Cours d'algorithmique et programmation 2

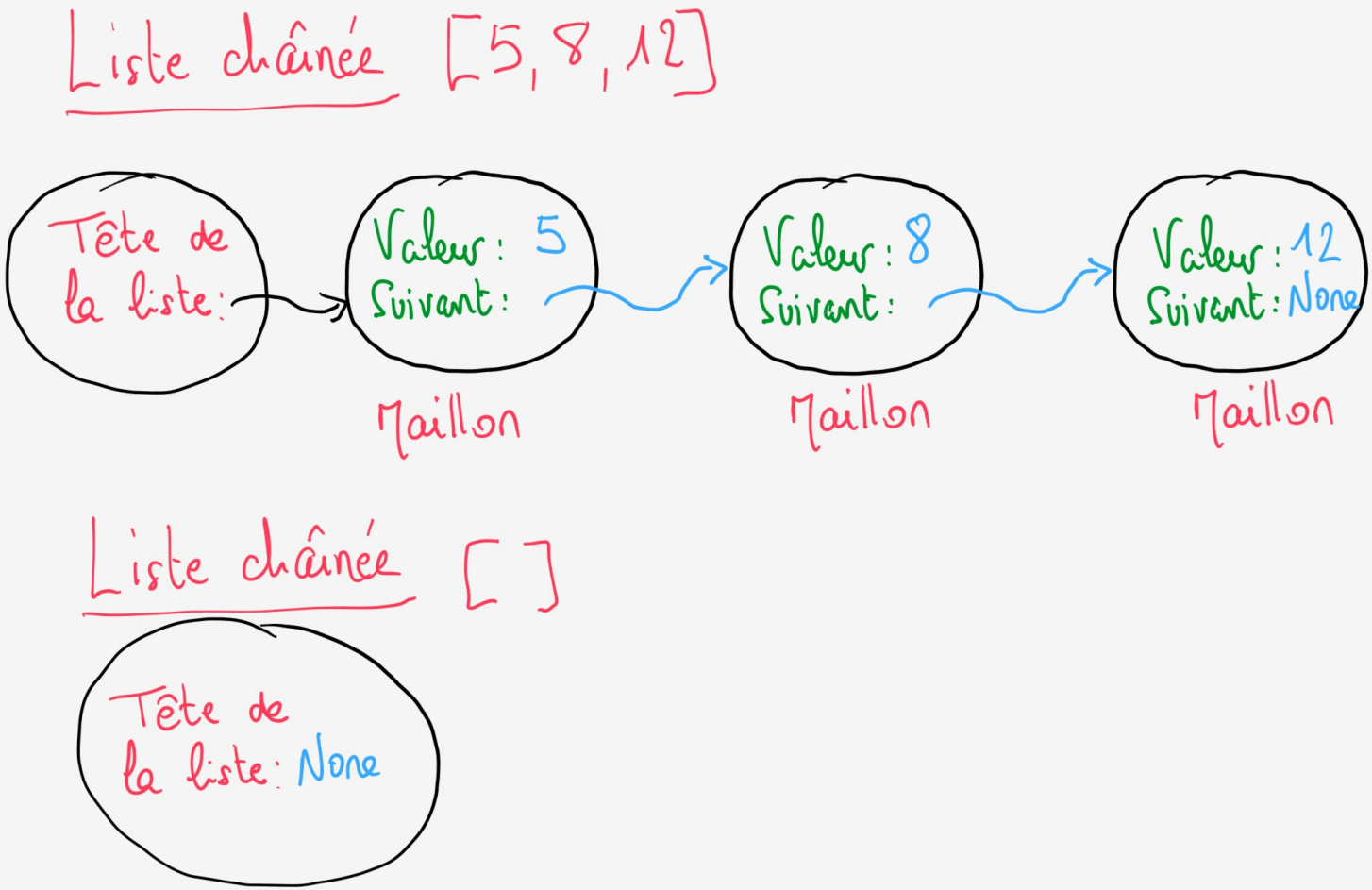
Anne Parrain

UFR des Sciences
Licence Sciences et Technologie
mentions Mathématiques et Informatique
Semestre 2

Section 1

Les listes chaînées

Construire sa propre liste (1)



Construire sa propre liste (3)

Une liste chaînée est une structure de données *récursive*.

Les éléments constitutifs d'une liste chaînée sont des maillons :

- soit un maillon vaut `None`
- soit un maillon possède deux attributs :
  - une valeur
  - un suivant, qui est lui-même un maillon

Une liste chaînée contient un attribut de type maillon, qui est la tête de la liste :

- si ce premier élément vaut `None`, la liste est vide
- sinon, elle contient toutes les valeurs des maillons qui se suivent (dans l'ordre où ils se suivent)

Construire sa propre liste (4)

```
1  class Maillon :
2      '''Classe qui modelise un maillon
3      d'une liste chainee.'''
4
5      def __init__(self,val,suiv=None):
6          '''Maillon, Objet, Maillon -> Maillon
7          construit un maillon d'une liste chainee.'''
8          self.__val = val
9          self.__suiv = suiv
10
11 class ListeChainee :
12     '''Classe qui modelise une liste sous la forme
13     recursive d'une liste chainee. '''
14
15     def __init__(self):
16         '''ListeChainee -> ListeChainee
17         une liste est toujours initialisee
18         a la liste vide.'''
19         self.__tete = None
```

Fonctionnalités basiques d'une liste

- savoir si la liste est la liste vide
- ajouter un élément au bout de la liste
- donner une représentation textuelle d'une liste
- connaître son nombre d'éléments
- récupérer la valeur du i^{eme} élément
- changer la valeur du i^{eme} élément
- insérer un élément à la position i
- supprimer un élément à la position i
- fournir une copie d'une liste
- ...

Tester si la liste est vide

Cela revient à regarder si la tête de liste vaut `None` !

```
1  class ListeChainee :
2
3      def est_vide(self) :
4          '''ListeChainee -> boolean
5
6          >>> l = ListeChainee()
7          >>> l.est_vide()
8          True
9          >>> l.append(1)
10         >>> l.est_vide()
11         False
12         '''
13         return self.__tete == None
14     ## fin est_vide
```

Section 2

Une version itérative pour les listes chaînées

Que savent faire les maillons ?

- ils permettent d'accéder aux valeurs de leurs attributs :
  - méthode `get()`
  - méthode `suivant()`
- ils permettent de poser un suivant sur un maillon :
  méthode `set_suiv(m)`
- ils sont capables de retourner une valeur textuelle d'eux-même :
  méthode `__str__()`

La classe `Maillon`

```
1  class Maillon :
2
3      def get(self):
4          '''Maillon -> Objet
5
6          >>> m = Maillon(1)
7          >>> m.get()
8          1
9          >>> m2 = Maillon(2,m)
10         >>> m2.get()
11         2
12         >>> m2.suivant().get()
13         1
14         '''
15         return self.__val
```

La classe `Maillon`

```
1  class Maillon :
2
3      def suivant(self):
4          '''Maillon -> Maillon
5
6          >>> m = Maillon(1)
7          >>> m.suivant()
8          >>> m2 = Maillon(2,m)
9          >>> m2.suivant().get()
10         1
11         '''
12         return self.__suiv
```

La classe `Maillon`

```
1  class Maillon :
2
3      def set_suiv(self,m) :
4          '''Maillon, Maillon -> None
5
6          >>> m = Maillon(1)
7          >>> m.set_suiv(Maillon(2))
8          >>> m.suivant().get()
9          2
10         '''
11         self.__suiv = m
```

La classe `Maillon`

```
class Maillon :

    def __str__(self):
        '''Maillon -> str

        >>> m = Maillon(1)
        >>> print(m)
        1
        >>> m2 = Maillon(2,m)
        >>> print(m2)
        2,
        '''
        mess = str(self.__val)
        if self.__suiv != None :
            mess += ', '
        return mess
## fin de la classe Maillon
```

Ajouter un élément au bout de la liste - À vous !

Cela revient à parcourir la liste chaînée. Arrivé sur le dernier maillon, il faut en créer un nouveau avec la valeur souhaitée.

```
1  class ListeChainee :
2      def append(self,val):
3          '''ListeChainee, Objet -> None
4
5          >>> l = ListeChainee()
6          >>> print(l)
7          []
8          >>> l.append(1)
9          >>> print(l)
10         [1]
11         >>> l.append(2)
12         >>> l.append(3)
13         >>> print(l)
14         [1, 2, 3]
15         '''
16         ... # a ecrire !
```

Ajouter un élément au bout de la liste (2)

```
1     def append(self,val):
2         '''ListeChainee -> str
3
4         tests unitaires ...
5         '''
6         if self.est_vide() :
7             self.__tete = Maillon(val)
8         else :
9             # prec : maillon auquel on va raccrocher
10            # le nouveau maillon
11            prec = self.__tete
12            while prec.suivant() != None :
13                prec = prec.suivant()
14            prec.set_suiv(Maillon(val))
15    ## fin append
```

Donner une représentation textuelle d'une liste - À vous !

```
1     def __str__(self) :
2         '''ListeChainee -> str
3
4         >>> l = ListeChainee()
5         >>> print(l)
6         []
7         >>> l.append(1)
8         >>> print(l)
9         [1]
10        >>> l.append(2)
11        >>> l.append(3)
12        >>> print(l)
13        [1, 2, 3]
14        >>> l.len()
15        3
16        '''
17        # a ecrire
```

Donner une représentation textuelle d'une liste (2)

```
1     def __str__(self) :
2         '''ListeChainee -> str
3         '''
4         mess = "["
5         prec = self.__tete
6         while prec != None :
7             mess += prec.__str__()
8             prec = prec.suivant()
9         mess += "]"
10        return mess
11    ## fin __str__
```

Le nombre d'éléments (1) - À vous !

Compte le nombre de maillons.

```
1     def len(self):
2         '''ListeChainee -> int
3
4         >>> l = ListeChainee()
5         >>> l.append(1)
6         >>> l.append(2)
7         >>> l.append(3)
8         >>> l.len()
9         3
10        '''
```

Le nombre d'éléments (2)

```
1     def len(self):
2         '''ListeChainee -> int
3         '''
4         nb = 0
5         prec = self.__tete
6         while prec != None :
7             prec = prec.suivant()
8             nb += 1
9         return nb
10    ## fin len
```

La valeur du i^{me} éléments (1) - À vous !

Retourne la valeur du maillon d'indice `ind`.

```
1     def get(self,ind):
2         '''ListeChainee, int -> Objet
3
4         >>> l = ListeChainee()
5         >>> l.append(1)
6         >>> l.get(0)
7         1
8         >>> l.append(2)
9         >>> l.append(3)
10        >>> l.get(1)
11        2
12        >>> l.get(2)
13        3
14        '''
```

La valeur du i^{me} éléments (2) - À vous !

```
1     def get(self,ind):
2         '''ListeChainee, int -> Objet
3         '''
4         assert self.__tete != None
5         cpt = 0
6         m = self.__tete
7         while cpt < ind :
8             assert m.suivant() != None
9             m = m.suivant()
10            cpt += 1
11        return m.get()
```

Supprimer le i^{me} élément (1) - À vous !

Supprime de la liste le maillon d'indice `ind`.

```
1     def delete(self,ind):
2         '''ListeChainee, int -> None
3
4         >>> l = ListeChainee()
5         >>> l.append(1)
6         >>> l.append(2)
7         >>> l.append(3)
8         >>> l.delete(1)
9         >>> print(l)
10        [1, 3]
11        >>> l.delete(0)
12        >>> print(l)
13        [3]
14        '''
```

Supprimer le i^{me} élément (2) - À vous !

```
1     def delete(self,ind):
2         '''ListeChainee, int -> None
3         '''
4         assert ind < self.len()
5         if ind == 0 :
6             self.__tete = self.__tete.suivant()
7         else :
8             prec = self.__tete
9             for i in range(ind-1):
10                prec = prec.suivant()
11            prec.set_suiv(prec.suivant().suivant())
12    ## fin delete
```

Insérer un élément à la i^{me} position (1) - À vous !

Insère dans la liste la valeur `val` à l'indice `ind`.

```
1     def insert(self,val,ind):
2         '''ListeChainee,Objet,int -> None
3
4         >>> l = ListeChainee()
5         >>> l.append(1)
6         >>> l.append(2)
7         >>> l.append(3)
8         >>> print(l)
9         [1, 2, 3]
10        >>> l.insert(8,0)
11        >>> print(l)
12        [8, 1, 2, 3]
13        >>> l.insert(12,2)
14        >>> print(l)
15        [8, 1, 12, 2, 3]
16        >>> l.insert(15,4)
17        >>> print(l)
18        [8, 1, 12, 2, 15, 3]
19        '''
```

Insérer un élément à la i^{me} position (2) - À vous !

```
1     def insert(self,val,ind):
2         '''ListeChainee,Objet,int -> None
3         '''
4         assert ind < self.len()
5         if ind == 0 :
6             self.__tete = Maillon(val,self.__tete)
7         else :
8             prec = self.__tete
9             for i in range(ind-1) :
10                prec = prec.suivant()
11            prec.set_suiv(Maillon(val,prec.suivant()))
12    ## fin insert
```