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 [1]:
```python
#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, t=None):
        '''tete : premiere cellule'''
        self.tete=t

    def __str__(self):
        '''renvoie une forme lisible de Lc'''
        if self.tete is None:
            return 'ø'
        else:
            cellule=self.tete
            valeurs='<'+ str(cellule.valeur)
            while cellule.suivant is not None:
                cellule=cellule.suivant
                valeurs=valeurs + '→ ' + str(cellule.valeur)

            return valeurs + '→ ø'



    #Ex 0
    def last(self):









        return

    #Ex 1
    def ajoute(self, nouvelle):









        pass

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









        pass

    #Ex 2
    def trouve(self,x):
```

```
    return

#Ex 3
def __len__(self):

    if self.tete is None:
        return 0
    else:
        return 1 + ...

#Ex 4
def __getitem__(self,i):

    if self.tete is None or i<0:
        return 'index error'
    elif i==0:
        return ...
    else:
        return ...
#Ex 5
def __eq__(self,L):

    cellule1=self.tete
    cellule2=L.tete

    if cellule1 is None and cellule2 is None:
        return ...

    while ...:
        cellule1=cellule1.suivant
        cellule2=cellule2.suivant

    if cellule1 is None and cellule2 is None:
        return ...
    else:
        return ...

#Ex 6
def __add__(self,L):

    if self.tete is None:
        return ...
    else:
        resultat=Lc(Cell(self.tete.valeur))
        cellule=...
        n_cellule=...

        while cellule.suivant is not None: #on crée une copie de self
            n_cellule.suivant=...
            cellule=...
            n_cellule=n_cellule.suivant

        ...=L.tete

        return resultat

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

    #Ex 7
    def reverse(self):









        pass
```

Exercice 0 : Derniere 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 [5]:
```
#Tests dernière cellule
L=Lc()
print(L.last()) #None

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

```
None
<0→ ø
0
```

In [6]:
```
L=Lc(Cell(1,Cell(1,Cell(2,Cell(3,Cell(5,Cell(8,None)))))))
print(L)
print(L.last()) #5
```

```
<1→ 1→ 2→ 3→ 5→ 8→ ø
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 [7]:
```
#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)
```

```
ø
<1→ ø
<1→ 1→ ø
<1→ 1→ 2→ ø
<1→ 1→ 2→ 3→ ø
<1→ 1→ 2→ 3→ 5→ ø
```

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 [8]:
```
#Tests trouve élément
L=Lc()
print(L.trouve(2)) #False
```

```
False
```

In [11]:
```
L=Lc(Cell(0))
print(L)
print(L.trouve(0)) #0
print(L.trouve(1)) #False
```

```
<0→ ∅
0
False
```

In [12]:
```
L=Lc(Cell(1,Cell(1,Cell(2,Cell(3,Cell(5,Cell(8,None)))))))
print(L)
print(L.trouve(1)) #0
print(L.trouve(3)) #3
print(L.trouve(8)) #5
print(L.trouve(7)) #False
```

```
<1→ 1→ 2→ 3→ 5→ 8→ ∅
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 [13]:
```
#Tests accès éléments
L=Lc()
print(L[2]) # index error
```

```
index error
```

In [14]:
```
L=Lc(Cell(0))
print(L[0]) #0
print(L[1]) #index error
```

```
0
index error
```

In [15]:
```
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
```

```
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 appellée avec les caractères `==` .

In [16]:
```
#Tests identiques
L1=Lc()
L2=Lc()
print(L1==L2) # True
```

```
True
```

In [17]:
```
L1=Lc(Cell(0))
L2=Lc(Cell(1))
print(L1==L2) # False
```

```
False
```

In [19]:
```
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 : ",L1)
print("L2 : ",L2)
print("L3 : ",L3)

print(L1==L2) #True
print(L2==L1) #True
print(L1==L3) #False
```

```
L1 :  <1→ 1→ 2→ 3→ 5→ 8→ ø
L2 :  <1→ 1→ 2→ 3→ 5→ 8→ ø
L3 :  <1→ 2→ 3→ 5→ 8→ ø
True
True
False
```

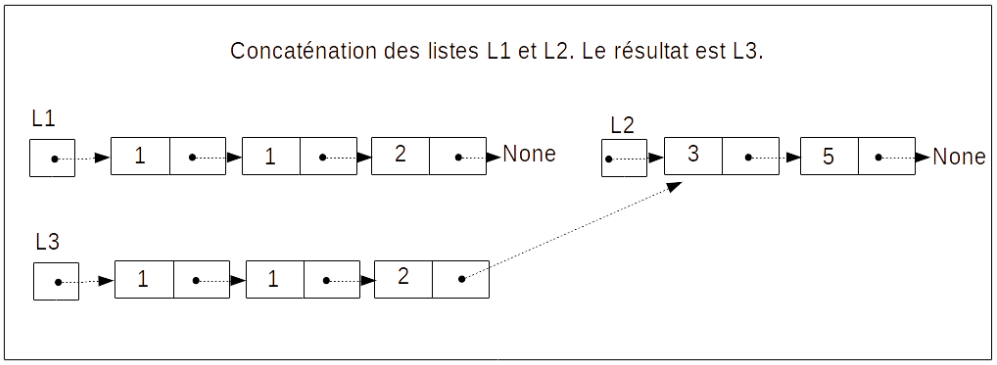
Exercice 6 : Concaténation

Ecrire la méthode `__add__(self,L)` qui concatène les listes passées en arguments et renvoie une nouvelle liste. On fera attention au cas ou 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 [22]:
```
#Tests concaténation
L1=Lc()
L2=Lc()
print(L1+L2) # ø
```

```
ø
```

In [23]:
```
L3=Lc(Cell(0))
L4=Lc(Cell(1))
print(L3+L4) # <0→ 1→ ø
print(L4+L3) # <1→ 0→ ø
print(L3+Lc()) #ø
```

```
<0→ 1→ ø
<1→ 0→ ø
<0→ ø
```

In [25]:
```
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→ ø
print(L6+L5) # <13→ 21→ 34→ 1→ 1→ 2→ 3→ 5→ 8→ ø
print(L5+L7) # <1→ 1→ 2→ 3→ 5→ 8→ ø
```

```
<1→ 1→ 2→ 3→ 5→ 8→ 13→ 21→ 34→ ø
<13→ 21→ 34→ 1→ 1→ 2→ 3→ 5→ 8→ ø
<1→ 1→ 2→ 3→ 5→ 8→ ø
```

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 élement de `self` et le reste de la liste concaténée est obtenu récursivement en concaténant le reste de `self` avec `L` .

In [26]:
```
#Tests concaténation
L1=Lc()
L2=Lc()
print(L1+L2) # ø
```

```
ø
```

In [28]:
```
L3=Lc(Cell(0))
L4=Lc(Cell(1))
print(L3+L4) # <0→ 1→ ø
print(L4+L3) # <1→ 0→ ø
print(L3+Lc()) # <0→ ø
```

```
<0→ 1→ ø
<1→ 0→ ø
<0→ ø
```

In [30]:
```
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→ ø
print(L6+L5) # <13→ 21→ 34→ 1→ 1→ 2→ 3→ 5→ 8→ ø
print(L5+L7) # <1→ 1→ 2→ 3→ 5→ 8→ ø
```

```
<1→ 1→ 2→ 3→ 5→ 8→ 13→ 21→ 34→ ø
<13→ 21→ 34→ 1→ 1→ 2→ 3→ 5→ 8→ ø
<1→ 1→ 2→ 3→ 5→ 8→ ø
```

Exercice 7 : Renverser une liste (avec une 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 [32]:
```
#Ex7 : Tests renverser
L=Lc(Cell(1,Cell(2,Cell(3,None))))
L.reverse()
print(L) # <3→ 2→ 1→ ø
L.reverse()
print(L)# <1→ 2→ 3→ ø

L=Lc()
L.reverse()
print(L) # ø
```

```
<3→ 2→ 1→ ø
<1→ 2→ 3→ ø
ø
```

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'éxécute sans erreur.

Réponse et remarques :

3.