Informatique et Programmation

Cours 3

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-py

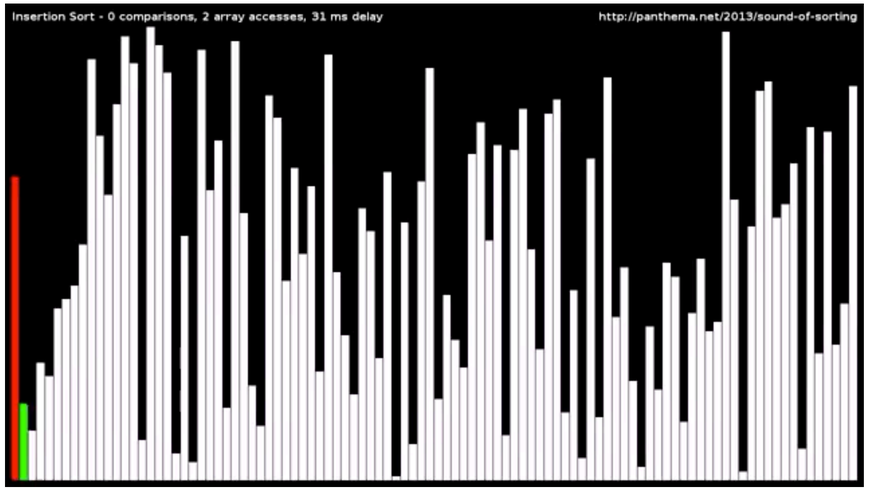
Plan

- exercices du cours 2
- fonctions récursives
- fonctions graphiques
- la petite tortue de Papert
- fractales
- fonctions récursives non numériques
- récursivité et raisonnement inductif

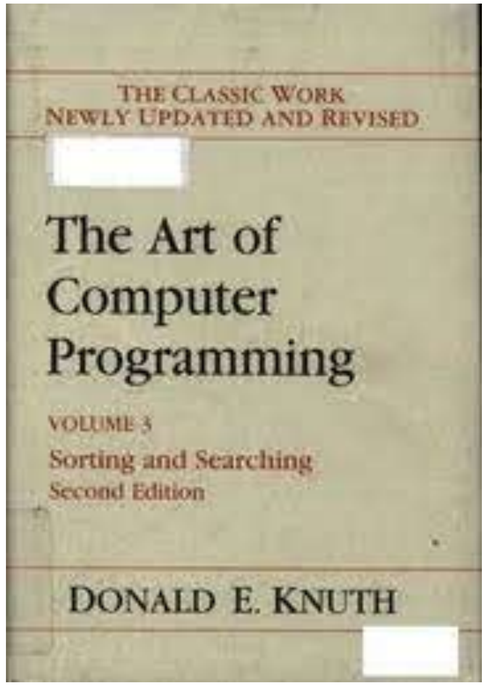
dès maintenant: télécharger Python 3 en `http://www.python.org`

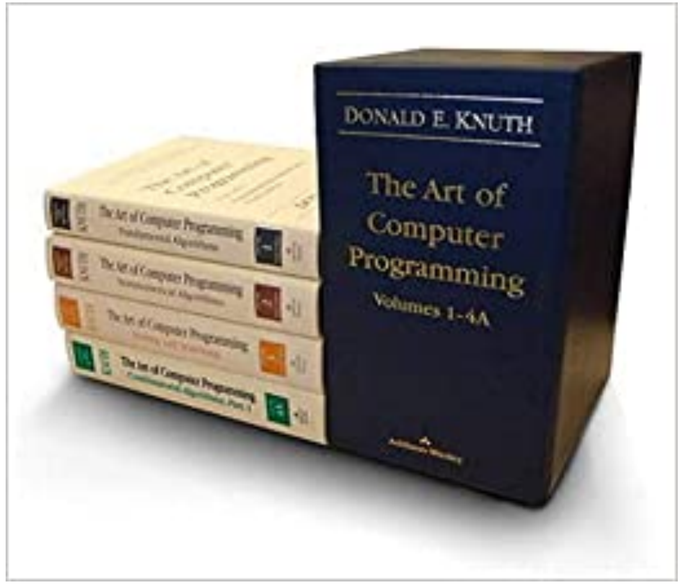
Exercices du cours 2

- écrire le tri par insertions



- quel est le meilleur de ces 3 tris ?

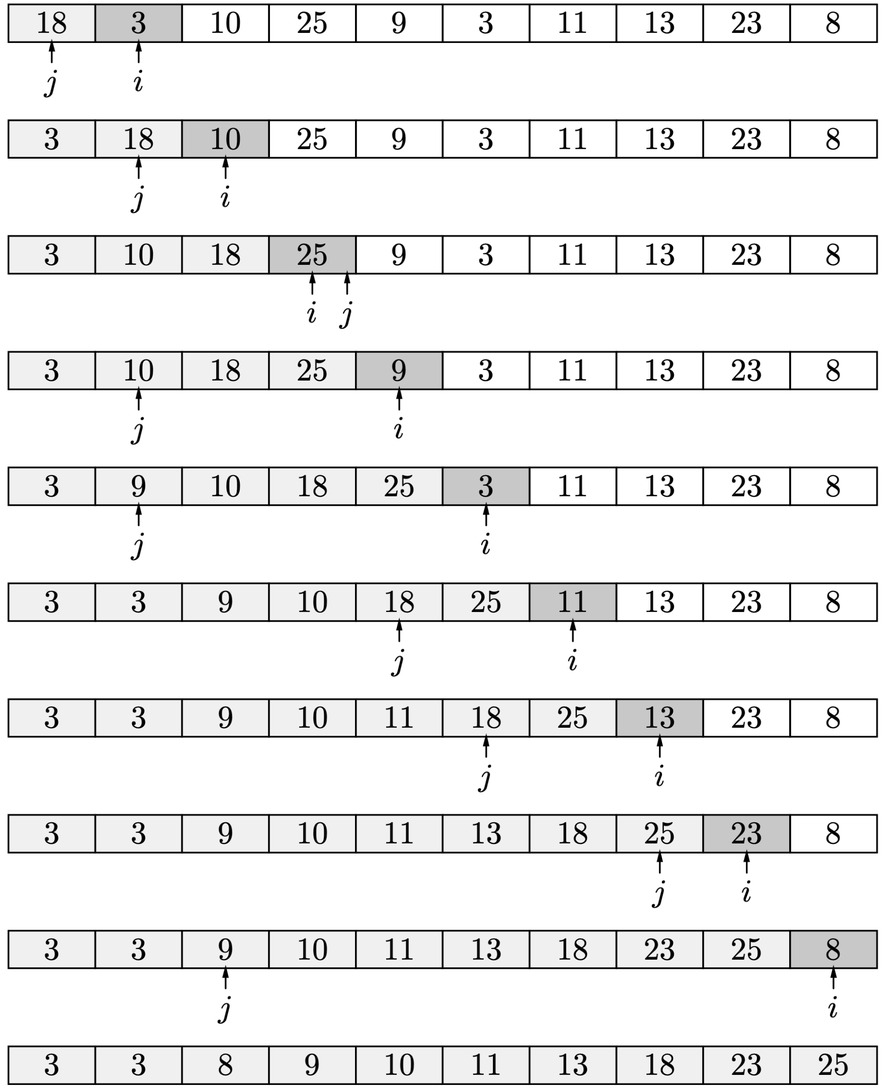




- prochain cours: la récursivité

Exercices du cours 2

```
def tri_insertion (a) :
    n = len (a)
    for i in range (1, n) :
        v = a[i]; j = i
        while (j > 0 and a[j-1] > v) :
            a[j] = a[j-1];
            j = j - 1
        a[j] = v
```



Fonctions récursives

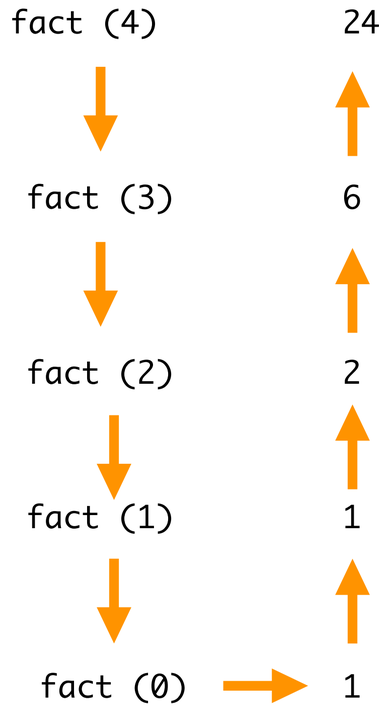
- une fonction qui se rappelle avec un argument plus petit

```
def fact (n) :
    if n == 0 :
      return 1
    else :
      return n * fact (n-1)
```

factorielle

```
>>> fact (3)
6
>>> fact (4)
24
>>> fact (10)
3628800
```

appels récursifs




```
fact (4) == 4 * fact (3) == 4 * 3 * fact (2) == 4 * 3 * 2 * fact (1) = 4 * 3 * 2 * 1 * fact (0) = 4 * 3 * 2 * 1 * 1
```

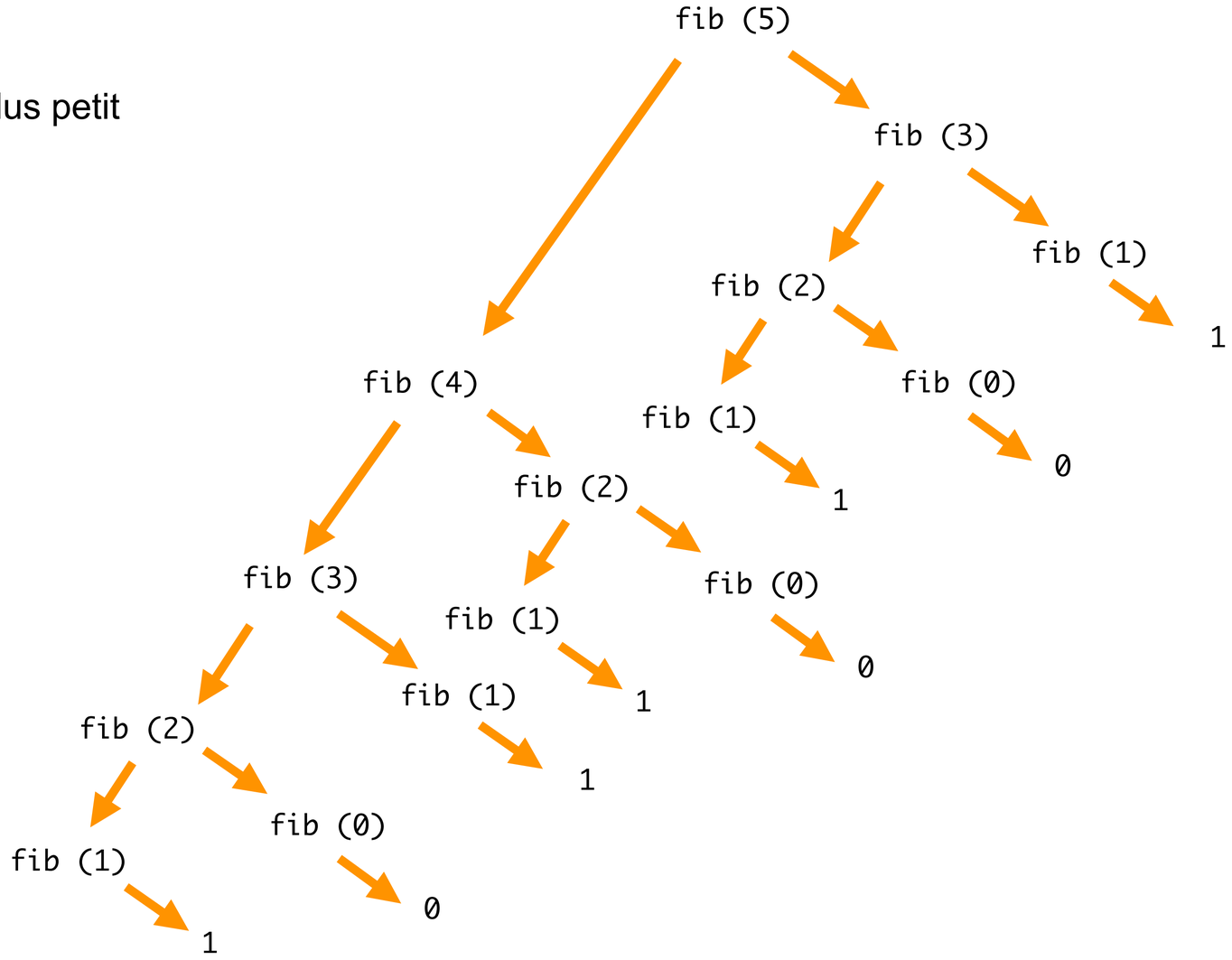
Fonctions récursives

- une fonction qui se rappelle avec un argument plus petit

```
def fib (n) :
    if n <= 1 :
      return n
    else :
      return fib (n-1) + fib (n-2)
```

fibonacci

```
>>> fib (10)
55
>>> fib (20)
6765
>>> fib (35)
9227465
```



- écrire une fonction qui calcule `fibonacci` plus rapidement

Fonctions récursives

- une fonction qui se rappelle avec un argument plus petit ?

```
def f (n) :
    if n > 100 :
        return n - 10
    else:
        return f(f(n+11))
```

- quelle est la valeur de cette fonction ?

```
def f (m, n) :
   if m == 0 :
     return 1
   else :
     f (m-1, f(m, n))
```

on calcule d'abord la valeur des arguments

appel par valeur

- quelle est la valeur de cette fonction ?

Fonctions récursives

- la fonction d'Ackermann croit très vite !

```
def A (m, n) :
    if m == 0 :
        return n + 1
    elif n == 0:
        return A (m-1, 1)
    else :
        return A (m-1, A (m, n-1))
```

Valeurs de $A(m, n)$

$m \backslash n$	0	1	2	3	4	n
0	1	2	3	4	5	$n + 1$
1	2	3	4	5	6	$n + 2$
2	3	5	7	9	11	$2n + 3$
3	5	13	29	61	125	$2^{n+3} - 3$
4	13	65533	$2^{65536} - 3$	$A(3, 2^{65536} - 3)$	$A(3, A(4, 3))$	$2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} - 3$ ($n + 3$ termes)
5	65533	$A(4, 65533)$	$A(4, A(5, 1))$	$A(4, A(5, 2))$	$A(4, A(5, 3))$	
6	$A(5, 1)$	$A(5, A(5, 1))$	$A(5, A(6, 1))$	$A(5, A(6, 2))$	$A(5, A(6, 3))$	

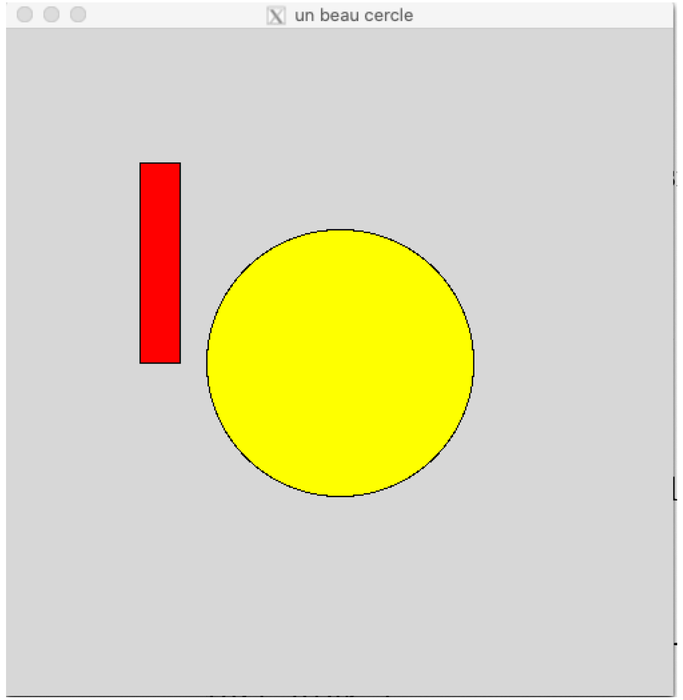
Graphique

- un paquetage `graphics.py` simple pour dessins 2D (a besoin d'installer le module `tkinter`)
  (cf. `http://mcsp.wartburg.edu/zelle/python/graphics`)

```
from graphics import *
```

* pour éviter de taper le préfixe `graphics.`

```
def dessin1 ():
    win = GraphWin("un beau cercle", 500, 500)
    c = Circle(Point(250,250), 100)
    c.setFill("yellow")
    c.draw(win)
    r = Rectangle(Point(100,100), Point(130, 250))
    r.setFill("red")
    r.draw(win)
    win.getMouse() # Pause to view result
    win.close()    # Close window when done
```

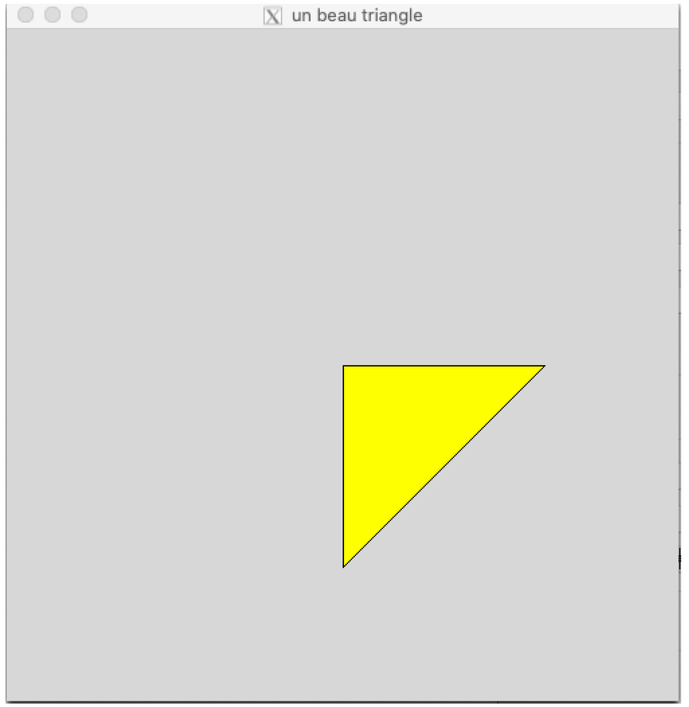


Graphique

- un paquetage `graphics.py` simple pour dessins 2D (a besoin d'installer le module `tkinter`)

  (cf. `http://mcsp.wartburg.edu/zelle/python/graphics`)

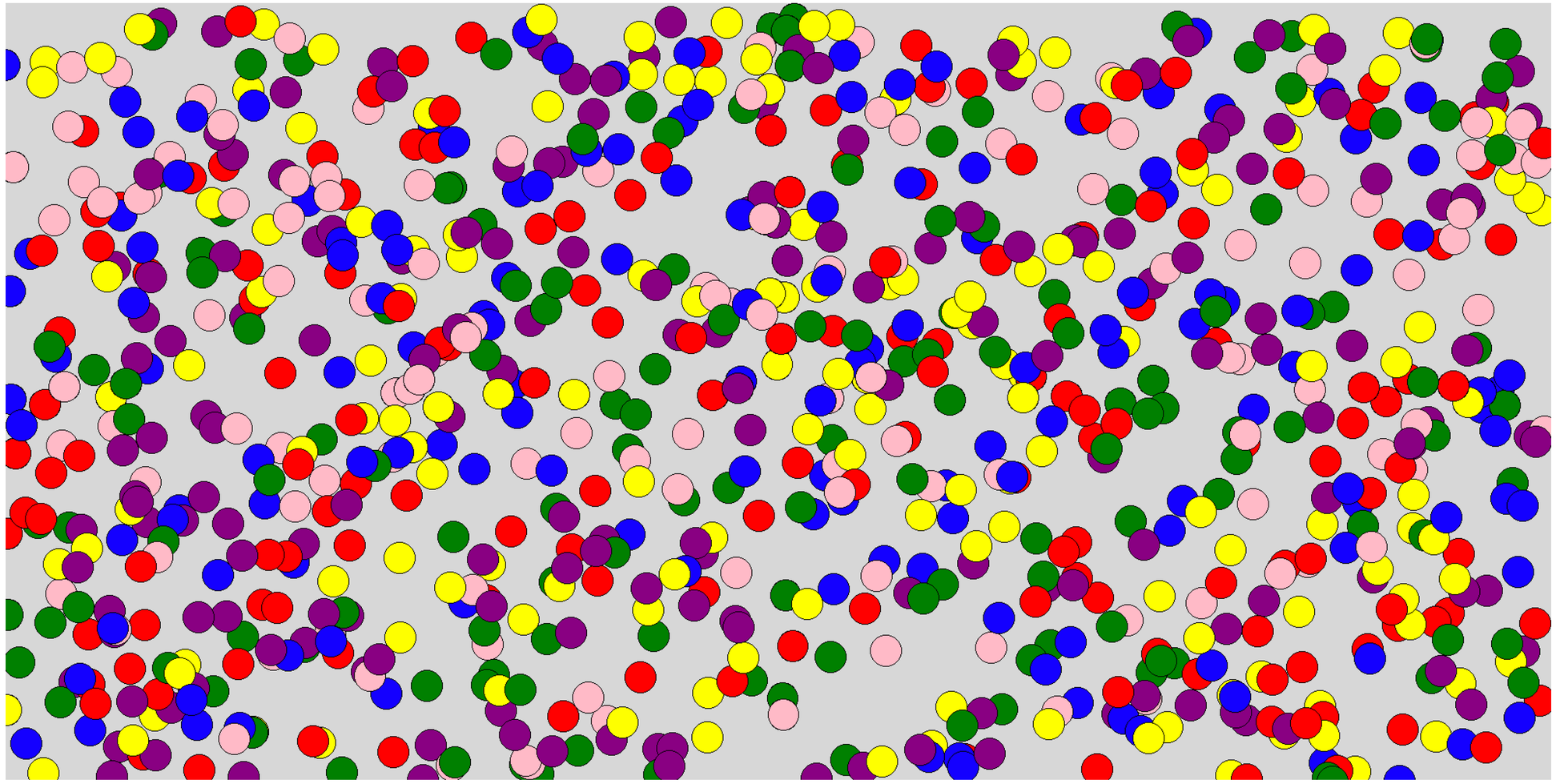
```
def dessin2 ():
    win = GraphWin("un beau triangle", 500, 500)
    p1 = Point (250,250)
    p2 = Point (400,250)
    p3 = Point (250,400)
    pol = Polygon(p1, p2, p3)
    pol.setFill("yellow")
    pol.draw(win)
    win.getMouse() # Pause to view result
    win.close()    # Close window when done
```



Graphique

- un paquetage `graphics.py` simple pour dessins 2D (a besoin d'installer le module `tkinter`)

  (cf. `http://mcsp.wartburg.edu/zelle/python/graphics`)



Graphique

- un paquetage `graphics.py` simple pour dessins 2D (a besoin d'installer le module `tkinter`)

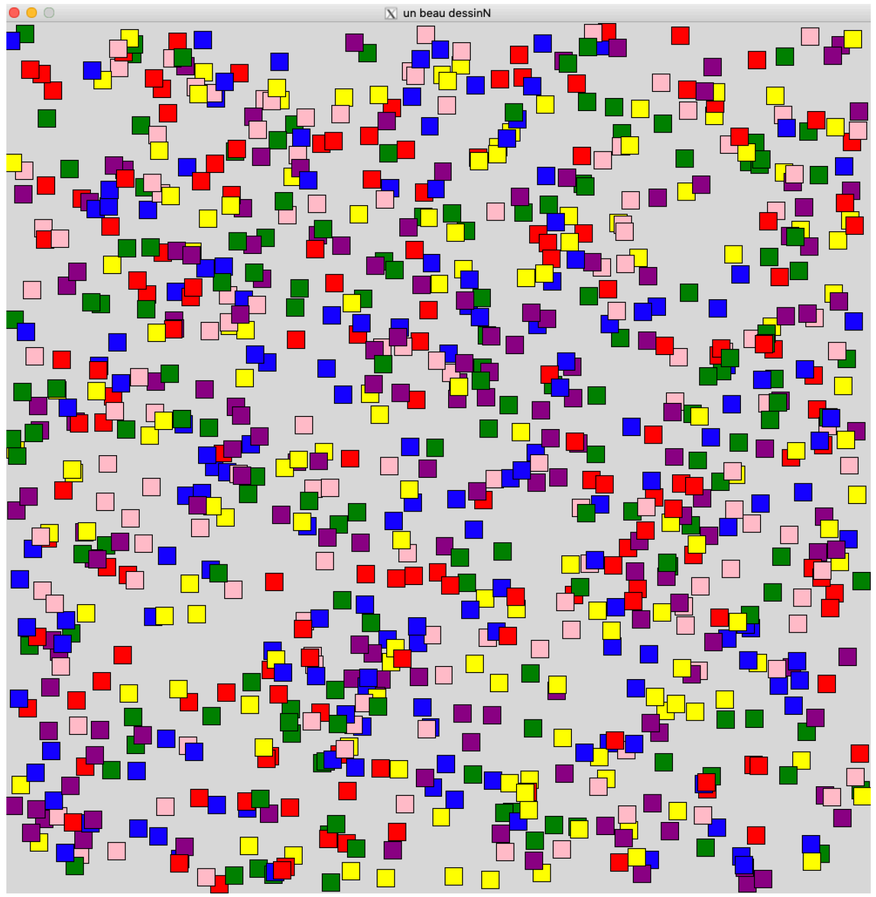
  (cf. `http://mcsp.wartburg.edu/zelle/python/graphics`)

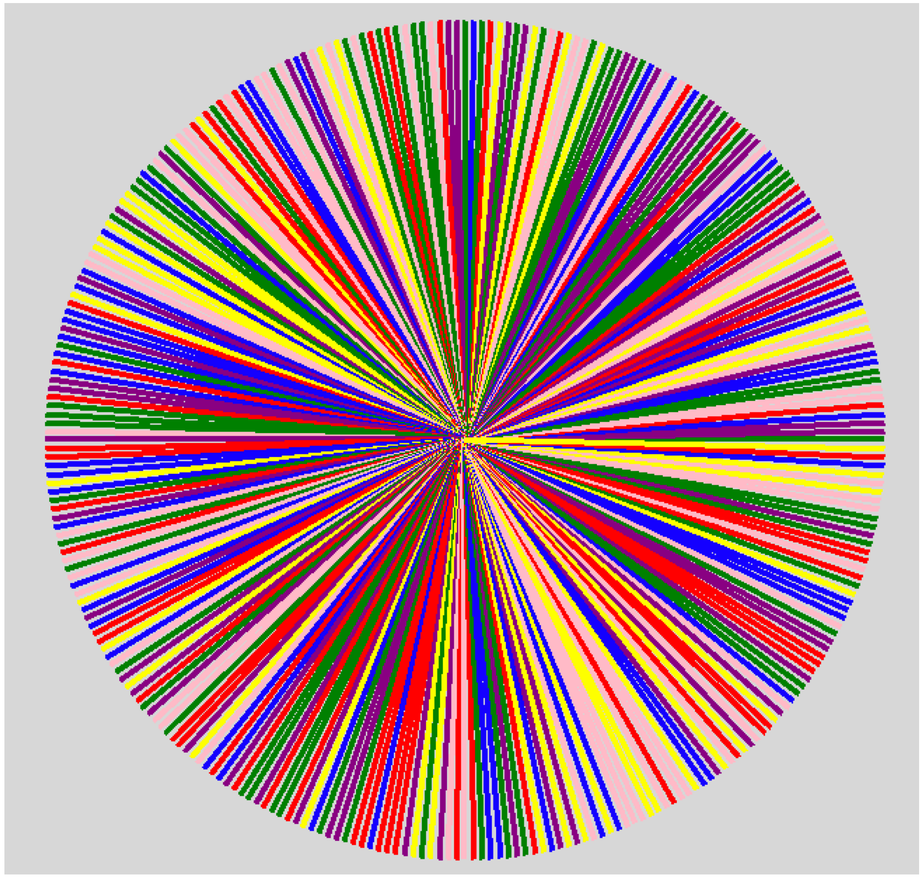
```
def dessinM ():
    winx = 2000; winy = 1000
    win = GraphWin("un beau dessinM", winx, winy, autoflush=False)
    win.setCoords (0, 0, winx, winy)
    cols = ("red", "yellow", "green", "blue", "purple", "pink")
    n = 600
    a = random.sample(range(winx-20),n)
    b = random.sample(range(winy-20),n)
    for i in range(n):
        c = Circle(Point(a[i], b[i]), 20)
        c.setFill (random.choice(cols))
        c.draw(win)
    win.update()
    win.getMouse() # Pause to view result
    win.close()    # Close window when done   r.draw()
```

Graphique

- un paquetage `graphics.py` simple pour dessins 2D (a besoin d'installer le module `tkinter`)

  (cf. `http://mcsp.wartburg.edu/zelle/python/graphics`)





Graphique

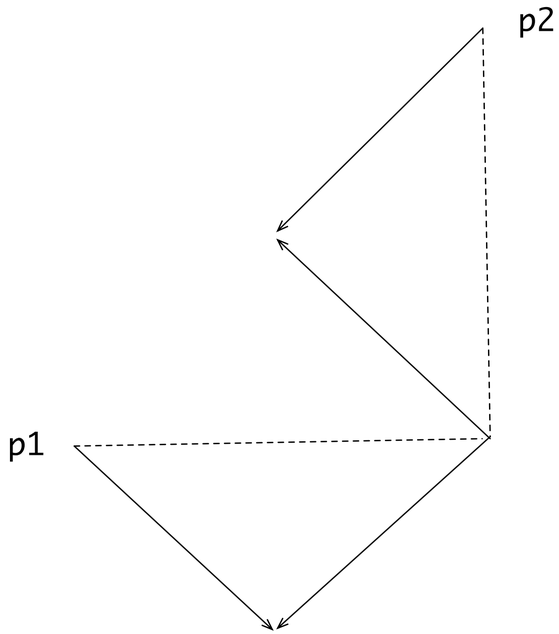
- un paquetage `graphics.py` simple pour dessins 2D (a besoin d'installer le module `tkinter`)

  (cf. `http://mcsp.wartburg.edu/zelle/python/graphics`)

```
def dessinQ (n, rho):
    winx = 2000; winy = 1000
    win = GraphWin("un beau cercle", winx, winy, autoflush=False)
    win.setCoords (0, 0, winx, winy)
    cols = ("red", "yellow", "green", "blue", "purple", "pink")
    PI = math.pi
    theta = 2*PI / n
    center = Point(winx // 2, winy // 2)
    for i in range(n):
      p = Point (center.x + rho*math.cos (i * theta), center.y + rho*math.sin (i * theta))
      l = Line(center, p)
      l.setWidth(4)
      l.setOutline (random.choice(cols))
      l.draw(win)
    win.update()
    win.getMouse() # Pause to view result
    win.close()    # Close window when done   r.draw()
```

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)
```



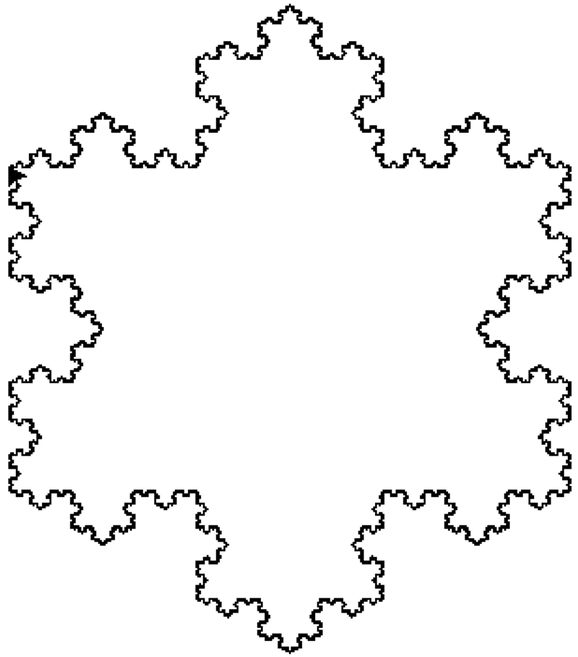
- on plie une feuille de papier n fois

Graphique avec la tortue

- un paquetage `turtle.py` simple pour apprendre l'informatique dans les écoles

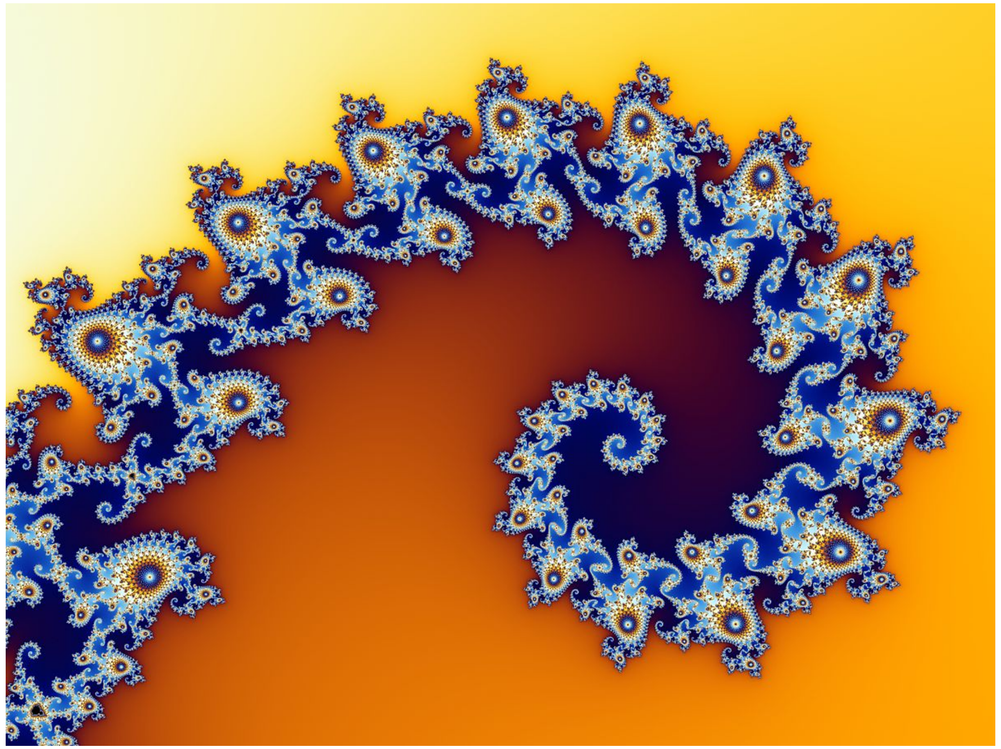
  (cf. http://fr.wikipedia.org/wiki/Seymour_Papert)

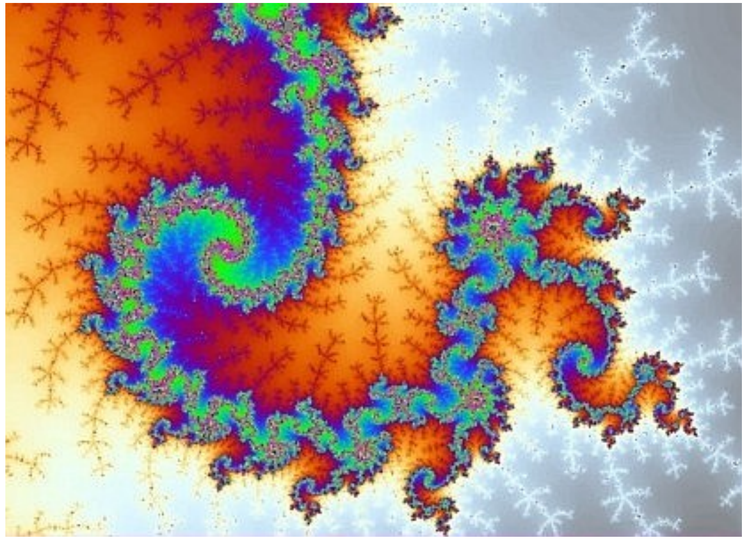
```
from turtle import *

def koch (l, n) :
    if n<=0 :
        forward (l)
    else:
        koch (l/3, n-1); left(60)
        koch (l/3, n-1); right (120)
        koch (l/3, n-1); left (60)
        koch (l/3, n-1)

def flocon (l, n) :
    koch (l, n); right (120)
    koch (l, n); right (120)
    koch (l, n)
```



Fractales

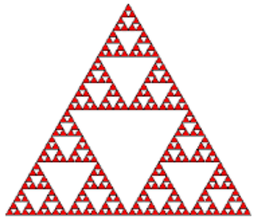
- les fractales de Benoît Mandelbrot

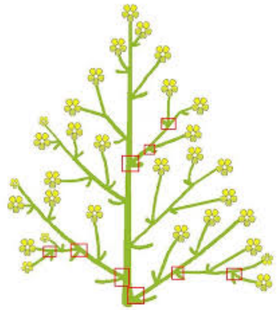


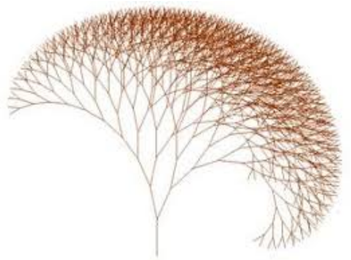


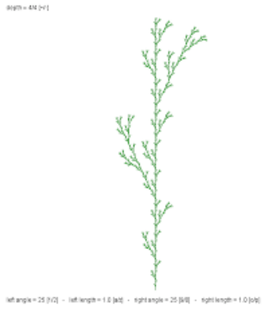
Fractales

- les fractales de Benoît Mandelbrot









Fractales

- la courbe de Hilbert

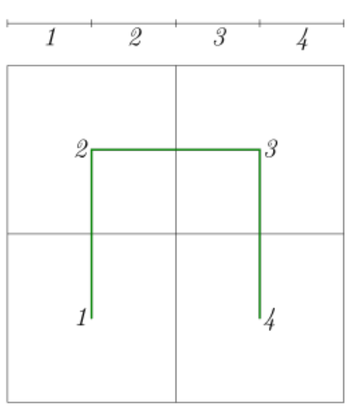


Fig. 1.

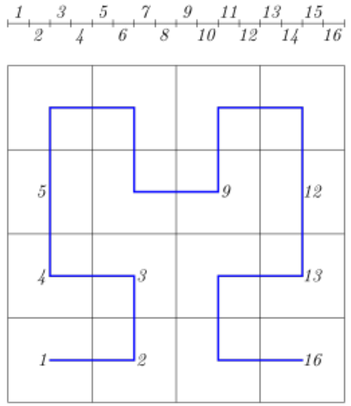


Fig. 2.

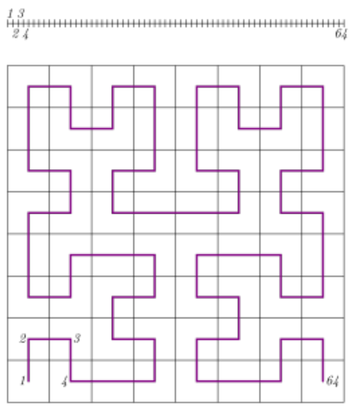
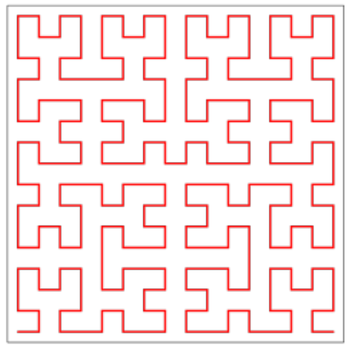
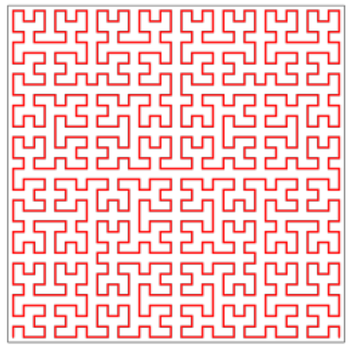
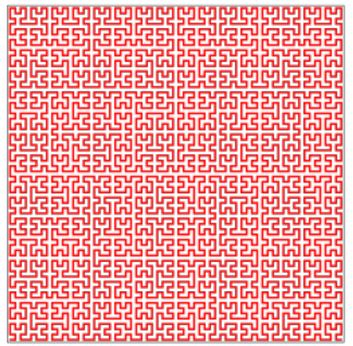


Fig. 3.







Exercice : écrire le programme qui la dessine

Récursivité et raisonnement inductif

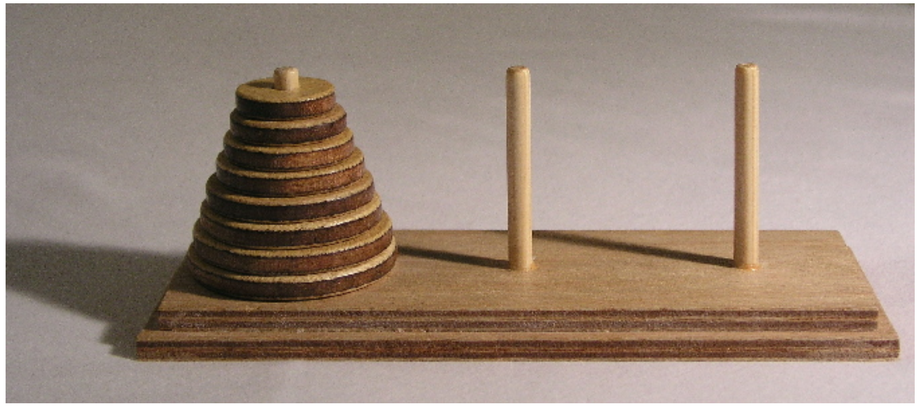
- le programme des tours de Hanoi

- supposons le problème résolu pour n-1 rondelles entre i et j

- j'amène les n-1 rondelles sur le pilier du milieu `i -> 6 - i - j`

- amener la grosse rondelle vers le pilier de droite

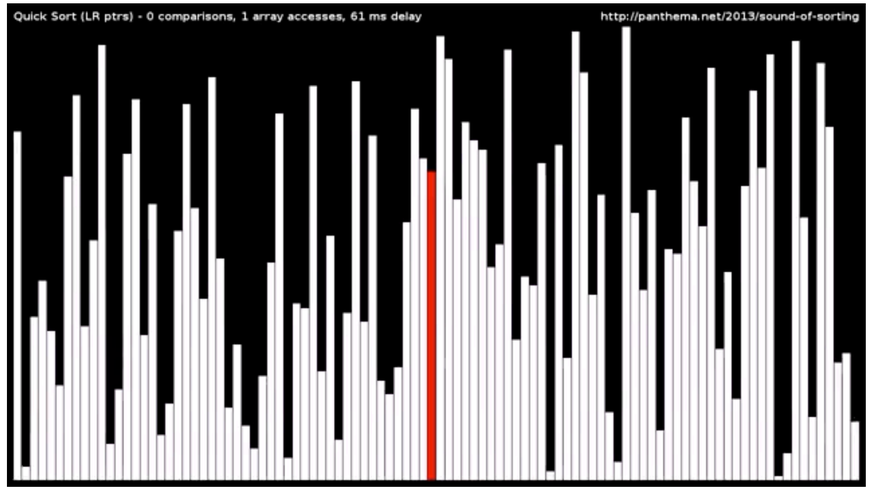
- ramener les n-1 rondelles du milieur vers la droite



```
def hanoi (n, i, j) :
    if n > 0 :
       hanoi (n-1, i, 6 - i -j)
       print ("%d --> %d" %(i, j))
       hanoi (n-1, 6 - i - j, j)
```

Tri rapide — Tri fusion

- tri et récursivité: quicksort



- tri et récursivité: mergesort