TP 5 – Listes
Année 2024-2025

Objectifs du TP : Représentations des listes chaînées, algorithmes élémentaires sur les listes. Vous pourrez vous appuyer sur la spécification des différentes méthodes de la classe, ainsi que sur l'implémentation de la classe *Liste* fournie sur Moodle (voir *listes.py*). Pour les nouvelles fonctions et méthodes demandées, prenez soin de bien compléter leur spécification, sur le modèle vu en cours et au précédent TP.

Exercice 1

Principe de la liste chaînée – Une liste chaînée est une structure de données permettant, comme les tableaux, de stocker plusieurs valeurs de même type. Mais, contrairement aux tableaux, cette structure de données est de taille variable. On la nomme ainsi car elle est similaire à une chaîne composée de cellules : chaque cellule est reliée à la cellule suivante, de telle sorte qu'à partir de la première cellule, on peut accéder (en « déroulant » la chaîne) à toutes les cellules de la chaîne. Par ailleurs, on peut facilement agrandir ou rétrécir la chaîne en lui ajoutant ou retirant des cellules.

Chaque élément de la liste est une cellule (class Cell) définie par la structure suivante :

```
class Cell():
    def __init__(self):
        self.data = object
        self.next = None
```

Chaque cellule porte une donnée (*data*, ici de type générique object), et pointe vers une autre cellule grâce à son champ suivant (*next*). Le nombre de cellules dans la liste est égal au nombre de valeurs à stocker. La dernière cellule n'a pas de suivant ; pour représenter cela, on donnera à son champ suivant la valeur None, qui représente en Python l'absence de valeur. Pour manipuler la liste, il suffit de garder une référence vers sa première cellule.

Une liste est définie par la spécification fournie sous Moodle (fichier *listes.py*). Cette spécification (noms et paramètres des fonctions) devra être respectée. Attention, elle n'est pas complète, vous devrez la compléter pour chacune des méthodes de la classe Liste.

Dans cet exercice, les listes seront implémentées par des listes chaînées simples. Certaines implémentations sont déjà données.

Écrivez les fonctions suivantes en Python dans le fichier *listes.py*, et testez les au fur et à mesure dans le fichier *listes_tests.py*. Pour faciliter les tests, vous testerez les méthodes sur des listes d'entiers.

1. Écrivez les fonctions simples sur les listes suivantes :
    - *__str__* qui affiche les éléments de la liste ;

- *length(self)* qui donne la longueur de la liste ;
- *get_ at(self,i)* qui renvoie le i-ème élément de la liste (de type class Cell) ;
- *get_ value(self,i)* qui renvoie la valeur du i-ème élément de la liste (de type object) ;
- *insert_ at(self,v, i)* qui renvoie la liste à laquelle on a inséré l'élément de valeur v en position i ;
- *delete_ value(self, v)* qui renvoie la liste à laquelle on a enlevé l'élément de valeur v (première occurrence de cet élément).

2. Construisez la liste [1,2,3,4,5,6] en utilisant les fonctions ci-dessus.

3. Écrivez les fonctions un peu plus avancées suivantes.

   - *map(self,f)* qui applique à chaque élément de la liste la fonction f. Par exemple, si l'on considère la liste L = [1,2,3] et la fonction f(x) qui permet d'incrémenter x de 1, L.map(f) renvoie la liste [2,3,4].
   - *count(self,v)* qui compte le nombre d'occurences des éléments de valeur v dans la liste (c'est-à-dire le nombre de fois qu'on trouve cette valeur dans la liste). Par exemple pour la liste L = [1,2,2,5,1,2] on a : L.count(2) qui retourne 3.
   - *filter(self,f)* qui renvoie la liste des éléments pour lesquels la fonction f renvoie True (on suppose que la fonction f ne renvoie que True ou False). Par exemple pour la liste L = [1,-2,3,-4,5] et la fonction f qui renvoie True si les éléments x sont positifs, L.filter(f) renvoir la liste L = [1,3,5]. On dit que l'on filtre les éléments de L en enlevant ceux qui ne vérifient pas la condition f(x) = True.
   - *reduce(self,f, x)* qui permet de calculer une valeur correspondant à la fonction f(x,y) propagée sur tous les élements de la liste. Plus précisément f commence par calculer $x1 = f(x, y0)$, où $y0$ est le premier élément de la liste et x est donné en paramètre, puis elle calcule $x2 = f(x1, y1)$ où $y1$ est le second élément de la liste, et ainsi de suite jusqu'à épuiser tous les éléments de la liste. On renvoie alors la dernière valeur obtenue.
     Par exemple, soit la liste L = [1,2,3] et la fonction f définie par f(x,y) = x + y, avec x qui vaut 1 à l'initialisation. Alors on a :
     x1 = x0 + L[0] = 1 + 1 = 2
     x2 = x1 + L[1] = 2 + 2 = 4
     x3 = x2 + L[2] = 4 + 3 = 7
     Par conséquent, L.reduce(f,1) retourne 7.
     L.reduce(f,0) retourne 6.

Exercice 2

Problème des trois couleurs – Le problème des trois couleurs consiste à regrouper par couleur une liste mélangée de boules rouges, vertes ou bleues.

1. Définissez une classe `Bowl` qui modélise une boule en vue de résoudre le problème des trois couleurs. Cette boule comporte un attribut `color` défini par une chaîne de caractères ("red", "green", ou "blue"). Vous écrirez les deux méthodes *__init__* et *__str__* qui permettent respectivement de construire une boule définie par sa couleur et d'afficher cette couleur.

2. Ecrivez une classe `BowlSortedList` qui permet de construire une liste de n boules, en insérant les boules à la bonne place. Les boules rouges précèdent les boules vertes qui elles-mêmes précèdent les boules bleues. Ecrivez les méthodes de la classe `BowlSortedList` suivantes :

   - `__init__` qui comporte les 3 attributs `nbRed`, `nbGreen` et `nbBlue` qui donnent les nombres de boules rouges, vertes et bleues (initialisés à 0), et l'attribut `list`, instance de la classe `Liste`.
   - `insert_bowl(self, bowl)` qui permet d'insérer une boule (de type `Bowl` à la bonne place dans la liste chaînée de boules. Vous utiliserez les fonctions d'insertion définies dans l'exercice 1 de ce TP.
   - `init_bowls(self,n)` qui permet de construire une liste chaînée de n boules à partir de la fonction `insert_bowl` définie ci-dessus. Vous tirerez aléatoirement la couleur de chacune des boules insérées en utilisant la fonction `randint` (`from random import randint`) entre 0 et 2 (par exemple 0 représente le rouge, 1 le vert et 2 le bleu).

3. Ecrivez la méthode `count_color(self, color)` qui affiche le nombre de boules de couleur donnée par le paramètre `color`. Par exemple, pour la liste de boules :

```
bowls: [red,red,red,red,red,green,green,green,blue,blue]
red:   5
green: 3
blue:  2
```

4. Ecrivez la méthode `first_bowl(self, color)` qui renvoie le couple (couleur, indice dans la chaîne) de la première occurrence de la boule de couleur donnée par le paramètre `color`. Par exemple, pour la liste de boules :

```
bowls: [red,red,green,green,green,green,blue,blue,blue,blue]
red:   ('red', 0)
green: ('green', 2)
blue:  ('blue', 6)
```

5. Ecrivez la méthode `keme_bowl(self, k, color)` qui renvoie le couple (couleur, indice) de la k-ème occurrence de la boule de couleur donnée par le paramètre `color`. Par exemple, pour k = 3 et la liste de boules :

```
bowls: [red,red,red,green,green,green,green,green,green,blue]
red:   ('red', 2)
green: ('green', 5)
blue:  None
```

6. Ecrivez la méthode `delete_bowls(self, color)` qui supprime toutes les boules d'une couleur donnée par le paramètre `color`. Par exemple, pour `color` = "green" et la liste de boules :

```
bowls: [red,red,green,green,green,blue,blue,blue,blue,blue]
on obtient :
bowls: [red,red,blue,blue,blue,blue,blue]
```

Exercice supplémentaire pour s'entraîner

Dans cet exercice on s'intéressera à l'implémentation des listes par des tableaux.

- Une liste est représentée par un tableau de taille MAX et un entier qui représente la taille de la liste.
- La liste vide est donc un tableau de taille MAX (comme toutes les autres) dont les valeurs n'ont pas d'importance et dont la taille est 0 ;
- La liste [10, 20, 30, 40] est représentée par un tableau de taille MAX , dont les 5 premières valeurs sont respectivement 10, 20, 30, 40 et l'entier 4 qui indique la taille de la liste.

Vous prendrez la valeur $MAX = 100$

1. Implémentez la classe *ListeTab* en vous inspirant de l'implémentation de la classe *Liste* de l'exercice 2.
2. Implémentez les fonctions suivantes :
   - *length(self)*
   - *get_ at(self,i)*
   - *insert_ at(self,v, i)*
   - *delete_ at(self, i)* (supprime l'élément situé à l'index i)