Programmation et IA

Cours 2

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-ia-25

Plan

- installation VScode
- listes
- itération sur les listes
- n-uplets
- ensembles
- dictionnaires
- notation compréhensive
- double itération sur les listes

dès maintenant: télécharger Python 3 en `http://www.python.org`

VScode

- visual studio code est un système intégré pour l'édition de programmes
- télécharger VScode en `https://code.visualstudio.com`
- ouvrir VScode et installer le mode Python
- créer des fichiers avec noms avec le suffixe `.py` (pour programmes Python)

Impressions formattées

- impression

```
def concat_print1 (s, n, x):
    print (f"{s} vaut {n} ou {x:.2f}")

concat_print1 ("Le resultat", 32, 28.5)
Le resultat vaut 32 ou 28.50


def concat_print2 (s, n, x):
    print ("{} vaut {} ou {:.2f}".format (s, n, x))

concat_print2 ("Le resultat", 32, 28.5)
Le resultat vaut 32 ou 28.50
```

```
for x in range(1, 11):
   print(f'{x:2d} {x*x:3d} {x*x*x:4d}')

 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

- voir en `www.programiz.com/python-programming/input-output-import`

les structures de données (1)

Listes

- les listes de Python sont des tableaux dynamiques modifiables

```
>>> x = [1, 3, 4, 2, 3, 5]
>>> type (x)
<class 'list'>
>>> x[0]
1
>>> x[1]
3
>>> x[3]
2
```

```
>>> x[3] = 99
>>> print (x)
[1, 3, 4, 99, 3, 5]
```

modification du 4ème élement

```
>>> len (x)
6
```

longueur de la liste

```
>>> x.append (9)
>>> x
[1, 3, 4, 2, 3, 5, 9]
```

- les listes de Python ne sont pas forcément homogènes

```
>>> y = [1, 3, 4, "kludge", 2, 3]
>>> print (y)
[1, 3, 4, 'kludge', 2, 3]
```

différent des listes de Lisp, Ocaml, Haskell, Java, …

Listes

• itération sur une liste (tableau)

```
>>> s = 0
>>> for m in x :
...   s = s + m
...
>>> s
27
```

itération sur tous les éléments de x

```
>>> x
[1, 3, 4, 2, 3, 5, 9]
>>> max = -1
>>> for m in x :
...   if m > max :
...     max = m
...
>>> max
9
```

itération sur tous les éléments de x

Exercice: valeur de `max si x vaut [ ]` ?

• idem avec une fonction

```
>>> def sum_of (x) :
...   s = 0
...   for m in x :
...       s = s + m
...   return s
...
>>> sum_of (x)
27
>>> a = [-3, 42, 23, 11, -30]
>>> sum_of (a)
43
```

Intervalles

- intervalles (*range*)

```
>>> for i in range (0, 10):
...   print (i)
...
0
1
2
3
4
5
6
7
8
9
```

intervalle semi-ouvert

```
>>> for i in range (10):
...   print (i)
...
0
1
2
3
4
5
6
7
8
9
```

- pas entre valeurs dans les intervalles

```
>>> for i in range (0, 10, 2) :
...   print (i)
...
0
2
4
6
8
```

2 est le pas

Exercice: quel est le sens de `range (10, 0, -1)` ?

Tranches

- tranche (*slice*)

```
>>> x
[1, 3, 4, 2, 3, 5, 9]
>>> x[3:6]
[2, 3, 5]
>>> x[3:6:2]
[2, 5]
>>> x[3:]
[2, 3, 5, 9]
>>> x[:6]
[1, 3, 4, 2, 3, 5]
>>> x[::2]
[1, 4, 3, 9]
```

intervalle semi-ouvert

listes délimitées par des [..]

n-uplets délimitées par des (..)

- une chaîne de caractère est une liste de caractères **non modifiables**

```
>>> s = "abcdefghijklmnopq"
>>> s[3:10]
‘defghij'
```

- un n-uplet (*tuple)* est une liste **non modifiable**

```
>>> b = (9, "novembre", 1989)
>>> b[0]
9
>>> b[1]
'novembre'
>>> b[2]
1989
```

Listes

- exemple de fonction sur une liste

```
>>> def max_of (x) :
      r = x[0]
      for m in x[1:] :
        if m > r :
          r = m
      return r

>>> max_of (x)
9
>>> max_of (['jj', 'andre', 'paul'])
'paul'
```

si x est une liste non vide

```
>>> def max_of (x) :
      r = x[0]
      for i in range (1, len(x)) :
        if x[i] > r :
          r = x[i]
      return r

>>> max_of (x)
9
>>> max_of (['jj', 'andre', 'paul'])
'paul'
```

Exercice: valeur de `max_of si x vaut [ ]` ?

Exercice: écrire la fonction `min_of (x)`

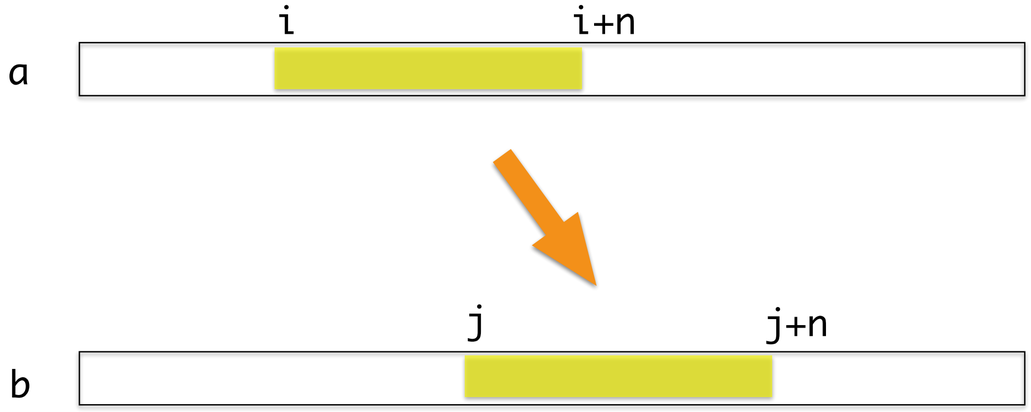
Valeur d'un tableau — Alias

Exercice: le programme suivant est-il correct ?

```
def copy (a, b, i, j, n) :
    for k in range (n) :
        b [j + k] = a [i + k]
```

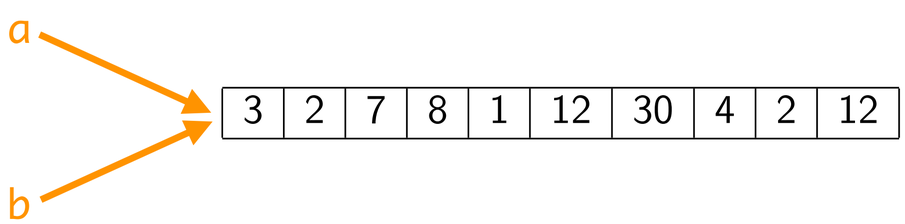
on suppose :

$$0 \leq i$$
$$0 \leq j$$
$$0 \leq n$$
$$i + n < \text{len(a)}$$
$$j + n < \text{len(b)}$$



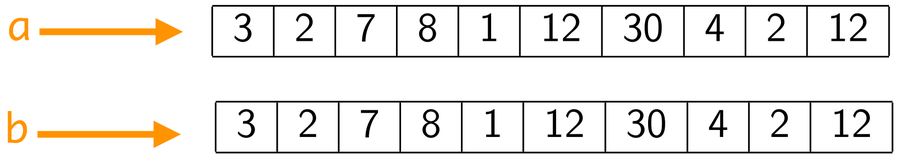
Valeur d'un tableau — Alias

• soient 2 tableaux a et b




```
a = [3, 2, 7, 8, 1, 12, 30, 4, 2, 12]
b = a

a[2] = 888

print (a)
[3, 2, 888, 8, 1, 12, 30, 4. 2. 12]

print (b)
[3, 2, 888, 8, 1, 12, 30, 4. 2. 12]
```

• les variables a et b sont 2 alias d'un même tableau




```
a = [3, 2, 7, 8, 1, 12, 30, 4. 2. 12]
b = [3, 2, 7, 8, 1, 12, 30, 4. 2. 12]

a[2] = 888

print (a)
[3, 2, 888, 8, 1, 12, 30, 4. 2. 12]

print (b)
[3, 2, 7, 8, 1, 12, 30, 4. 2. 12]
```

• les variables a et b sont 2 tableaux distincts

la valeur de a ou de b est l'adresse mémoire de son premier élément

données modifiables et alias sont sources de bugs

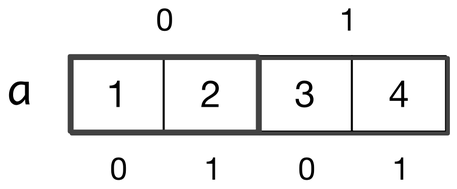
tableaux multi-dimmensionnels

Tableaux multi-dimensionnels

- une matrice est une liste de listes

```
>>> a = [[1,2], [3,4]]

>>> a[0][0]
1
>>> a[0][1]
2
>>> a[1][0]
3
>>> a[1][1]
4
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$



- une itération sur les matrices

```
def print_matrix (a) :
    for line in a :
        for elt in line :
            print (f'{elt:2d} ', end = ' ')
        print ()
```

impression sur une ligne

```
print_matrix (a)
 1   2
 3   4
```

Tri

- on cherche le minimum et on le met en tête..
  et on recommence à partir du deuxième élément, etc…

```
def triSelection (a) :
    n = len (a)
    for i in range (n-1) :
        jmin = index_min_of (a, i)
        xchange (a, i, jmin)
```

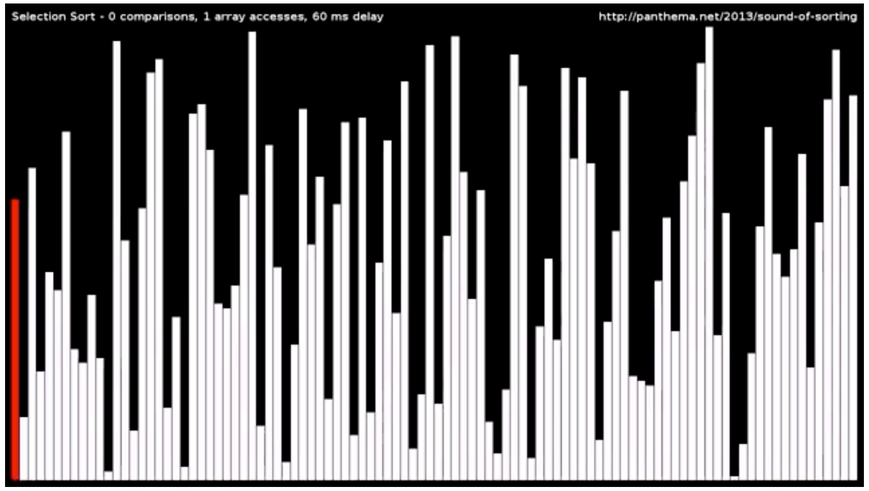
échanger les valeurs de `a[i]` et `a[jmin]`

index du minimum dans `a` à partir de `i`

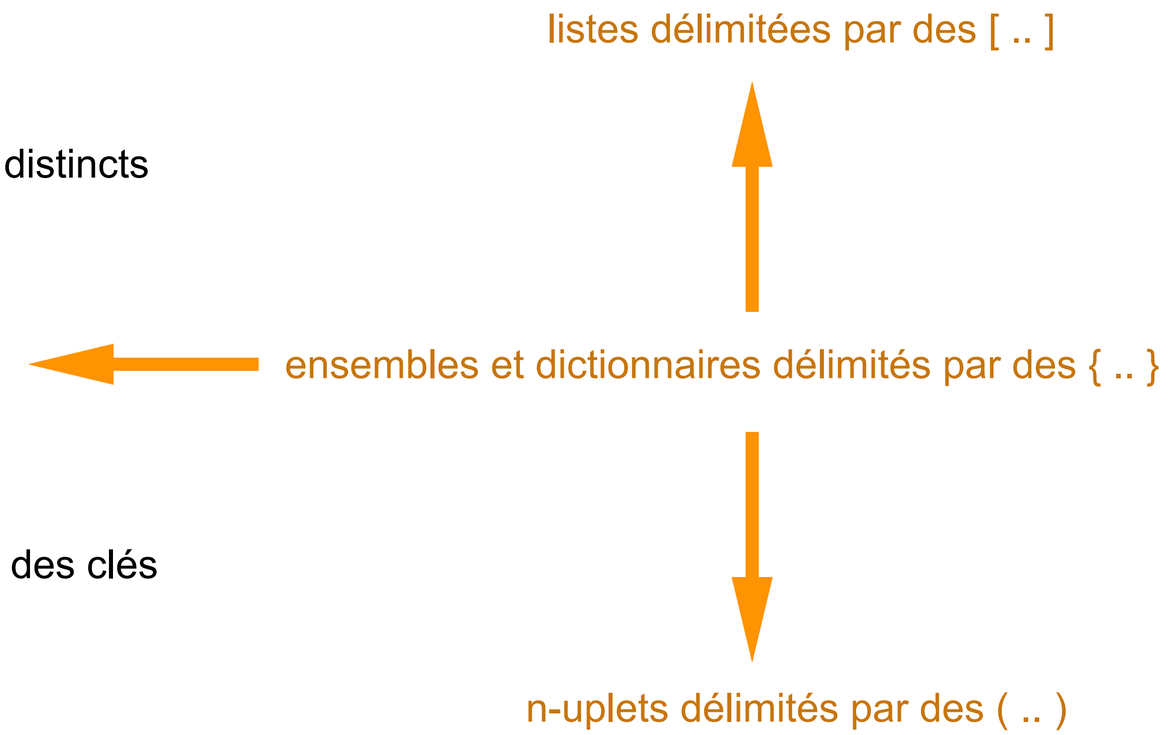
```
def index_min_of (a, i) :
    jmin = i
    for j in range (i+1, len(a)):
        if a[j] < a[jmin] :
            jmin = j
    return jmin


def xchange (a, i, j) :
    temp = a[i]
    a[i] = a[j]
    a[j] = temp
```

http://visualgo.net/en/sorting

```
def triSelection (a) :
    n = len (a)
    for i in range (n-1) :
        jmin = i
        for j in range (i+1, n):
            if a[j] < a[jmin] :
                jmin = j
        t = a[i]; a[i] = a[jmin]; a[jmin] = t
```



Ensembles — Dictionnaires



- les **ensembles** sont des listes non ordonnées d'éléments tous distincts

```
print ({10, 2, 3} == {3, 2, 10})
True

print ({10, 2, 3} == {5, 2, 10})
False
```

- les **dictionnaires** sont des listes non ordonnées indexées par des clés

```
age = {"alice": 22, "bob": 27, "charlie": 30}

table1 = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

table2 = {1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343}

print(age["bob"])

print(table1[5])

print(table2[6])
```

Compréhension

- on peut **générer** des tableaux, listes, ensembles, dictionnaires avec la notation compréhensive

```
a = [x**2 for x in range(21) if x*2 < 21]
print (a)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
b = {2 * x for x in range (20)}
print (b)
{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38}
```

```
c = {x : x**2 for x in range (10)}
print (c)
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

Modules

- les fonctions ou données (de librairie..) sont regroupées en modules
- par exemple, on charge le module `random` avec

```
import random
```

- notation qualifiée avec nom de module `random.sample`

```
def rand_array (n, p) :
    return random.sample (range(p), n)
```

- on peut raccourcir la notation qualifiée `rd.sample` le nom du module avec

```
import random as rd
```

- on peut utiliser la notation simple `sample` sans le nom du module avec

```
from random import *
```

- la liste des modules disponibles s'obtient avec

```
help()
modules
. . .
quit
```

Modules

Exercice: faire sa propre bibliothèque de modules, par exemple en y incorporant `sum_of ([ ])` et `max_of ([ ])` ?

- on crée un fichier `myLib.py`

```
import myLib
```

- ou

```
import myLib as my
```

- ou

```
from myLib import *
```

Prochain cours

- récursivité
- réviser les classes et objets
- classes et objets
- structures de données
- structures arborescentes
- parcours d’arbres