INFO1

Magistère de Mathématiques de Rennes
1ère année

25 octobre 2016

école normale supérieure rennes

UNIVERSITÉ DE RENNES 1

Structure de données

Une structure de données est l'implémentation explicite d'un ensemble organisé d'objets, avec la réalisation des opérations d'accès, de construction et de modification afférentes.

Structure de données
Exemple : les tableaux

Création

- créer un tableau par une définition en extension
- créer un tableau de taille N avec une valeur par défaut V
- créer un tableau comme la concaténation de deux autres tableaux

```
t1 = [0, 3 , 4]
t2 = [10] * 5
t3 = t1+t2
```

Accès

- consulter la longueur du tableau
- consulter le contenu de la i-ème case du tableau

```
l = len(t2)
v = t1[1]
```

Modification

- modification de la case i-ème élément
- ajout d'un élément à la fin du tableau

```
t1[1] = 1
t1.append(10)
```

(appelés listes en Python)

Structure de données
Exemple : les entiers

Création

- créer un entier par une définition en extension
- créer un entier somme de deux autres entiers

```
i1 = 0
i2 = i1 + i0
```

Accès

- consulter le signe d'un entier
- calculer la représentation textuelle d'un entier

```
b = i2 > 0
s = str(i2)
```

Modification

les entiers sont *immuables*

Structure de données
Exemple : les rationnels

Exercice

Mutable/Immutable

Certaines structures de données sont immutables car elle ne proposent pas d'opérations de modification. Les changement de valeur doivent alors passer obligatoirement par des créations.

Les structures de données mutables privilégient la modification sur la création (allocation) de nouvelles valeurs. Attention cependant à bien comprendre les effets de bord d'une opération de modification et l'impact d'une copie.

Quizz

```
x = 1
y = x
x = 2

print "x = ", x
print "y = ", y
```

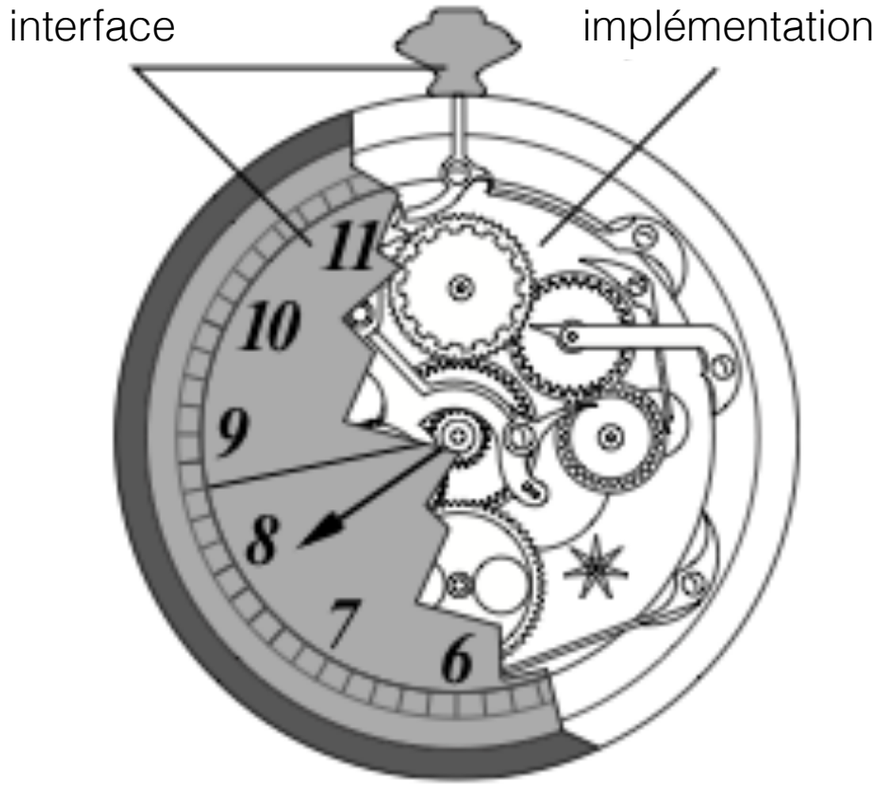
```
x = 1
t = [x, x, x]
x = 42

print "t =", t
```

```
t1 = [1 , 2, 3]
t2 = t1
t1[0] = 42

print "t1 =", t1
print "t2 =", t2
```

```
l = [1 , 2, 3]
t1 = [l, l ,l]
t2 = t1
l[0] = 42

print "t1 =", t1
print "t2 =", t2
```

Distinction

*Il faut distinguer **l'implémentation** d'une structure de donnée, de son **interface***



Étude de cas

- Ecrire les fonctions suivantes

```
def empty():
    """renvoie l'ensemble vide"""

def add(s,x)
    """modifie l'ensemble s, en lui ajoutant x"""

def remove(s,x)
    """modifie l'ensemble s, en lui enlevant x"""

def string(s):
    """renvoie une représentation de s sous forme
       de chaîne de caractères"""
```

- Sans utiliser les opérations python pré-définies **`append`**, **`extends`**, **`remove`** sur les tableaux

setm.py

```python
# -*- coding: utf-8 -*-

# codage d'un ensemble sous forme d'un tableau s = [t]
# avec t un tableau sans doublon
# afin de disposer d'un niveau d'indirection supplémentaire
# en attendant le cours sur la programmation objet...

def empty():
    """renvoie l'ensemble vide"""
    return [[]]

def add(s,x):
    """modifie l'ensemble s, en lui ajoutant x"""
    t = s[0]
    if not x in t:
        s[0] = t+[x]

def remove(s,x):
    """modifie l'ensemble s, en lui enlevant x"""
    t = s[0]
    if x in t:
        s2 = []
        for i in range(len(t)):
            if x != t[i]:
                s2 = s2+[t[i]]
        s[0] = s2

def string(s):
    """renvoie une représentation de s sous forme
       de chaîne de caractères"""
    res = "{"
    t = s[0]
    if len(t) > 0:
        res = res + str(t[0])
    for i in range(len(t)-1):
        res = res + ", " + str(t[i+1])
    res = res + "}"
    return res
```

```python
from setm import *

s = empty()
print string(s)

add(s,1)
print string(s)

add(s,2)
print string(s)

remove(s,1)
print string(s)
```

Programmation disciplinée

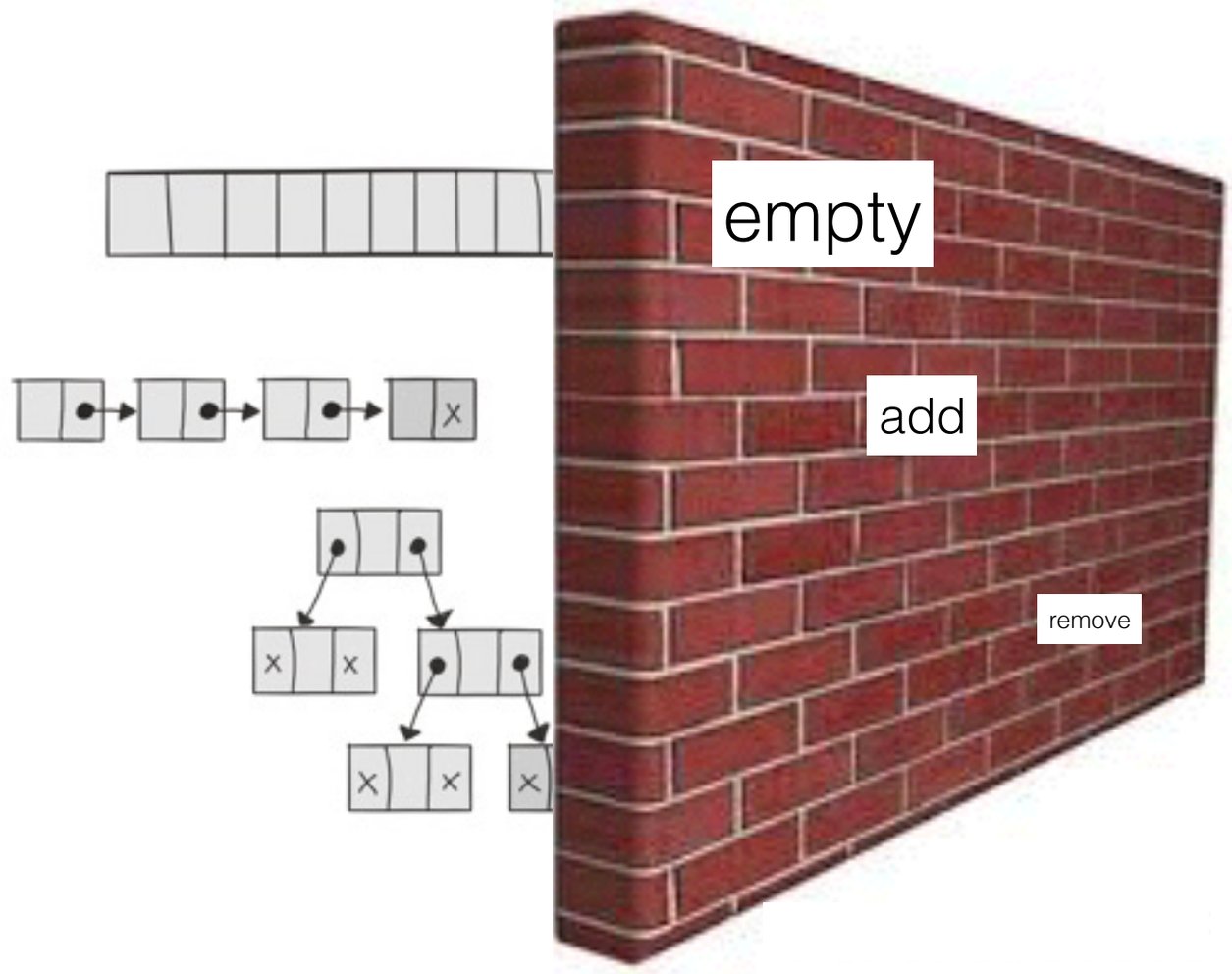
*L'**interface** spécifie les opérations disponible sur une structure de donnée **abstraite** (type abstrait)*

*La **librairie** propose une implémentation **concrète** conforme à la spécification décrite dans l'interface*

*Le **client** ne manipule la structure que par les opérations décrites dans l'interface. Il ne dépend pas des choix de l'implémentation concrète.*

L'efficacité des opérations reste cependant une information utile pour le client.

Librairie / Client



{1,3,4}

∅ {1,5}

Remarques

- La solution précédente n'est pas très élégante, mais elle illustre le besoin d'un niveau supplémentaire d'indirection

- Dans le cas présent, on aurait aussi pu s'appuyer sur les méthodes `append` et `remove` des listes Python

- Dans le cas général, il est utile de savoir manipuler la couche *objet* du langage de programmation Python

setm.py

```python
# -*- coding: utf-8 -*-

# codage d'un ensemble sous forme d'un tableau sans doublon

def empty():
    """renvoie l'ensemble vide"""
    return []

def add(s,x):
    """modifie l'ensemble s, en lui ajoutant x"""
    if not x in s:
        s.append(x)

def remove(s,x):
    """modifie l'ensemble s, en lui enlevant x"""
    if x in s:
        s.remove(x)

def string(s):
    """renvoie une représentation de s sous forme
       de chaîne de caractères"""
    res = "{"
    if len(s) > 0:
        res = res + str(s[0])
    for i in range(len(s)-1):
        res = res + ", " + str(s[i+1])
    res = res + "}"
    return res
```

```python
from setm import *

s = empty()
print string(s)

add(s,1)
print string(s)

add(s,2)
print string(s)

remove(s,1)
print string(s)
```

Programmation objet

- Toutes les fonctions précédentes agissent sur un élément de type *ensemble*

- On peut regrouper ces fonctions dans une *classe* pour former les *méthodes* d'un objet

- Le premier argument **`self`** de chaque méthode

  ```
  def m(self,x,y):…
  ```

  joue le rôle de receveur de l'action.

- L'appel de méthode suit la syntaxe `o.m(x,y)`

argument correspondant au paramètre self

Utilisation

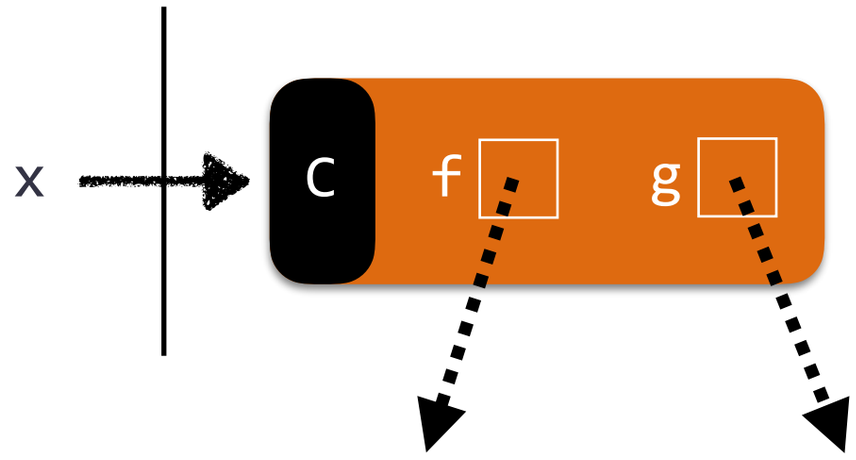
Version immutable

```
from seti import Seti

s = Seti()
print s

s = s.add(1)
print s

s = s.add(1)
print s

s = s.add(2)
print s

s = s.remove(1)
print s
```

Version mutable

```
from setm import Setm

s = Setm()
print s

s.add(1)
print s

s.add(1)
print s

s.add(2)
print s

s.remove(1)
print s
```

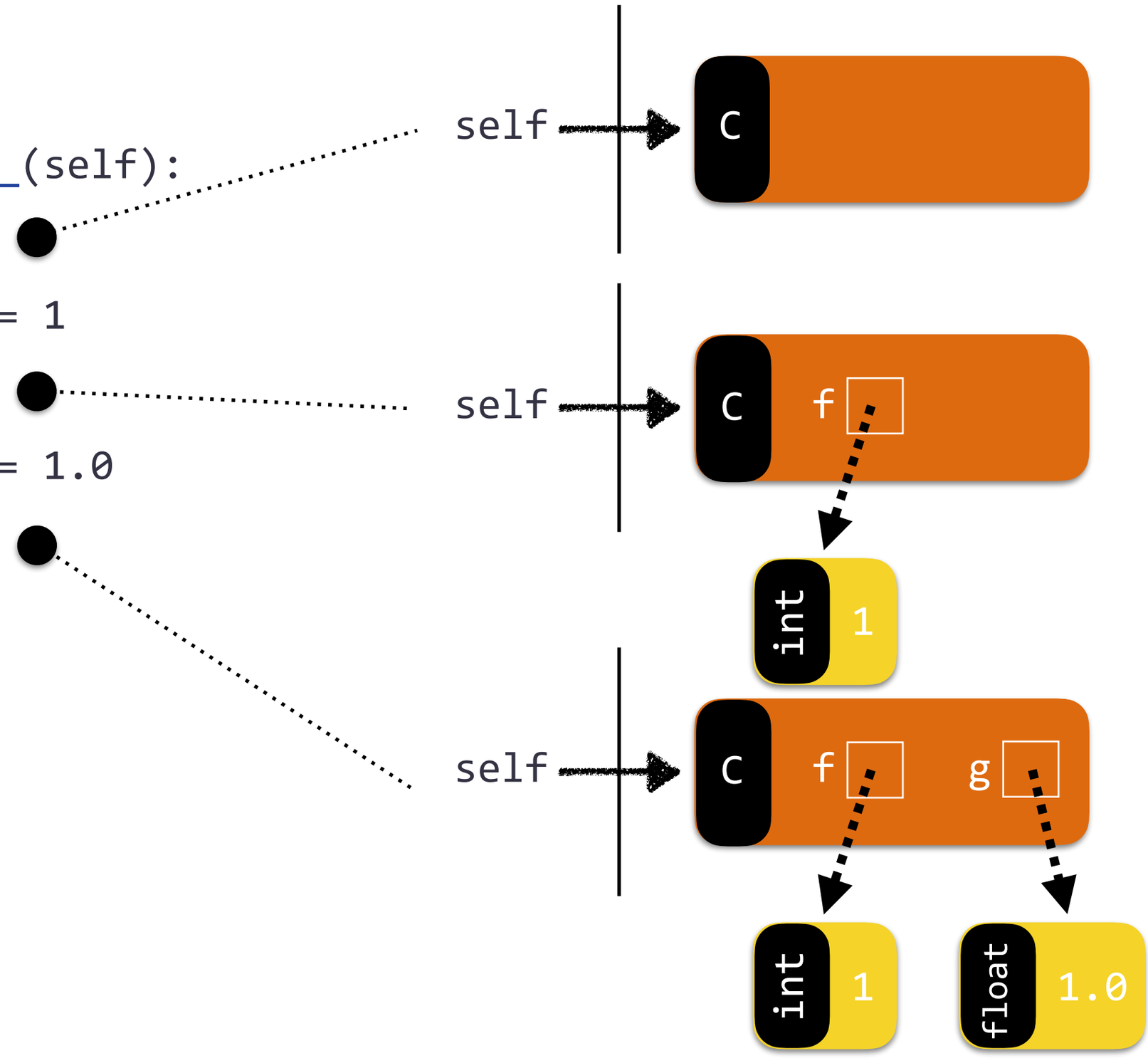
Un objet




```
x = C()

… = x.f
… = x.g

x.f = …
x.g = …
```

Construction

```
def __init__(self):
    self.f = 1
    self.g = 1.0
```



setm.py

```python
# -*- coding: utf-8 -*-

class Setm:

    def __init__(self):
        """initialise un ensemble vide"""
        self.set = []

    def add(self,x):
        """modifie l'ensemble self, en lui ajoutant x"""
        if not x in self.set:
            self.set = self.set+[x]

    def remove(self,x):
        """modifie l'ensemble self, en lui enlevant x"""
        if x in self.set:
            s2 = []
            for y in self.set:
                if x != y:
                    s2 = s2+[y]
            self.set = s2

    def __str__(self):
        """renvoie une représentation de s sous forme
           de chaîne de caractères"""
        res = "{"
        if len(self.set) > 0:
            res = res + str(self.set[0])
        for i in range(len(self.set)-1):
            res = res + ", " + str(self.set[i+1])
        res = res + "}"
        return res
```

```python
from setm import Setm

s = Setm()
print s

s.add(1)
print s

s.add(2)
print s

s.remove(1)
print s
```

Programmation objet de structures de données classiques

- Pile avec un tableau
- Pile avec une liste simplement chaînée
- File avec un tableau
- File avec une liste simplement chaînée cyclique

Pour des raisons pédagogiques on ne s’autorisera pas le redimensionnement de tableau

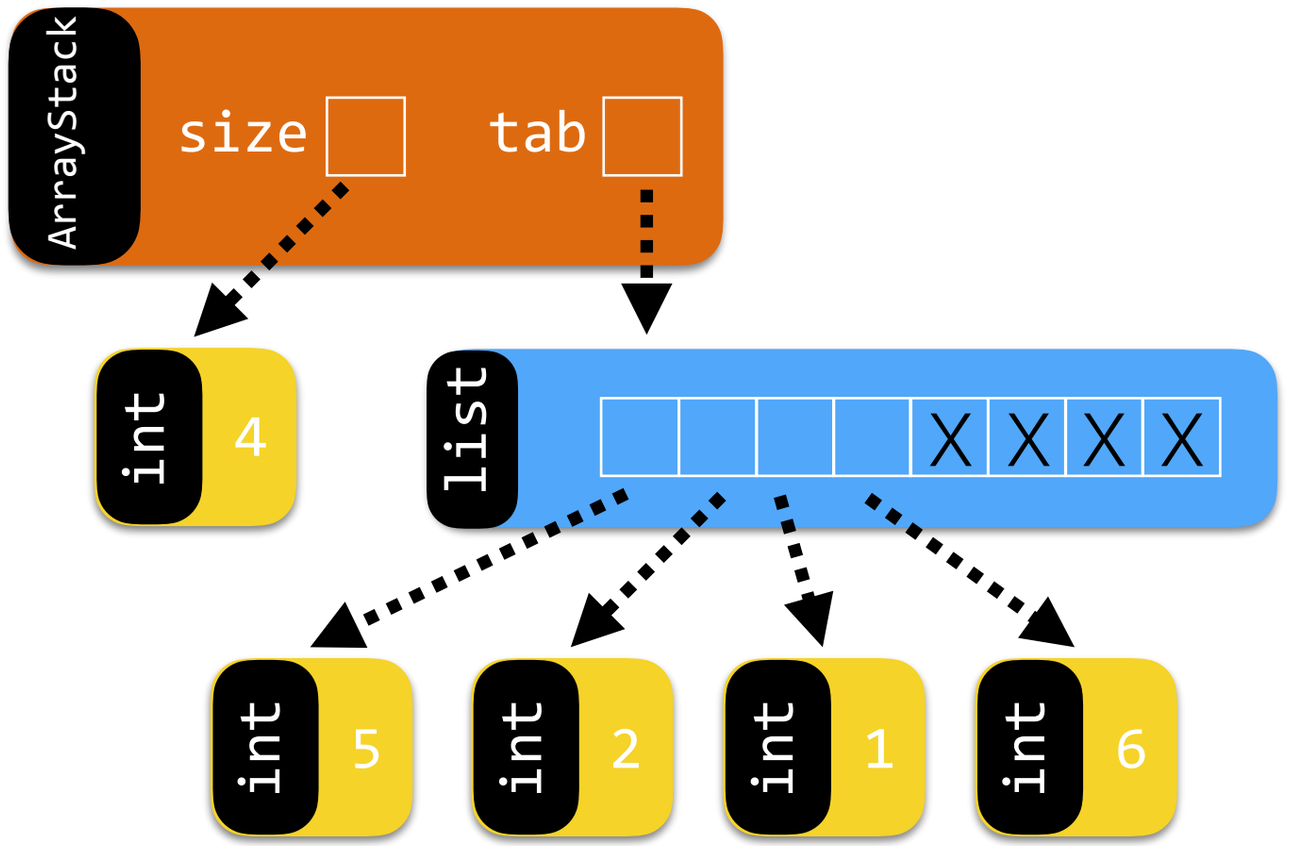
Flot de contrôle avancé
les exceptions

- On peut interrompre un calcul en lançant une exception

```
raise ValueError
```

- Le client peut ainsi être notifié d'une mauvaise utilisation d'une fonction de librairie
  Exemple : extraire un élément d'une collection vide

Pile (mutable) implémentée avec un tableau



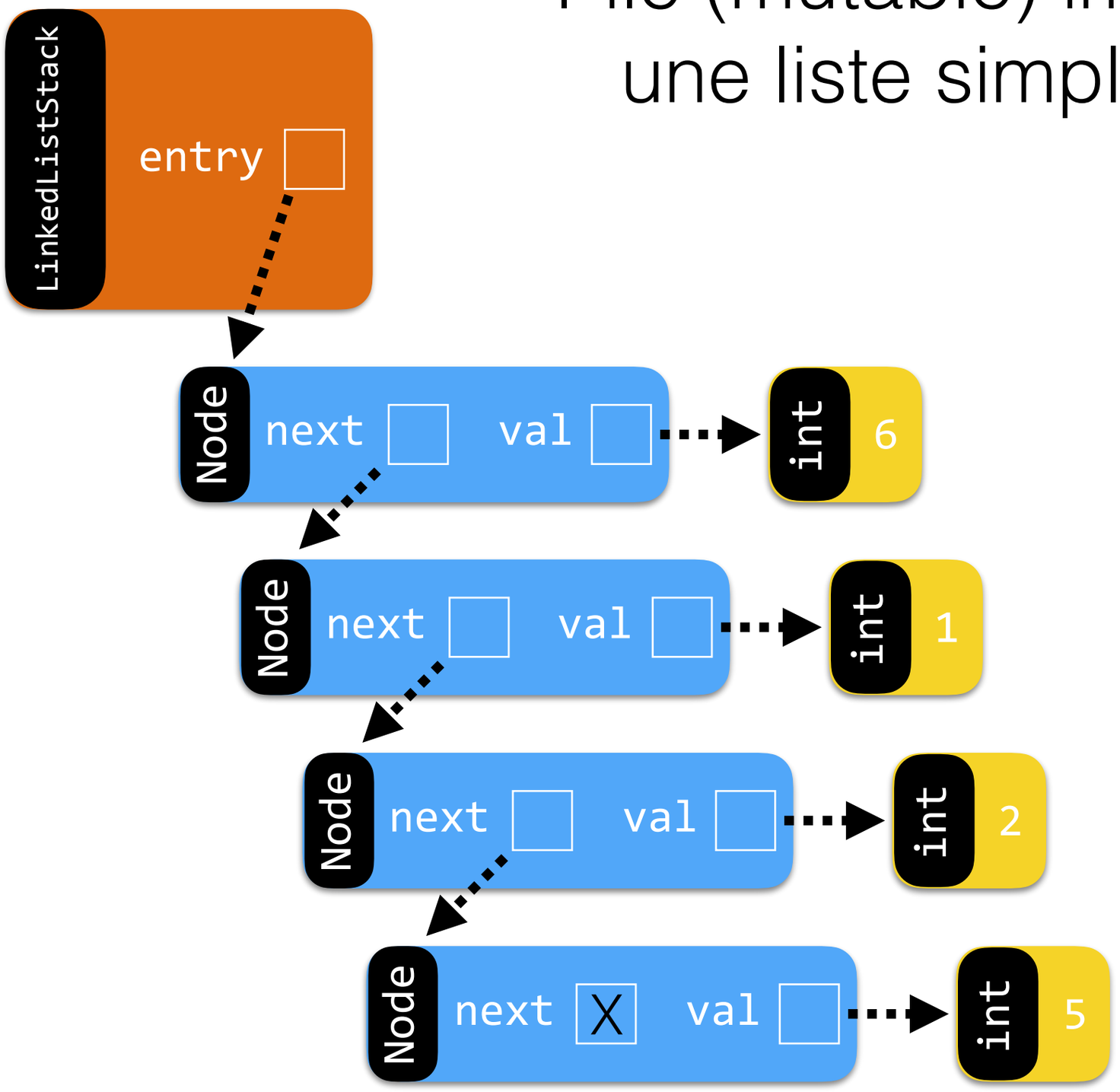

```
s = ArrayStack(8)
s.push(5)
s.push(2)
s.push(1)
s.push(6)
```

```python
class ArrayStack:

    def __init__(self,size_max):
        """Initialise une pile vide capacité max size_max."""
        self.size = 0
        self.tab = [None] * size_max

    def top(self):
        """Renvoie la tete de la pile self.
           Lance une exception ValueError si la pile est vide."""
        if self.size>0:
            return self.tab[self.size-1]
        else:
            raise ValueError

    def is_empty(self):
        """Teste si la pile self est vide."""
        return (self.size==0)
```

```python
    def push(self,x):
        """Ajoute l'élément x en tete de la pile self.
           Lance une exception ValueError si la pile est pleine."""
        if self.size < len(self.tab):
            self.tab[self.size] = x
            self.size = self.size + 1
        else:
            raise ValueError

    def pop(self):
        """Supprime la tete de la pile self.
           Lance une exception ValueError si la pile est vide."""
        if self.size > 0:
            self.tab[self.size-1] = None # inutile
            self.size = self.size -1
        else:
            raise ValueError
```

```
if __name__ == '__main__':
    s = ArrayStack(4)
    s.push(1)
    s.push(2)
    s.push(3)
    print "top(s) =", s.top()
    s.pop()
    print "top(s) =", s.top()
    s.pop()
    print "top(s) =", s.top()
    print "s vide ?", s.is_empty()
    s.pop()
    print "s vide ?", s.is_empty()
```

Pile (mutable) implémentée avec une liste simplement chaînée




```
s = LinkedListStack()
s.push(5)
s.push(2)
s.push(1)
s.push(6)
```

```python
class Node:
    def __init__(self,v):
        self.val = v
        self.next = None # inutile

class LinkedListStack:

    def __init__(self):
        """Initialise une pile vide capacité max size_max."""
        self.entry = None # inutile

    def top(self):
        """Renvoie la tete de la pile self.
           Lance une exception ValueError si la pile est vide."""
        if self.entry==None:
                raise ValueError
        return self.entry.val

    def is_empty(self):
        """Teste si la pile self est vide."""
        return (self.entry==None)

    def push(self,x):
        """Ajoute l'élément x en tete de la pile self."""
        n = Node(x)
        n.next = self.entry
        self.entry = n

    def pop(self):
        """Supprime la tete de la pile self.
           Lance une exception ValueError si la pile est vide."""
        if self.entry == None:
            raise ValueError
        self.entry = self.entry.next
```

```
if __name__ == '__main__':
    s = LinkedListStack()
    s.push(1)
    s.push(2)
    s.push(3)
    print "top(s) =", s.top()
    s.pop()
    print "top(s) =", s.top()
    s.pop()
    print "top(s) =", s.top()
    print "s vide ?", s.is_empty()
    s.pop()
    print "s vide ?", s.is_empty()
```