Lycée Les 3 Sources **Bourg-Lès-Valence**
NSI 1re – Thème 6 Année **2024-25**

Les listes Python

Les bases des listes

En Python, comme dans la majorité des langages, il est possible de construire un objet contenant un ensemble ordonné de valeurs, appelé **liste**. Une liste est délimitée par des crochets et les éléments sont séparés par des virgules :

```
>>> liste_val = [4, 1, 2, 6, 7]
```

La liste vide est notée `[]`. Il est possible de rajouter un élément à la fin d'une liste :

```
>>> liste_val.append(15)
>>> liste_val.append(7)
>>> liste_val
[4, 1, 2, 6, 7, 15, 7]
```

Il est possible d'accéder à n'importe quel élément d'une liste, comme pour un texte :

```
>>> liste_val[0]
4
>>> liste_val[1]
1
```

```
>>> liste_val[-1]
7
>>> liste_val[-2]
15
```

Pour concaténer 2 listes, il suffit d'utiliser l'opérateur + ou `extend` :

```
>>> [6, 3, 1] + [4, 8]          # Crée une nouvelle liste
[6, 3, 1, 4, 8]
>>> liste.extend([4, 8])        # Modifie la liste et ne renvoie rien
```

Exercice 1 :

1) On considère la liste `valeurs = [2, 6, 14, 8, 0, -3]`.
   Déterminer les valeurs des expressions suivantes :
   a) `valeurs[1]` b) `valeurs[5]` c) `valeurs[valeurs[0]]`
2) On considère les instructions suivantes :

```
>>> liste1 = [3, 6, 1]
>>> liste2 = [4, 5]
```

   Déterminer les valeurs des expressions suivantes :
   a) `liste1+liste2` b) `liste2+liste1` c) `liste1+liste1` d) `liste2+liste2`
3) Que vaut `donnees` après les instructions suivantes :

```
>>> donnees = [-8, 3, 2]
>>> donnees.append(5)
>>> donnees.append(9)
```

Génération des listes

Il y a plusieurs façons pour définir une liste de façon explicite :

```
>>> [0] * 5
[0, 0, 0, 0, 0]
>>> [1, 2] * 3
[1, 2, 1, 2, 1, 2]
```

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list('bonjour')
['b', 'o', 'n', 'j', 'o', 'u', 'r']
```

Python permet également de générer une liste en appliquant une fonction à un ensemble de valeurs :

```
[f(x) for x in ITERABLE]
```

Cela permet de créer une liste en une seule ligne au lieu de passer par une boucle :

```
>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
>>> [x//2 for x in [3, 1, 7]]
[1, 0, 3]
```

Il est même possible de filtrer la liste obtenue à l'aide d'une condition :

```
[f(x) for x in ITERABLE if TEST]
```

On peut obtenir la liste des entiers inférieurs à 50 qui sont multiples de 3 mais pas de 4 :

```
>>> [x for x in range(50) if x%3 == 0 and x%4 != 0]
[3, 6, 9, 15, 18, 21, 27, 30, 33, 39, 42, 45]
```

Exercice 2 : Déterminer la valeur des expressions suivantes :

1) `[i+1 for i in range(5)]`
2) `[2*i for i in range(5)]`
3) `[2*i for i in range(50) if i < 4]`
4) `[i for i in range(10) if i%3 == 0]`

En Python, les listes sont polymorphiques. C'est-à-dire qu'elles peuvent contenir plusieurs types de données différents dans une même liste, y compris d'autres listes, comme `["mot", 4, 5.1, True, [1, 2]]`.

Parcours de listes

Les listes peuvent être parcourues par valeur ou par indice.

```
for v in liste:
    print(v)
```

```
for i in range(len(liste)):
    print(liste[i])
```

Lorsqu'on veut parcourir une boucle, il faut donc se poser la question si l'indice est important, et dans ce cas on parcourt par indice, sinon on peut parcourir par valeurs. En cas de doute, le parcours par indice permet toujours de faire ce qu'on aurait fait avec un parcours par valeurs.

Exercice 3 : On veut écrire la fonction `longueur` qui prend en paramètre une liste `liste` et qui renvoie la longueur de `liste`, c'est-à-dire, le nombre d'éléments qu'elle contient.

1) Déterminer le résultat attendu pour les listes suivantes :
   a) `longueur([3, 4])`
   b) `longueur([9, 8, 1, 7])`
   c) `longueur([0])`
   d) `longueur([])`
2) Comment déterminer le résultat de l'expression suivante :
   `longueur([2, 1, 351, 7841, 112, 0, -12134, 215, 41, -9, 0, 1, 2, 0, 7, 8])`
3) Compléter le code de la fonction :

```
def longueur(liste):
    . . . .
    for v in liste:
        . . .
    return . . .
```

```
>>> longueur([])
0
```

```
>>> longueur([4, 5, 2])
3
```

Cette fonction existe déjà en Python et se note `len(liste)`.

Exercice 4 : Compléter le code de la fonction `est_dans` qui prend comme paramètres une valeur `e` et une liste `liste`, et qui renvoie un booléen indiquant si `e` est dans `liste`.

```
def est_dans(e, liste):
    for v in liste:
        if . . . . . .:
            return . . .
    return . . .
```

```
>>> est_dans(4, [2, 3])
False
```

```
>>> est_dans(3, [2, 3])
True
```

Cette fonction existe déjà en Python et se note "`e in liste`".

Exercice 5 : On souhaite écrire une fonction `indice` qui prend comme paramètres une valeur `e` et une liste `liste`, et qui renvoie l'indice de la première occurrence de `e` dans `liste`, en comptant à partir de 0 et −1 s'il n'est pas dans `liste`.

1) Déterminer le résultat des expressions suivantes :
   a) `indice(3, [3, 4])`
   b) `indice(4, [3, 4])`
   c) `indice(5, [3, 4])`
   d) `indice(1, [2, 1, 7, 1, 3])`
   e) `indice(9, [8, 3, 7, 1, 3, 8])`
   f) `indice(2, [2, 3, 7, 1, 9, 1, 2])`
2) Déterminer le résultat de l'expression suivante :
   `indice(3233323223, [323323223, 323332323, 3233323223, 32233323223])`
3) Compléter le code de la fonction `indice(e, liste)`.

```
def indice(e, liste):
    for i in range(len(liste)):
        if . . . . . . . . .:        # quelque chose avec liste[i]
            return . . .
    return . . .
```

Cette fonction existe déjà en Python et se note `liste.index(e)`. Par contre, si `e` n'est pas dans `liste`, cela provoque une erreur.

Exercice 6 : On souhaite écrire une fonction `compter` qui prend comme paramètres une valeur `e` et une liste `liste`, et qui renvoie le nombre de fois où `e` apparaît dans `liste`.

1) Déterminer le résultat des expressions suivantes :
   a) `compter(5, [5, 5, 1])`
   b) `compter(1, [5, 5, 1])`
   c) `compter(9, [5, 5, 1])`
   d) `compter(2, [])`
   e) `compter(4, [3, 4, 4, 1])`
   f) `compter(4, [4, 1, 4, 3])`
2) Est-ce qu'il est nécessaire de connaître les indices des éléments parcourus ou juste leur valeur ?
3) Compléter le code de la fonction `compter(e, liste)`.

```
def compter(e, liste):
    . . .
    for . . . in . . . . . . . .:
        if . . . . .:
            . . . . .
    return . . .
```

Cette fonction existe déjà en Python et se note `liste.count(e)`.

Exercice 7 : On souhaite écrire une fonction qui permet de déterminer le maximum d'une liste.

1) Déterminer le maximum de la liste suivante :
   `[32232323, 32222323, 32322323, 32232333, 33232323, 32233223]`
2) Est-ce qu'il est nécessaire de connaître les indices des éléments parcourus ou juste leur valeur ?
3) Écrire le code de la fonction `maximum` qui prend comme paramètre une liste non vide `nombres` d'entiers et qui renvoie le maximum des valeurs de `nombres`.

```
def maximum(nombres):
```

```
>>> maximum([5, 1, 8, 14, 2])
14
>>> maximum([15])
15
```

```
>>> maximum([-9, -27, -2, -48])
-2
>>> maximum([9, 2, -57, 1, 8])
9
```

Cette fonction existe déjà en Python et se note `max(nombres)`. De même, il existe `min(nombres)`.

Exercice 8 : On souhaite faire une fonction qui calcule la moyenne des valeurs d'une liste.

1) Expliquer comment calculer la moyenne de la liste `[15, 12, 17, 15, 14]`.
2) Est-ce qu'il est nécessaire de connaître les indices des éléments parcourus ou juste leur valeur ?
3) Écrire le code de la fonction `moyenne` qui prend en paramètre une liste non vide `valeurs` de nombres qui calcule la moyenne des valeurs de `valeurs`.
   Vous ne devez pas utiliser la fonction `len`.

```
def moyenne(valeurs):
```

```
>>> moyenne([20])
20.0
>>> moyenne([20, 10])
15.0
```

```
>>> moyenne([5, 5, 5, 5, 5])
5.0
>>> moyenne([3, 2, 5, 1, 7, 4])
3.6666666666666665
```