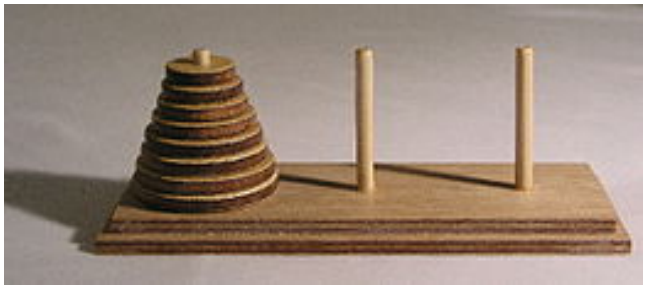
# Algorithmique et programmation

26 octobre 2016

David Pichardie

# Exercice : tours de Hanoï



- On dispose de 3 piquets et de N disques sur ces piquets
- On ne peut déplacer plus d'un disque à la fois
- On ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide
- Au départ, tous les disques sont sur le premier piquet
- Comment déplacer les disques pour positionner tous les disques sur le troisième piquet ?

```
def hanoi(t1,t2,t3,n):
    """affiche a l 'écran les déplacement pour déplacer
        n disques de t1 vers t3 en s'aidant de t2"""
    if n>0:
        hanoi(t1,t3,t2,n-1)
        print "déplacer disque", n, "de", t1, "à", t3
        hanoi(t2,t1,t3,n-1)


hanoi("A","B","C",3)
```

# 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 changements 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
```

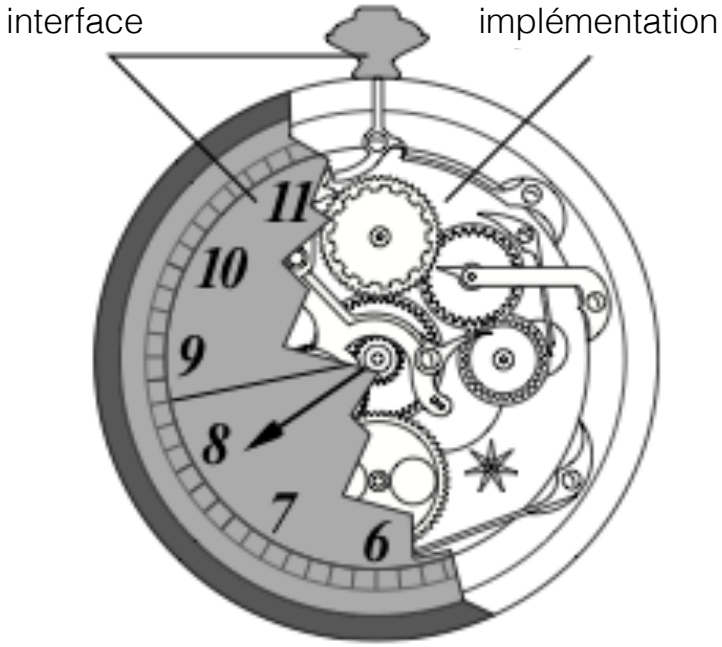
```
t1 = [1 , 2, 3]
t2 = t1
t1[0] = 42

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

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

print "t =", t
```

```
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

```
# -*- 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
```

```
from setm import *

s = empty()
print string(s)

add(s,1)
print string(s)

add(s,1)
print string(s)

add(s,2)
print string(s)

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

# Programmation disciplinée

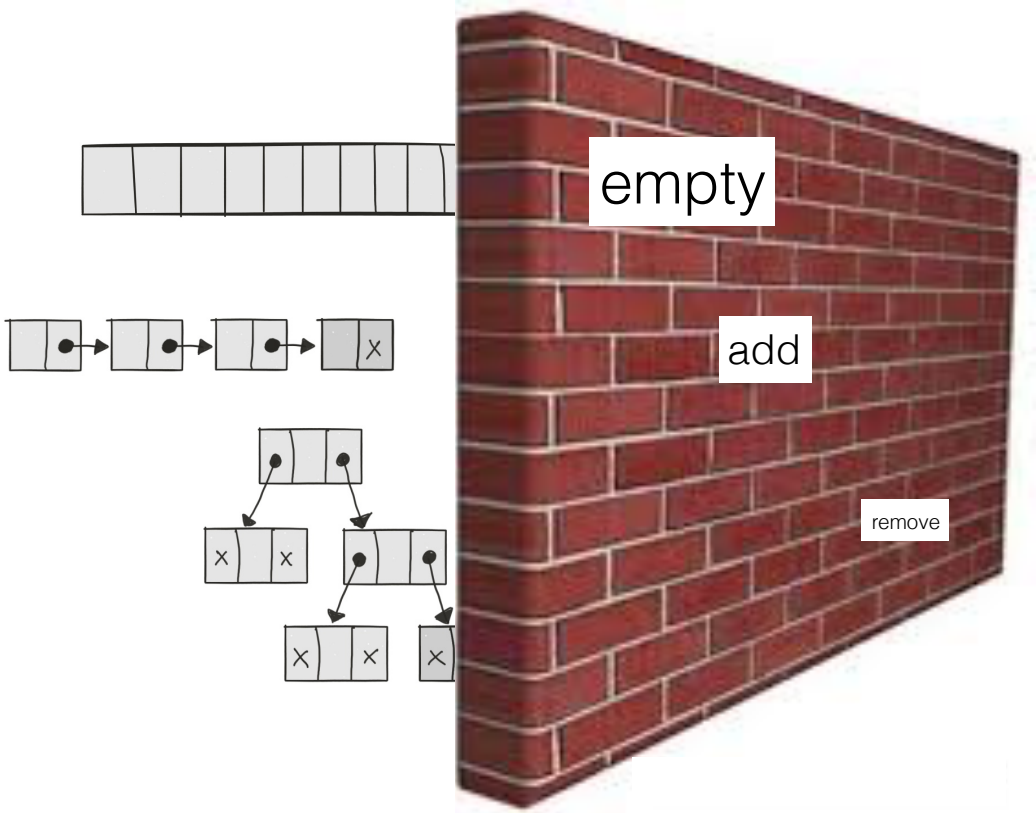
*L'**interface** spécifie les opérations disponible sur une structure de donnée **abstraite** (type abstrait)*

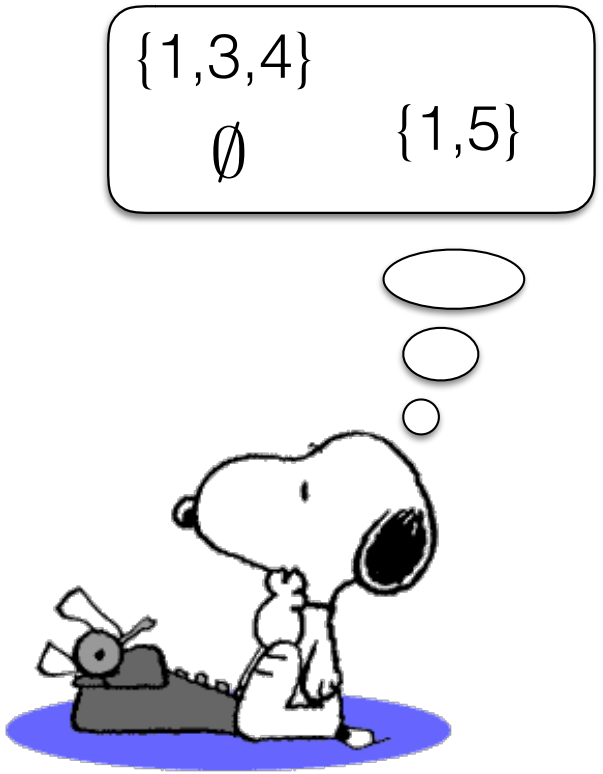
*La **librairie** propose une implémentation **concrète** conforme à la spécification décrite dans l'interface*

*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



# 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,1)
print string(s)

add(s,2)
print string(s)

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

# Exercice

- Ecrire les fonctions suivantes

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

def add(s,x)
    """renvoie l'union de s et {x}"""

def remove(s,x)
    """renvoie la différence ensembliste s\{x}"""

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

- On prendra soin de donner cette fois une version immutable de l'interface

seti.py

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

# codage d'un ensemble sous forme d'un tableau sans doublon
# en version immutable

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

def copy(s):
    """renvoie une copie du tableau s, sans partage immediat avec s"""
    return s[:]

def add(s,x)
    """renvoie l'union de s et {x}"""
    if not x in s:
        return s+[x]
        # une copie n'est pas necessaire car on ne modifie jamais l'entrée
    else:
        return s

def remove(s,x)
    """renvoie la différence ensembliste s\{x}"""
    if not x in s:
        return s
    else:
       t = copy(s).remove(x)
       return t

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
```

copie nécessaire !

on peut aussi faire une boucle comme dans setm.remove

```
from seti import *

s = empty()
print string(s)

s = add(s,1)
print string(s)

s = add(s,1)
print string(s)

s = add(s,2)
print string(s)

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

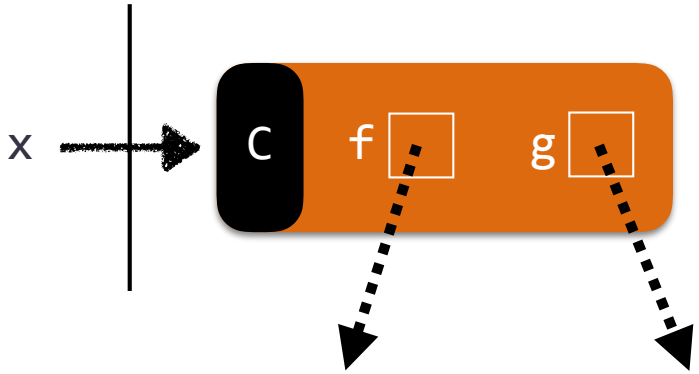
## Version immutable

```
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

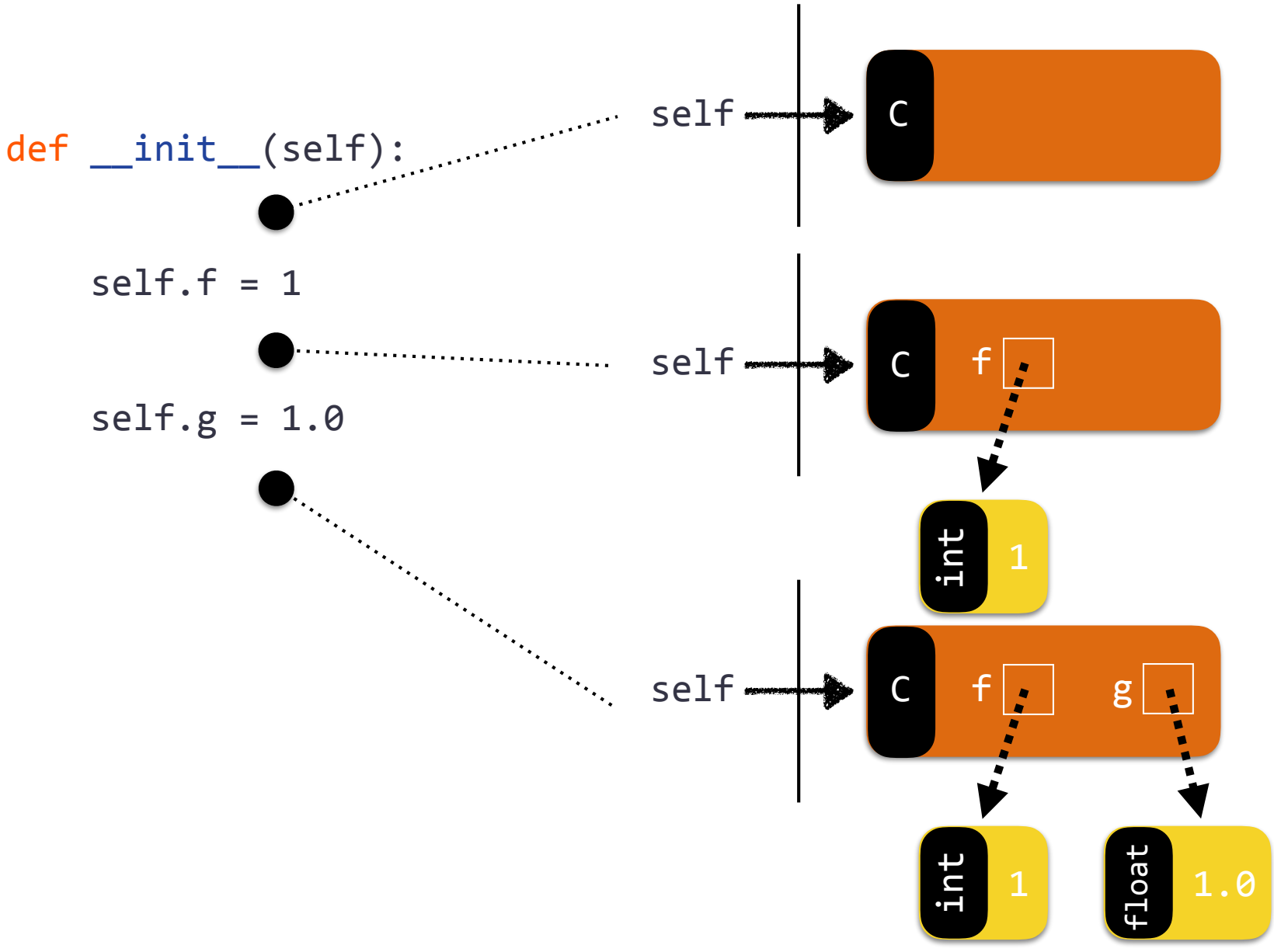
```
from setm import Setm

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



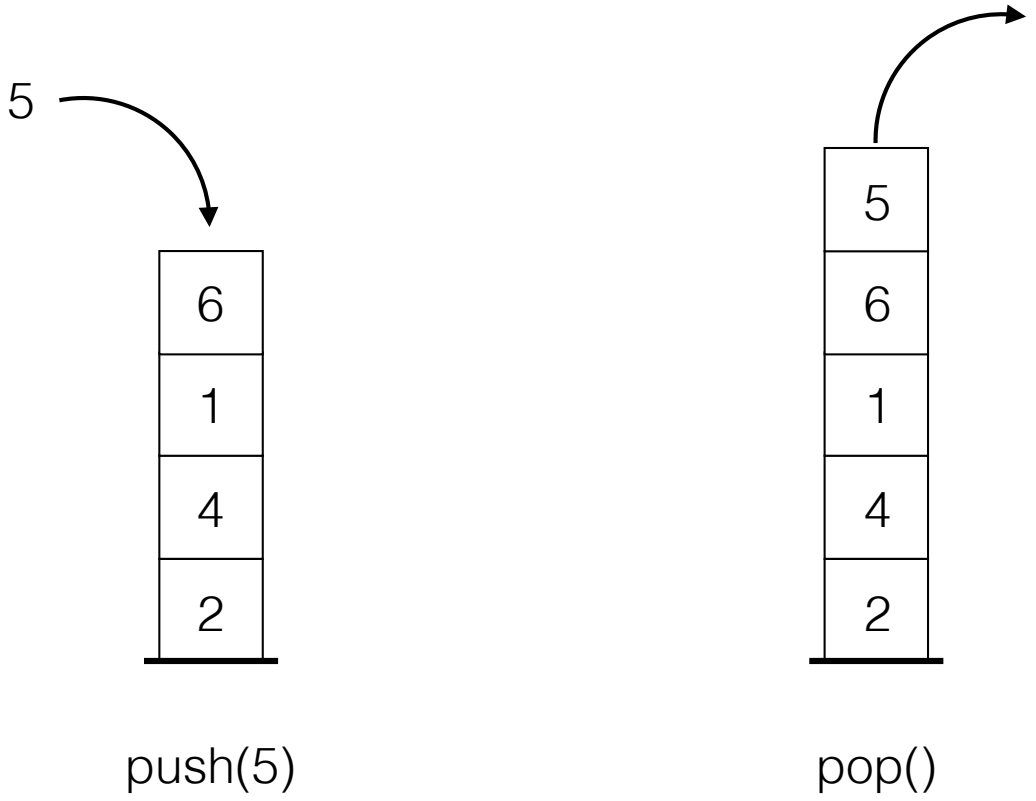
setm.py

```
# -*- 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
```

```
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
```

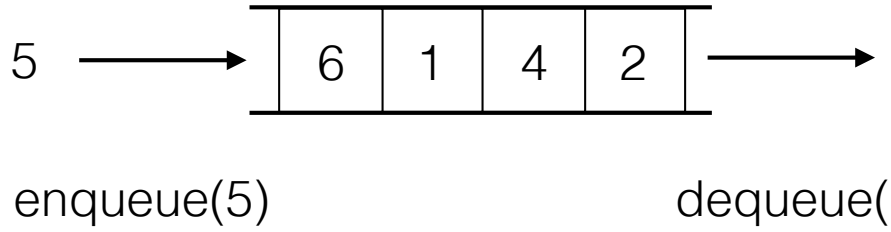
# Structures de données fondamentales

# Pile




# Last In First Out (LIFO)

# File d’attente




First In First Out (FIFO)

# 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

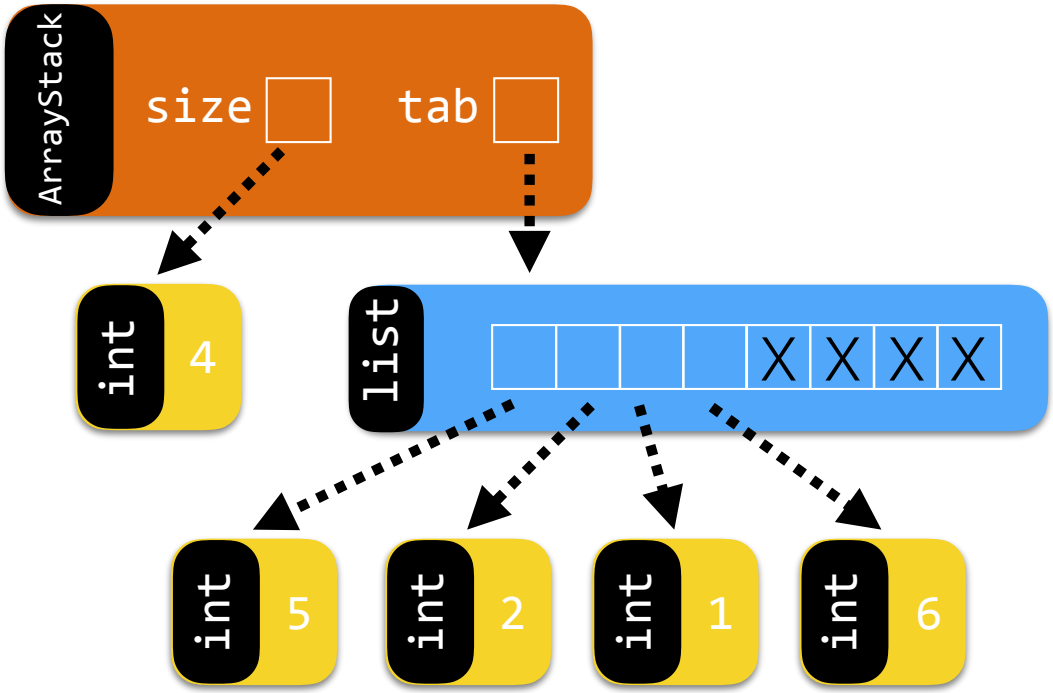
# Flot de contrôle avancé
# les exceptions

- 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 de capacité max size_max."""



    def top(self):
        """Renvoie la tete de la pile self.
           Lance une exception ValueError si la pile est vide."""



    def is_empty(self):
        """Teste si la pile self est vide."""



    def push(self,x):
        """Ajoute l'élément x en tete de la pile self.
           Lance une exception ValueError si la pile est pleine."""



    def pop(self):
        """Supprime la tete de la pile self.
           Lance une exception ValueError si la pile est vide."""
```

```python
class ArrayStack:

    def __init__(self,size_max):
        """Initialise une pile vide de 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)

    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()
```

# Exercice
# Notation polonaise inversée

- Cette notation représente les expressions arithmétiques en plaçant les opérandes avant les opérateurs
- Exemples :
  - (5 + 5) est noté 5 5 +
  - (6 + (4 * 3)) / 2 est noté 6 4 3 * + 2 /
- Plus besoin de parenthèses !!!
- **Exercice** : écrire une fonction python `eval(e)` qui calcule le résultat de l'évaluation de l'expression `e`, en s'aidant d'une pile. L'expression `e` est donnée par une liste d'entiers et de symboles `'+'`, `'-'`, `'*'` et `'/'`.
- Exemple : `eval([6,4,3,'*','+',2,'/'])  ->  9`

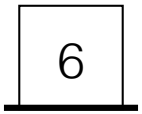
# Notation polonaise inversée


—

(au départ la pile est vide)

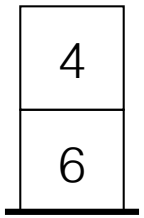
# Notation polonaise inversée

```
[6,4,3,'*','+',2,'/']
```



# Notation polonaise inversée

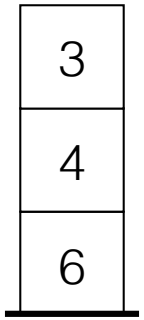
```
[6,4,3,'*','+',2,'/']
```



# Notation polonaise inversée

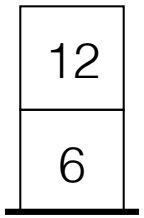
```
[6,4,3,'*','+',2,'/']
```





# Notation polonaise inversée

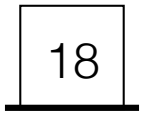
```
[6,4,3,'*','+',2,'/']
```




12=4*3

# Notation polonaise inversée

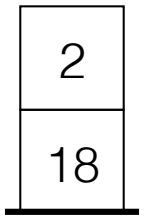
```
[6,4,3,'*','+',2,'/']
```




18=6+12

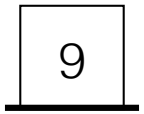
# Notation polonaise inversée

```
[6,4,3,'*','+',2,'/']
```



# Notation polonaise inversée

```
[6,4,3,'*','+',2,'/']
```

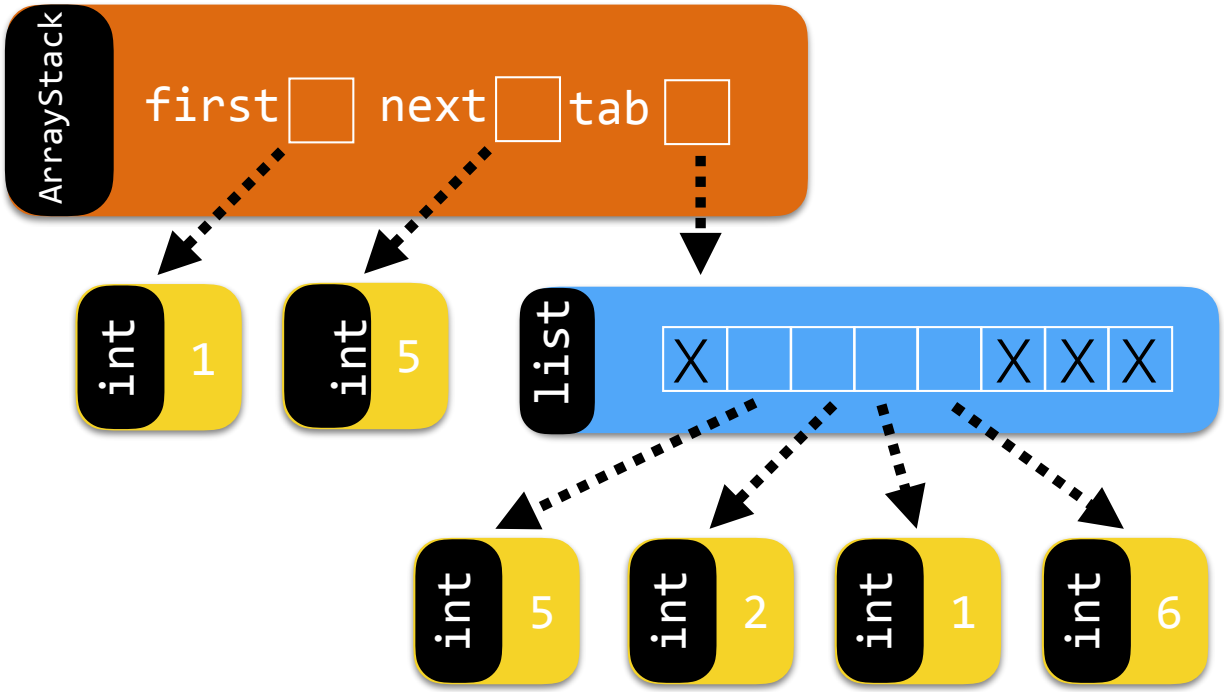



9=18/2

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

from arraystack import ArrayStack

def eval(expr):
    s = ArrayStack(30)
    for i in expr:
        if type(i)==int:
            …
        else:
            …




print eval([6,4,3,'*','+',2,'/'])
```

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

from arraystack import ArrayStack

def eval(expr):
    s = ArrayStack(30)
    for i in expr:
        if type(i)==int:
            s.push(i)
        else:
            b = s.top()
            s.pop()
            a = s.top()
            s.pop()
            if i=='+':
                s.push(a+b)
            elif i=='-':
                s.push(a-b)
            elif i=='*':
                s.push(a*b)
            elif i=='/':
                s.push(a // b)
    return s.top()

print eval([6,4,3,'*','+',2,'/'])
```

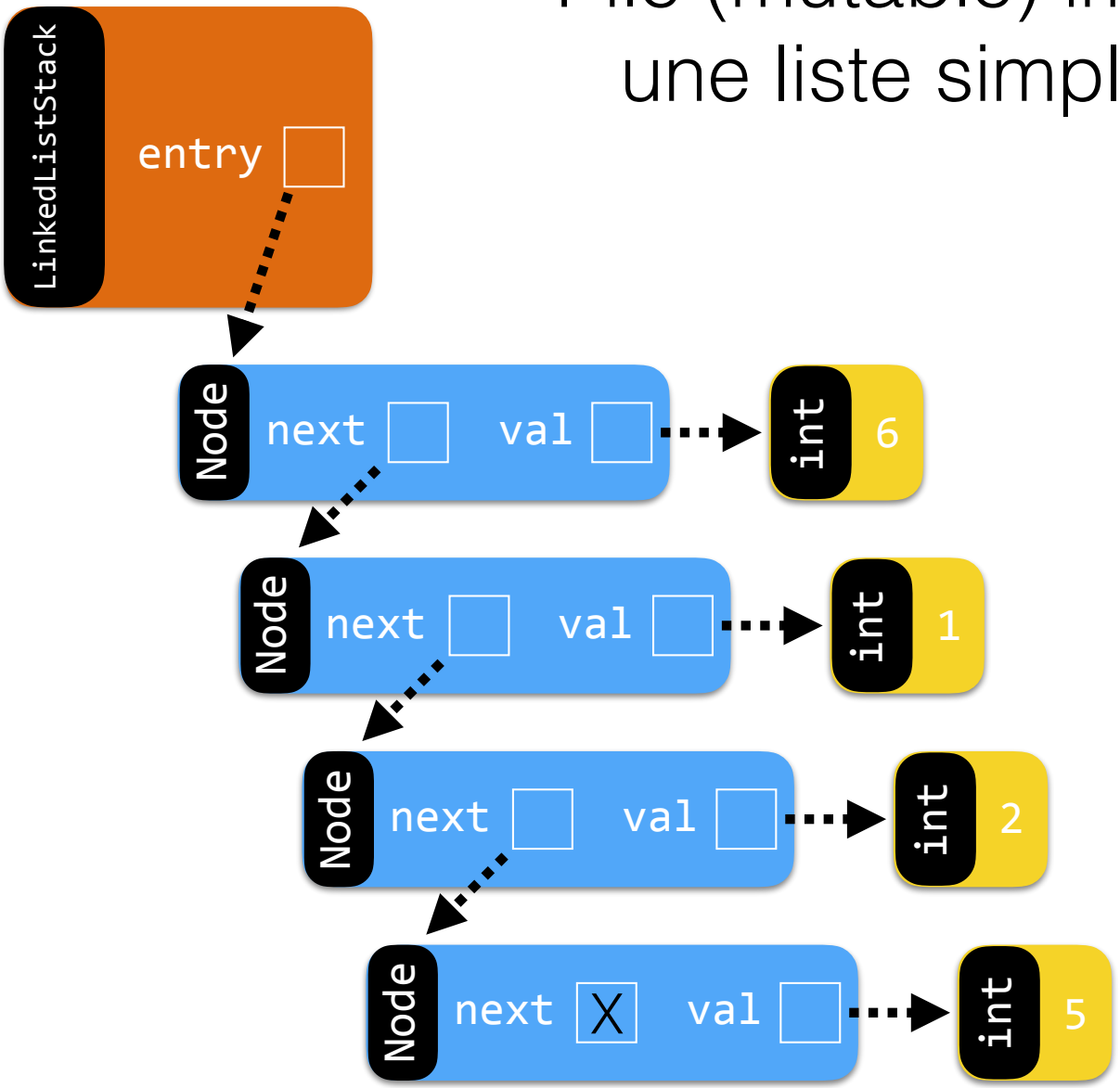
# File (mutable) implémentée avec un tableau




```
s = ArrayQueue(8)
s.enqueue(0)
s.enqueue(5)
s.enqueue(2)
s.dequeue()
s.enqueue(1)
s.enqueue(6)
```

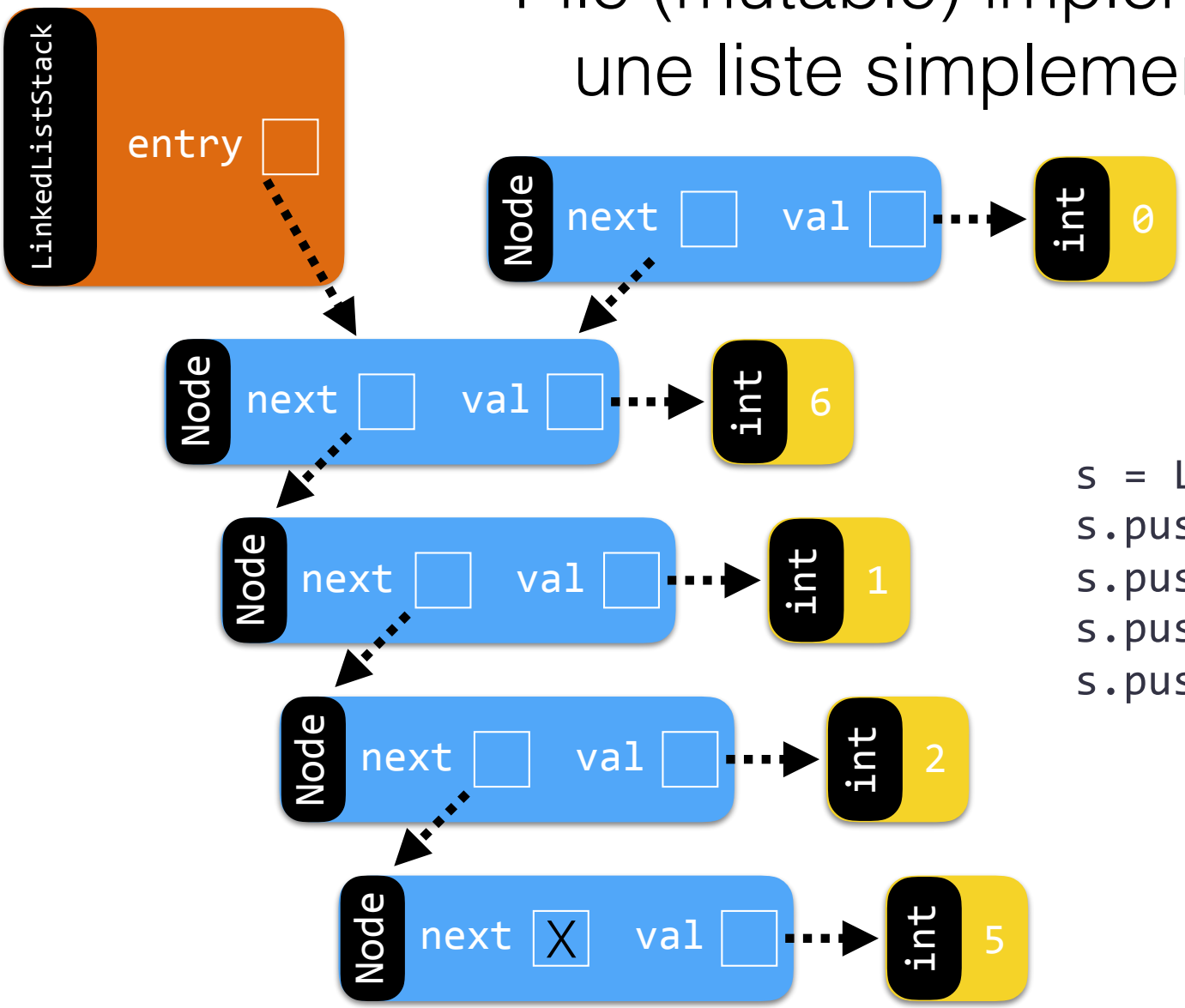
```python
class ArrayQueue:

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

    def head(self):
        """Renvoie la tete de la file self.
           Lance une exception ValueError si la file est vide."""



    def is_empty(self):
        """Teste si la file self est vide."""




    def add(self,x):
        """Ajoute l'élément x en fin de la file self."""



    def remove(self):
        """Supprime la tete de la file self.
           Lance une exception ValueError si la file est vide."""
```

# Pile (mutable) implémentée avec une liste simplement chaînée



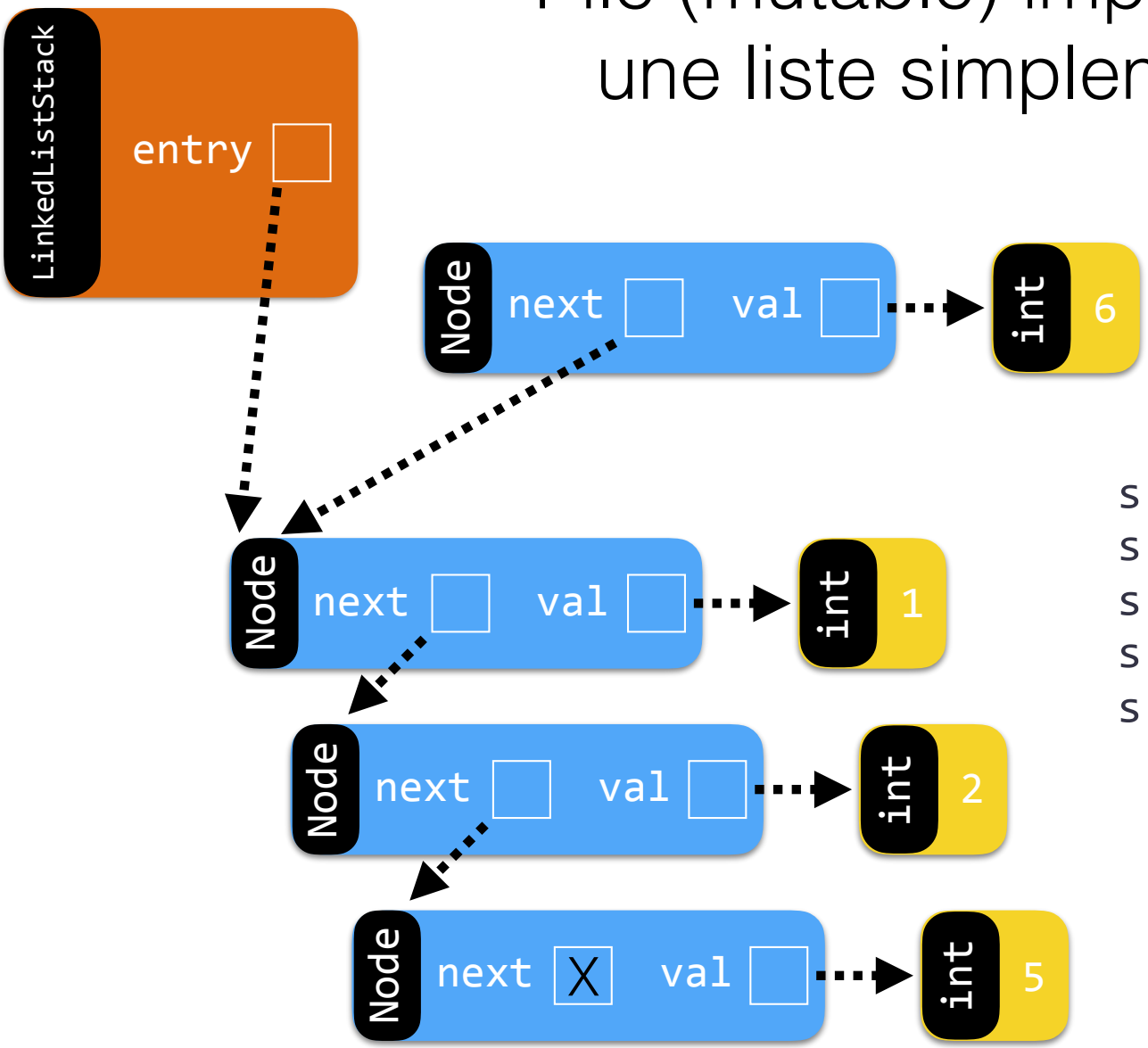

```
s = LinkedListStack()
s.push(5)
s.push(2)
s.push(1)
s.push(6)
```

# Pile (mutable) implémentée avec une liste simplement chaînée




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

# Pile (mutable) implémentée avec une liste simplement chaînée

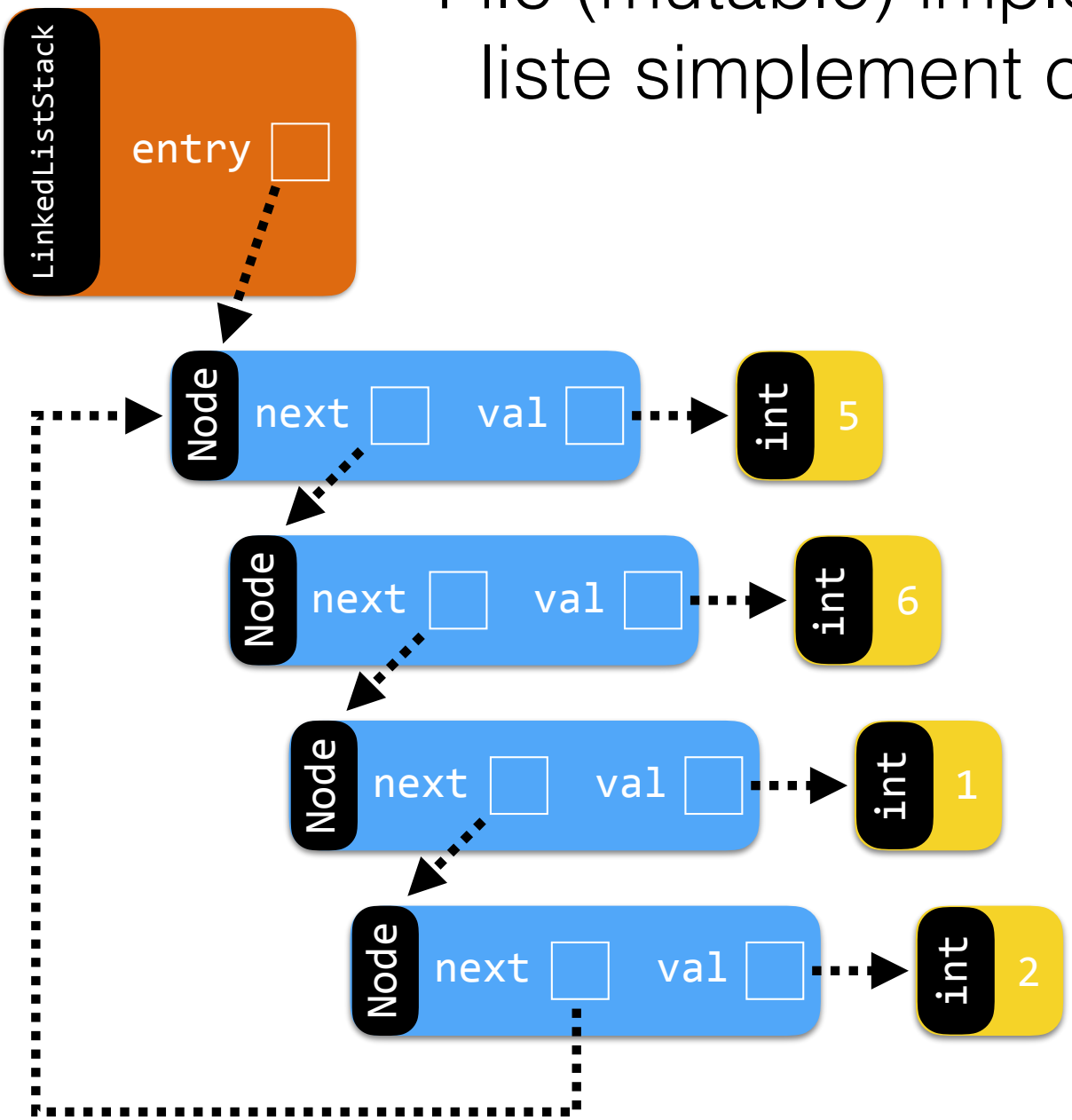



```
s = LinkedListStack()
s.push(5)
s.push(2)
s.push(1)
s.push(6)
```

```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()
```

# File (mutable) implémentée avec une liste simplement chaînée circulaire




```
s = LinkedListQueue()
s.add(5)
s.add(2)
s.add(1)
s.add(6)
```

```
class LinkedListQueue:

    def __init__(self):
        """Initialise une file vide"""

    def head(self):
        """Renvoie la tete de la file self.
           Lance une exception ValueError si la file est vide."""

    def is_empty(self):
        """Teste si la file self est vide."""

    def add(self,x):
        """Ajoute l'élément x en fin de la file self."""

    def remove(self):
        """Supprime la tete de la file self.
           Lance une exception ValueError si la file est vide."""
```

```python
class Node:

    def __init__(self,val):
        self.val = val
        self.next = None # inutile

class LinkedListQueue:

    def __init__(self):
        """Initialise une file vide"""
        self.entry = None # inutile

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

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


    def add(self,x):
        """Ajoute l'élément x en fin de la file self."""
        if self.entry==None:
            n = Node(x)
            n.next = n
            self.entry = n
        else:
            n = Node(x)
            n.next = self.entry.next
            self.entry.next = n
            self.entry = n

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