1 Introduction

Jusqu'à présent, nous avons manipulé des types simples : valeurs numériques (entières ou flottantes) et booléens.

En informatique, comme dans la vie pratique, ces types de données sont rarement suffisants. Par exemple, si on désire étudier l'évolution d'un prix sur plusieurs années, on ne va pas regarder uniquement un prix en particulier, mais l'ensemble des prix sur la période. Il faut donc être capable de stocker toutes ces valeurs dans une même structure afin de pouvoir y accéder sans difficulté. La structure la plus simple que l'on puisse utiliser est un tableau (en général unidimensionnel, c'est-à-dire avec une seule ligne).

En `Python`, un tableau est assimilé à une structure spécifique appelée **liste** (type `list`).

Le but de ce chapitre est de comprendre et manipuler ce nouveau type de donnée.

Commençons par une petite introduction à l'aide du Notebook *NSI Première Partie 1 Chapitre 6 Listes introduction*.

2 Le type `list`

2.1 Définition des listes

Définition 2.1 Une *liste* est une collection ordonnée (ou séquence) d'éléments encadrée par des crochets, les éléments étant séparés par des virgules.

Exemple 2.2 `[1, 2, -5, 10]` est une liste de quatre éléments.

Remarque 2.3 Les listes peuvent être composées d'éléments de types différents mais nous ne n'en manipulerons pas cette année.

Définition 2.4 (Longueur d'une liste) On appelle *longueur d'une liste* le nombre d'éléments qui la compose.
Cette longueur peut-être obtenue à l'aide de la fonction `len`.

Exemple 2.5 L'instruction `len([1, 2, -5, 10])` renvoie `4`.

Définition 2.6 Il existe plusieurs manières de définir une liste :

- **Liste vide :** elle est définie par l'instruction `[]`
- **Liste par extension :** elle est définie en indiquant directement entre crochets les éléments de la liste.
  Exemple : la liste `[1, 2, -5, 10]` est une liste définie par extension.
- **Liste par compréhension :** elle est définie en indiquant entre crochets la manière dont la liste est construite. On utilise pour cela la syntaxe suivante :

  ```
  [fonction(i) for i in range(valeur_initiale, valeur_finale + 1) if condition(i)]
  ```

  où `fonction(i)` est une fonction de paramètre `i` et où `condition(i)` est une condition que l'on impose sur `i`. Cette condition est optionnelle.
  Exemple : les deux listes suivantes sont des listes définies par compréhension :
  - `[i for i in range(5)]` correspond à la liste `[0, 1, 2, 3, 4]`
  - `[i**2 for i in range(5) if i%2 == 1]` correspond à la liste `[1, 9]`
- **Liste par transtypage :** elle est définie en appliquant la fonction `list` sur certains types de données (par exemple des chaînes de caractères).
  Exemple : l'instruction `list('Python')` donne la liste `['P', 'y', 't', 'h', 'o', 'n']`

Remarque 2.7 On peut également définir des listes par ajout. Ceci sera abordé au paragraphe 4.

2.2 Indexation

⋆ Les éléments d'une liste étant ordonnés, ils sont repérés à l'aide d'un indice de la façon suivante :

- le premier élément de la liste a pour indice `0` ;
- le deuxième élément de la liste a pour indice `1` ;
- le troisième élément de la liste a pour indice `2` ;
- etc.

Ainsi, **l'indice d'un élément dans une liste est égal à la `position - 1` de l'élément dans cette liste.**.

⋆ L'élément d'indice `j` d'une liste `L` est défini par l'instruction `L[j]`.
Exemple : on considère la liste `L = [1, 2, -5, 10]`. Alors :

- `L[0]` correspond à `1` (élément d'indice `0`);
- `L[1]` correspond à `2` (élément d'indice `1`);
- `L[2]` correspond à `-5` (élément d'indice `2`);
- `L[3]` correspond à `10` (élément d'indice `3`).

Remarque : si on veut accéder à un indice qui n'existe pas, on déclenche une erreur de type

`IndexError: list index out of range`,

c'est-à-dire que l'on a voulu utiliser un indice en dehors de la plage autorisée (de `0` jusqu'à `len - 1`).

⋆ Les éléments `L[0]`, `L[1]`, etc. d'une liste `L` peut être également vus comme des variables. A ce titre, on peut les manipuler comme toutes les variables et écrire des instructions d'affectation avec celles-ci. Ceci aura pour effet de modifier la valeur de la liste située à l'indice que l'on utilise.
Exemple : on considère la liste `L = [1, 2, -5, 10]`. Alors, l'instruction `L[2] = 0` modifie la valeur `-5` en la valeur `0`. La liste `L` devient donc `[1, 2, 0, 10]`.

⋆ Les instructions suivantes permettent d'extraire des sous-listes d'une liste `L` à partir des indices de ses éléments (slicing).

- `L[i:j+1]` renvoie la sous-liste de `L` constituée des éléments dont les indices sont compris entre `i` et `j`.
  Exemple : si `L = [1, 5, 4, 8, 2, 3]`, alors `L[1:4]` correspond à la sous-liste `[5, 4, 8]`.
  Remarque : si `i > len(L) - 1`, alors la sous-liste obtenue est la liste vide.
- `L[:j+1]` renvoie la sous-liste de `L` constituée des éléments dont les indices sont compris entre `0` et `j`.
  Exemple : si `L = [1, 5, 4, 8, 2, 3]`, alors `L[:3]` correspond à la sous-liste `[1, 5, 4]`.
- `L[i:]` renvoie la sous-liste de `L` constituée des éléments dont les indices sont compris entre `i` et `len(L) - 1`.
  Exemple : si `L = [1, 5, 4, 8, 2, 3]`, alors `L[2:]` correspond à la sous-liste `[4, 8, 2, 3]`.

2.3 Opérateurs

Les deux opérateurs + et ⋆ sont compatibles avec les listes :

⋆ L'opérateur + est un **opérateur de concaténation**, c'est-à-dire qu'il permet de rassembler deux listes en une seule.
Exemple : l'instruction `[1, 2] + [3]` renvoie la liste `[1, 2, 3]`.

⋆ L'opérateur ⋆ est un **opérateur de duplication**, c'est-à-dire qu'il renvoie une liste dont les éléments sont ceux d'une liste qui ont été dupliqués plusieurs fois.
Exemple : l'instruction `[0]*3` renvoie la liste `[0, 0, 0]` constituée de l'élément `0` dupliqué trois fois.

2.4 Tests

⋆ On peut tester si deux listes sont égales ou différentes grâce aux opérateurs de comparaison == et `!=`
Exemples :

- L'instruction `[1, 2, 3] == [1, 2, 3]` renvoie `True` (les deux listes sont égales).
- L'instruction `[1, 2, 3] != [1, 2, 3]` renvoie donc `False`.
- L'instruction `[1, 2, 3] == [1, 3, 2]` renvoie `False` (les deux listes ne sont pas égales).
- L'instruction `[1, 2, 3] != [1, 3, 2]` renvoie donc `True`.

⋆ On peut également tester si un élément appartient à une liste ou non grâce aux opérateurs `in` et `not in`
Exemples :

- L'instruction `5 in [1, 2, 4, 5, 8]` renvoie `True` (`5` est un élément de la liste).
- L'instruction `5 not in [1, 2, 4, 5, 8]` renvoie donc `False`.
- L'instruction `6 in [1, 2, 4, 5, 8]` renvoie `False` (`6` n'est pas un élément de la liste).
- L'instruction `6 not in [1, 2, 4, 5, 8]` renvoie donc `True`.

2.5 Exercices

Exercice 2.8 (Listes et indexation) Compléter le Notebook *NSI Première Partie 1 Chapitre 6 Listes définition et indexation*.

Exercice 2.9 (Listes par compréhension) Compléter le Notebook *NSI Première Partie 1 Chapitre 6 Listes compréhension*.

Exercice 2.10 (QCM)

1. On considère la liste `L = [1,5,8,2,3]`. Que renvoie l'instruction `L[2]` ?
   (a) `3` (b) `4` (c) `8` (d) une erreur

2. On considère la liste `L = [1,5,8,2,3]`. Que renvoie l'instruction `L[5]` ?
   (a) `1` (b) `2` (c) `3` (d) une erreur

3. On considère la liste `L = [3,1,5,7]`. Parmi les affirmations suivantes, lesquelles sont vraies ?
   (a) la liste a quatre éléments
   (b) la liste est définie par extension
   (c) la liste est définie par compréhension
   (d) l'instruction `len(L)` renvoie `4`

4. On considère les instructions suivantes :

```
L = list('mIths')
L[1] = 'a'
```

   Parmi les affirmations suivantes, lesquelles sont vraies ?
   (a) `L` est une chaîne de caractères
   (b) `L` contient la chaîne de caractères `'maths'`
   (c) `L` est une liste
   (d) `L` contient la liste `['m','a','t','h','s']`

5. On considère la liste `L = [1,2,6,8,6]`. Quel est le contenu de `L` après l'instruction `del(L[2])` ?
   (a) `[1,2,8]` (b) `[1,6,8,6]` (c) `[1,2,8,6]` (d) la liste `L` est supprimée

6. On considère la liste `L = [1,2,5,8,6]`. Quel est le contenu de `L` après l'instruction `L[2] = 6` ?
   (a) `[1,6,5,8,6]` (b) `[1,6,5,8,2]` (c) `[1,2,5,8,2]` (d) `[1,2,6,8,6]`

7. On considère la liste suivante : `[0 for i in range(5)]`. Parmi les affirmations suivantes, lesquelles sont vraies ?
   (a) la liste est définie par compréhension
   (b) la liste est définie par extension
   (c) la liste peut aussi être définie par l'instruction `[0]*5`
   (d) la liste est égale à la liste `[0, 0, 0, 0, 0]`

8. On considère la liste `L = [1,5,8,2,3]`. Que renvoie l'instruction `'8' not in L`?
   (a) une erreur (b) `False` (c) un indice (d) `True`

9. On considère la liste `L = [1,5,8,2,3]`. Que renvoie l'instruction `5 in L`?
   (a) une erreur (b) `False` (c) un indice (d) `True`

10. On considère la liste `L = [1,2,5,3,4]`. Que renvoie l'instruction `L[:3] + [0] + L[3:]` ?
    (a) la liste `[1,2,5,3,0,3,4]`
    (b) la liste `[5,0,3]`
    (c) la liste `[1,2,5,0,3,4]`
    (d) une erreur car on ne peut pas additionner des listes

3 Parcours de listes

3.1 Syntaxe et exemples

Une liste est une structure qui peut être parcourue afin d'obtenir des informations sur chacun de ses éléments.

Il existe deux types de parcours : le **parcours par indice** et le **parcours par élément**.

Définition 3.1 On donne ci-dessous la syntaxe Python pour chacun des deux parcours pour une liste L :

Parcours par indice :

```
for i in range(len(L)):
    bloc d'instructions
```

Parcours par élément :

```
for elem in L:
    bloc d'instructions
```

Remarque 3.2

- Lorsqu'on parcourt une liste par indice, on accède à chaque élément en lecture et en écriture, c'est-à-dire que l'on va pouvoir modifier des éléments de la liste si on le désire.
- En revanche, lorsqu'on parcourt une liste par élément, on accède à chaque élément uniquement en lecture, c'est-à-dire que l'on ne va pas pouvoir modifier les éléments.

Exemple 3.3 Le tableau ci-dessous donne le code Python permettant d'afficher tous les éléments de la liste L définie à la première ligne suivant ces deux types de parcours :

Parcours par indice :

```
L = [1, 5, 3, 8, 9, 10, 7, 8]
for i in range(len(L)):
    print(L[i])
```

Parcours par élément :

```
L = [1, 5, 3, 8, 9, 10, 7, 8]
for elem in L:
    print(elem)
```

L'exécution de ces deux codes donne le même affichage :

```
1
5
3
8
9
10
7
8
```

Remarque 3.4 Les listes étant des éléments parcourables, on peut aussi les utiliser pour définir des listes par compréhension. L'exemple ci-dessous permet de définir une telle liste (on donne la syntaxe avec les deux types de parcours) :

- Parcours par indice :

```
L = [1, 5, 3, 8, 9, 10, 7, 8]
M = [L[i] for i in range(len(L)) if L[i]%2 == 1]
```

- Parcours par élément :

```
L = [1, 5, 3, 8, 9, 10, 7, 8]
M = [elem for elem in L if elem%2 == 1]
```

Ces deux codes produisent la même liste M constituée des éléments de L qui sont impairs : [1, 5, 3, 9, 7].

3.2 Exercices

Exercice 3.5 Compléter le Notebook *NSI Première Partie 1 Chapitre 6 Listes parcours*.

Exercice 3.6 (QCM)

1. On considère la liste `L = [1,5,8,2,3]`.
   Parmi les instructions suivantes, quelles sont celles qui permettent de parcourir la liste `L` ?
   (a) `for i in L`
   (b) `for i in range(len(L))`
   (c) `for i in len(L)`
   (d) `for i in range(L)`

2. On considère la fonction suivante :

```
def recherche2(L): # L de type list
    for i in range(len(L)):
        if L[i] == 2:
            return i
    return -1
```

   Parmi les affirmations suivantes, lesquelles sont vraies ?
   (a) l'instruction `recherche2([1,5,2,3])` renvoie `2`
   (b) l'instruction `recherche2([1,5,2,3])` renvoie `-1`
   (c) l'instruction `recherche2([1,3,4,0])` renvoie `-1`
   (d) l'instruction `recherche2([1,5,2,3])` renvoie `3`

3. On considère les instructions suivantes :

```
L = [1, 5, 6, -1, -2, 0, 5, 4]
M = [elem**2 for elem in L if elem%2 == 0 and elem > 0]
```

   Quel est le contenu de la liste `M` ?
   (a) `[36,4,0,16]` (b) `[36,0,16]` (c) `[36,16]` (d) `[16,36]`

4 Fonctions et méthodes agissant sur les listes

Il existe un grand nombre de fonctions et méthodes agissant sur les listes. On ne les présente pas toutes ici. La liste complète peut être obtenue avec l'instruction `help(list)`.

4.1 Fonctions

On donne ci-dessous quatre fonctions importantes dans le cadre des listes numériques, c'est-à-dire à valeurs entières ou flottantes.

- L'instruction `sum(L)` renvoie la somme des éléments de la liste numérique `L`.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `sum(L)` renvoie `9`.

- L'instruction `min(L)` renvoie le plus petit élément de la liste numérique `L`.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `min(L)` renvoie `-1`.

- L'instruction `max(L)` renvoie le plus grand élément de la liste numérique `L`.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `max(L)` renvoie `5`.

- L'instruction `sorted(L)` renvoie une liste dont les éléments sont ceux de la liste numérique `L` rangés par ordre croissant.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `sorted(L)` renvoie la liste `[-1, 1, 1, 3, 5]`.

4.2 Méthodes

Les méthodes sont des fonctions particulières qui agissent directement sur les listes. Lorsqu'elles provoquent des modifications de la liste (ajout, suppression, etc.), ces modifications ont lieu **en place**, c'est-à-dire que la liste est modifiée intrinsèquement et que l'on perd la liste initiale.

On donne ci-dessous huit méthodes importantes :

- L'instruction `L.append(elem)` ajoute à la fin de la liste `L` l'élément `elem` (modification en place).
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.append(0)` modifie la liste `L` en la liste `[1, 5, -1, 3, 1, 0]`.

- L'instruction `L.remove(elem)` supprime de la liste `L` le premier élément dont la valeur est `elem` (modification en place). Si l'élément `elem` n'existe pas, une erreur de type `ValueError` est déclenchée.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.remove(1)` modifie la liste `L` en la liste `[5, -1, 3, 1]`.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.remove(0)` déclenche l'erreur `ValueError: list.remove(x): x not in list`.

- L'instruction `L.pop(i)` supprime l'élément d'indice `i` de la liste `L` et le renvoie (modification en place). Si le paramètre `i` est omis, c'est le dernier élément de la liste qui est supprimé et renvoyé.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.pop(1)` modifie la liste `L` en la liste `[1, -1, 3, 1]` et la valeur `5` (élément d'indice `1`) est renvoyé.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.pop()` modifie la liste `L` en la liste `[1, 5, -1, 3]` et la valeur `1` (dernier élément de la liste) est renvoyé.

- L'instruction `L.insert(i, elem)` insère l'élément `elem` à l'indice `i` dans la liste `L` (modification en place).
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.insert(2,6)` modifie la liste `L` en la liste `[1, 5, 6, -1, 3, 1]`.

- L'instruction `L.sort()` range par ordre croissant les éléments de la liste `L` si `L` est une liste numérique (modification en place).
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.sort()` modifie la liste `L` en la liste `[-1, 1, 1, 3, 5]`.

- L'instruction `L.count(elem)` renvoie le nombre d'occurences de l'élément `elem` dans la liste `L` (pas de modification).
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.count(1)` renvoie `2`.

- L'instruction `L.index(elem)` renvoie l'index de la première occurence de l'élément `elem` dans la liste `L` si `elem` appartient à `L`, et renvoie une erreur sinon.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.index(5)` renvoie `1`.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.index(1)` renvoie `0`.
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.index(6)` renvoie `ValueError: list.index(x): x not in list`
- L'instruction `L.copy()` renvoie une copie de la liste `L` si elle ne contient pas de sous-listes (pas de modification).
  Exemple : si `L = [1, 5, -1, 3, 1]`, alors `L.copy()` renvoie la liste `[1, 5, -1, 3, 1]`.

Remarque 4.1 Lorsque des modifications en place ont lieu sur une liste, il y a une réindexation automatique de ses éléments.

4.3 Exercices

Exercice 4.2 (Fonctions et méthodes (hors `append`)) Compléter le Notebook *NSI Première Partie 1 Chapitre 6 Listes fonctions méthodes*.

Exercice 4.3 (Méthode `append`) Compléter le Notebook *NSI Première Partie 1 Chapitre 6 Listes méthode append*.

Exercice 4.4 (QCM)

1. On considère la liste `L = [1,3,4,8,3]`. Que renvoie l'instruction `sum(L)` ?
   (a) une erreur (b) une somme (c) une moyenne (d) `19`

2. On considère la liste `L = [1,3,4,8,3]`. Que renvoie l'instruction `L.count(3)` ?
   (a) `[1,3,4]` (b) `8` (c) `2` (d) `4`

3. On considère la liste `L = [1,2,5,8]`. Quel est le contenu de `L` après l'instruction `L.append(6)` ?
   (a) `[1,2,5,8,6]`
   (b) la liste `L` n'est pas modifée
   (c) `[6,1,2,5,8]`
   (d) `[1,2,5,6]`

4. On considère les instructions suivantes :

```
L = []
for i in range(5):
    L.append(i)
```

   Parmi les affirmations suivantes, lesquelles sont vraies ?
   (a) l'instruction `[i for i in range(5)]` définit la même liste
   (b) la liste `L` contient cinq éléments
   (c) la liste `L` est la liste `[1,2,3,4]`
   (d) la liste `L` est définie par compréhension

5. On considère la liste `L = [1,3,4,8,3]`. Quel est le contenu de la liste `L` à l'issue des instructions suivantes :

```
x = L.pop()
L.clear()
L.append(x)
```

   (a) rien
   (b) la liste `[3]`
   (c) la liste vide
   (d) la liste `[x]`

6. On considère les instructions suivantes :

```
L = [1,1]
for i in range(6):
    L.append(L[0]+L[1])
    L.pop(0)
```

   Parmi les instructions suivantes, lesquelles sont vraies à l'issue de ces instructions ?
   (a) `L` contient la liste `[13,21]`
   (b) `L[1]` contient la valeur `21`
   (c) `L` contient la liste `[1,1,2,3,5,8,13,21]`
   (d) `L` est vide

7. On considère la fonction suivante :

```
def suppr_max(liste): # liste de type list
    liste.sort()
    liste.pop()
    return liste
```

   Que renvoie l'instruction `suppr_max([5,2,0,5,4])` ?
   (a) la liste `[2,0,5,4]`
   (b) la liste `[2,0,4]`
   (c) la liste `[0,2,4,5]`
   (d) la liste `[0,2,4]`

8. On considère les instructions suivantes :

```
L = []
for i in range(6):
    L.append(i**2)
    if i in L:
        L.remove(i)
```

   Quelle liste contient la variable `L` à l'issue de ces instructions ?
   (a) `[0,1,4,9,16,25]`
   (b) `[9,16,25]`
   (c) `[4,9,16,25]`
   (d) `[]`