INFO2

Magistère de Mathématiques de Rennes
2ème année

22 février 2016

école normale supérieure rennes

UNIVERSITÉ DE RENNES 1

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"""
```

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
import setm

s = setm.empty()
print setm.string(s)

setm.add(s,1)
print setm.string(s)

setm.add(s,2)
print setm.string(s)

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

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

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)`

Utilisation

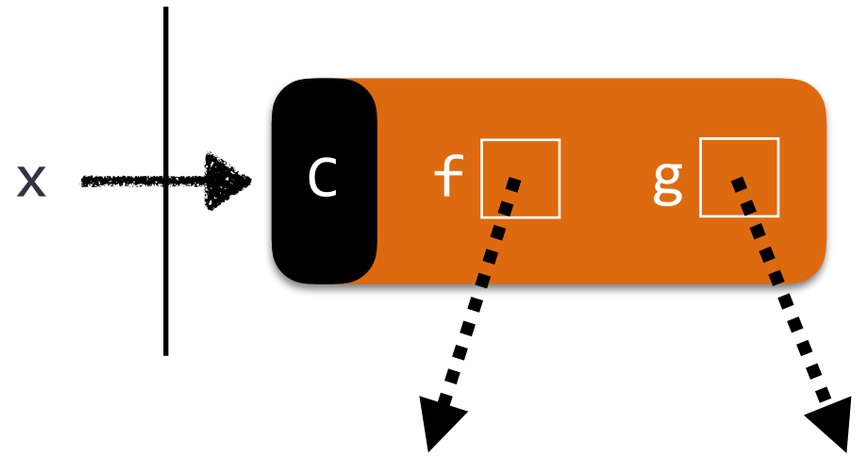
Version immutable

```
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
```

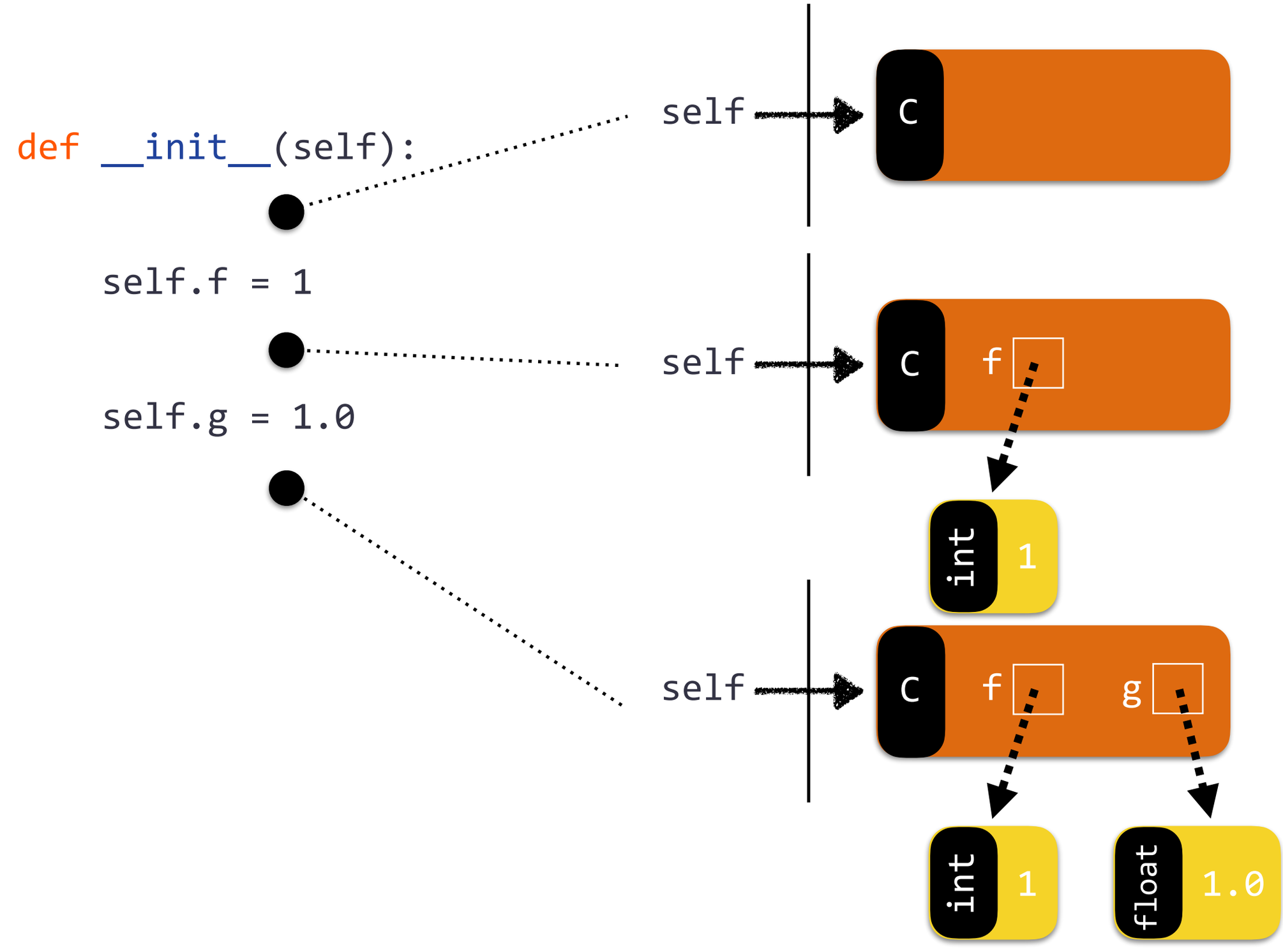
Un objet




```
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

```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()
```

```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()
```

```
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."""
```

Exercices

- Pile non-mutable avec une liste simplement chaînée
- File non-mutable avec une liste simplement chaînée cyclique
- Pile (liste) mutable avec une liste doublement chaînée et
  - une opération de concaténation
  - une opération pour donner la taille
  - une opération de recherche d'un élément