

LISTE CHAINEES 2/2 : Exercices

- Dans cette feuille, on complètera au fur et à mesure le code ci-dessous qui définit les classes permettant de manipuler les listes chaînées.
- Rappels :
 - Une liste chaînée est une structure de données pour représenter une séquence finie d'éléments.
 - Chaque élément est contenu dans une cellule, qui fournit par ailleurs un moyen d'accéder à la cellule suivante.
 - Les opérations sur les listes chaînées se programment sous forme de parcours qui suivent ces liaisons, en utilisant la récursivité ou une boucle.

```

In [13]: #liste chaînée

class Cell:
    '''cellule d'une liste chainee'''
    def __init__(self,valeur,suivant=None):
        self.valeur=valeur
        self.suivant=suivant

    def __str__(self):
        return str(self.valeur)

class Lc:
    '''Liste chaînée'''
    def __init__(self, tete=None):
        '''tete : premiere cellule'''
        self.tete=tete

    def __str__(self):
        '''renvoie une forme lisible de Lc'''
        if self.tete is None:
            return '[]'
        else:
            cellule=self.tete
            valeurs=[cellule.valeur]
            while cellule.suivant is not None:
                cellule=cellule.suivant
                valeurs.append(cellule.valeur)

            return str(valeurs)

#Ex 0
def last(self):
    '''renvoie la derniere cellule de la liste'''
    pass

#Ex 1
def ajoute(self, nouvelle):
    '''ajoute la cellule nouvelle en fin de liste'''
    pass

#Ex 1 v2
def ajoute(self, nouvelle):
    pass

#Ex 2
def trouve(self,x):
    '''renvoie l'index compté à partir de 0
    de la premiere occurrence de x dans la liste'''
    pass

#Ex 3
def __len__(self):
    pass

#Ex 4
def __getitem__(self,i):
    pass

#Ex 5
def __eq__(self,L):
    '''renvoie True si et seulement si
    les deux listes sont identiques'''
    pass

#Ex 6
def __add__(self,L):
    '''concatene les deux listes
    dans une nouvelle liste'''
    pass

#Ex 6bis
def __add__(self,L):

```

```

'''concatene les deux listes
dans une nouvelle liste'''
if self.tete is None:
    return L
else:
    #une nouvelle liste dont la valeur de la tete est celle de l1
    #et qui pointe vers la tete de la concaténation de la liste
    # dont la valeur de la tete est celle de self.tete.suivant
    # avec L
    pass

#Ex 7
def reverse(self):
    '''renverse la liste'''
    rev=Lc()
    pass

    self.tete=rev.tete #on pointe self vers la liste créée

```

Exercice 0 : Dernière cellule

Ecrire la méthode `last(self)` qui renvoie la dernière cellule de la liste chaînée. Si la liste est vide, la valeur `None` est renvoyée.

```

In [9]: #Tests dernière cellule
L=Lc()
print(L.last()) #None

L=Lc(Cell(0))
print(L.last()) #0

L=Lc(Cell(1,Cell(1,Cell(2,Cell(3,Cell(5,Cell(8,None)))))))
print(L.last()) #5

None
0
8

```

Exercice 1: ajouter une cellule

Ecrire la méthode `ajoute(self,nouvelle)` qui ajoute une cellule passée en argument à la fin de `self`. On pourra éventuellement utiliser la méthode de l'exercice précédent.

```

In [13]: #tests ajoute
L=Lc()
print(L)
L.ajoute(Cell(1))
print(L)
L.ajoute(Cell(1))
print(L)
L.ajoute(Cell(2))
print(L)
L.ajoute(Cell(3))
print(L)
L.ajoute(Cell(5))
print(L)

[]
[1]
[1, 1]
[1, 1, 2]
[1, 1, 2, 3]
[1, 1, 2, 3, 5]

```

Exercice 2 : Trouve élément

Ecrire la méthode `trouve(self, x)` qui renvoie l'index, numéroté à partir de 0 de la première occurrence de `x` dans la liste. Si `x` n'est pas une valeur contenue dans une des cellules, alors `False` est renvoyé.

```
In [31]: #Tests trouve élément
L=Lc()
print(L.trouve(2)) #False

L=Lc(Cell(0))
print(L.trouve(0)) #0
print(L.trouve(1)) #False

L=Lc(Cell(1,Cell(1,Cell(2,Cell(3,Cell(5,Cell(8,None)))))))
print(L.trouve(1)) #0
print(L.trouve(3)) #3
print(L.trouve(8)) #5
print(L.trouve(7)) #False

False
0
False
0
3
5
False
```

Exercice 3 : Longueur de la liste (récursivité)

Ecrire de façon récursive la méthode `__len__(self)` qui renvoie le nombre de cellules contenues dans la liste. Une liste vide a pour longueur 0 et sinon la longueur de la liste est égale à 1 + la longueur de la liste avec une cellule en moins.

```
In [32]: #Tests Longueur de Liste
L=Lc()
print(len(L)) #0
L=Lc(Cell(0))
print(len(L)) #1
L=Lc(Cell(1,Cell(2,Cell(3,Cell(4,Cell(5,None))))))
print(len(L)) #5

0
1
5
```

Exercice 4 : Accès aux éléments (récursivité)

Ecrire la méthode récursive `__getitem__(self,i)` qui renvoie la valeur de la cellule d'index donné, numéroté à partir de 0.

```
In [33]: #Tests accès éléments
L=Lc()
print(L[2]) # index error

L=Lc(Cell(0))
print(L[0]) #0
print(L[1]) #index error

L=Lc(Cell(1,Cell(1,Cell(2,Cell(3,Cell(5,Cell(8,None)))))))
print(L[0]) #1
print(L[4]) #5
print(L[5]) #8
print(L[6]) #Index error

index error
0
index error
1
5
8
index error
```

Exercice 5 : Identiques

Ecrire la méthode `__eq__(self,L)` qui renvoie le booléen `True` si les deux listes passées en arguments sont identiques (c'est à dire si les valeurs contenues dans les cellules sont les mêmes et dans le même ordre) et `False` sinon.

Rappel : cette méthode spéciale peut être appelée avec les caractères `==` .

```
In [105]: #Tests identiques
L1=Lc()
L2=Lc()
print(L1==L2) # True

L1=Lc(Cell(0))
L2=Lc(Cell(1))
print(L1==L2) # False

L1=Lc(Cell(1,Cell(1,Cell(2,Cell(3,Cell(5,Cell(8,None))))))
L2=Lc(Cell(1,Cell(1,Cell(2,Cell(3,Cell(5,Cell(8,None))))))
L3=Lc(Cell(1,Cell(2,Cell(3,Cell(5,Cell(8,None))))))
print(L1==L2) #True
print(L2==L1) #True
print(L1==L3) #False
```

```
True
False
True
True
False
```

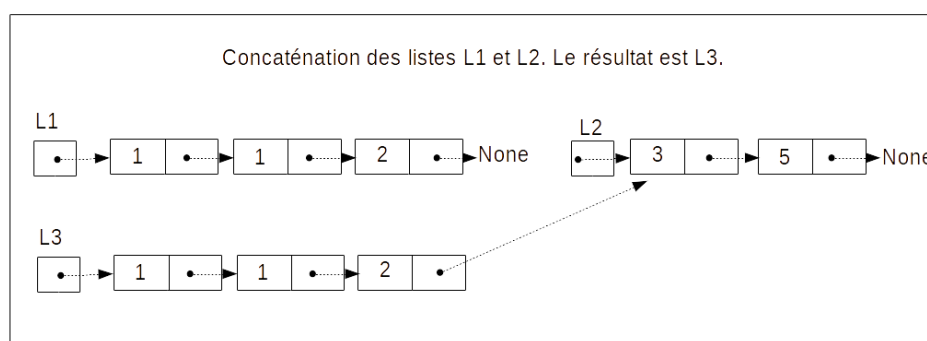
Exercice 6 : Concaténation

Ecrire la méthode `__add__(self,L)` qui concatène les listes passées en arguments et renvoie une nouvelle liste. On fera attention au cas où la liste `self` est vide.

Rappel : cette méthode peut être appelée avec l'opérateur `+` .

Attention :

- Pour deux listes `L1` et `L2` données, `L1+L2` est en général différent de `L2+L1`
- Il s'agit bien ici de construire une nouvelle liste constituée d'une copie de la liste `self` et dont la dernière cellule de `self` pointe vers la tête de `L` (voir schéma ci-dessous).



```
In [116]: #Tests concaténation
L1=Lc()
L2=Lc()
print(L1+L2) #[]

L3=Lc(Cell(0))
L4=Lc(Cell(1))
print(L3+L4) #[0,1]
print(L4+L3) #[1,0]
print(L3+Lc()) #[0]

L5=Lc(Cell(1,Cell(1,Cell(2,Cell(3,Cell(5,Cell(8, None)))))))
L6=Lc(Cell(13,Cell(21,Cell(34, None))))
L7=Lc()
print(L5+L6) # [1,1,2,3,5,8,13,21,34]
print(L6+L5) # [13,21,34,1,1,2,3,5,8]
print(L5+L7) # [1,1,2,3,5,8]

[]
[0, 1]
[1, 0]
[0]
[1, 1, 2, 3, 5, 8, 13, 21, 34]
[13, 21, 34, 1, 1, 2, 3, 5, 8]
[1, 1, 2, 3, 5, 8]
```

Exercice 6bis : Concaténation (récursivité)

Modifier la méthode `__add__(self, L)` qui concatène les listes passées en arguments et renvoie une nouvelle liste en écrivant une version récursive.

Cas de base : Si la liste `self` est vide, on renvoie `L`.

Cas récursif : Le premier élément de la liste concaténée est le premier élément de `self` et le reste de la liste concaténée est obtenu récursivement en concaténant le reste de `self` avec `L`.

```
In [5]: #Tests concaténation
L1=Lc()
L2=Lc()
print(L1+L2) #[]

L3=Lc(Cell(0))
L4=Lc(Cell(1))
print(L3+L4) #[0,1]
print(L4+L3) #[1,0]
print(L3+Lc()) #[0]

L5=Lc(Cell(1,Cell(1,Cell(2,Cell(3,Cell(5,Cell(8, None)))))))
L6=Lc(Cell(13,Cell(21,Cell(34, None))))
L7=Lc()
print(L5+L6) # [1,1,2,3,5,8,13,21,34]
print(L6+L5) # [13,21,34,1,1,2,3,5,8]
print(L5+L7) # [1,1,2,3,5,8]

[]
[0, 1]
[1, 0]
[0]
[1, 1, 2, 3, 5, 8, 13, 21, 34]
[13, 21, 34, 1, 1, 2, 3, 5, 8]
[1, 1, 2, 3, 5, 8]
```

Exercice 7 : Renverser une liste (boucle)

Ecrire la méthode `reverse(self)` qui renverse la liste passée en argument. Ainsi, la liste 1, 2, 3 devient 3, 2, 1. Si la liste est vide, on renvoie une liste vide.

```
In [15]: #Ex7 : Tests renverser
L=Lc(Cell(1,Cell(2,Cell(3,None))))
L.reverse()
print(L) #[3,2,1]
L.reverse()
print(L)#[1,2,3]

L=Lc()
L.reverse()
print(L) #[]
```

```
[3, 2, 1]
[1, 2, 3]
[]
```

Exercice 8 :

1. Compléter le code du début avec les méthodes vues dans la première feuille sur les listes chaînées et non présentes ici.
2. Enregistrer ce code dans un fichier `LC.py`.
3. Ouvrir le fichier `LC_tests.py` avec un éditeur (IDLE par exemple) :
 - Que contient-il ?
 - Ajouter l'instruction qui convient en début de script pour que ce fichier s'exécute sans erreur.

Réponse et remarques :

1.
 - Il contient des tests des méthodes des classes. Un test est réussi lorsqu'aucun message d'erreur ne s'affiche.
 - L'instruction à écrire est `from LC import *`. On importe le fichier contenant les classes et les méthodes définissant une liste chaînée, comme un module python !