Informatique et Programmation

Cours 2

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-py

Plan

- exercices du cours 1
- tableaux et listes
- itération sur les listes
- tableaux multi-dimensionnels, matrices
- tri d’un tableau
- exercices

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

Exercices du cours 1

- PGCD par l'algorithme d'Euclide

```
def pgcd (m, n) :
    while m != n :
        if m > n :
            m = m - n
        else:
            n = n - m
    return m
```

suppose m > 0 et n > 0

- suite de Syracuse

```
def syracuse (n) :
   while n != 1 :
     print (n, end = ' ')
     if n % 2 == 0 :
       n = n // 2
     else :
       n = 3 * n + 1
   print (n)
```

impression sur la même ligne

```
>>> syracuse(19)
19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

Exercices du cours 1

- vérification de la date de Pâques

```
def verif_paques (y, ymax) :
    while y <= ymax :
        paques (y)
        y = y + 1
```

```
>>> verif_paques(2020, 2047)
12 avril 2020
4 avril 2021
17 avril 2022
9 avril 2023
31 mars 2024
20 avril 2025
5 avril 2026
28 mars 2027
16 avril 2028
1 avril 2029
21 avril 2030
13 avril 2031
28 mars 2032
17 avril 2033
9 avril 2034
25 mars 2035
13 avril 2036
5 avril 2037
25 avril 2038
10 avril 2039
1 avril 2040
21 avril 2041
6 avril 2042
29 mars 2043
17 avril 2044
9 avril 2045
25 mars 2046
14 avril 2047
```

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

• idem avec une fonction

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

x est l'argument de la fonction

x est ici la variable globale de l'environnement

Listes

- itération sur une liste (tableau)

```
>>> 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

- idem avec une fonction

```
>>> import sys
>>> MIN_INT = -sys.maxsize
>>>
>>> def max_of (x) :
...   r = MIN_INT
...   for m in x :
...     if m > r :
...       r = m
...   return r
...
>>> max_of (x)
9
```

entier minimum sur 64 bits

Exercice: valeur de `max_of ([ ])` ?

Exercice: écrire la fonction `min_of ([ ])`

Listes

- intervalles (*range*)

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

intervalle semi-ouvert

- abréviation et pas dans les intervalles

```
>>> range (10)
range(0, 10)
>>> 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)` ?

Listes

- 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

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

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

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

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

Listes

- itération sur les indices d'un tableau

```
>>> def is_palindrome (s) :
...     n = len(s)
...     for i in range(n) :
...         if s[i] != s[n-1-i] :
...             return False
...     return True
...
>>> is_palindrome("kayak")
True
>>> is_palindrome("kayok")
False
```

on peut optimiser avec `range(n//2)`

- une autre itération sur les indices d'un tableau

```
def index_max_of (x) :
    n = len (x)
    m = MIN_INT; imax = -1
    for i in range (n):
        if x[i] > m :
            m = x[i]; imax = i
    return imax
```

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
```

- addition et multiplication de matrices

```
def add (a, b) :
    m = len (a); n = len (a[0])
    if m != len(b) or n != len(b[0]) :
        print("Erreur !"); return
    r = new_matrix (m, n)
    for i in range(m) :
        for j in range(n):
            r[i][j] = a[i][j] + b[i][j]
    return r
```

- une itération sur les matrices

```
def print_matrix (a) :
    for line in a :
        for elt in line :
            print (elt, end = ' ')
        print ()
```

impression sur une ligne

```
def mul (a, b) :
    m = len (a); n = len (a[0]); p = len (b[0])
    if n != len(b) :
        print("Erreur !"); return
    r = new_matrix (m, p)
    for i in range(m) :
        for j in range(p):
            r[i][j] = 0
            for k in range(n):
                r[i][j] = r[i][j] + a[i][k] * b[k][j]
    return r
```

Tableaux multi-dimensionnels

- création d'une matrice pleine de zéros

```
def new_matrix (m, n) :
    a = []; z = [0]*n
    for i in range(m): a.append (z.copy())
    return a
```

à comprendre plus tard !

- carré magique

```
def magique (n) :
    a = new_matrix (n, n)
    i = n - 1
    j = n // 2
    for k in range (n*n) :
        a[i][j] = k+1
        if (k+1) % n == 0 :
            i = (i - 1) % n
        else :
            i = (i + 1) % n
            j = (j + 1) % n
    return a
```

n impair

```
>>> print_matrix(magique(3))

4 9 2
3 5 7
8 1 6
```

somme 15 sur lignes et colonnes

somme 15 sur les 2 diagonales

```
>>> print_matrix2(magique(7))

 22  31  40  49   2  11  20
 21  23  32  41  43   3  12
 13  15  24  33  42  44   4
  5  14  16  25  34  36  45
 46   6   8  17  26  35  37
 38  47   7   9  18  27  29
 30  39  48   1  10  19  28
```

Recap

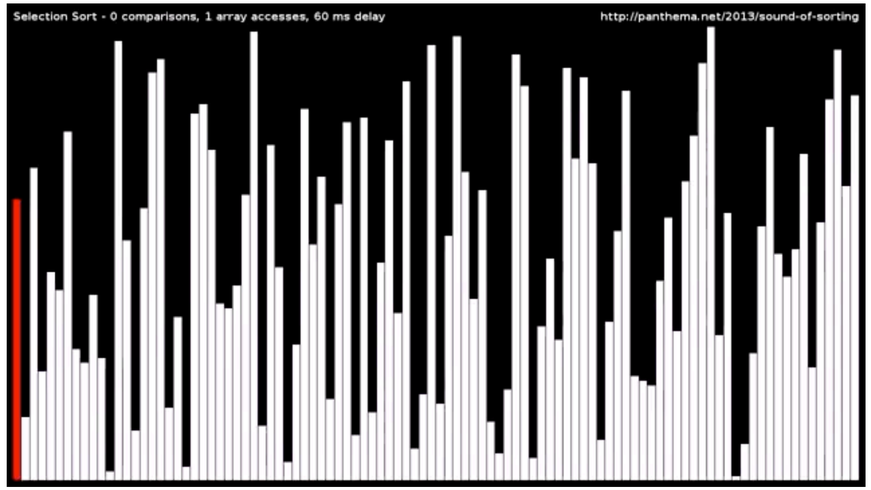
- mots clés en Python (déjà vus en rouge)

```
>>> help()
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

Tri sélection



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

```
def tri_selection (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
```

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

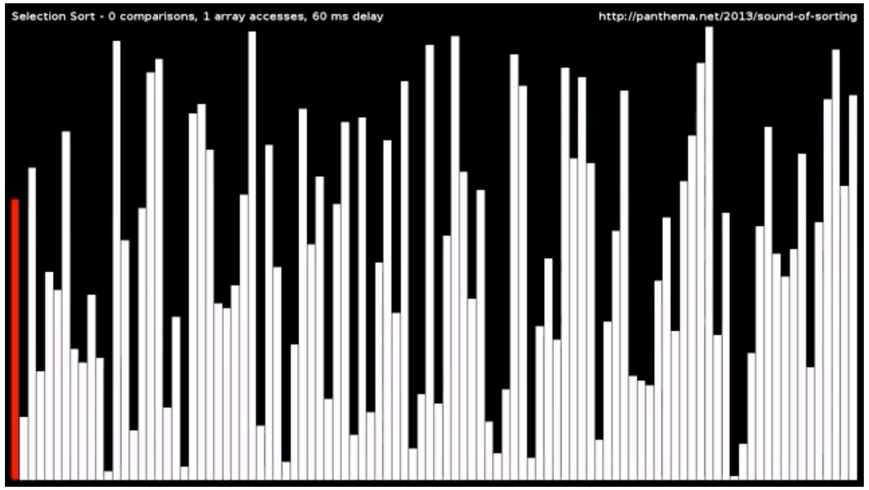
```
>>> x
[1, 3, 4, 2, 3, 5, 9]
>>> tri_selection(x)
>>> x
[1, 2, 3, 3, 4, 5, 9]
```

```
>>> import random
>>> x = random.sample(range(100), 20)
>>> x
[59, 30, 23, 69, 67, 42, 20, 53, 72, 82, 0, 49, 31, 95, 90, 19, 46, 34, 66, 26]

>>> tri_selection (x)
>>> x
[0, 19, 20, 23, 26, 30, 31, 34, 42, 46, 49, 53, 59, 66, 67, 69, 72, 82, 90, 95]
```

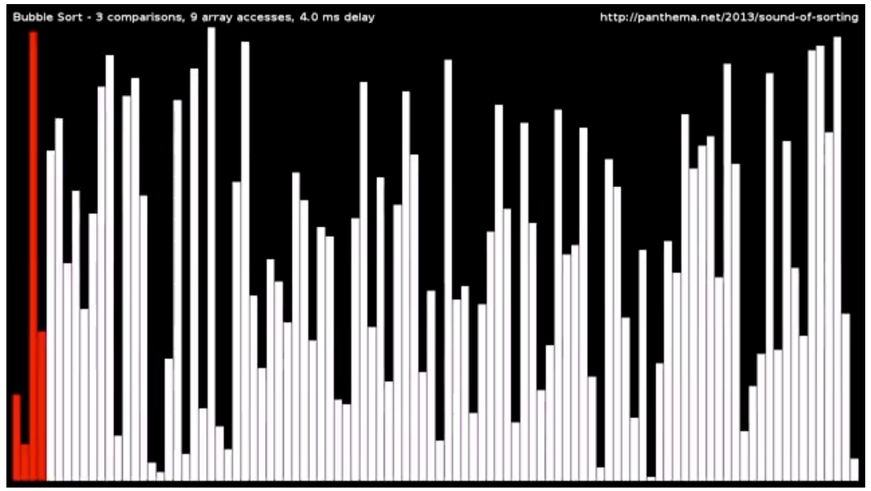
générer un tableau de 20 nombres aléatoires entre 0 et 99

Tri bulle



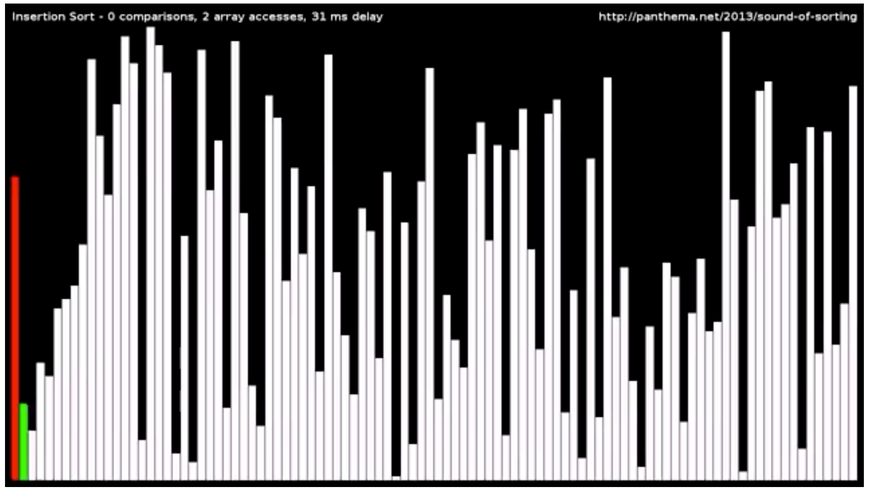
- un bulle pousse le maximum vers la fin.. et on recommence jusqu'à l'avant-dernier élément, etc…

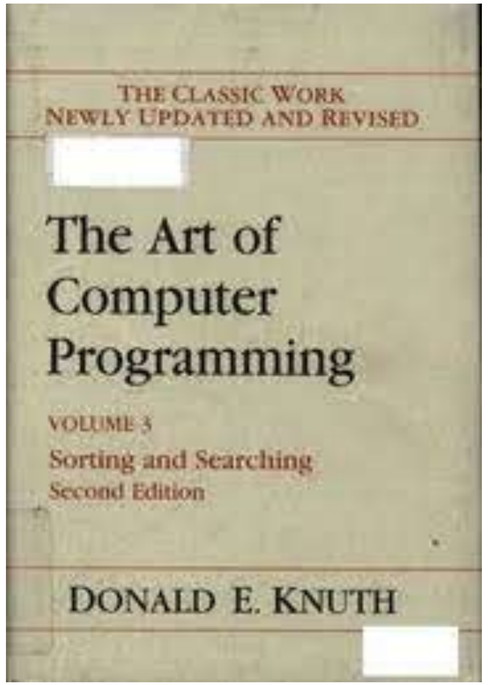
```
def tri_bulle (a) :
    n = len (a)
    for j in range (n-1, 0, -1) :
      for i in range (0, j):
        if a[i] > a [i+1] :
          t = a[i]; a[i] = a[i+1]; a[i+1] = t
```

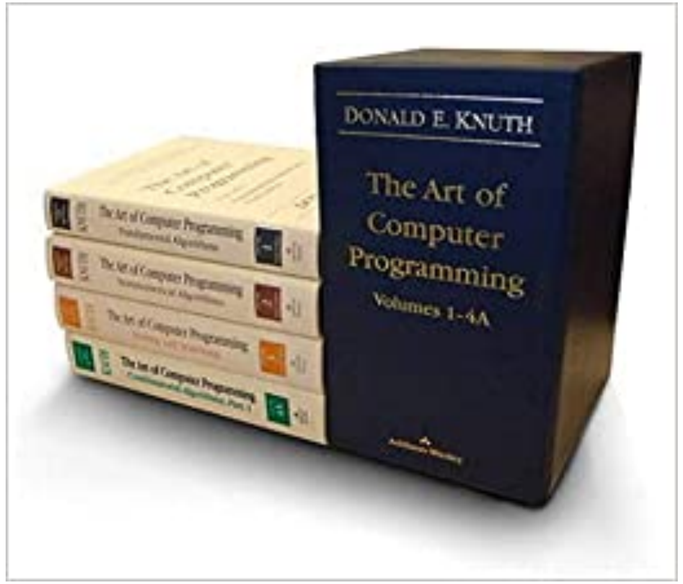


Exercices

- écrire le tri par insertions
- quel est le meilleur de ces 3 tris ?







- prochain cours: la récursivité