# Python appliqué à la finance : listes chaînées

**Exercice 1 : Liste simplement chaînée**

Une liste chaînée désigne une structure de données représentant une collection ordonnée et de taille arbitraire d'éléments.

— L'accès aux éléments d'une liste se fait de manière séquentielle
— Chaque élément permet l'accès au suivant (contrairement au cas du tableau dans lequel l'accès se fait de manière absolue, par adressage direct de chaque cellule dudit tableau).
— Un élément contient un accès vers une donnée
— Le principe de la liste chaînée est que chaque élément possède, en plus de la donnée, des pointeurs vers les éléments qui lui sont logiquement adjacents dans la liste.
— Liste simplement chaînée : un seul pointeur vers l'élément suivant

On supposera que le code suivant est déjà implémenté :

```
class Maillon(object):
    def __init__(self,donnee):
        self.__donnee = donnee
        self.__suivant = None

    @property
    def donnee(self):
        return self.__donnee

    @donnee.setter
    def donnee(self,value):
        self.__donnee = value

    @property
    def suivant(self):
        return self.__suivant

    @suivant.setter
    def suivant(self,value):
        if not isinstance(value,Maillon) and value is not None:
            raise TypeError("Le suivant doit etre un maillon!")
        else:
            self.__suivant = value

    def __str__(self):
        return "Maillon (%s, suivant %s)" % (str(self.donnee),repr(self.
    suivant))

class LC(object):
    def __init__(self,premier=None):
        self.premier = premier

    @property
    def premier(self):
        return self.__premier

    @premier.setter
    def premier(self, maillon):
        if not isinstance(maillon,Maillon) and maillon is not None:
            raise TypeError("Le premier doit etre un maillon!")
        else:
            self.__premier = maillon
```

(a) Écrivez les méthodes suivantes de la classe LC en mettant éventuellement à jour les primitives précédentes :
— estVide(self) renvoie vrai si la liste est vide et faux sinon.

- supprimerPremier(self) supprime le premier élément de la liste.
- vider( self ) supprime tous les éléments de la liste.
- ajouterEnTete(x, self) ajoute x au début de la liste.
- ajouterApres(self , x, y) ajoute x dans la liste après y.
- supprimerSuivant(self,x) supprime le suivant de x dans la liste.
- __len__( self ) renvoie le nombre d'éléments de la liste.

(b) Écrire les méthodes __str__ des classes Maillon et LC.

(c) Écrire la classe itLC créant un itérateur permettant d'exécuter le code `for m in LC:`

(d) On veut améliorer la complexité de la méthode __len__(L). Pour ce faire, on introduit un nouvel attribut __taille qui contient le nombre d'éléments de la liste. Réécrivez toutes les méthodes précédentes en tenant compte de cet attribut.

(e) On suppose que l'on peut ordonner les maillons suivant leur donnée. Écrire une méthode de la classe Maillon rendant licite l'utilisation de l'instruction m1<m2.

(f) Écrire la méthode inserer( self , elt ,debut=None) qui insère le maillon elt dans une liste supposée triée après debut en respectant l'ordre des maillons.

(g) Soient deux listes chaînées triées $L1$ et $L2$. Proposez une méthode pour insérer les éléments de $L2$ dans $L1$ pour que $L1$ soit triée à la fin. On peut détruire $L2$.

**Exercice 2 : Liste doublement chaînée**

Une liste doublement chaînée est une liste qui, en plus de permettre l'accès au suivant d'un élément, permet l'accès au précédent d'un élément. Pour représenter cette nouvelle classe de liste, on introduit l'attribut precedent, qui contient une référence à l'élément précédent.

(a) Corrigez la classe Maillon pour inclure le precedent.

(b) Écrivez les méthodes suivantes en reprenant éventuellement le code de l'exercice 1.
- supprimerPremier(self) supprime le premier élément de la liste.
- vider( self ) supprime tous les éléments de la liste.
- ajouterApres(self,m1,m2) ajoute $m1$ dans la liste après $m2$.
- supprimer(self,m) supprime $m$ dans la liste.

(c) On veut maintenant gérer le dernier élément de la liste. On pourrait introduire un attribut contenant le dernier élément, mais, comme vous l'avez montré, cela complique les méthodes. Pour résoudre ce problème, on peut travailler avec des listes circulaires : on connaît le premier et le précédent du premier est le dernier élément de la liste. Afin de résoudre les problèmes de l'existence d'une donnée dans la liste, on introduit un élément fictif que l'on appelle sentinelle. Le premier élément réel de la liste devient donc le suivant de la sentinelle et le dernier élément de la liste est le précédent de la sentinelle. On peut donc supprimer l'attribut premier. On le remplace par l'attribut sentinelle. Écrire les méthodes suivantes :
- premier( self ) renvoie le premier élément de la liste.
- dernier( self ) renvoie le dernier élément de la liste.
- estVide( self ) renvoie vrai si la liste est vide et faux sinon.
- supprimerPremier(self) supprime le premier élément de la liste.
- vider( self ) supprime tous les éléments de la liste.
- ajouterApres(self,m1,m2) ajoute m1 dans la liste après m2.
- supprimer(self,m) supprime m dans la liste.

(d) On considère que L1 et L2 sont deux listes doublement chaînées circulaires avec sentinelle. Écrivez ensuite les méthodes ajouterEnFin(L1, L2) (ajoute L2 en fin de L1), et ajouterAuDebut(L1, L2).