Plan du Cours

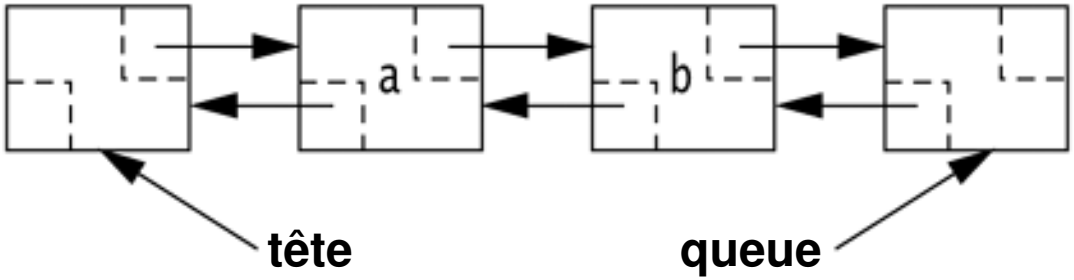
- Allocation Dynamique - Introduction
- Introduction au concept de Liste
  - Implémentation : Classe TabListe
  - Problèmes liés à l'utilisation d'un tableau
- Listes Chaînées (Simples)
  - Introduction
  - Ajout
  - Suppression
  - Maillon Tête Factice
- Accès
- Listes Chaînées Doubles
  - Insertion
  - Suppression
- Listes Chaînées Circulaires

Listes Chaînées Doubles (LCD)

- On veut également pouvoir parcourir la liste en sens inverse
- Une liste chaînée double permet le parcours bidirectionnel de la liste en utilisant un lien ***prec*** supplémentaire entre chaque maillon et son maillon précédent

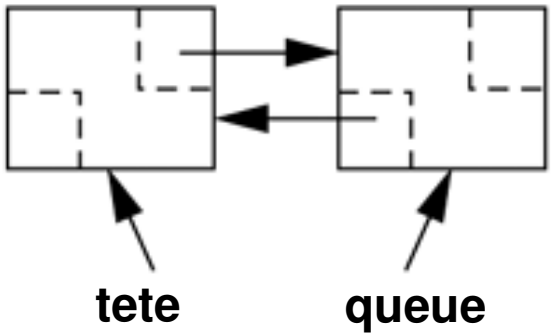
```
class ListeChaineeDouble :

        class MaillonDouble:
                def __init__(self, val):
                        self._valeur = val
                        self._prec = None
                        self._suiv = None
```

LCD avec Tête et Queue Factices



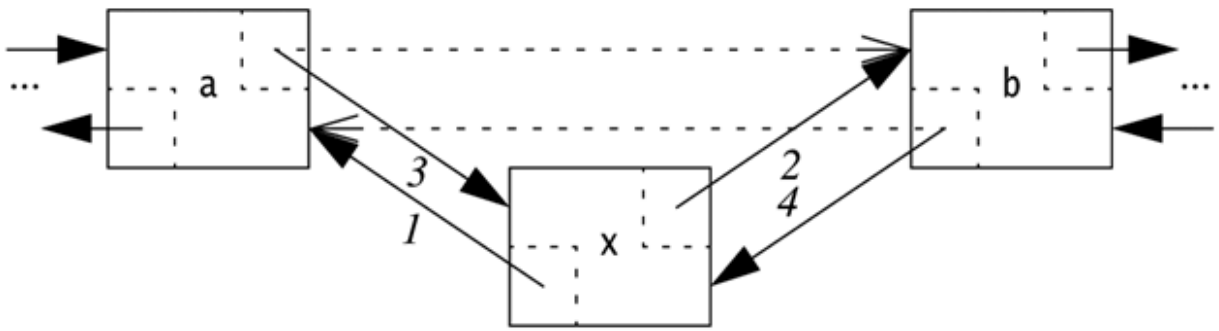
- Un maillon ***queue*** factice est utilisé en plus du noeud tête factice
  - Une liste vide consiste en deux maillons *tete* et *queue* initialisés tel que

```
tete._prec = None
tete._suiv = queue
queue._prec = tete
queue._suiv = None
```



Insertion d'un Elément

```
tmp = ListeChaineeDouble.MaillonDouble(x)  // création d'un maillon et insertion de x
tmp._prec = courant                        // 1. courant précède tmp
tmp._suiv = courant._suiv                  // 2. suivant de courant suit tmp
tmp._prec._suiv = tmp                      // 3. tmp suit courant
tmp._suiv._prec = tmp                      // 4. tmp précède son suivant
```



Suppression d'un Elément

```
courant._prec._suiv = courant._suiv      // le maillon courant est contourné dans
                                         // la direction 'suiv'

courant._suiv._prec = courant._prec      // le maillon courant est contourné dans
                                         // la direction 'prev'
```

- Contrairement à la liste chaînée simple, l'élément courant est désormais supprimé à la place de l'élément qui le suit sachant que nous pouvons avoir la référence au maillon précédent en utilisant le lien ***prec***

- L'insertion et la suppression nécessitent deux fois plus de changements de liens que pour une liste chaînée simple

Classe LCDListe : Interface

```
def __init__(self):
 """
 :sortie self:
 :post-cond: liste chaînée avec tête et queue factices
 """

def ajout(self,pos,x):
 """
 :entrée-sortie self:
 :entrée x: object
 :entrée pos: int
 :pré-cond: --
 :post-cond: ajout de x à LCDListe à la position pos si elle existe
 """

def retourner_pos(self,x):
 """
 :entrée self:
 :entrée x: object
 :sortie i: int
 :pré-cond: l'élément x se trouve dans la liste
 :post-cond: i est sa position
 """
```

```
def set(self,id,nouvVal):
 """
 :entrée-sortie self:
 :entrée id: int
 :entrée nouvVal: object
 :sortie vieilleVal: object
 :pré-cond: l'élément à modifier est à la position id
 :post-cond: retourne l'ancien élément à la position id
 """

def suppr(self,id):
 """
 :entrée self:
 :entrée id:int
 :sortie supprVal: object
 :pré-cond: l'élément à supprimer est à la position id
 :post-cond: retourne l'élément supprimé supprVal
 """
```