Cours d'algorithmique et programmation 2

Anne Parrain

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

Section 1

Une version récursive pour les listes chaînées

Maillon : une structure de données récursive

Pour le moment :

- c'est la liste qui fait tous les calculs
- les maillons ont des méthodes très limitées

On peut aussi décider de tirer parti de la nature récursive de la structure de données :

- la structure de données maillon se définit en fonction d'elle-même : un maillon, c'est une valeur et un suivant qui est un maillon
- de la même façon, on peut écrire des fonctions (ou des méthodes) qui sont récursives et se définissent en fonction d'elles-mêmes.

Nous allons implémenter une version récusive des listes chaînées avec les classes `ListeChaineeRec` et `MaillonRec`.

La classe `ListeChaineeRec`

- Dans un premier temps, la classe `ListeChaineeRec` va se contenter de déléguer le travail aux `MaillonRec`. Sa seule responsabilité est de gérer le cas de la liste vide.
- Le constructeur et la méthode `est_vide()` sont identiques
- Les tests unitaires sont identiques.

```
1     def append(self,val):
2         '''ListeChaineeRec -> str
3         '''
4         if self.est_vide() :
5             self.__tete = MaillonRec(val)
6         else :
7             self.__tete.append(val)
```

La classe ListeChaineeRec

```
    def len(self):
        '''ListeChaineeRec -> int
        '''
        if self.est_vide():
            return 0
        else :
            return self.__tete.len()

    def __str__(self) :
        '''ListeChaineeRec -> str
        '''
        mess = "["
        if not self.est_vide() :
            mess += self.__tete.__str__()
        mess += "]"
        return mess
```

La classe `MaillonRec`

Maintenant la classe `MaillonRec`

- n'a plus à fournir de méthodes pour manipuler ses attributs
- doit implémenter toutes les méthodes `append`, `len`, `__str__`, etc. . .attendues pour une liste
- un maillon n'est plus seulement un élément de la liste, il est le début de la liste qui commence par lui !

```
1     def append(self,val):
2         '''MaillonRec, Objet -> None
3         '''
4         if self.__suiv == None :
5             self.__suiv = MaillonRec(val)
6         else :
7             self.__suiv.append(val)
```

La récursivité

Les fonctions (ou méthodes récursives), comme une suite numérique, se composent

- d'une formule générale qui définit le calcul à faire en fonction du paramètre courant, et de cette même fonction rappelée sur une valeur (souvent plus petite) du paramètre
- d'un cas d'arrêt : une formule pour une valeur particulière du paramètre qui stoppe la récursivité.

Dans le cas général, les valeurs successives des paramètres appelés par récurrence doivent suivre une fonction qui converge en un temps fini vers la valeur du cas d'arrêt.

Dans le cas de la classe `MaillonRec`, le schéma général de progression du paramètre est le passage au maillon suivant, et le cas général est la fin de liste.

Fonctions récursives - un exemple

On choisit une fonction mathématiques qui peut se définir facilement par récurrence : la fonction factorielle

$n!$ se définit par :

- si $n = 0$ alors $0\,! = 1$
- sinon, $\forall n > 0, n! = n \times (n-1)!$

Fonctions récursives - un exemple

On choisit une fonction mathématiques qui peut se définir facilement par récurrence : la fonction factorielle

$n!$ se définit par :

- si $n = 0$ alors $0\,! = 1$
- sinon, $\forall n > 0, n! = n \times (n-1)!$

```
1 def fact(n) :
2     '''int -> int
3     n est un entier positif ou nul. Retourne n!
4     '''
5     assert n >= 0
6     if n == 0 :
7         return 1
8     else :
9         return n * fact(n-1)
```

La méthode `len`

Dans la classe `ListeChaineeRec`

```
1     def len(self):
2         '''ListeChaineeRec -> int
3         '''
4         if self.est_vide():
5             return 0
6         else :
7             return self.__tete.len()
```

Dans la classe `MaillonRec`

```
1     def len(self):
2         '''MaillonRec -> int
3         '''
4         if self.__suiv == None :
5             return 1
6         else :
7             return 1 + self.__suiv.len()
```

La méthode `__str__`

```
1     def __str__(self):
2         '''MaillonRec -> str
3
4         >>> m = MaillonRec(1)
5         >>> print(m)
6         1
7         >>> m2 = MaillonRec(2,m)
8         >>> print(m2)
9         2, 1
10        '''
11        mess = str(self.__val)
12        if self.__suiv != None :
13            mess += ", "+self.__suiv.__str__()
14        return mess
```

Les listes chaînées : une structure de données récursive

Une liste peut-être vue essentiellement comme une structure de données récursive :

```
1  class ListeChaineeRec :
2
3      def __init__(self):
4          '''ListeChaineeRec -> ListeChaineeRec
5          '''
6          self.__tete = None
7
8
9  class MaillonRec :
10
11     def __init__(self,val,suiv=None):
12         ''' MaillonRec, Objet, MaillonRec -> MaillonRec
13         '''
14         self.__val = val
15         self.__suiv = suiv
```

La méthode append - À vous !

```
1     def append(self,val):
2         '''ListeChaineeRec (inout), Objet -> None
3
4         >>> l = ListeChaineeRec()
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        '''
15        ## a ecrire
```

et au niveau de la classe `MaillonRec` ?

La méthode append (2)

Dans la classe `ListeChaineeRec`

```
1     def append(self,val):
2         '''MaillonRec, Objet -> None
3         '''
4         if self.__suiv == None :
5             self.__suiv = MaillonRec(val)
6         else :
7             self.__suiv.append(val)
```

Dans la classe `MaillonRec`

```
1     def append(self,val):
2         '''MaillonRec, Objet -> None
3         '''
4         if self.__suiv == None :
5             self.__suiv = MaillonRec(val)
6         else :
7             self.__suiv.append(val)
```

La méthode `get` - À vous !

```
1     def get(self,ind):
2         '''ListeChaineeRec, int -> Objet
3
4         >>> l = ListeChaineeRec()
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        '''
15        ## a ecrire
```

et au niveau de la classe `MaillonRec` ?

La méthode `get` (2)

Dans la classe `ListeChaineeRec`

```
1     def get(self,ind):
2         '''ListeChaineeRec, int -> Objet
3         '''
4         assert ind < self.len()
5         return self.__tete.get(ind)
```

Dans la classe `MaillonRec`

```
1     def get(self,ind):
2         '''MaillonRec, int -> Objet
3
4         '''
5         assert ind < self.len()
6         if ind == 0 :
7             return self.__val
8         else :
9             return self.__suiv.get(ind-1)
```

La méthode `set` - À vous !

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

et au niveau de la classe `MaillonRec` ?

La méthode `set` (2)

Dans la classe `ListeChaineeRec`

```
1     def set(self,ind, val):
2         '''ListeChaineeRec (inout), int, Objet -> None
3         '''
4         assert ind < self.len()
5         self.__tete.get(ind)
```

Dans la classe `MaillonRec`

```
1     def set(self,ind, val):
2         '''MaillonRec (inout), int, Objet -> None
3         '''
4         assert ind < self.len()
5         if ind == 0 :
6             self.__val = val
7         else :
8             self.__suiv.set(ind-1,val)
```

La méthode `delete` - À vous !

```
1     def delete(self,ind):
2         '''ListeChaineeRec, int -> None
3
4         >>> l = ListeChaineeRec()
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        '''
15        ## a ecrire
```

et au niveau de la classe `MaillonRec` ?

La méthode delete (2)

Dans la classe ListeChaineeRec

```
1     def delete(self,ind):
2         '''ListeChaineeRec, int -> None
3         '''
4         assert ind < self.len()
5         if ind == 0 :
6             self.__tete = self.__tete.suivant()
7         else :
8             self.__tete.delete(ind-1)
```

Dans la classe MaillonRec

```
1     def delete(self,ind):
2         '''MaillonRec, int -> None
3         '''
4         assert ind < self.len()
5         if ind == 0 :
6             self.__suiv = self.__suiv.__suiv
7         else :
8             self.__suiv.delete(ind-1)
```

La méthode `insert` - À vous !

```
1     def insert(self,val,ind):
2         '''ListeChaineeRec,Objet,int -> None
3
4         >>> l = ListeChaineeRec()
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        '''
20        ## a ecrire
```

La méthode insert (2)

Dans la classe ListeChaineeRec

```
1     def insert(self,val,ind):
2         '''ListeChaineeRec,Objet,int -> None
3         '''
4         assert ind < self.len()
5         if ind == 0 :
6             self.__tete = MaillonRec(val,self.__tete)
7         else :
8             self.__tete.insert(val,ind-1)
```

Dans la classe MaillonRec

```
1     def insert(self,val,ind):
2         '''MaillonRec,Objet,int -> None
3         '''
4         assert ind < self.len()
5         if ind == 0 :
6             self.__suiv = MaillonRec(val,self.__suiv)
7         else :
8             self.__suiv.insert(val,ind-1)
```

La méthode copy - À vous !

```
1     def copy(self):
2         '''ListeChaineeRec -> ListeChaineeRec
3         '''
4         ## a ecrire
```

et au niveau de la classe `MaillonRec` ?

La méthode copy (2)

Dans la classe ListeChaineeRec

```
1     def copy(self):
2         '''ListeChaineeRec -> ListeChaineeRec
3         '''
4         l = ListeChaineeRec()
5         if not self.est_vide() :
6             l.__tete = self.__tete.copy()
7         return l
```

Dans la classe MaillonRec

```
1     def copy(self):
2         '''MaillonRec -> MaillonRec
3         '''
4         m = MaillonRec(self.__val)
5         if self.__suiv is not None :
6             m.__suiv = self.__suiv.copy()
7         return  m
```