TP 5 – Listes
Année 2019-2020

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 *tp5-liste.py*). Pour les nouvelles fonctions et méthodes demandée, prenez soin de bien donner la spécification.

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 qui soit de taille variable, contrairement aux tableaux. 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 Cellule) définie par la structure suivante :

```
class Cellule():
    def __init__(self):
        self.data = object
        self.suivant = None
```

Chaque cellule porte une *data* (ici de type générique object), et pointe vers une autre cellule grâce à son champ suivant. Le nombre de cellules dans la liste sera égal au nombre de valeurs à stocker. La dernière cellule n'aura 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 donnée en annexe (que nous appellerons aussi $interface$).

Dans cet exercice, les listes seront implémentées par des listes chaînées simples. Une partie de l'implémentation des méthodes de la liste est donnée en annexe.

Écrivez les fonctions suivantes en Python :

1. Écrivez les fonctions simples sur les listes suivantes, et testez les au fur et à mesure.
   - *__str__* qui affiche les éléments de la liste ;
   - *longueur(self)* qui donne la longueur de la liste ;
   - *get_at(self,i)* qui renvoie le i-ème élément de la liste ;
   - *insere_at(self,e, i)* qui renvoie la liste à laquelle on a inséré l'élément e en position i;

- *supprime_at(self, i)* qui renvoie la liste à laquelle on a enlevé le i-ème 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. Vous pourrez éventuellement utiliser la fonction *get_valeur(self,i)* qui permet de récupérer la valeur de la donnée de l'élément en position i de la chaîne.

   - *map(self,f)* qui applique à chaque élément de la liste la fonction f ;
   - *count(self,e)* qui compte le nombre d'occurences de e dans la liste ;
   - *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) ;
   - *reduce(self,f, x)* : on suppose que la fonction f prend 2 arguments. Elle commence par calculer $x1 = f(x, y0)$, où $y0$ est le premier élément de la liste, 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.

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.

2. Ecrivez une classe `BowlSort` 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.

3. Ecrivez une méthode qui affiche le nombre de boules de couleur rouge, verte ou bleue.

4. Ecrivez une méthode qui renvoie l'indice de la première occurrence de boule verte.

5. Ecrivez une méthode qui renvoie l'indice de la k-ème occurrence de boule verte.

6. Ecrivez une méthode qui supprime toutes les boules d'une certaine couleur.

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 simples sur les listes de l'exercice 2 : *longueur(self)*, *insere_ at(self,e, i)*, *get_ at(self,i)*, *supprime_ at(self, i)*.

Annexe

Spécification de la classe Liste

```
class Liste:
    def __init__(self):
        """
        :sortie : self
        :post-cond: self est une liste vide
        """
    def est_liste_vide(self):
        """
        :entree : self
        :sortie v: bool
        :renvoie True si et seulement si self est vide
        """
    def tete(self):
        """
        :entree : self
        :sortie : object
        :pre-cond: self n'est pas vide
        :renvoie le premier element de la liste
        """
    def premiere_valeur(self):
        """
        :entree : self
        :sortie : object
        :pre-cond: self n'est pas vide
        :renvoie la valeur du premier element de la liste
        """
    def insere_tete(self, v):
        """
        :entree/sortie : self
        :entree v : object
        :insere v en tete de liste
        """


class Cellule():
    def __init__(self):
        self.data = object
        self.suivant = None

class Liste:
```

```
def __init__(self):
    self.mpremier = None
    self.taille = 0

def est_liste_vide(self):
    v = (self.mpremier == None)
    return v

def tete(self):
    if self.mpremier == None:
        print("liste vide")
    else:
        p = self.mpremier
        return p

def premiere_valeur(self):
    if self.mpremier == None:
        print("liste vide")
    else:
        s = self.mpremier.data
        return s

def insere_tete(self, v):
    m = Cellule()
    m.data=v
    if self.mpremier == None:
        m.suivant = None
    else:
        m.suivant=self.mpremier
    self.mpremier = m
    self.taille += 1
```