

22

Algorithme

15

Programmation

Algorithme & Programmation

# Structures Séquentielles

## Liste

$l = [ \text{'lundi'}, 13, \text{'janv'}, \text{lolo} ]$

$l = []$

- Fonction
- $\text{list}(\text{'timo'}) \rightarrow [ \text{'t'}, \text{'i'}, \text{'m'}, \text{'o'} ]$
  - $\text{list}(\text{range}(5)) \rightarrow [0, 1, 2, 3, 4]$
  - $[0] * 4 \rightarrow [0, 0, 0, 0]$
  - $l_1 + l_2 \rightarrow [ \text{'lundi'}, 13, \text{'janv'}, \text{lolo}, 1 ]$

## Modif dt d'une liste :

$l[i] = <\text{expr}>$

@  $l_1 = l_1 + 1$   
(listes st mutables Mais pas chaine, intervalle, tuple)

## Méthodes (index, count, copy, append, pop, reverse, sort)

- $l_1.\text{index}(\text{'lundi'}) \rightarrow 0$
- $l_1.\text{count}(\text{'o'}) \rightarrow 2$
- $l_2 = l_1.\text{copy}()$
- $l_2.\text{append}(1) \rightarrow \text{None}$
- $l_2.\text{pop}() \rightarrow 1$
- $l_2.\text{reverse} \rightarrow \text{None}$
- $l_2.\text{sort} \rightarrow \text{None}$

QHV o égalité logique  $\neq$  égalité physique

$\rightarrow l_2 = l_1 . \text{copy}()$  &  $l_3 = l_1$   
 $\rightarrow l_1 \text{ is } l_2 \rightarrow \text{False}$     $\rightarrow l_1 \text{ is } l_3 \rightarrow \text{True}$

## Effet de bord sur paramètres de f

```
def compte(elt, l):  
    : param elt: (any)  
    : param l: (list)  
    : return: (int)  
  
    res = 0  
    while l != []:  
        x = l.pop()  
        if elt == x:  
            res += 1  
    return res
```

⚠ pop(): sur listes non-vide

(la liste est mutable et elle a été vidée à fin du code)

```
def compte_V2(elt, l):
```

```
    l2 = l.copy()  
    res = 0  
    while l2 != []:  
        x = l2.pop()  
        if elt == x:  
            res += 1
```

return res

② CPC.CX/9x9

## Constructions en Compréhension

$t1 = \text{tuple}(\text{k} * \text{k} \text{ for } \text{k} \text{ in range}(10))$   
 $\rightarrow (0, 1, 4, 9, 16, 25, 36, 49, 64, 81)$   
CEC    $\langle \text{expr} \rangle \text{ for } \langle \text{var} \rangle \text{ in } \langle \text{iterable} \rangle$   
 $\langle \text{expr} \rangle \text{ for } \langle \text{var} \rangle \text{ in } \langle \text{iterable} \rangle \text{ if } \langle \text{expr bool} \rangle$   
 $\langle \text{expr} \rangle \text{ for } \langle \text{var} \rangle \text{ in } \langle \text{iterable1} \rangle \text{ for } \langle \text{var} \rangle \text{ in } \langle \text{iterable2} \rangle$

## Ch 2: Ensembles & dictionnaires structures itérables non séquentielles

### Ensembles:

Ens: s'ce données non-séquentielles d'elts distincts  
 $\rightarrow \emptyset$  indice, ordre élts pas de sens,  $\times$  f compte

### Constructions linéaires

$s1 = \{1, 2, 3\} \rightarrow \{1, 2, 3\} \& \gg \{3, 1, 2\}$   
type(s1)  $\rightarrow$  (set)  
set('timoleum')  $\rightarrow \{'e', 'i', 'l', 'm', 'n', 'o', 't'\}$

⚠ Appari & 1 seule fois le the o.

⚠ set()  $\neq \{\}$   
↓      ↓  
set      dict

③

⚠ V des SAUF valeurs mutables

• Itéra<sup>o</sup>: for fruits in tarifs:

print (fruit, tarifs [fruits])  
»» banane 3.5  
orange 1.95  
pomme 1.95

Chgt valeur ↔ à clé

tarifs ['banane'] = tarifs ['banane'] \* (1-0,1)  
tarifs ['banane'] → 3.15

Ajout élé

Supr élé

tarifs ['fraise'] = 10.95

• pop

def creer\_date (j, m, a):

return { 'jour': j, 'mois': m, 'annee': a }

»» creer\_date (20, 1, 2020)

→ { "annee": 2020, "jour": 20, "mois": 1 }

⚠ Ces données mutables ne sont pas admises  
dans les ensembles.

{ [3, 1, 4]} → Type Error  
{ s1 } → Type Error

- $s_1 = \{ \} \cup \{ 2 \} \rightarrow \text{True}$  &  $s_1$  is  $s_2 \rightarrow \text{False}$   
 $\downarrow$   
= b' logiq  
= b' physiq
- $s_1. add(4) \rightarrow s_1 \rightarrow \{ 1, 2, 3, 4 \}$
- $s_1. remove(4) \rightarrow s_1 \rightarrow \{ 1, 2, 3 \}$

## Dictionnaires

Dictionnaire : 5<sup>e</sup> données contenant  $\leftrightarrow$  clé / valeur.

## Constantes & littérale

tarifs = { 'banane': 2.59, 'pomme': 1.95, 'orange': 1.95 }

## f dict

dict([('banane', 2.59), ('orange', 1.95)])

→ { 'banane': 2.59, 'orange': 1.95 }

dict(jour = 20, mois = 1, année = 2020)

→ { 'année': 2020, 'jour': 20, 'mois': 1 }

## % import timeit

number = 10 000

tps = [timeit.timeit(stmt='fact(2:d)', format(n),  
globals=globals(), number=number) / number  
(paramètres) (cas défaut) (cas défini)]

## import pylab

pylab.plot(abcisses, tps, 'r')

mem\_fact = [0:1]

for n in range(1, 10):

mem\_fact[n] = n \* mem\_fact[n-1]

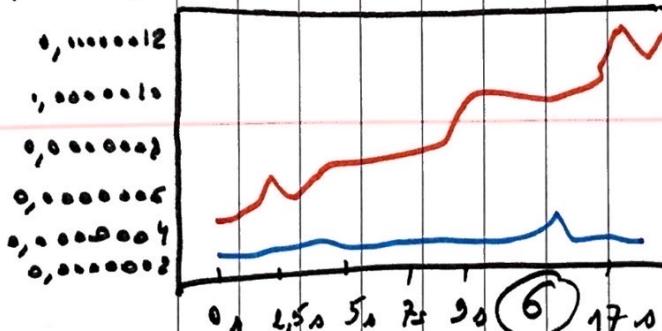
def fact2(n):

return mem\_fact[n]

tps2 = [timeit.timeit(stmt='fact2(2:d)', format(n),  
globals=globals(), number=number) / number

for n in abcisses

pylab.plot(abcisses, tps, 'r', abcisses, tps2, 'g')



93\* import timeit

number = 10 000

tps = [timeit.timeit(stmt='fact(2:d)'), format(n),  
globals=globals(), number=number) / number  
(paramètres) (rôle des inputs) (rôle des outputs)

import pylab

pylab.plot(abcisses, tps, 'x')

mem\_fact = [0:1]

for m in range(1, 10):

mem\_fact[m] = m \* mem\_fact[m-1]

def fact2(m):

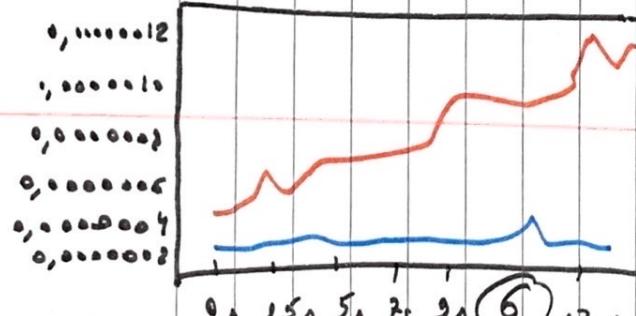
return mem\_fact[m]

tps2 = [timeit.timeit(stmt='fact2(2:d)'), format(n),

globals=globals(), number=number) / number

for m in abcisses

pylab.plot(abcisses, tps, 'x', abcisses, tps2, 'g')



RF / RIF: moins de temps pour recherche infructueuse  
recherche fructueuse.

f comparaison trivaluée:  
cv: type(x) == type(y) compare(x, y) =  $\begin{cases} 0 & \text{si } x = y \\ -1 & \text{si } x < y \\ 1 & \text{si } x > y \end{cases}$

Algo recherche ségRe laborieuse:

trouve = Faux

Pour i variant de a à b-1

si comp(elt, seq[i]) == 0

trouve = vrai

Renvoyer trouve

def rech\_laborieuse(elt, seq, a=0, b=None, comp=compare):

if b is None:

b = len(sequence)

trouve = False

for i in range(a, b):

if comp(elt, seq[i]) == 0:

trouve = True

return trouve

$$C_{\text{laborieuse}}(n) = \sum_{i=0}^{n-1} 1 = n$$

$$C_{\text{RF}}(n) = C_{\text{RIF}}(n).$$

def rech\_seq\_laborieuse(elt, seq, a=0, b=None):

if b is None:

b = len(sequence)

trouve = False

i = a

while not trouve and i < b:

if comp(elt, seq[i]) == 0:

i += 1

return trouve

7

$$\text{MNC} \rightarrow C(m) \text{ RF} = \frac{m}{2} \quad \& \quad C(m) \text{ RIF} = m$$

$\sum_{i=a}^k 1 = k - a + 1 \leq m$   
meilleur cas      pire cas

Recherche de 5<sup>e</sup> séquence triée

- Séqs triées :  $0 \leq i < m-1$       seq[i] < seq[i+1]
- constitue range.

trouve = Faux, fimi = Faux, i = a  
while **not** trouve and **not** fimi :

res = comp(elt, seq[i])

if res == 0:

trouve = Vrai

elif res == -1:

fimi = Vrai

i += 1

if i == b:

fimi = Vrai

return trouve

**MNC**  $C(m) \text{ RF} = \frac{m}{2} = C(m) \text{ RIF}$  triée

(8)

## Algo Recherche dichotomique

$$m = 10$$

$$f =$$

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

$$\text{elt} = 2, a = 0, b = 10$$

$$m \leftarrow (d + p) // 2$$

comp(elt, f[m])		d	1	m
		0	9	
-1		0	4	4
0		0	2	2

return TRUE car

$$C(m) = (\Sigma 1) + 1$$

$$m=8$$

$$0..7$$

$$0..3$$

$$0..1$$

$$2..3$$

$$4..7$$

$$4..5$$

$$6..7$$

$$6..6$$

$$7..7$$

$$0..1$$

$$2..2$$

$$3..4$$

$$5..5$$

$$6..7$$

$$8..8$$

$$9..9$$

$$10..10$$

$$3..3$$

$$4..4$$

$$6..6$$

$$7..7$$

$$4 = 3 + 1 \leq C(11) \leq 4 + 1 = 5$$

$$3 \text{ branches}$$

$$i$$

$$C(8) = 3 + 1 = 4 = c(2^3)$$

$$C(m) = c(2^p) = p + 1$$

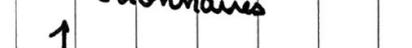
$$\Leftrightarrow m = 2^p \Leftrightarrow p = \log_2 m$$

$$\lfloor \log_2 m \rfloor + 1 \leq C(m) \leq \lceil \log_2 m \rceil + 1$$

$$x \in R, \lceil x \rceil = + \text{grad entier} \leq x$$

$$x \in R, \lceil x \rceil = + \text{ptt entier} > x.$$

Dictionnaires



timeit

(9)

11

## exo 4 : Algorithme de tri

Tri en Python

o f sorted → renvoie liste  
↳ p EDB

o M sort → Δ que pour liste  
↳ renvoie rien  
↳ p EDB.

f anonymes :

lambda x: 2\*x

(1 seule expression)

TRI PAR SELECTION :

```
def select_indice_min (t, a, b, comp = compare):
    ind_min = a
    for i in range (a+1, b):
        if comp (t[i], t[ind_min]) == -1:
            ind_min = i
    return ind_min
```

```
def echanger (t, i, j):
    t[i], t[j] = t[j], t[i]
```

```
def tri_selection (t, comp = compare):
```

$n = \text{len}(t)$

for i in range (n-1):

indice\_min = select\_indice\_min (t, i, n, comp = comp)

echanger (t, i, indice\_min)

⑩ Δ pas de return

```
    => sorted ('TIMOLEON')
    => [' ', 'i', ' ', 'l', ' ', 'm', ' ', 'o', ' ', 'n']
```

=> f. sort()
 (reverse = True)
 tri ordre décissant

## 6: Recherche dichotomique

d = a

j = b - 1

TQ d < j :

m = (d+j) // d

si comp (t[m], x) < 0 :

d = m + 1

sinon

j = m

fin TQ

si x = t[d] :

renvoie True

sinon

renvoie False

• Tri p sélect;

@ si l'ic → du minimum  
[+] échanger

t = TIMOLEON  
EIMOLTON  
EILONTON  
EILMOTON  
EILMNTOO  
EILMNOTO  
EILMNOTT

s'arrêter avec min  
éph.

$$C_{\text{tri-select}}(n) = \sum_{i=0}^{n-1} C_{\text{select-indice-min}}(n-i) = \sum_{i=0}^{n-1} (n-i-1)$$

$$\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2 - n}{2}$$

⑪

## TRI PAR INSERTION

def insérer(l, i, comp = compare):

aux = l[i]

k = i

while k >= i and comp(aux, l[k-1]) == -1:

l[k] = l[k-1]

k = k - 1

l[k] = aux

def tri\_inser(l, comp = compare):

m = len()

for i in range(l, m):

insérer(l, i, comp = comp)

$$m-1 = \sum_{i=1}^{m-1} 1 \leq C_{\text{tri-inser}}(m) = \sum_{i=1}^{m-1} C_{\text{insérer}}(m, i) \leq \sum_{i=1}^{m-1} \frac{m(m-1)}{2}$$

cas pire

renfin si liste bien triée

Rg: C. m.  $\log m$  : r<sup>pe</sup> + pt. le possible p. tri: (M sort)

Rg: Nc pas comparer 2 dictionnaires en Python.

@ tri par insert

T I M O L E O N  
 Z I M O L E O N  
 I M T O L E O N  
 I M O T L E O N  
 I L M O T C O N  
 E I L M O T O N  
 E I L M O S T N  
 E I L M N O S T

(12)

## échanger

□ M1 pas satisfaisante

def mélanger - ☹(l):

m = len(l)

for i in range(m):

j = random. randrange(m)

echanger(l, i, j)

□ M2 correcte

def mélanger ☺(l):

m = len(l)

for i in range(m-1):

j = random. randrange(i, m)

echanger(l, i, j)

pas équiprobable

& intervalle  
fluctuant à chaque échange

(13)

(14)