(len(semaine))
```

Quel est le résultat attendu après l'exécution de ce programme ?
Vérifiez votre réponse.

3. Il est **possible d'ajouter un élément à une liste** (en **fin de liste** plus précisément) grâce à la fonction **apend()**.
Pour ajouter l'élément **« b » à la fin de la liste ma_liste**, il faut utiliser une syntaxe un peu particulière : **ma_liste.append(b)**

a) Soit le programme suivant :

```
semaine = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']
for jour in semaine:
      print(jour)
print("Rajoutons le samedi")
semaine.append('samedi')
for jour in semaine:
      print(jour)
print("Voilà, comme vous pouvez le constater c'est chose faite !")
```

Quel est le résultat attendu après l'exécution de ce programme ?
Vérifiez votre réponse.

b) Avec des nombres

On donne le programme suivant :

```
liste_nombres = []      # cette instruction créer une liste vide
for element in range(0, 5):
      liste_nombres.append(element)
```

Quel est le résultat attendu après l'exécution de ce programme ?
Vérifiez votre réponse.

4. Indice négatif

La liste peut également être indexée avec des nombres négatifs selon le modèle suivant :

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

ou encore

```
liste          : ['A','B','C','D','E','F']
indice positif :   0   1   2   3   4   5
indice négatif :  -6  -5  -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.
L'avant-dernier élément a lui l'indice -2, l'avant-avant dernier l'indice -3, etc.

III. Créer une liste par compréhension

Nous avons vu qu'il était possible de "remplir" un tableau en renseignant les éléments du tableau les uns après les autres :

```
liste_nombres = [5, 8, 6, 9]
```

ou encore à l'aide de la méthode "append"

1. Les fonctions range() et list()

L'instruction `range()` est une fonction spéciale en Python qui génère des nombres entiers compris dans un intervalle.

Lorsqu'elle est utilisée en combinaison avec la fonction `list()`, on obtient une liste d'entiers.

Par exemple :

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

La commande `list(range(10))` a généré une liste contenant tous les nombres entiers de 0 inclus à 10 **exclu**.

Dans l'exemple ci-dessus, la fonction `range()` a pris un argument, mais elle peut également prendre deux ou trois arguments, voyez plutôt :

```
>>> list(range(0, 5))
[0, 1, 2, 3, 4]

>>> list(range(15, 20))
[15, 16, 17, 18, 19]

>>> list(range(0, 1000, 200))
[0, 200, 400, 600, 800]

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

L'instruction `range()` fonctionne sur le modèle `range([début,] fin[, pas])`.
Les arguments entre crochets sont optionnels.
Pour obtenir une liste de nombres entiers, il faut l'utiliser systématiquement avec la fonction `list()`.

Il faudrait, par exemple, préciser un pas de -1 pour obtenir une liste d'entiers décroissants :

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

2. Liste par compréhension

Il est aussi possible d'obtenir exactement le même résultat qu'à la question 3b) en une seule ligne grâce à la compréhension de tableau :

a) Quel est le contenu du tableau référencée par la variable mon_tab après l'exécution du programme ci-dessous ? (utilisez la console pour vérifier votre réponse)

liste_nombres = `[p for p in range(0, 5)]`

b) Les compréhensions de tableau permettent de rajouter une condition (if) :

Quel est le contenu du tableau référencé par la variable `nombres` après l'exécution des programmes ci-dessous ? (utilisez la console pour vérifier votre réponse)

- `listes = [1, 7, 9, 15, 5, 20, 10, 8]`

  `nombres = [p for p in listes if p > 10]`

- `listes = [1, 7, 9, 15, 5, 20, 10, 8]`

  `nombres = [p**2 for p in l if p < 10]`

  *Rappel : p**2 permet d'obtenir la valeur de p élevée au carrée*