INFO2

Magistère de Mathématiques de Rennes
2ème année

9 février 2016

école normale supérieure rennes

UNIVERSITÉ DE RENNES 1

Dans l’épisode précédent…

- Modes d’exécution
- Types de bases
- Flot de contrôle
  - boucles
  - fonctions

Comprendre le *modèle mémoire* de Python

- 4 programmes simples
- Notion d'indirection
- Modèle uniforme
  - allocation
  - copie superficielle/profonde
  - desallocation (fuites mémoires, ramasses miettes)

```
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
```

Comprendre le *modèle mémoire* de Python

- Valeurs immutables
  - valeurs numériques
  - chaînes
- Valeurs mutables

Programmation modulaire

- Clients / Interfaces / Implémentation

Étude de cas

- On souhaite représenter un ensemble de valeurs par un tableaux de valeurs, sans répétition

- Ecrire les fonctions suivantes

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

def add(s,x)
    """renvoie l'ensemble s, auquel x a été ajouté"""

def remove(s,x)
    """renvoie l'ensemble s, auquel x a été enlevé"""

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

seti.py

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

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

def add(s,x):
    """renvoie l'ensemble s, auquel x a été ajouté"""
    if not x in s:
        return s+[x]
    else:
        return s

def remove(s,x):
    """renvoie l'ensemble s, auquel x a été enlevé"""
    if x in s:
        s2 = []
        for y in s:
            if x != y:
                s2 = s2+[y]
        return s2
    else:
        return s

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
```

```
import seti

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

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

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

s = seti.remove(s,1)
print seti.string(s)
```

Étude de cas

- On souhaite réduire la consommation mémoire et opter pour une approche mutable

- 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)
```

Autre quizz

```
def inc(x):
    x = x+1

i = 99
inc(i)
print "i =", i
```

```
def inc(x):
    x[0] = x[0]+1

i = [99,99,99]
inc(i)
print "i =", i
```