

Informatique et Programmation

Cours 6

Jean-Jacques Lévy

`jean-jacques.levy@inria.fr`

`http://jeanjacqueslevy.net/prog-py-22`

Plan

- interaction graphique
- solution des exercices
- tri récursif
- alias
- dictionnaires
- recherche en table
- recherche en table par dichotomie

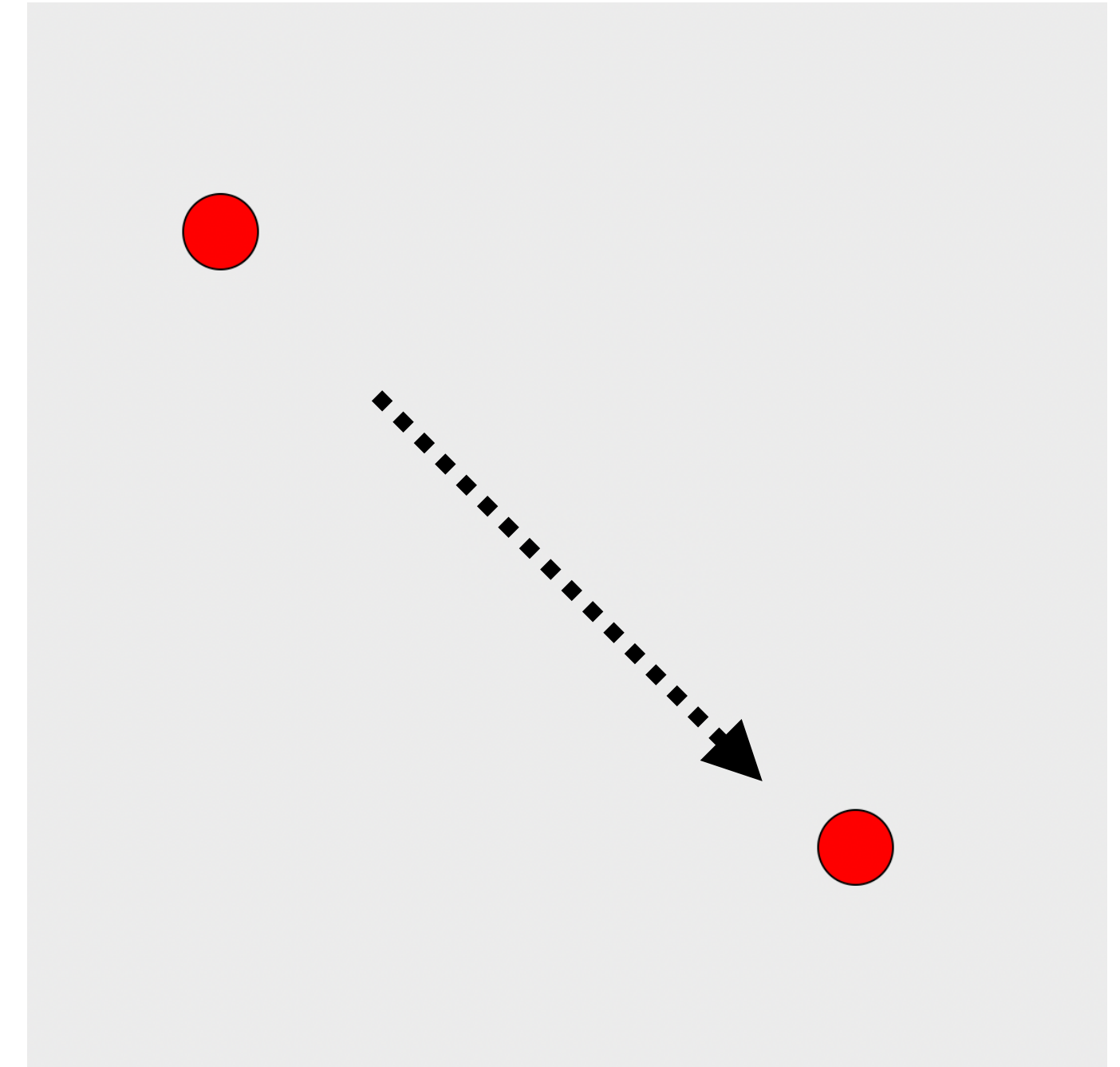
dès maintenant: **télécharger Python 3 en** `http://www.python.org`

Graphique ++

- utilisation de la souris avec le module graphics

```
from graphics import *

win = GraphWin("Update Example", 1320, 1200, autoflush=False)
p0 = win.getMouse()
x = p0.x; y = p0.y
for i in range(50):
    c = Circle(Point(x, y), 10)
    c.setFill("red"); c.draw(win)
    update(50)
    c.undraw()
    x = x + 10; y = y + 10
win.getMouse() # Pause to view result
win.close()
```



- getMouse **attend** un clic souris et renvoie le Point où se trouve alors la souris (module graphics)

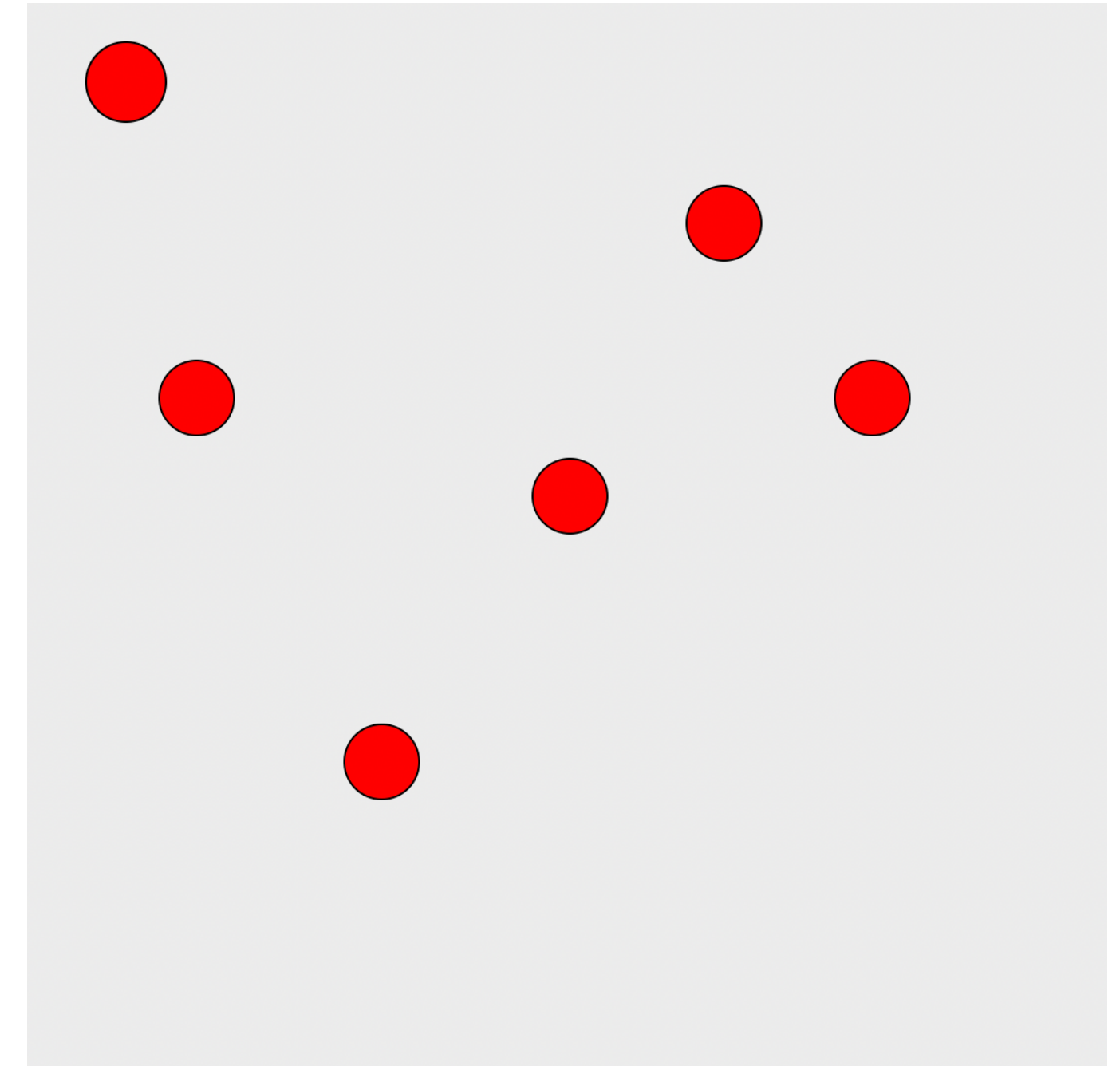
Graphique ++

- utilisation de la souris avec le module graphics

```
from graphics import *

win = GraphWin("CheckMouse Example", 1320, 1200, autoflush=False)
p0 = win.getMouse()
c = Circle(p0, 20)
c.setFill("red"); c.draw(win)
for i in range(6):
    while True:
        p = win.checkMouse()
        if p != None:
            center = c.getCenter()
            steps = 35
            xincr = (p.x - center.x) / steps
            yincr = (p.y - center.y) / steps
            for _ in range(steps):
                c.move(xincr, yincr)
            break

win.getMouse() # Pause to view result
win.close()
```



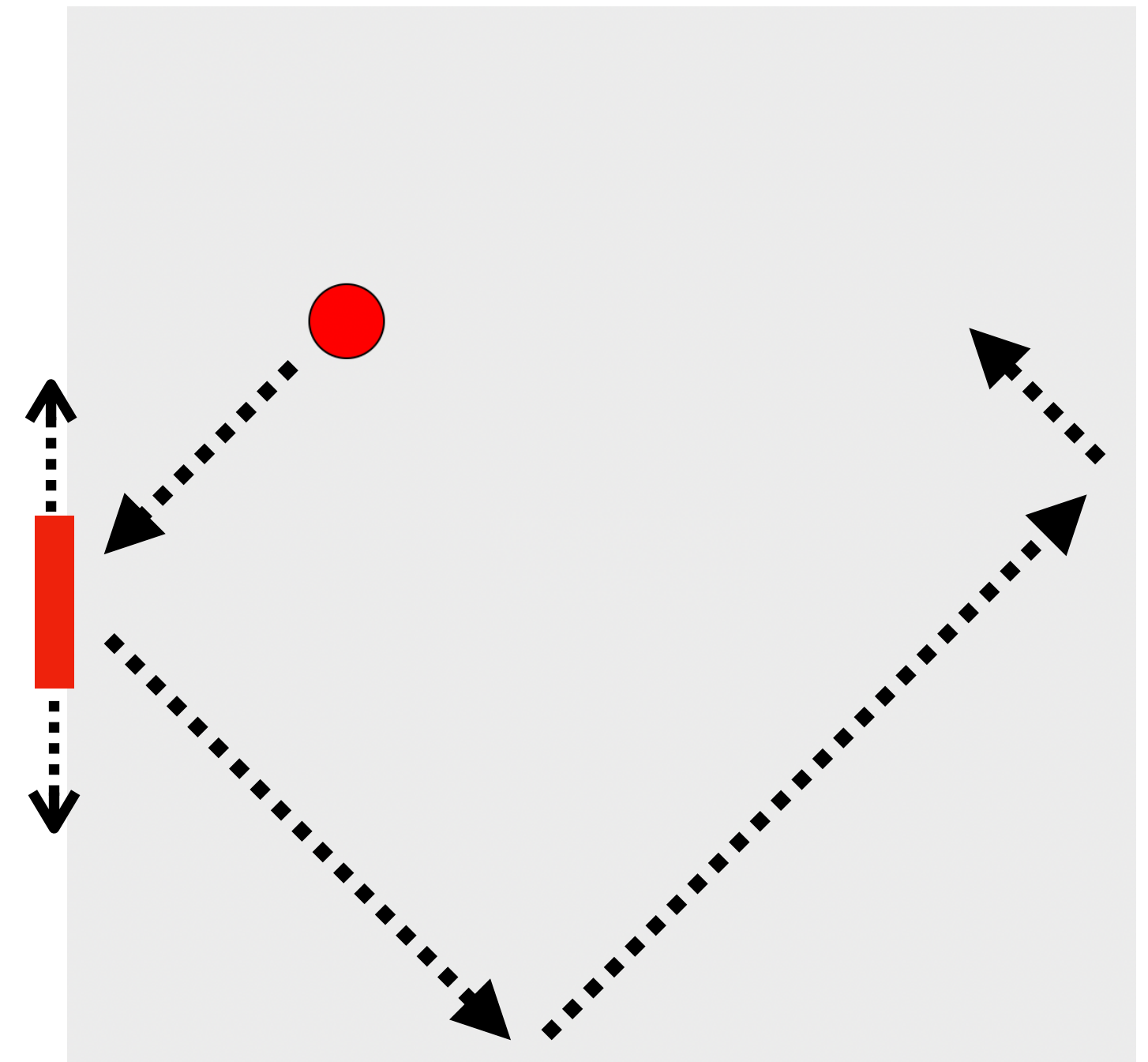
- checkMouse n'est **pas bloquant** et renvoie le Point où se trouvait le dernier clic (None si pas de clic)

Graphique ++

Exercice 1 Faire rebondir une balle dans un rectangle.

Exercice 2 Faire une raquette qui suit les mouvements verticaux de la souris

Exercice 3 Faire un jeu de pong

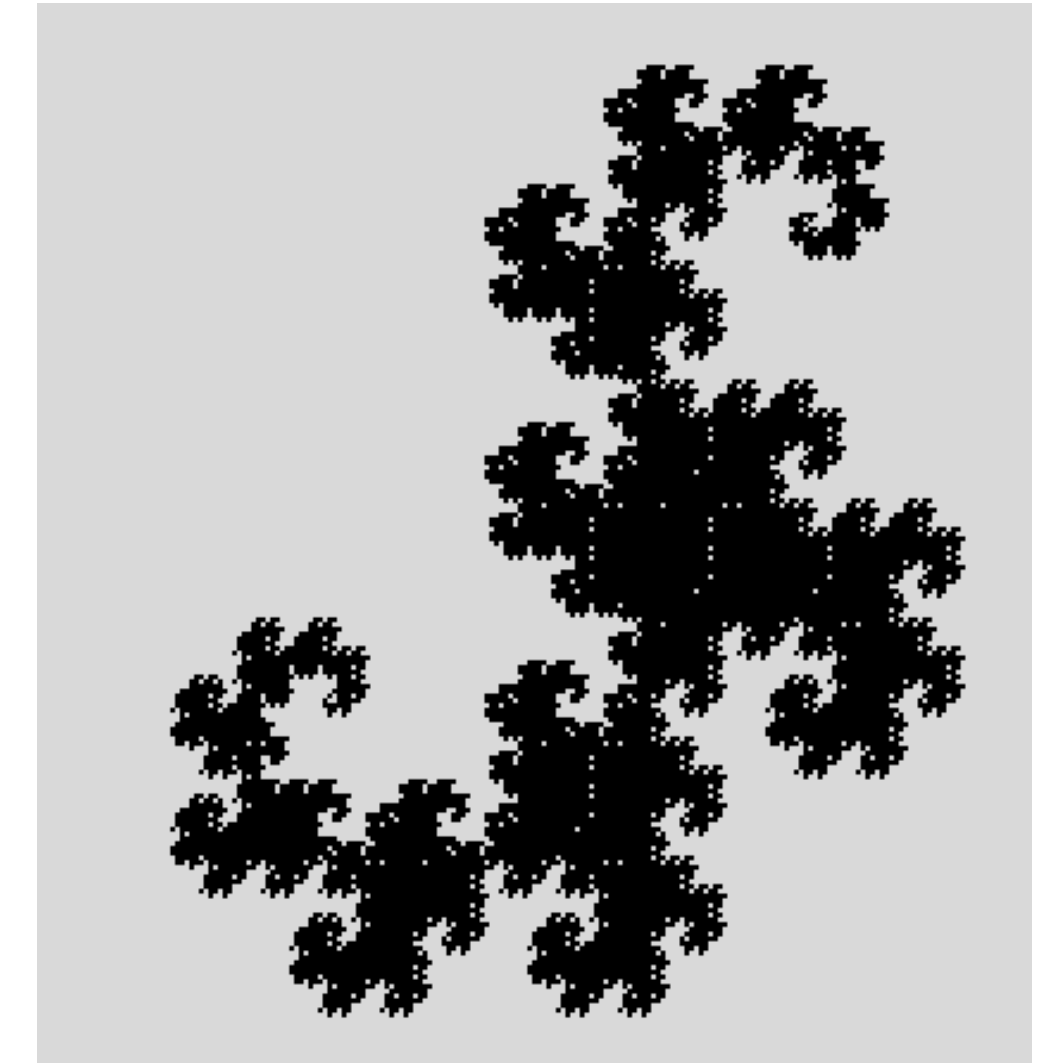
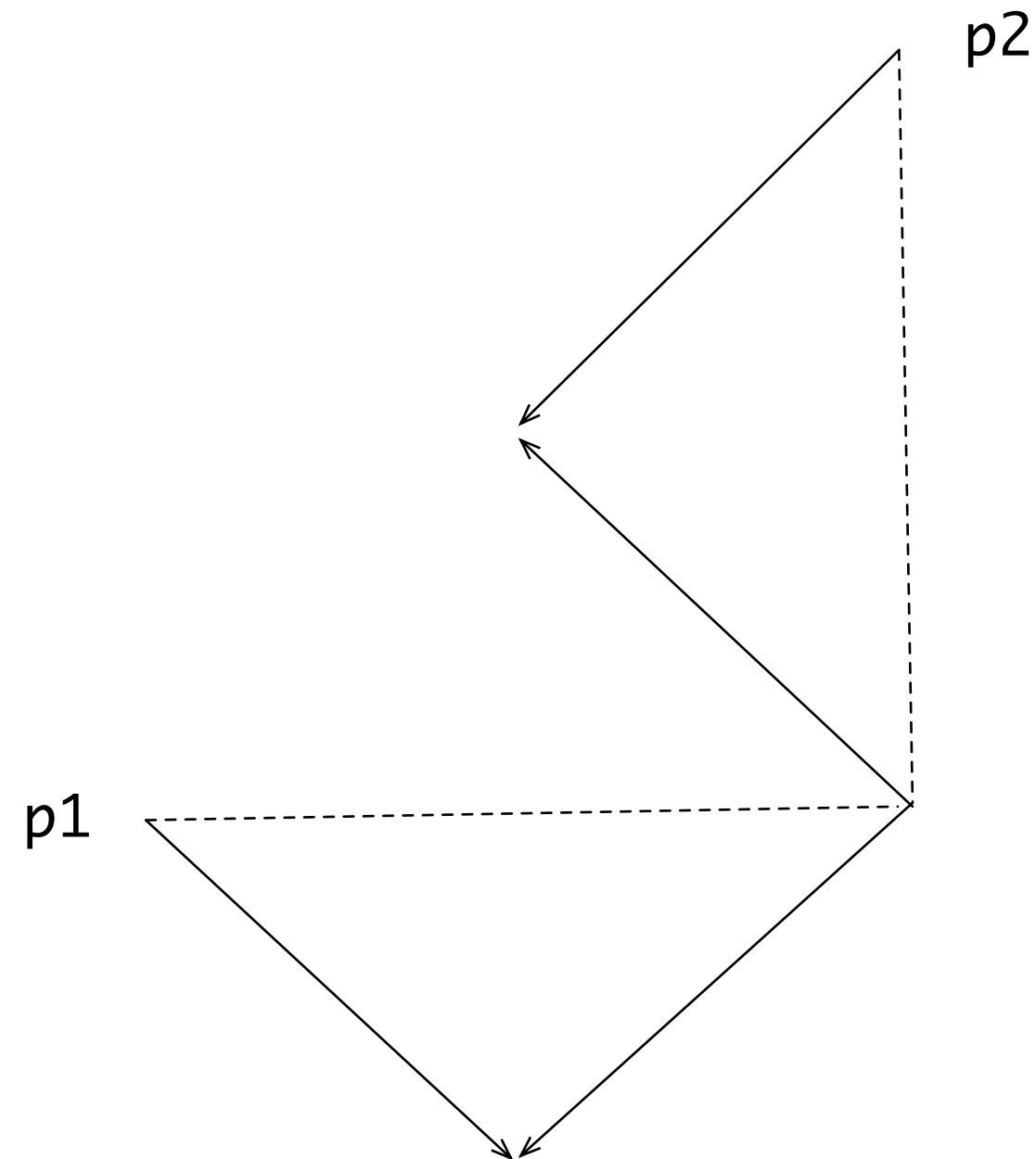


Rappel

- courbe du dragon

```
def dragon (win, n, x, y, z, t):  
    if n == 1 :  
        p1 = Point (x,y)  
        p2 = Point (z,t)  
        l = Line (p1, p2)  
        l.draw(win)  
    else :  
        u = (x + z + t - y) // 2  
        v = (y + t - z + x) // 2  
        dragon (win, n-1, x, y, u, v)  
        dragon (win, n-1, z, t, u, v)
```

- on plie une feuille de papier n fois



```
from graphics import *  
  
def dessinF (n, x, y, z, t):  
    winx = 1000; winy = 1000  
    win = GraphWin("un beau dessinM", winx, winy, autoflush=False)  
    win.setCoords (0, 0, winx, winy)  
    dragon (win, n, x, y, z, t)  
    win.update()  
    win.getMouse() # Pause to view result  
    win.close()   # Close window when done    r.draw()
```

les sources sont en <http://jeanjacqueslevy.net/prog-py-22/progs/c6a.py>

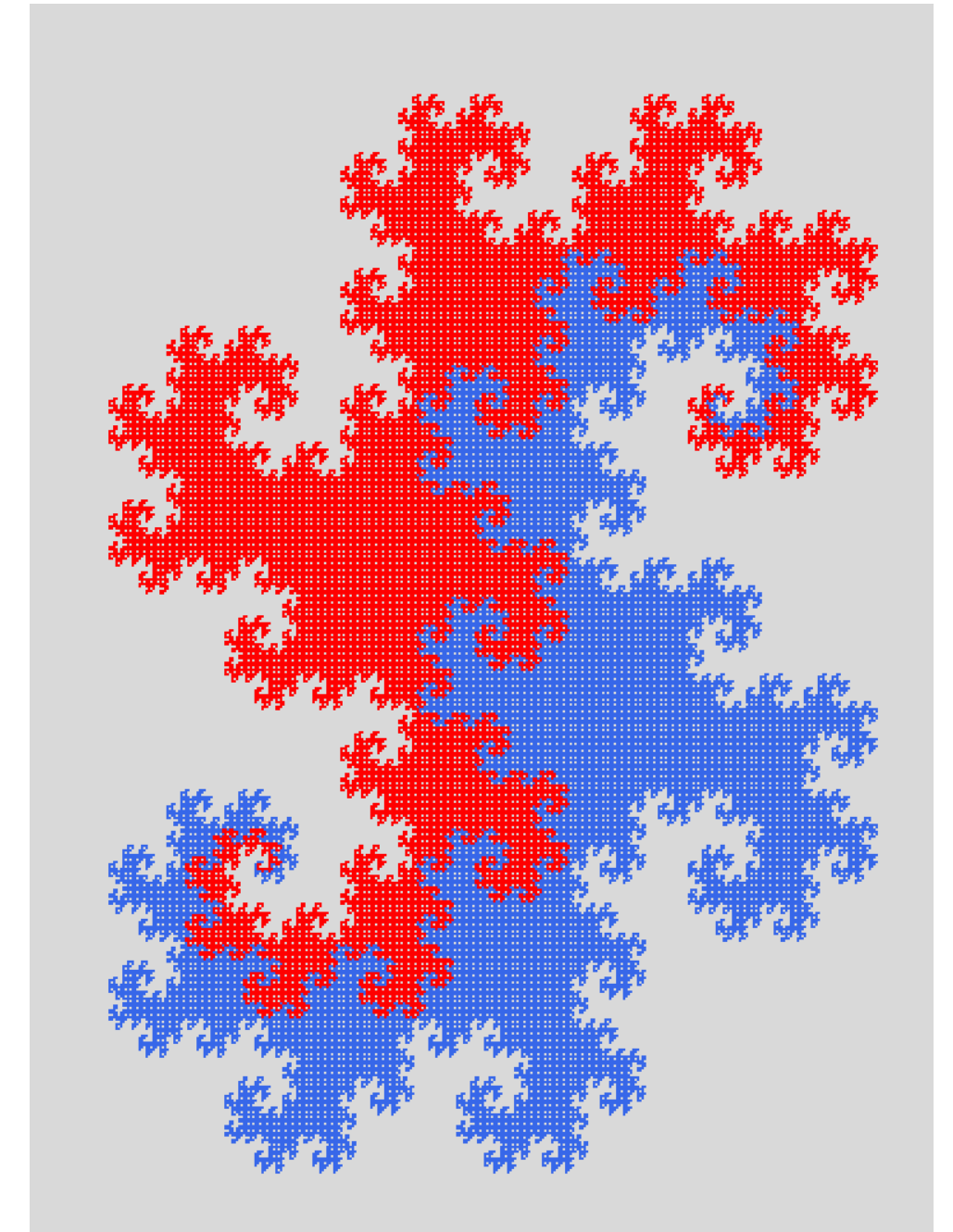
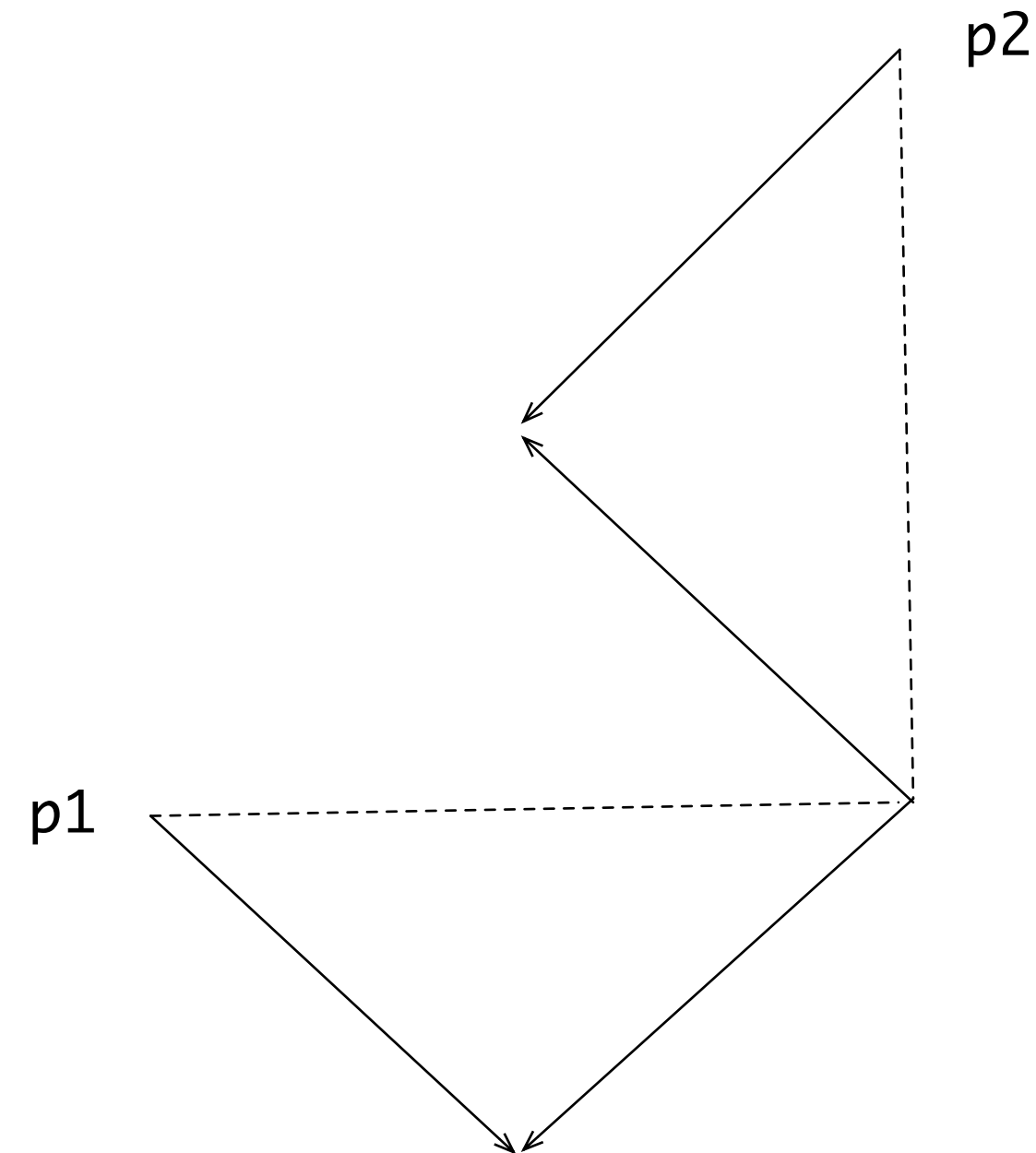
Solution des exercices

- twin dragons

```
def dragonTC (win, n, x, y, z, t, s, color) :  
    if n <= 1 :  
        p1 = Point (x,y); p2 = Point (z,t)  
        l = Line (p1, p2);  
        l.setOutline (color)  
        l.draw(win)  
    else :  
        u = (x + z + s*t - s*y) / 2  
        v = (y + t - s*z + s*x) / 2  
        dragonTC (win, n-1, x, y, u, v, 1, color)  
        dragonTC (win, n-1, u, v, z, t, -1, color)
```

- rajoutés:

1. couleur
2. sens (1 si à l'endroit, -1 si à l'envers)



```
def dessinG (n, x, y, z, t) :  
    winx = 1000; winy = 1000  
    win = GraphWin("un beau dessinM", winx, winy, autoflush=False)  
    win.setCoords (0, 0, winx, winy)  
    dragonTC (win, n, x, y, z, t, 1, 'royal blue')  
    dragonTC (win, n, x, y, z, t, -1, 'red') # twin dragon  
    win.update()  
    win.getMouse() # Pause to view result  
    win.close() # Close window when done r.draw()
```

Solution des exercices

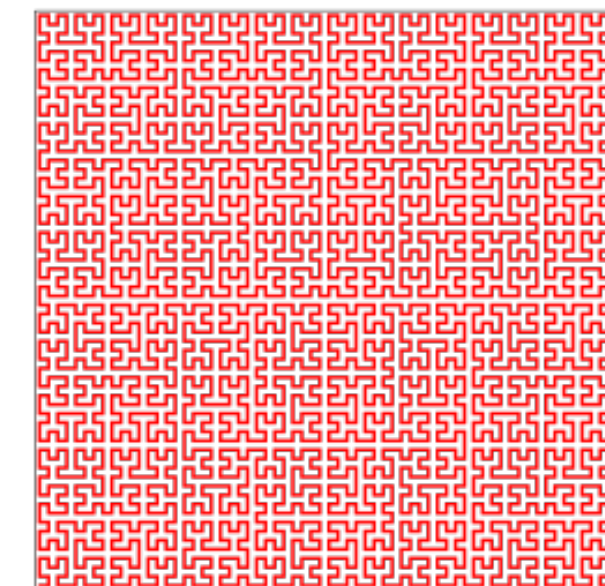
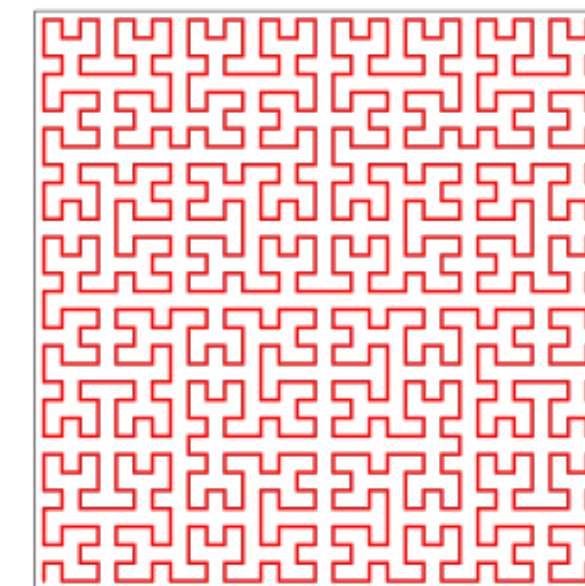
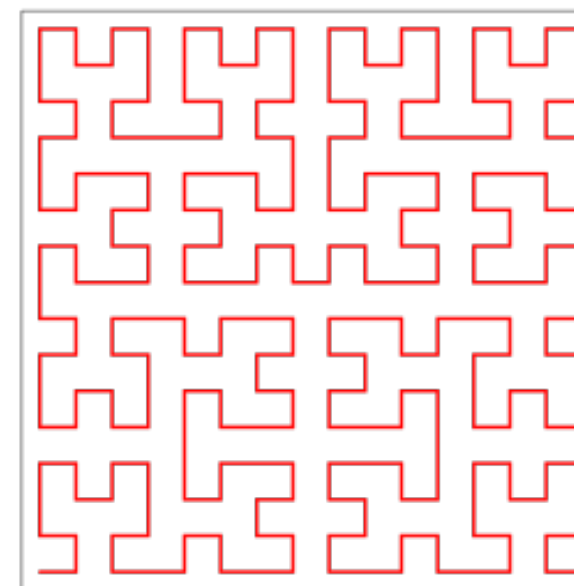
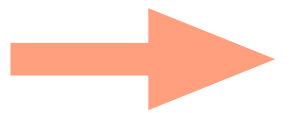
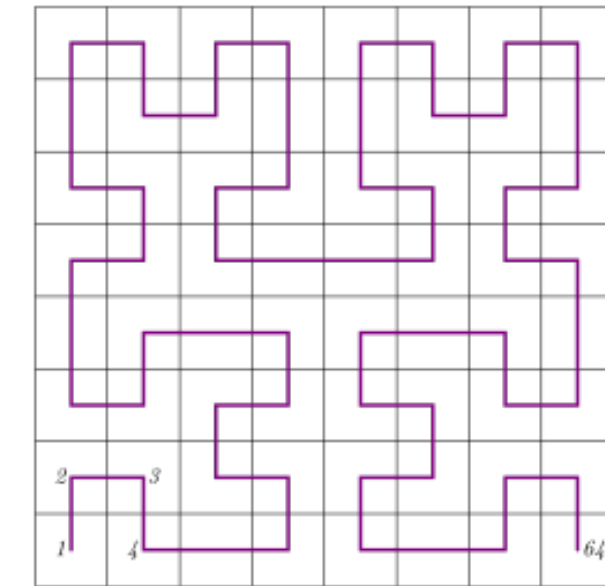
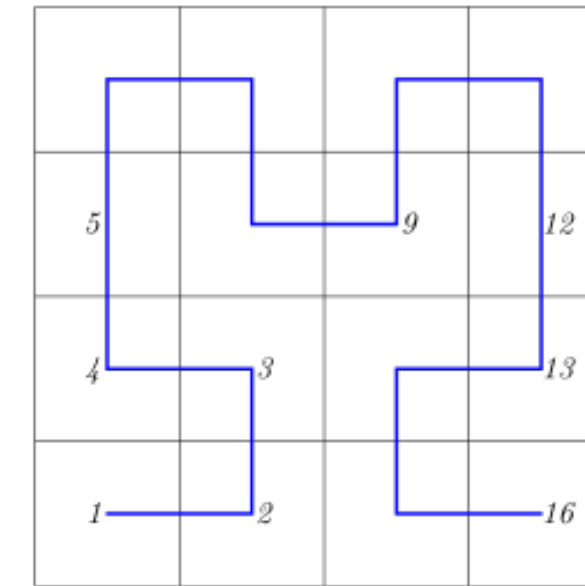
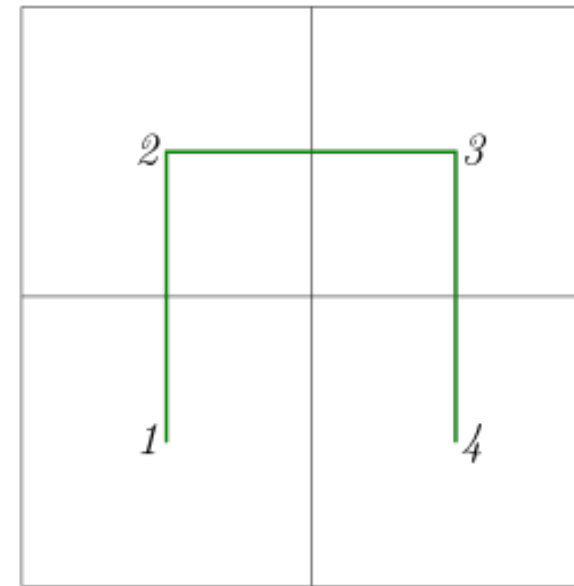
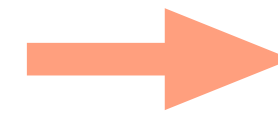
- la courbe de Hilbert

```
c = 30
```

```
def hilbertL (n) :  
    if n > 0 :  
        left (90); hilbertR (n-1)  
        forward (c)  
        right (90); hilbertL (n-1);  
        forward (c)  
        hilbertL (n-1); right(90)  
        forward (c);  
        hilbertR (n-1); left(90)
```

```
def hilbertR (n) :  
    if n > 0 :  
        right (90); hilbertL (n-1)  
        forward (c)  
        left (90); hilbertR (n-1)  
        forward (c)  
        hilbertR (n-1); left(90)  
        forward (c)  
        hilbertL (n-1); right(90)
```

```
from turtle import *  
def reset() :  
    home() ; clear(); speed (20)
```



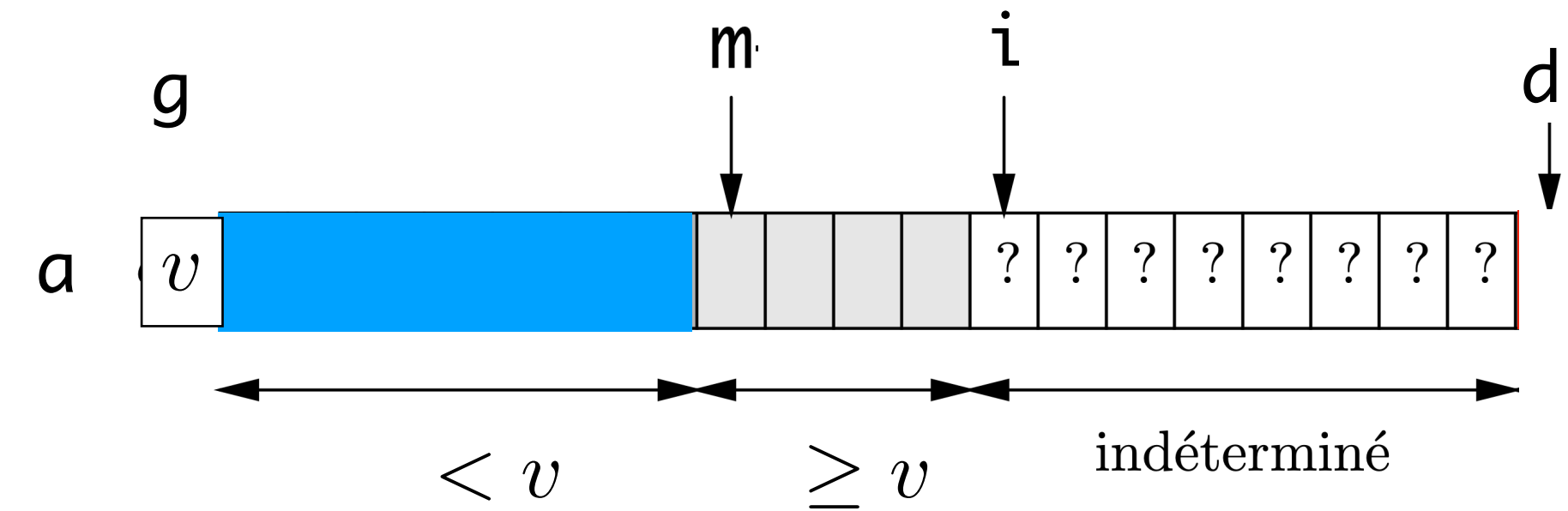
Tri récursif

- tri rapide (*Quicksort*)

```
def tri_rapide (a) :  
    tri_rapide1 (a, 0, len(a))
```

```
def tri_rapide1 (a, g, d) :  
    if g < d - 1 :  
        v = a[g]  
        m = g + 1  
        for i in range (g+1, d):  
            if a[i] < v :  
                t = a[m]; a[m] = a[i]; a[i] = t  
                m = m + 1  
        a[g] = a[m-1]; a[m-1] = v  
        tri_rapide1 (a, g, m-1)  
        tri_rapide1 (a, m, d)
```

- bonne méthode de tri en moyenne



- on met $a[g]$ à sa place dans a trié
- et on recommence sur les parties gauche et droite

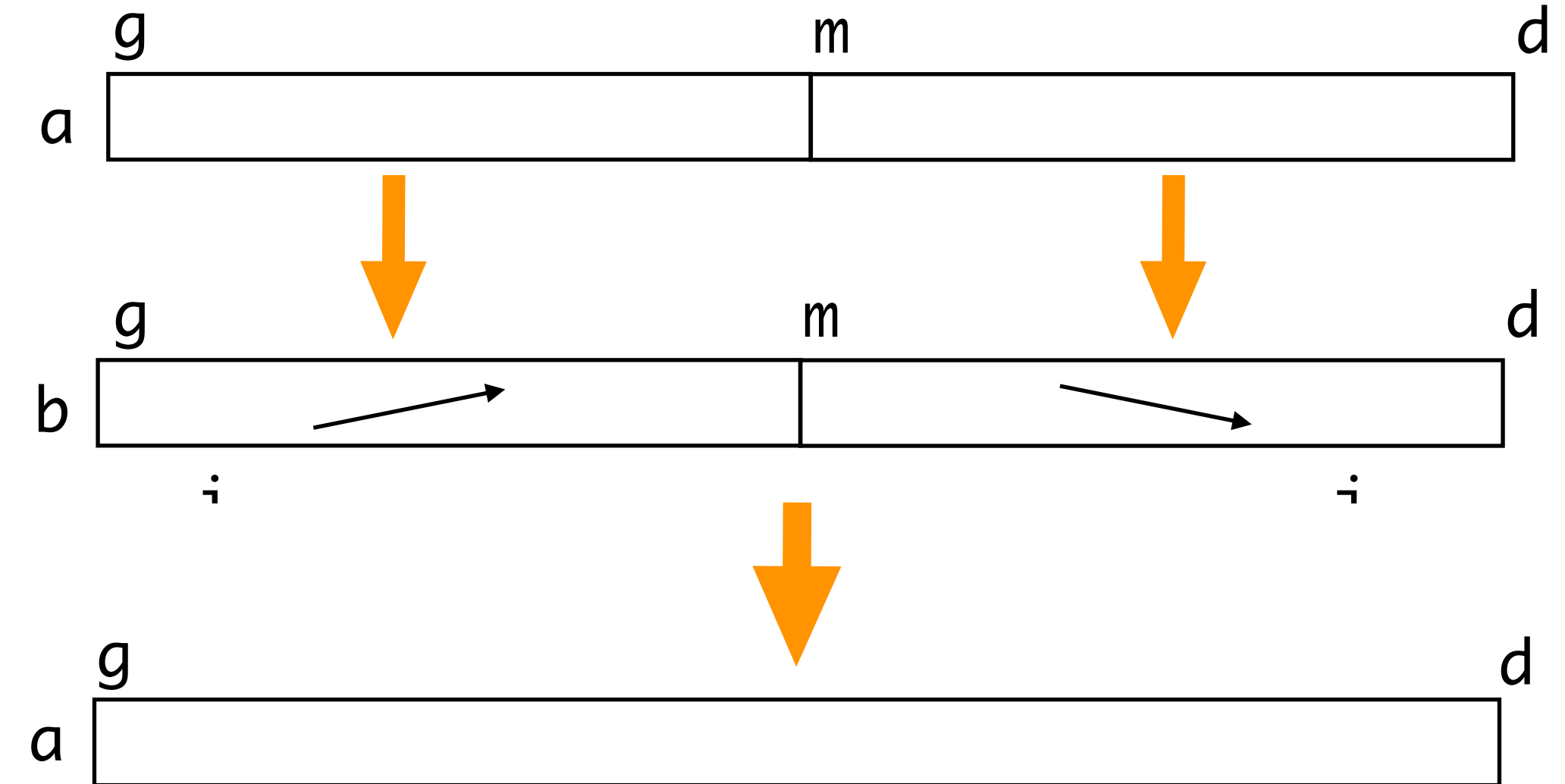
Tri récursif

- tri fusion (*merge sort*)

```
def tri_fusion (a) :  
    n = len (a)  
    b = [0 for i in range(n)] #tableau intermédiaire  
    tri_fusion1 (a, 0, n, b)
```

```
def tri_fusion1 (a, g, d, b) :  
    if g < d - 1 :  
        m = (g + d) // 2  
        tri_fusion1 (a, g, m, b)  
        tri_fusion1 (a, m, d, b)  
        for i in range (g, m) :  
            b[i] = a[i]  
        for j in range (m, d) :  
            b[m + d - 1 - j] = a[j]  
        i = g; j = d - 1;  
        for k in range (g, d) :  
            if b[i] < b[j] :  
                a[k] = b[i]; i = i + 1  
            else :  
                a[k] = b[j]; j = j - 1
```

- très bonne méthode de tri



- a) on coupe **a** en 2
- b) on trie les moitiés gauche et droite
- c) on copie les résultats dans un tableau annexe **b**
- d) on fusionne les 2 moitiés dans le tableau **a**

Valeur d'un tableau — Alias

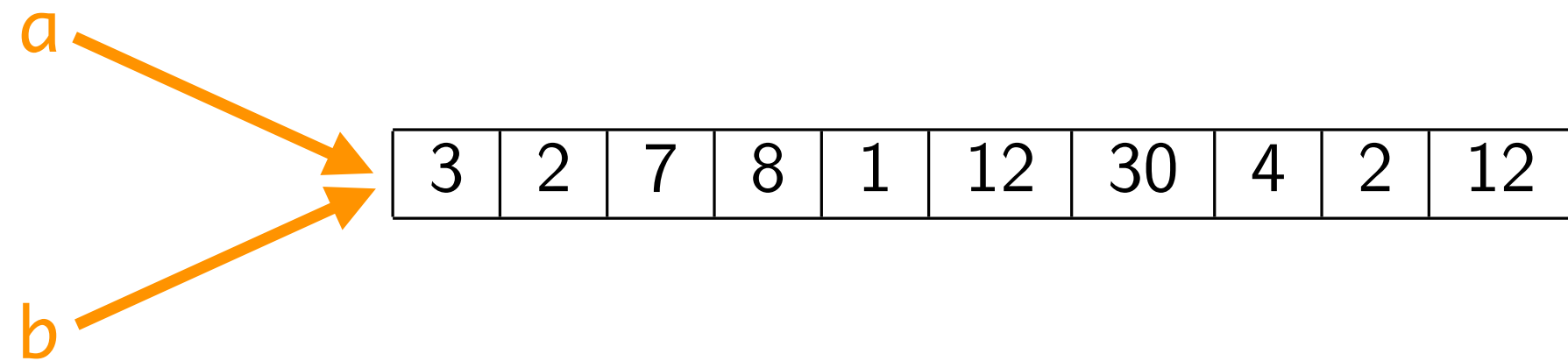
- soient 2 tableaux **a** et **b**

```
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]
```



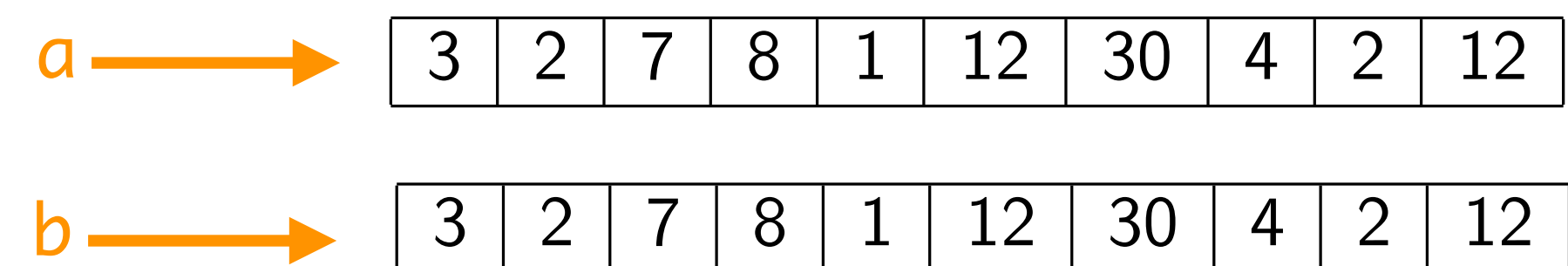
- les variables **a** et **b** sont 2 alias d'un même tableau
- la valeur de **a** ou de **b** est l'adresse mémoire de son premier élément

```
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

données modifiables et alias sont sources de bugs

Dictionnaires

- les dictionnaires sont des listes d'associations : clé — valeur

```
adresse = {'JJ': (32, 'boulevard St Michel', Paris, 'F', 75005),
           'Robert': (3, 'rue Chaban-Delmas', Bordeaux, 'F', 33000),
           'maman': (4, 'corniche André de Joly', Nice, 'F', 06300),
           'Santa': (6, 'route du Ciel', Marmande, 'F', 31330),
           'Xi': (2, 'pte de la Paix céleste', Beijing, 'PRC', 0001),
           'Manuel': (3, 'faubourg St Honoré', Paris, 'F', 75001)}
```

- les dictionnaires sont des listes d'associations : clé — valeur

```
age = {'JJ':35, "marie": 25, "alice": 38, "bob": 29}
print (age)
→ {'bob': 29, 'marie': 25, 'alice': 38, 'JJ': 35}
print (age['marie'])
→ 25
print (age['JJ'])
→ 35
age['raymond'] = 29
print (age)
→ {'bob': 29, 'marie': 25, 'alice': 38, 'JJ': 35, 'raymond': 29}
age ['JJ'] = 49
print (age)
→ {'bob': 29, 'marie': 25, 'alice': 38, 'JJ': 49, 'raymond': 29}
```

```
print (age['henry'])
→ Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'henry'
```

```
'henry' in age
→ False
'alice' in age
→ True
```

Ensembles — Compréhension

- 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
```

- 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}
```

Recherche en table

- les dictionnaires permettent d'accéder à une valeur par une clé
- le mécanisme de base est la recherche d'une clé en table
- exemple: recherche d'un numéro de téléphone dans un annuaire

```
def recherche (v, a) :  
    for p in a :  
        if p[0] == v :  
            return p[1]  
    return 'None'
```

- **def** il vaut mieux retourner une exception quand la clé n'est pas trouvée

```
def recherche (v, a) :  
    for p in a :  
        if p[0] == v :  
            return p[1]  
    raise ValueError
```



type du résultat plus cohérent avec le type des valeurs trouvées

a

('jj', '0543221800')
('marie', '0728423301')
('raymond', '0924828202')
('bob', '0630204100')
('alice', '0629359210')
('ali', '0330013917')

- on verra plus tard comment retourner une exception plus précise

Recherche en table

- la recherche précédente fait n opérations si n est la longueur de a
- si le tableau est trié dans l'ordre croissant des clés, on ramène ce nombre d'opérations à $\log(n)$

```
def recherche_dichotomique (v, a) :  
    g = 0; d = len(a)  
    while g < d :  
        m = (g + d) // 2  
        if v == a[m] :  
            return True  
        elif v < a[m] :  
            d = m  
        else :  
            g = m + 1  
    return False
```

a

('ali', '0330013917')
('alice', '0629359210')
('bob', '0630204100')
('jj', '0543221800')
('marie', '0728423301')
('raymond', '0924828202')

← $n = 230000$ donne $\log(230000) \sim 18$ opérations

Recherche par interpolation

- on a une connaissance de la distribution des clés (par exemple uniforme)
- au lieu de comparer à la clé du milieu, on peut comparer à une clé plus proche de la distribution

```
def recherche_interpolation (v, a) : # v supposé nombre entier
    g = 0; d = len(a)
    while g < d and a[g] <= v <= a[d-1] :
        lo = a[g]; hi = a[d-1]
        if hi == lo:
            m = g
        else :
            m = round ((g * (hi - v) + (d - 1) * (v - lo) ) / (hi - lo))
        if v == a[m] :
            return True
        elif v < a[m] :
            d = m
        else :
            g = m + 1
    return False
```


$$m = \frac{g(hi-v) + (d-1)(v-lo)}{(hi-lo)}$$

- pour une distribution uniforme, cela donne de l'ordre de $\log(\log(n))$ opérations

4.16 **opérations pour table de 239210 entrées !!**

Recherche par interpolation

- par exemple, voici un test :

```
a = rand_array (23921, 10000)
tri_insertion (a)
v = 60

print (a)
print (recherche_interpolation (v, a))
```

- qui donne en 3 itérations :

```
[Running] python -u "z4.py"
[0, 1, 2, 2, 2, 2, 3, 4, 4, 4, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 9, 9, 9, 10, 10, 10, 10, 10, 11,
12, 12, 12, 12, 13, 13, 13, 14, 14, 15, 15, 16, 17, 18, 19, 20, 20, 21, 21, 22, 22, 23, 23, 23, 24,
25, 26, 27, 28, 29, 29, 29, 32, 33, 33, 33, 33, 34, 34, 35, 35, 35, 35, 35,
...
9994, 9994, 9994, 9994, 9994, 9995, 9995, 9996, 9996, 9996, 9997, 9998, 9998, 9998, 9999, 10000, 10000]
0 60 23921
0 60 144
True
```

Lecture de fichier

- pour lire un fichier, on l'ouvre en mode lecture seule 'r' (read)
et on le lit en séparant les lignes

```
def lire_lignes (nom) :  
    f = open (nom, 'r')  
    return f.read().splitlines()
```

- dans le répertoire courant, il y a un dictionnaire de français french-iso de 239210 mots

```
>>> a = lire_lignes ('french-iso')  
>>> len(a)  
239210
```

```
>>> for i in range (10) :  
...     print (a[i])  
...  
a  
abaca  
abacule  
abaissa  
abaissable  
abaissai  
abaissaient  
abaissais  
abaissait  
abaissant
```

application de la méthode read à f
puis de la méthode splitlines

Recherche d'un mot français

- on vérifie les mots de français

```
a = lire_lignes ('french-iso')
tri_Fusion (a)
recherche_dict ('jouer', a)
→ True
recherche_dict ('jouera', a)
→ True
recherche_dict ('jouerra', a)
→ False
```

Recherche d'un mot français

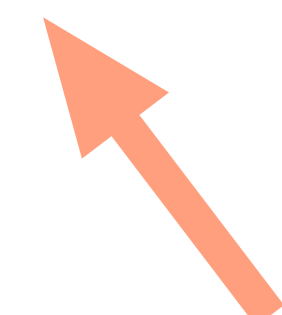
- et on peut faire un vérificateur de français

```
a = lire_lignes ('french-iso')
tri_Fusion (a)

b = lire_mots_fichier ('texte.txt')
for w in b:
    if not recherche_dichotomique (w, a):
        print (w)
```

- le vérificateur n'est pas parfait
 - le dictionnaire french-iso ne contient pas les noms propres
 - la méthode split() ne tient pas compte des ponctuations

```
def lire_mots_fichier (nom) :
    f = open (nom, 'r')
    return f.read.split()
```



application de la méthode read à f
puis de la méthode split

Conclusion

VU:

- tri rapide, tri fusion
- alias
- recherche en table
- ensembles, dictionnaires

TODO list

- listes
- classes et objets
- types de données structurées