S54MA2M7 : Informatique 2
Types de données inductifs et dynamiques : la double vie des listes

Luigi Santocanale
LIS, Aix-Marseille Université

12 février 2019

Plan

Types inductifs

- Intro
- Crash course en théorie des catégories
- Algèbre initiale d'un foncteur

Programmer les listes en **python**

Plan

Nombres naturels, listes, arbres

- Un *nombre naturel* est ou bien
  - 0, ou bien
  - le successeur d'un autre nombre naturel.

- Une *liste* (avec valeurs dans A) est ou bien
  - la liste vide `Nil`, ou bien
  - c'est une tête (portant un étiquette un valeur) suivie par une autre liste (la queue).

- Un arbre binaire (complet) à valeur dans A est ou bien
  - une feuille (noté `Leaf`), ou bien
  - une valeur de A, et une couple (l, r) où l, r sont deux autres arbres binaires à valeur dans A.

Dans les langages de prog. fonctionnels

En Haskell :

```haskell
data Nat = Zero | Succ Nat
data List a =  Nil | Cons a (List a)
data Arbre a = Leaf | Noeud a (Arbre a) (Arbre a)
```

En Ocaml :

```ocaml
type nat = Zero | Succ of nat
type 'a list = Nil | Cons of ('a * 'a list)
type 'a tree = Leaf | Node of ('a * 'a tree * 'a tree)
```

Équations ensemblistes

On a :

$$\begin{aligned} \mathbb{N} &\simeq \{0\} + \mathbb{N} \\ \mathit{List}(A) &\simeq \{\mathtt{Nil}\} + A \times \mathit{List}(A) \\ \mathit{Tree}(A) &\simeq \{\mathtt{Leaf}\} + A \times \mathit{Tree}(A) \times \mathit{Tree}(A)\,. \end{aligned}$$

Attention, on peut avoir plusieurs solutions :

$$\begin{aligned} \mathbb{N}^+ &\simeq \{0\} + \mathbb{N}^+ \\ \mathit{List}^+(A) &\simeq \{\mathtt{Nil}\} + A \times \mathit{List}^+(A) \\ \mathit{Tree}^+(A) &\simeq \{\mathtt{Leaf}\} + A \times \mathit{Tree}^+(A) \times \mathit{Tree}^+(A)\,, \end{aligned}$$

où

$$\begin{aligned} \mathbb{N}^+ &= \mathbb{N} \cup \{\infty\} \\ \mathit{List}^+(A) &= \text{listes possiblement infinies avec valeur dans } A \\ \mathit{Tree}^+(A) &= \text{arbres binaires, possiblement infinis, avec étiquettes dans } A \end{aligned}$$

Objectif

Dénotons par 1 un sigleton (e.g. $1 := \{0\}$).

On souhaite donner un sens à l'énoncé suivant :

Proposition

$\mathbb{N}$, $List(A)$, $Tree(A)$ sont, respectivement, les plus petites solutions des systèmes d'équations ensemblistes

$$X \simeq 1 + X$$
$$X \simeq 1 + A \times X$$
$$X \simeq 1 + A \times X \times X\,.$$

Définition de catégorie

Une catégorie $\mathcal{C}$ consiste de

- une classe d'objets $Obj(\mathcal{C})$,
- un ensemble de flèches $\hom_{\mathcal{C}}(C, D)$, pour tout couple d'objets C, D,
- identités et loi de composition :
  - $id_C \in \hom_{\mathcal{C}}(C, C)$, pour tout objet C,
  - $g \circ f \in \hom_{\mathcal{C}}(C, E)$, pour $f \in \hom_{\mathcal{C}}(C, D)$ et $g \in \hom_{\mathcal{C}}(D, E)$.
- ces données satisfont :

$$f \circ id_C = f = id_D \circ f$$
$$h \circ (g \circ f) = (h \circ g) \circ f$$

chaque fois que ces flèches sont composables.

On écrit $f : C \to D$ pour $f \in \hom_{\mathcal{C}}(C, D)$.

Exemples

- $\mathcal{E}ns$: la catégorie des ensembles et fonctions entre ensembles.
- Ensembles et relations binaires (entre ensembles).
- Espaces topologiques et fonctions continues.
- Groupes et leur homomorphismes.
- Espaces vectoriels et fonctions linéaires.
- Ensembles partiellement ordonnés et fonctions croissantes.
- Un groupe (ou un monoide).
- Un ensembles partiellement ordonné (poset).

Objets initial et terminal

- Objet terminal : $1 \in \mathcal{C}$ tel que, pour tout $C \in \mathcal{C}$ il existe un et un seul flèche de C vers 1.
- Objet initial : $0 \in \mathcal{C}$ tel que, pour tout $C \in \mathcal{C}$ il existe un et un seul flèche de 0 vers C.
- Quels sont les objets terminale/initiale dans $\mathcal{E}ns$?
- . . . et dans un poset ?
- . . . dans un groupe ?

Lemme

Si A, B sont deux objets initiaux dans la mémé catégorie, alors il existe $f : A \to B$ et $g : B \to A$ tels que

$$g \circ f = id_A\,, \qquad\qquad f \circ g = id_B\,.$$

C'est-à-dire, ils sont "isomorphes".

Foncteur, définition

C'est un homomorphismes de catégories ! ! !

Un *foncteur* $F : \mathcal{C} \to \mathcal{D}$ consiste en la donnée de

- $F(C) \in \mathcal{D}$, pour tout $C \in Obj(\mathcal{C})$,
- $F(f) \in \hom_{\mathcal{D}}(F(C), F(C'))$,
  pour tout $C, C' \in Obj(\mathcal{C})$ et tout $f \in \hom_{\mathcal{C}}(C, C')$,

tels que

- $F(id_C) = id_{F(C)}$,
- $F(g \circ f) = F(g) \circ F(f)$.

Exemples

Les suivants

$$F(X) := 1 + X,$$
$$F(X) := 1 + A \times X,$$
$$F(X) := 1 + A \times (X \times X),$$

sont des foncteurs de $F : \mathcal{E}ns \to \mathcal{E}ns$.

Exercice

Quel est la valeur de $F(f)$ pour $f : B \to C$ et F ci-dessus ?

Remarque

X satisfait l'équation

$$X \simeq 1 + X, \qquad \text{resp.,} \quad X \simeq 1 + A \times X, \quad X \simeq 1 + A \times X \times X,$$

ssi $X \simeq F(X)$, où F est comme ci-dessus.

Catégories des algèbres d'un foncteur

- Soit $F : \mathcal{C} \to \mathcal{C}$ un foncteur.
- Vous pouves penser que $\mathcal{C} = \mathcal{E}ns$.
- On définit la catégorie des algèbres de F :

Objets :

$$\begin{array}{c} F(C) \\ \downarrow \alpha \\ C \end{array}$$

Flèches :

$$\begin{array}{ccc} F(C) & \xrightarrow{F(f)} & F(D) \\ \downarrow \alpha & & \downarrow \beta \\ C & \xrightarrow{f} & D \end{array}$$

Algèbre initiale d'un foncteur

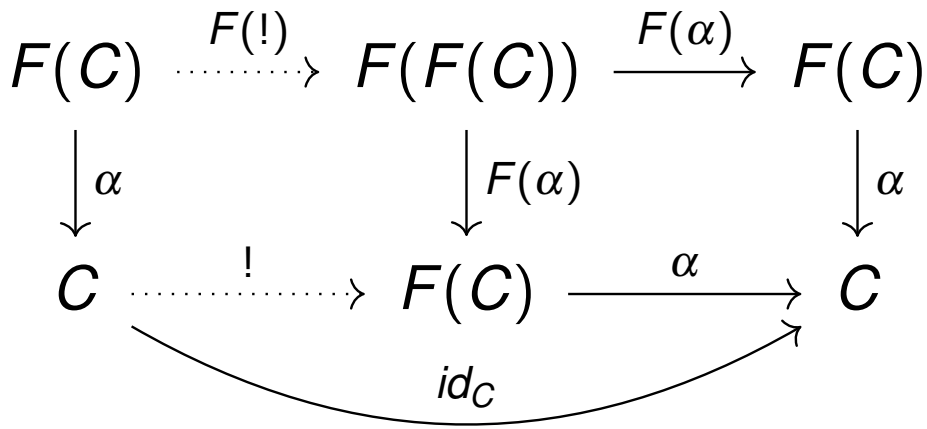
Algèbre initiale : algèbre (α, C) tel que, pour tout autre algèbre (β, D), il existe un unique morphisme $! : (\alpha, C) \to (\beta, D)$:

$$\begin{array}{ccc} F(C) & \overset{F(!)}{\dashrightarrow} & F(D) \\ \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \beta} \\ C & \overset{!}{\dashrightarrow} & D \end{array}$$

Le lemme de Lambek

Lemme

Si $\alpha : F(C) \to C$ est une algèbre initiale, alors α est inversible.



De $\alpha \circ ! = id_C$, on a aussi

$$! \circ \alpha = F(\alpha) \circ F(!) = F(\alpha \circ !) = F(id_C) = id_{F(C)}\,.$$

Le théorème de Tarski

Le lemme de Lambek généralise aux catégories le théorème de point fixe de Tarski :

Théorème

Soit L un treillis complet et $f : L \to L$ une fonction croissante. Alors

$$\bigwedge \{ x \in L \mid f(x) \leq x \}$$

est un point fixe de L, nécessairement le plus petit.

Retour aux systèmes d'équations fonctoriels

Un algèbre initiale du foncteur

- $F(X) = 1 + X$ est

$$\{0, \mathit{succ}\} : 1 + \mathbb{N} \to \mathbb{N}$$

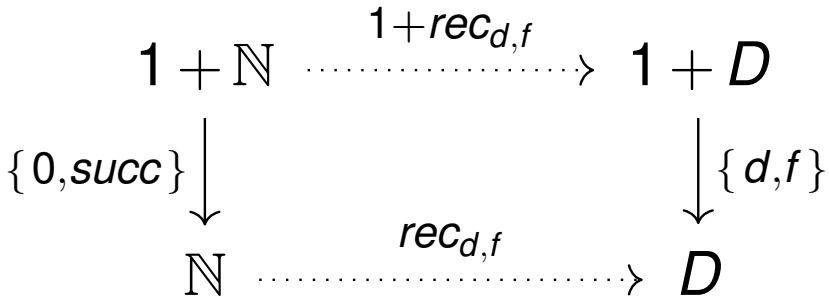
- $F(X) = 1 + A \times X$ est

$$\{\mathtt{Nil}, \mathtt{Cons}\} : 1 + A \times \mathit{List}(A) \to \mathit{List}(A)$$

- $F(X) = 1 + A \times X \times X$ est

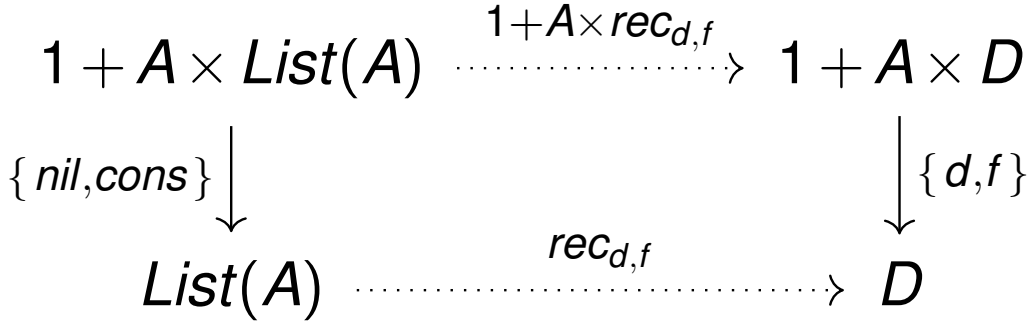
$$\{\mathtt{Leaf}, \mathtt{Node}\} : 1 + A \times \mathit{Tree}(A) \times \mathit{Tree}(A) \to \mathit{Tree}(A)\,.$$

Récursion et initialité : nombres naturels



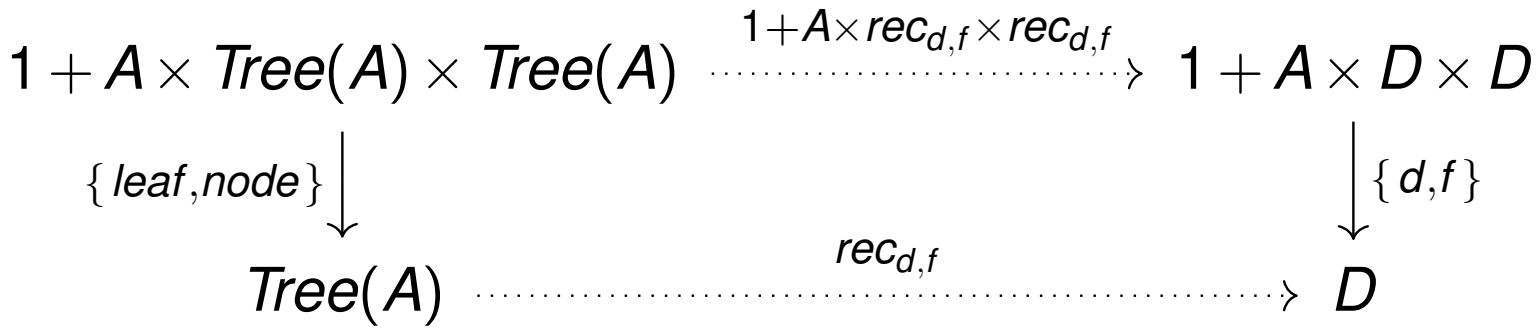
$$rec_{d,f}(0) = d\,,$$
$$rec_{d,f}(succ(n)) = f(rec_{d,f}(n))\,.$$

Récursion et initialité : listes



$$rec_{d,f}(nil) = d\,,$$
$$rec_{d,f}(cons(head, tail)) = f(head, rec_{d,f}(tail))\,.$$

Récursion et initialité : arbres



$$rec_{d,f}(leaf) = d\,,$$
$$rec_{d,f}(node(label, leftson, rightson)) = f(\,label, rec_{d,f}(leftson), rec_{d,f}(rightson))\,.$$

Moralité

- A vous de définir un type inductif adapté,
  il y aura toujours un principe de récursion associé.
- Avec les langages de programmation fonctionnels,
  il existe une mécanisme pour définir des types inductifs à votre goût.
- En utilisant les propriétés formelles des fonctions (par exemple, en utilisant les catégories)
  vous pouvez faire de preuves de correction des programmes.
- Pas nécessairement l'approche le plus efficace, en ce qui concerne le coût des calculs ou l'utilisation de la mémoire.

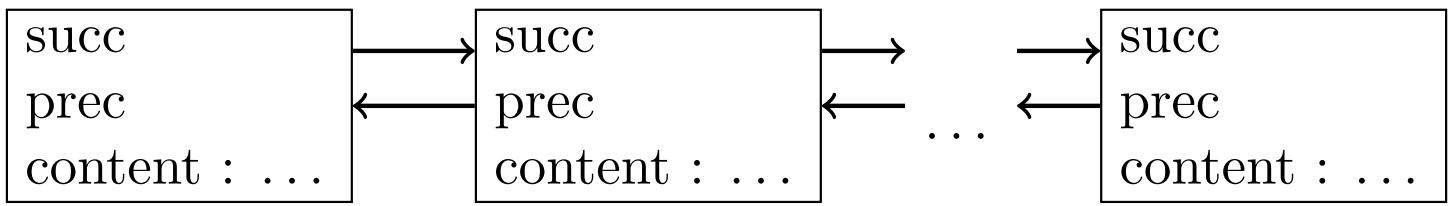
Plan

Structures de données (types) dynamiques

- On ne peut pas prévoir la taille de la mémoire nécessaire à stoker un objet de tel type.
- Exemples : listes, dictionnaires, arbres, integer
- Non-exemples : int, float, double, struct.
- Il faut allouer/libérer la mémoire à la demande.
- En langage C : il faut utiliser `malloc`, `free`.
- Dans la plupart des langages de programmation d'aujourd'hui, la mémoire est libéré par ramasse miette (en anglais : garbage collector).
- Même avec une ramasse miette, un grande nombre d'erreurs origine de la gestion de la mémoire pour ces types.
- Attention : types dynamiques $\neq$ types inductifs.

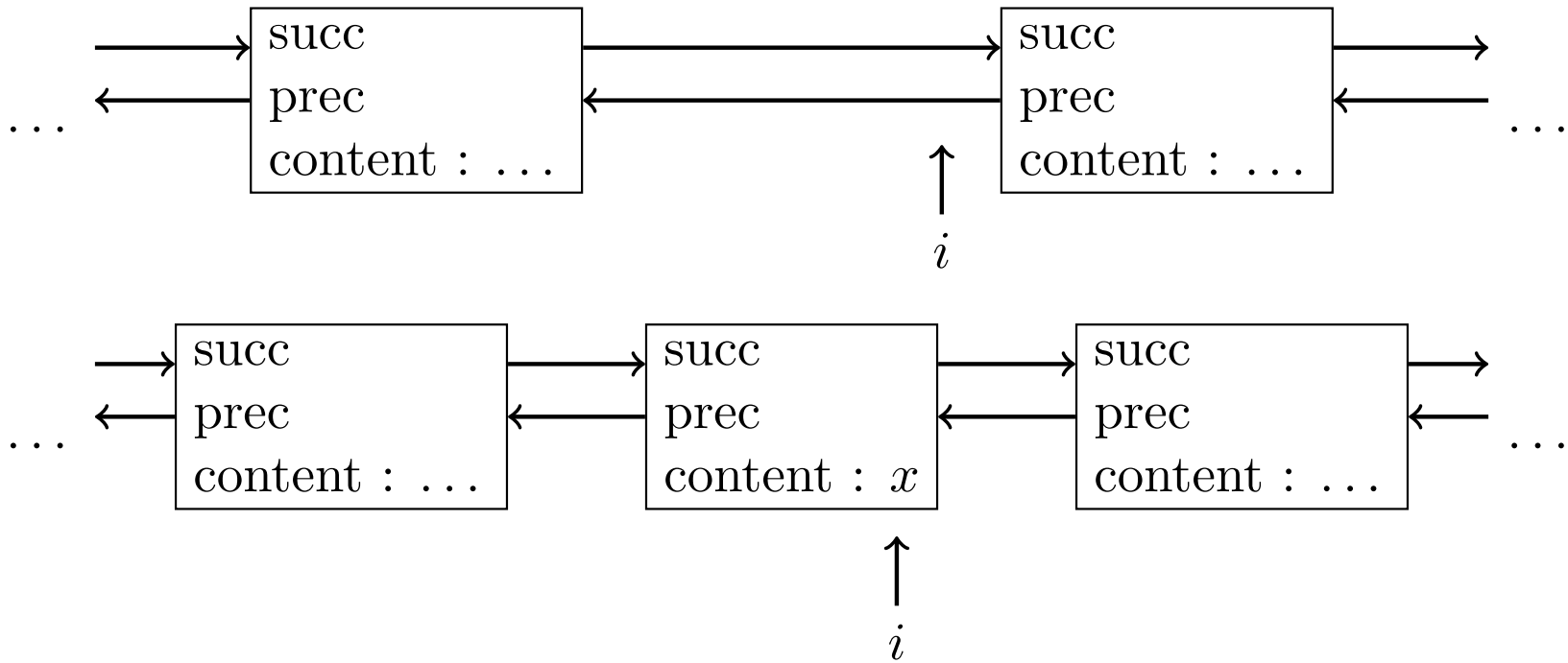
Implémentation des listes

Une liste, pour un ordinateur, est un collection de maillots reliés par des pointeurs :



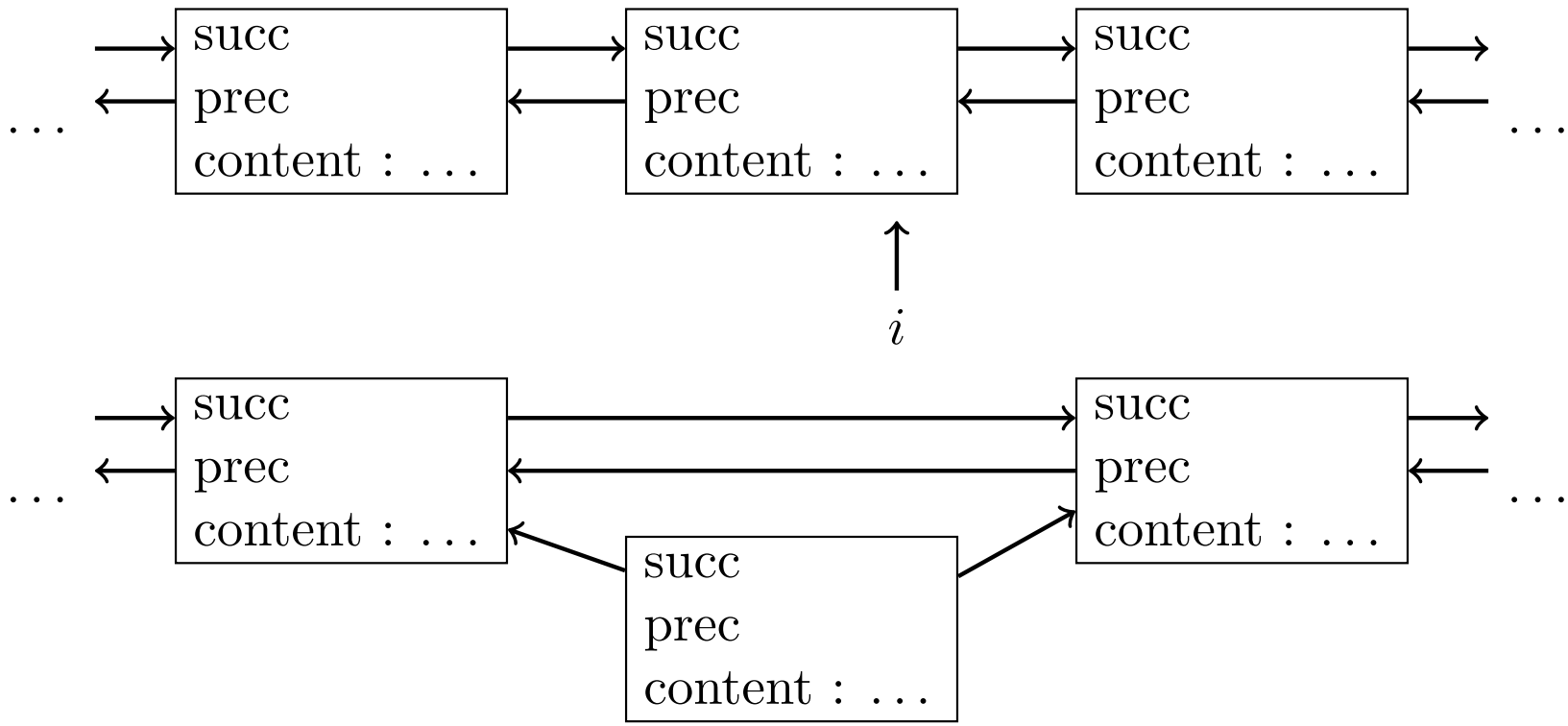
- Chaque conteneur est une structure, appelé *maillot de liste*.
- `succ`, `prec` et, possiblement, `content`, sont des pointeurs.
- On peut, en principe, se débarrasser de `prec`.

Insertion à la position i de x



- Attention à la première et à la dernière position ! ! !

Retirer de la position i



La classe abstraite

```
9  class ListAbstract():
10     """Classe abstraite des listes"""
11
12     errOutBounds = "Index {} out of range"
13     errEmptyList = "Operation not allowed on empty list"
14
15     @classmethod
16     def nil(cls):
17         """Retourne une liste vide"""
18         return cls.__init__()
19
20     @staticmethod
21     def cons(head, tail):
22         """Retourne la liste avec head en tête
23         et la liste tail à la suite"""
24         return tail.copy().insert(0, head)
25
26     def length(self):
27         """Return the length of a list"""
28
29     def head(self):
30         """Return the first element of the list"""
31
32     def tail(self):
33         """Return the a copy of the list, without the first
   ↪   element"""
```

La classe abstraite (II)

```
35      def append(self, element):
36          "Add an item to the end of the list."
37
38      def extend(self, iterable):
39          """Extend the list by appending all the items from the iterable. """
40
41      def insert(self, i, element):
42          """Insert an item at a given position.
43          Positions are counted starting from 0.
44          The first argument is the index of the element before which to insert."""
45
46      def remove(self, element):
47          """Remove the first item from the list whose value is equal to element.
48          It raises a ValueError if there is no such item."""
49
50      def pop(self, i=0):
51          """Remove the item at the given position in the list, and return it.
52          If no index is specified, a.pop() removes and returns the last item in
    ↪     the list."""
53
```

La classe abstraite (III)

```
54      def clear(self):
55          """Remove all items from the list."""
56
57      def count(self, element):
58          "Return the number of times element appears in the
    ↪  list."
59
60      def reverse(self):
61          "Reverse the elements of the list in place."
62
63      def copy(self):
64          "Return a shallow copy of the list."
```

La classe concrète

```
9  from listAbstract import ListAbstract
10
11 class MaillotDeListe():
12     """Classe pour isoler les operations
13     sur les maillots de liste"""
14
15     def __init__(self, value, succ=None, prev=None):
16         self.content = value
17         self.succ = succ
18         self.prev = prev
19
20     def link(self, other):
21         """Fait le lien entre deuc maillots.
22         Other peut etre None"""
23        #  We allow other be None
24         self.succ = other
25         if other is not None:
26             other.prev = self
27
28     def set_last(self):
29         """Un maillot devient le dernier"""
30         self.succ = None
31
32     def set_first(self):
33         """Un maillot devient le premier"""
34         self.prev = None
```

La classe concrète (II)

```
36 class List(ListAbstract):
37     """Implémnation des listes basée sur les maillots"""
38
39     def __init__(self):
40         self.first = None # Le premier maillot de liste
41         self.len = 0
42
43     def length(self):
44         return self.len
45
46     def head(self):
47         if self.len == 0:
48             raise Exception(self.errEmptyList)
49         return self.first.content
50
51     def tail(self):
52         if self.len == 0:
53             raise Exception(self.errEmptyList)
54         copy = self.copy()
55         copy.len = self.len -1
56         copy.first = self.first.succ
57         return copy
```

La classe concrète (III)

```
59      def __str__(self):
60          maillot, ret = self.first, "[ "
61          while maillot is not None:
62              ret += maillot.content.__str__() +", "
63              maillot = maillot.succ
64          return ret+" ]"
```

```
67      def __get_maillot(self, i):
68          """Retourne le maillot i,
69          en comptant à partir de 0"""
70          maillot, j = self.first, 0
71          while True:
72              if maillot is None:
73                  raise Exception(self.errOutBounds.format(i))
74              if j == i:
75                  return maillot
76              maillot, j = maillot.succ, j+1
77
78      def lookup(self, i):
79          maillot = self.__get_maillot(i)
80          return maillot.content
```

La classe concrète (IV)

```
82      def insert(self, i, element):
83          new = MaillotDeListe(element)
84          if i == 0:
85              new.link(self.first)
86              self.first = new
87          else:
88              previous = self.__get_maillot(i-1)
89              successor = previous.succ
90              previous.link(new)
91              new.link(successor)
92          self.len += 1
93          return self
94
95      def append(self, element):
96          return self.insert(self.len, element)
97
98      def extend(self, iterable):
99          for element in iterable:
100             self.append(element)
101         return self
```

La classe concrète (V)

```
103     def pop(self, i=-1):
104         if i == -1:
105             i = self.len
106 
107         maillot = self.__get_maillot(i)
108         content = maillot.content
109         previous = maillot.prev
110         if previous is None:
111             self.first = maillot.succ
112             if self.first is not None:
113                 self.first.set_first()
114         else:
115             previous.link(maillot.succ)
116         self.len -= 1
117         return content
```

La classe concrète (VI)

```
119     def remove(self, element):
120         maillot, i = self.first, 0
121         while maillot is not None:
122             if maillot.content == element:
123                 self.pop(i)
124                 return self
125             maillot, i = maillot.succ, i+1
126         raise ValueError
127 
128     def clear(self):
129         while self.first is not None:
130             self.pop(0)
131         return self
132 
133     def count(self, element):
134         maillot, tot = self.first, 0
135         while maillot is not None:
136             if maillot.content == element:
137                 tot += 1
138             maillot = maillot.succ
139         return tot
```

La classe concrète (VII)

```
141     def reverse(self):
142         len_ = self.length()
143         for i in range(len_):
144             content = self.pop(0)
145             self.insert(len_ -1 - i, content)
146 
147     def copy(self):
148         new_list = List()
149         return new_list
```

Programmer les arbres en **python** ? ? ?

A vous, en TP