Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2023 – 2024
DEV1 – DEV1L

DEV1 – DEV1L – Laboratoires Python

TD 12 – Listes (suite)

Dans ce TD nous continuons l'exploration des listes. Vous allez découvrir de nouvelles méthodes, fonctions et de nouveaux opérateurs pour créer, modifier et manipuler des listes.

Table des matières

Rappel

Vous devez absolument avoir fait et compris les TD précédents *avant* d'aborder celui-ci. Nous supposons que vous êtes capable de : créer une liste vide ou comprenant des littéraux, de parcourir une liste en utilisant des boucles `while` et `for`, de faire la distinction entre des objets mutables et immutables, d'utiliser l'indexation pour accéder aux éléments d'une liste.

Version du document
13 novembre 2023

1 Les listes - []

1.1 Résumé des principales méthodes, fonctions et opérateurs de la classe `list`

Méthode	Description
`[]`	construit une liste vide
`l[i]`	retourne la valeur de l'élément d'indice i de la liste
`len(l)`	donne le nombre d'élément de la liste
`x in l`	retourne `True` si x est dans la liste `False` sinon
`del l[i]`	supprime l'élément d'indice i de la liste
`l1 + l2`	concaténation
`l * 3`	répétition
`l[i : j]`	slicing : retourne la sous-liste composée des éléments de la liste d'indice i à j exclu
`l[i] = x`	assigne la valeur x à l'élément d'indice i de la liste
`l.append(x)`	ajoute x à la fin de la liste
`l.copy()`	crée et retourne une nouvelle copie de la liste
`l.count(x)`	compte et retourne le nombre d'occurrences de x dans la liste
`l.extend(l1`	ajoute les éléments de la liste `l1` à la fin de la liste
`l.index(x)`	retourne l'indice de la première occurrence de x
`l.insert(i, x)`	insère x dans la liste avant l'élément d'indice i
`l.pop() ou l.pop(i)`	extrait le dernier élément de la liste ou l'élément d'indice i
`l.remove(x)`	supprime la première occurrence de x dans la liste
`l.reverse()`	inverse l'ordre des éléments de la liste

TABLE 1 – Les opérations sur les listes.

Attention

Dans la liste ci-dessus, il faut distinguer les fonctions, méthodes et opérateurs qui modifient une liste (par exemple : la méthode `reverse`) de ceux qui créent une nouvelle liste (par exemple : la méthode `copy`, les opérateurs `+` et `*`).
Il faut aussi distinguer les fonctions et méthodes qui retournent une valeur (par exemple : les méthodes `index` et `pop`) de celles qui ne retournent rien, c'est-à-dire la valeur `None` de type `NoneType`, par exemple : les méthodes `append`, `extend` et `remove`.

1.2 Création de listes contenant des éléments aléatoires

Pour tester les fonctions que vous écrirez lors de ce TD, nous vous présentons ci-dessous 2 exemples de code Python qui permettent de créer des listes d'éléments choisis aléatoirement dans une séquence (tuples, listes, chaines de caractères...).
Le code ci-dessous crée une liste de 10 entiers choisis aléatoirement entre 0 et 5 inclus :

listeEntiersAleatoiresEx.py

```
1 import random
2
3 ma_liste = []
4
5 for i in range(10):
6     ma_liste.append(random.randint(0,5))
7
8 print(ma_liste)
```

Le code ci-dessous crée une liste de 10 caractères minuscules choisis aléatoirement dans les lettres de l'alphabet latin :

listeCaracteresAleatoiresEx.py

```
import random

caracteres = "abcdefghijklmnopqrstuvwxyz"

ma_liste = []

for i in range(10):
    ma_liste.append(random.choice(caracteres))

print(ma_liste)
```

1.3 Opérations sur les listes

Exercice 1 Manipulations de base

Dans le shell interactif de Python, à l'aide d'une ou plusieurs instructions (en vérifiant à chaque fois le résultat) :

- ▷ créez une liste `liste1` contenant quelques entiers ;
- ▷ vérifiez la longueur de `liste1` ;
- ▷ à l'aide de l'opérateur `in` vérifier la présence de la valeur 0 ;
- ▷ ajoutez à cette liste, en dernière position, la valeur 15 ;
- ▷ inversez l'ordre des éléments de la liste ;
- ▷ utilisez le slicing pour créer une nouvelle liste (`liste4`) contenant les éléments d'indice pair de la `liste1` ;
- ▷ grâce à la méthode `count`, comptez le nombre d'occurrences dans cette liste de la valeur 3 ;
- ▷ insérez en deuxième position la valeur 7 ;
  On distingue *insérer* et *modifier*. Comment mettez vous en évidence la différence ?
- ▷ en utilisant le slicing, remplacez le deuxième et le troisième élément de `liste1` par les éléments de la liste [10,12,14,16] ;
- ▷ à l'aide de la méthode `copy` de la classe `list`, créez une copie de `liste1` nommée `liste2`. Exécutez ensuite l'instruction `liste3 = liste1`. Quelle est la différence entre ces deux instructions ?
- ▷ extrayez de la `liste1` le dernier élément ;
- ▷ extrayez de la `liste1` l'élément en 4ème position ;
- ▷ triez `liste1` en ordre ascendant (du plus petit au plus grand) ;
- ▷ triez `liste2` en ordre descendant (du plus grand au plus petit) ;
- ▷ recherchez l'indice d'une valeur présente dans `liste2` ;
- ▷ recherchez l'indice d'une valeur non présente dans `liste2` ;
- ▷ supprimez une valeur présente dans `liste2` ;
- ▷ que se passe-t-il si vous supprimez une valeur non présente dans `liste2` ?
- ▷ à l'aide de la méthode `extend` de la classe `list`, ajoutez à la fin de `liste1` les éléments de `liste2` situés entre la $3^{\text{ième}}$ et la $7^{\text{ième}}$ position inclues ;
- ▷ supprimez toutes les valeurs de `liste2` ;
- ▷ à l'aide de la fonction `del` supprimez le premier élément de `liste1`.

Exercice 2 — Rien c'est pas grand chose ... mais c'est important !

Dans le shell ou dans un script Python, exécutez le code suivant :

appendReturnNone.py
```
mots = ['Carottes', 'Poireaux', 'Tomates', 'Aubergines', 'Courgettes']

mots = mots.append("Brocolis")
```

Que contient la liste ?
Comment pouvez-vous expliquer le résultat ?

Indice

Quelle valeur est retournée par la méthode `append` de la classe `list` ?

Exercice 3 — Le parcours d'une liste est semé d'embûches

Dans le shell Python ou dans un script, exécutez les deux morceaux de code ci-dessous.

Qu'observez-vous ?
Pouvez-vous expliquez les résultats obtenus à l'aide des propriétés des listes rencontrées au TD précédent ?

boucleSansModifListe.py
```
mots = ['Carottes', 'Poireaux', 'Tomates', 'Aubergines', 'Courgettes']

for m in mots[:] :
    mots.append(m)
```

boucleAvecModifListe.py
```
mots = ['Carottes', 'Poireaux', 'Tomates', 'Aubergines', 'Courgettes']

for m in mots :
    mots.append(m)
```

1.4 Listes en compréhension

La technique des listes en compréhension permet de créer, de manière optimisée, des listes en évaluant une expression pour chaque élément d'une séquence. Sa syntaxe est proche de la notation qu'on retrouve en mathématique pour la définition en compréhension d'un ensemble.

Par exemple l'ensemble des nombres naturels dont le carré est inférieur ou égal à 25 ($\{0, 1, 2, 3, 4, 5\}$) s'écrit :

$$\{i \mid i \in \mathbb{N},\ i^2 \leq 25\}$$

Pour créer la liste en compréhension en Python, on écrira par exemple :

creationListeComprehension.py
```
ma_liste = [x for x in range(25) if x**2 <= 25]
```

La même liste peut être créée à l'aide d'un boucle `for` :

creationListeBoucleFor.py
```
ma_liste = []
for x in range(25):
    if x**2 <= 25:
        ma_liste.append(x)
```

Dans une liste en compréhension la condition `if` est optionnelle.

Par exemple, le code suivant :

listeEntiersAleatoires.py
```
import random
ma_liste = [random.randint(0,20) for i in range(10)]
```

crée une liste de 10 nombres entiers choisis aléatoirement entre 0 et 20 inclus.

Exercice 4 Liste d'entiers aléatoires

Écrivez une fonction `liste_entiers_aléatoires(n, nmin, nmax)` qui retourne une liste de taille `n` de nombres entiers choisis aléatoirement entre `nmin` et `nmax` inclus.

Utilisez une liste en compréhension.

Exercice 5 Additionner un nombre

Écrivez une fonction `addition(l1, valeur)` qui reçoit en paramètre une liste d'entiers (`l1`) et un nombre entier (`valeur`). Votre fonction retourne une nouvelle liste obtenue en ajoutant à chacun des éléments de la liste `l1` le nombre entier `valeur`. Par exemple si l1 = [1, 2, 3, 4, 5] et valeur = 2, votre fonction retournera la liste [3, 4, 5, 6, 7].

Utilisez une liste en compréhension.

Exercice 6 Conversion

Écrivez une fonction `char_to_int(liste)` qui transforme une liste de caractères minuscules (non accentués) en une liste d'entiers compris entre 0 et 25. Par exemple `"a"` sera renvoyé sur `0`, `"b"` sera renvoyé sur `1` , etc.

Par exemple si `liste = ['b', 'o', 'n', 'j', 'o', 'u', 'r']`, votre fonction retournera la liste d'entiers `[1, 14, 13, 9, 14, 20, 17]`. Pour tester votre fonction, vous pouvez facilement créer une liste de caractères à l'aide de la fonction [1] `list` qui transforme une chaine de caractères en une liste de caractères.

On pourra remarquer que `ord("a")` renvoie 97 et `ord("z")` renvoie 122.

2 Exercices récapitulatifs

Exercice 7 Moyenne

Écrivez une fonction `moyenne(ma_liste)` qui reçoit en paramètre une liste de nombres flottants et retourne la valeur moyenne des éléments de la liste.

Exercice 8 Liste des éléments communs

Écrivez une fonction `liste_elements_communs(l1, l2)` qui reçoit deux listes d'entiers en paramètre et retourne la liste des éléments communs aux deux listes. Veillez à ce que les éléments de la liste retournée soient deux à deux distincts.

Exercice 9 Le nettoyage

Écrivez une fonction `suppression_occurrences(ma_liste, mot)` qui reçoit une liste de chaines de caractères et un mot (chaine de caractères) en paramètre et supprime de cette liste toutes les occurrences de `mot`. Votre fonction retournera le nombre de suppressions effectuées.

1. Le terme exact est constructeur.

Exercice 10 Suppression des doublons

Écrivez une fonction `suppression_doublons(ma_liste)` qui reçoit en paramètre une liste d'entiers avec de possibles doublons et qui supprime les doublons consécutifs de la liste.

Exemple : Si la liste est [7, 3, 3, 8, 8, 8, 7], le résultat est [7, 3, 8, 7].

a) Faites l'exercice en créant et retournant une **nouvelle liste** (la liste de départ reste inchangée)

b) Refaites l'exercice en **modifiant** la liste de départ (pas de nouvelle liste)

3 En résumé ...

Principaux points de matière du TD

Voici les principaux points abordés lors de ce TD. Vous devez absolument être à l'aise avec ceux-ci avant d'aborder la prochaine séance d'exercice.

1. Les méthodes de la classe `list` et les fonctions pré-définies pour manipuler des listes.
2. La création de listes en compréhension.
3. Mettre en pratique les points précédents pour écrire des scripts résolvants des problèmes divers.