Informatique et Programmation

Cours 5

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

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

Plan

- exercices du cours 4
- 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`

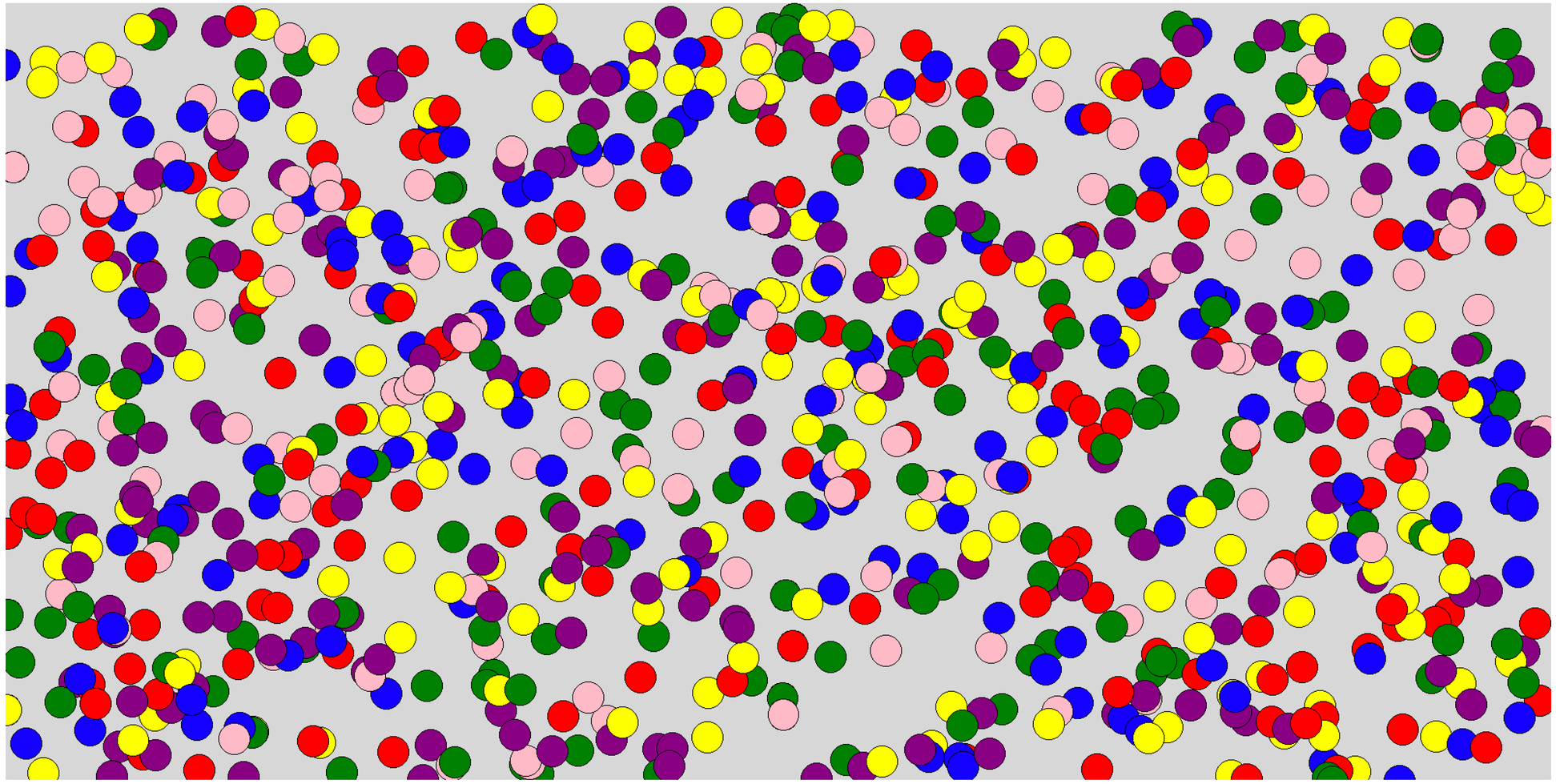
un bon tutoriel en `http://www.programiz.com/python-programming`

Python à distance en en `http://pythonAnywhere.com`

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()
```

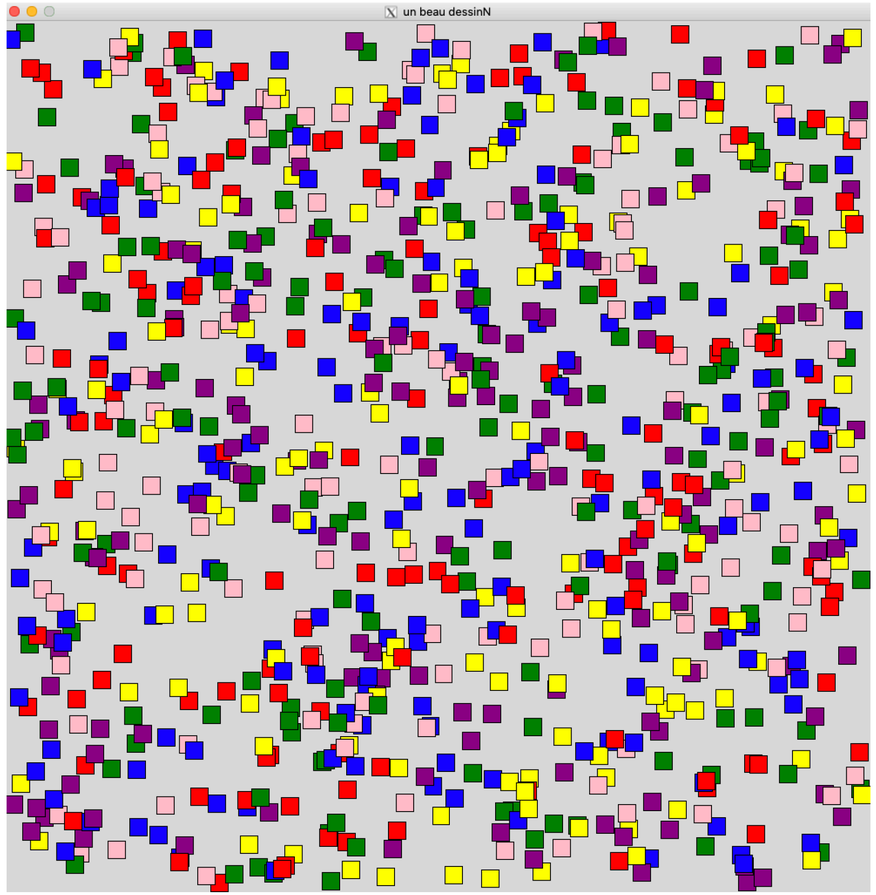
les sources sont en `http://jeanjacqueslevy.net/prog-py-22/progs/c5a.py`
et le module graphics en `http://jeanjacqueslevy.net/prog-py-22/progs/graphics.py`

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 dessinR (n, largeur):
    winx = 1000; winy = 1000
    win = GraphWin("un beau dessinM", winx, winy, autoflush=False)
    win.setCoords (0, 0, winx, winy)
    cols = ("red", "yellow", "green", "blue", "purple", "pink")
    a = random.sample(range(winx-largeur),n)
    b = random.sample(range(winy-largeur),n)
    for i in range(n):
        p1 = Point(a[i], b[i])
        p2 = Point(a[i]+largeur, b[i]+largeur)
        r = Rectangle (p1, p2)
        r.setFill (random.choice(cols))
        r.draw(win)
    win.update()
    win.getMouse() # Pause to view result
    win.close()    # Close window when done   r.draw()
```



Fonctions récursives

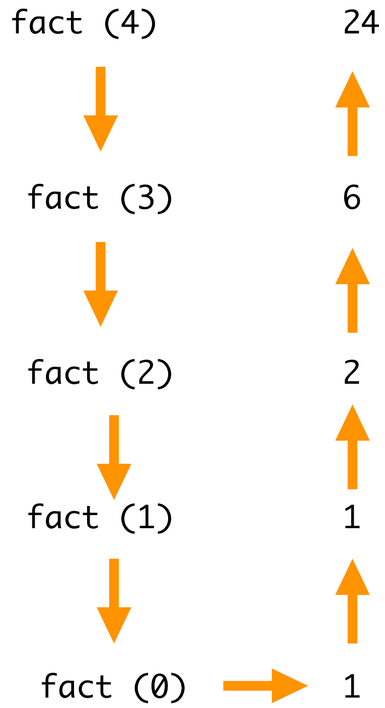
- une fonction qui se rappelle avec un argument plus petit

```
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
```

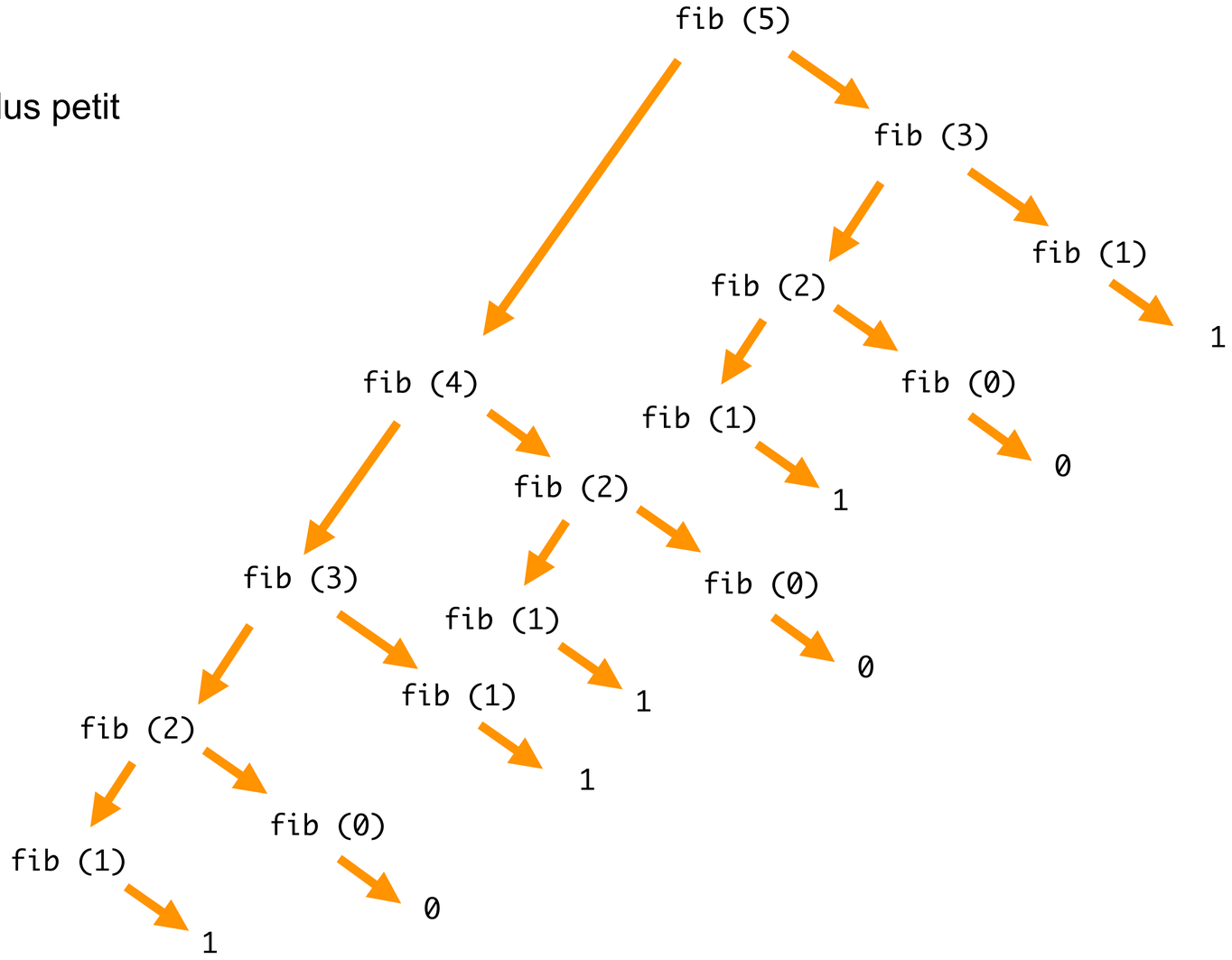
Fonctions récursives

- une fonction qui se rappelle avec un argument plus petit

```
def fib (n) :
    if n == 0 or 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 !

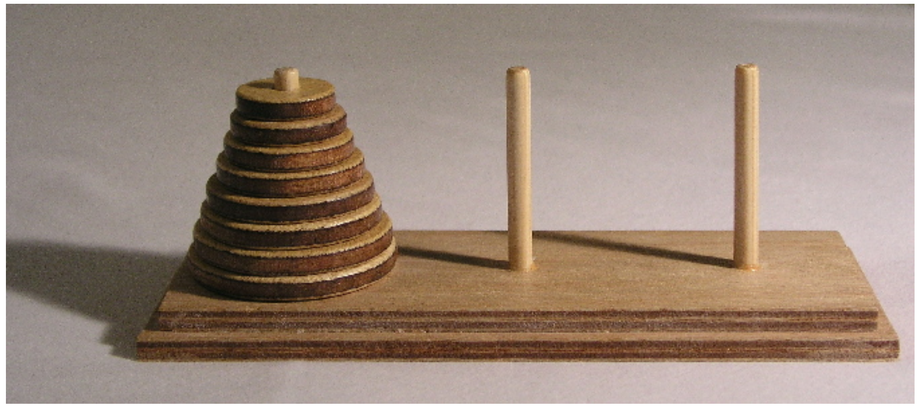
```
def A (m, n) :
    if m == 0 :
        return n + 1
    elif n == 0:
        return A (m-1, 1)
    else :
        return A (m-1, A (m, n-1))
```

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	$A(5, 1)$	$A(5, A(5, 1))$	$A(5, A(6, 1))$	$A(5, A(6, 2))$	$A(5, A(6, 3))$	

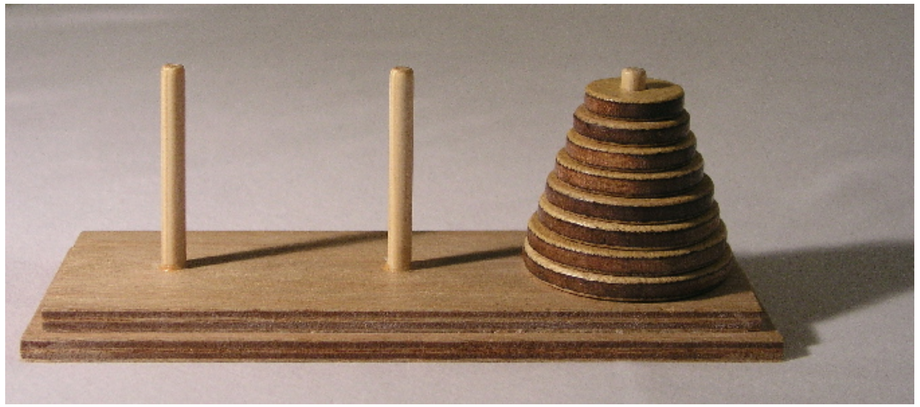
Les tours de Hanoi

- on a 3 piles et n rondelles sur la pile 1
- jamais une rondelle grosse au-dessus d'une rondelle petite



pile 1 pile 2 pile 3

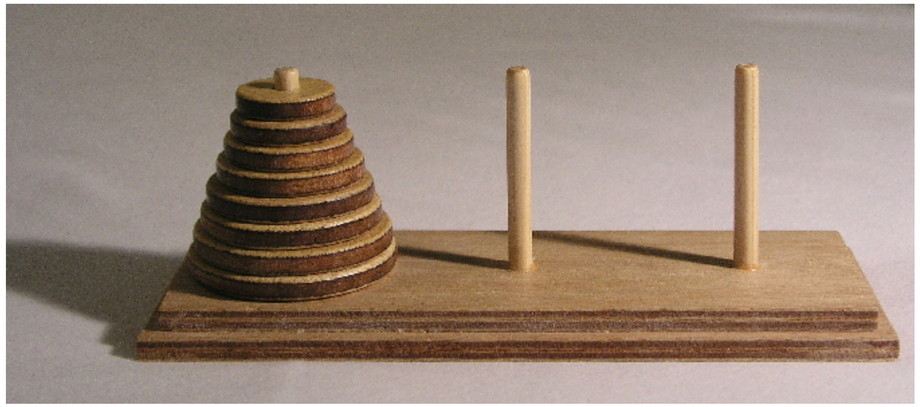
- il faut amener les n rondelles sur la pile 3
- on ne déplace qu'une seulle rondelle à la fois
- et on ne met jamais une rondelle au-dessus d'une plus petite



pile 1 pile 2 pile 3

Les tours de Hanoi

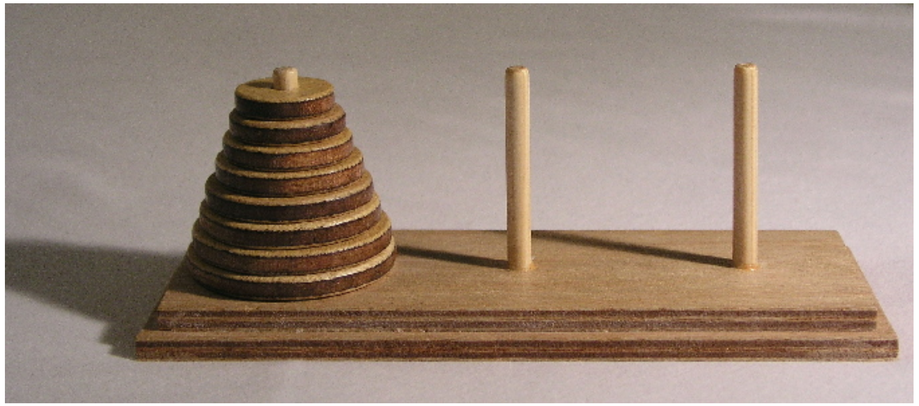
- on a 3 piles et n rondelles sur la pile 1
- jamais une rondelle grosse au-dessus d'une rondelle petite



pile 1 **pile 2** **pile 3**

- il faut amener les n rondelles sur la pile 3
- on ne déplace qu'une seule rondelle à la fois
- et on ne met jamais une rondelle au-dessus d'une plus petite

Les tours de Hanoi



pile i pile $6 - i - j$ pile j

- on généralise le problème pour aller de la pile i à la pile j
  où $1 \leq i \leq 3$ et $1 \leq j \leq 3$

  la troisième pile est alors $6 - i - j$

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

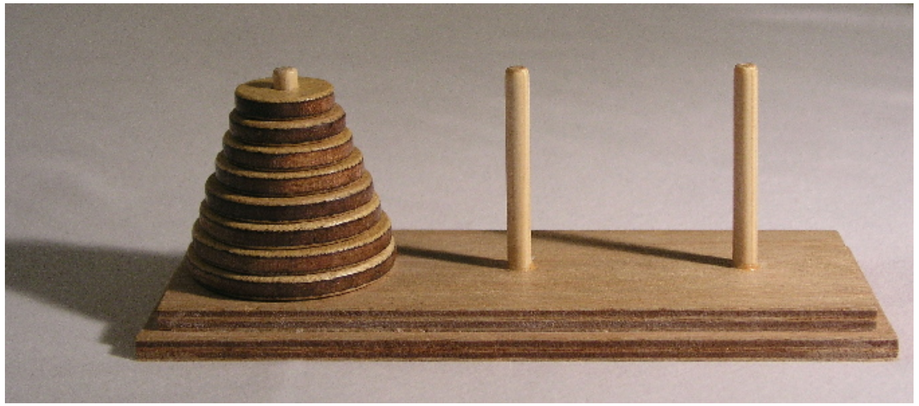
- j'amène les n-1 rondelles du dessus de la pile i sur la troisième pile
- j'amène la grosse rondelle de la pile i vers la pile j
- j'amène les n-1 rondelles de la troisième pile vers la pile j

```
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)
```

Les tours de Hanoi

```
>>> hanoi (2, 1, 3)
1 --> 2
1 --> 3
2 --> 3
>>> hanoi (3, 1, 3)
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
>>> hanoi (4, 1, 3)
1 --> 2
1 --> 3
2 --> 3
1 --> 2
3 --> 1
3 --> 2
1 --> 2
1 --> 3
2 --> 3
2 --> 1
3 --> 1
2 --> 3
1 --> 2
1 --> 3
2 --> 3
```

```
>>> hanoi (5, 1, 3)
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
1 --> 2
3 --> 2
3 --> 1
2 --> 1
3 --> 2
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
2 --> 1
3 --> 2
3 --> 1
2 --> 1
2 --> 3
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
```



pile i pile $6 - i - j$ pile j

- les tours de Hanoi sont un exemple du raisonnement inductif

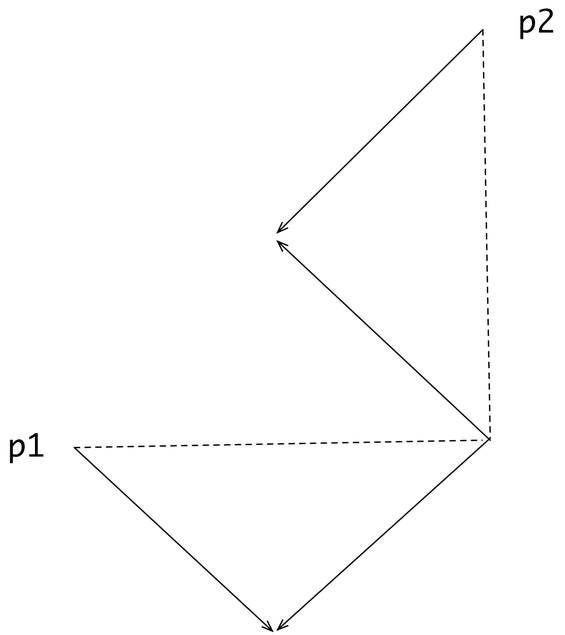
  (aussi appelé raisonnement par récurrence)

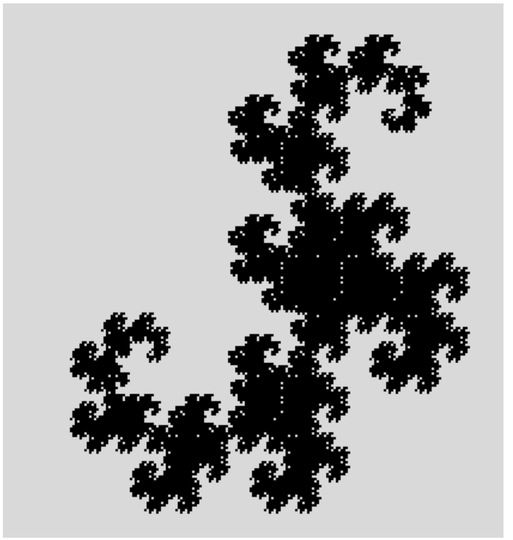
récursivité ⟷ raisonnement inductif

Courbe du dragon

- dessin récursif

```
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

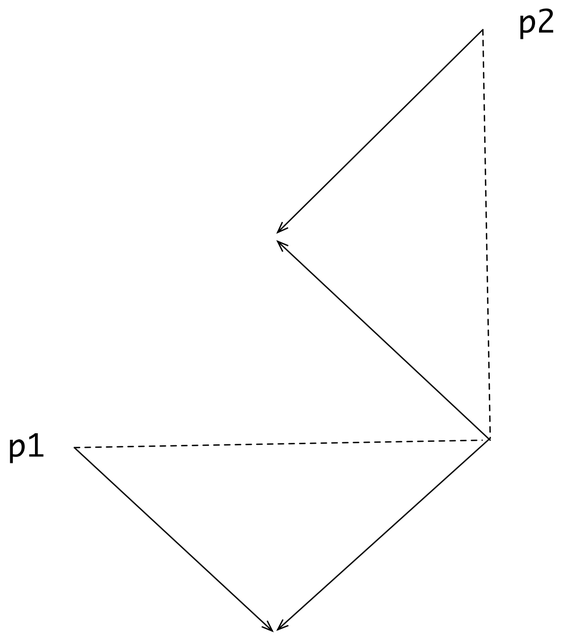
```
def dessinF (n, x, y, z, t):
    winx = 1000; winy = 1000
    win = GraphWin("un beau dessinM", winx, winy, autoflush=False)
    win.setCoords (0, 0, winx, winy)
    dragon (win, n, x, y, z, t)
    win.update()
    win.getMouse() # Pause to view result
    win.close()    # Close window when done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

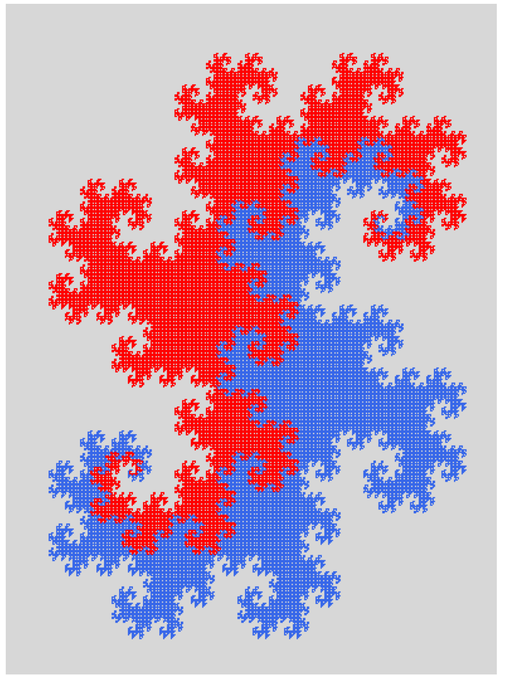
les sources sont en http://jeanjacqueslevy.net/prog-py-22/progs/c5a.py

Courbe du dragon

- dessin récursif

```
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)
```



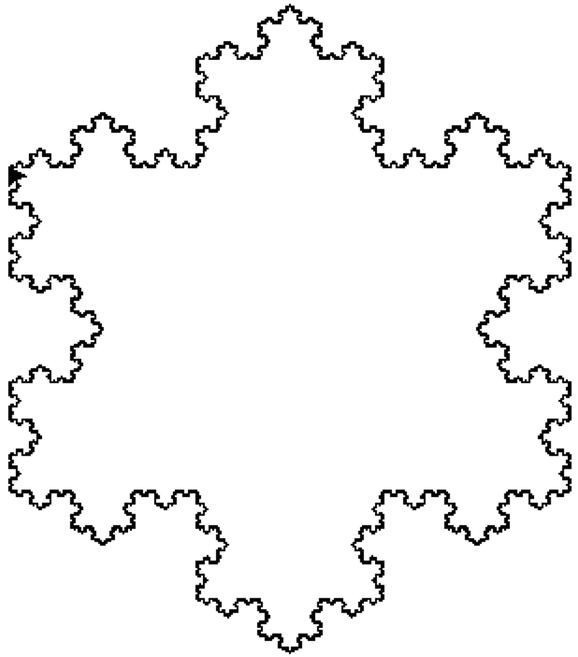


```
def dessinG (n, x, y, z, t) :
    winx = 1000; winy = 1000
    win = GraphWin("un beau dessinM", winx, winy, autoflush=False)
    win.setCoords (0, 0, winx, winy)
    dragonTC (win, n, x, y, z, t, 1, 'royal blue')
    dragonTC (win, n, x, y, z, t, -1, 'red')        # twin dragon
    win.update()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

Graphique avec la tortue

- un paquetage `turtle.py` simple pour apprendre l'informatique dans les écoles
  (cf. http://fr.wikipedia.org/wiki/Seymour_Papert)

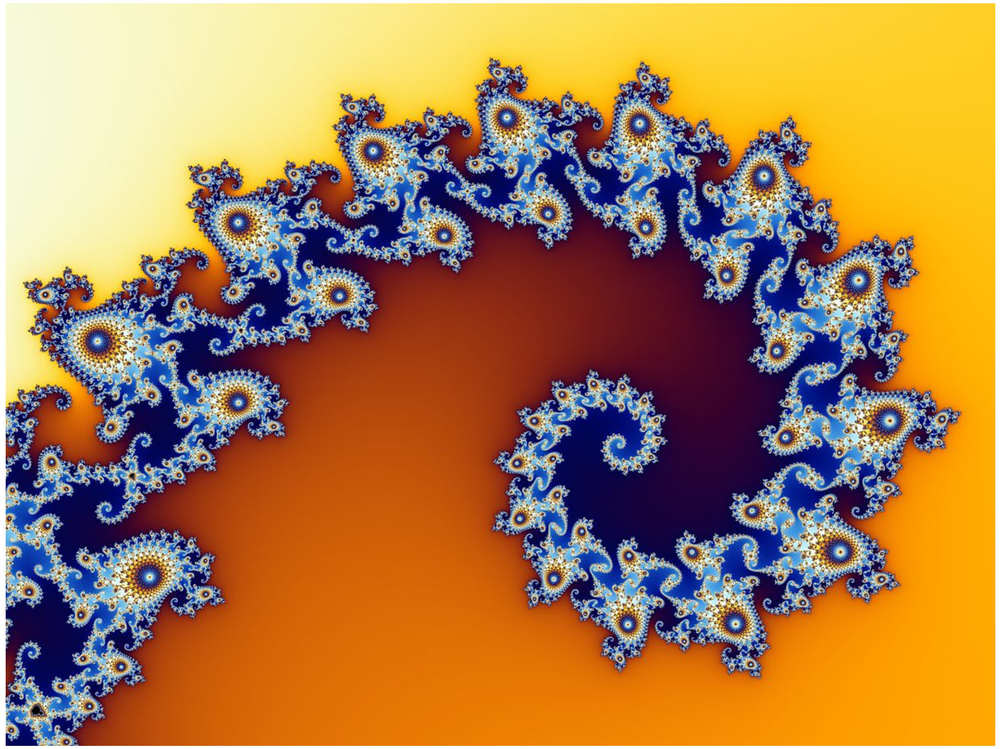
```
from turtle import *

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)
```

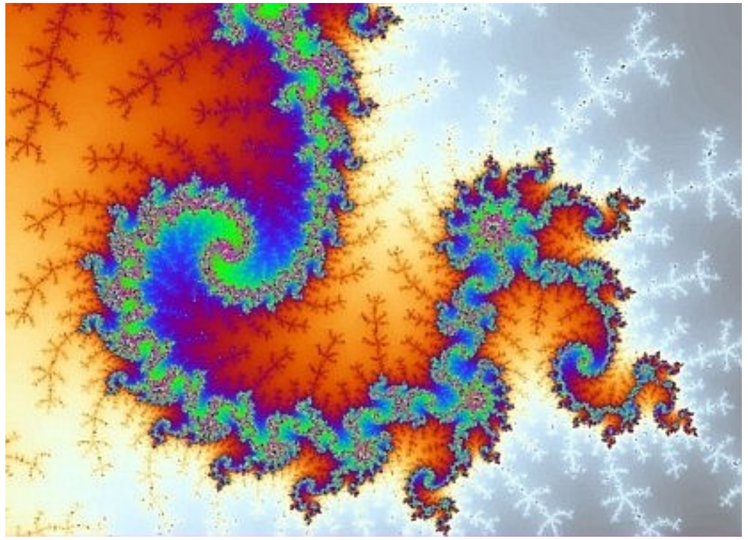


les sources sont en http://jeanjacqueslevy.net/prog-py-22/progs/c5a.py

Fractales

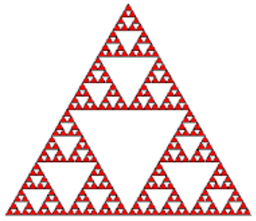
- les fractales de Benoît Mandelbrot

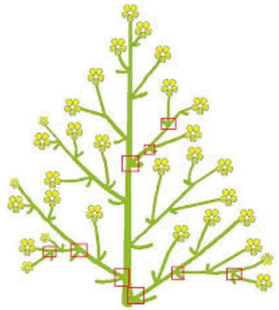


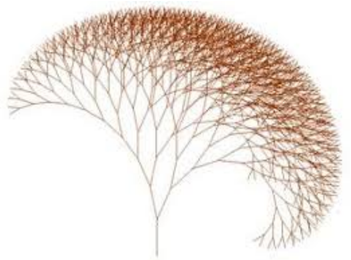


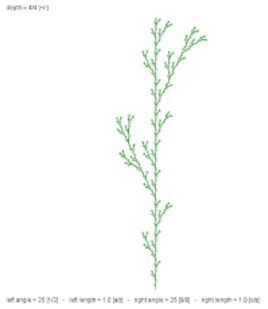
Fractales

- les fractales de Benoît Mandelbrot









Fractales

- la courbe de Hilbert

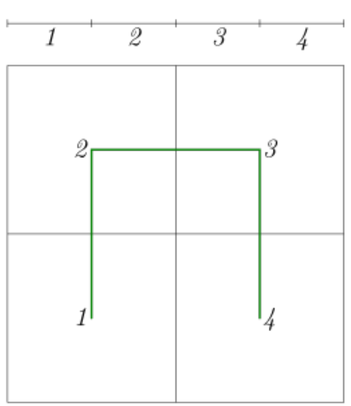


Fig. 1.

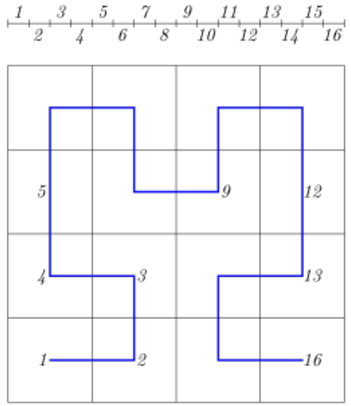


Fig. 2.

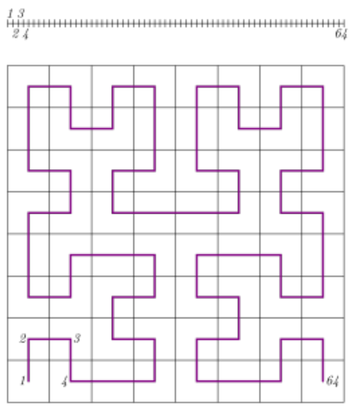
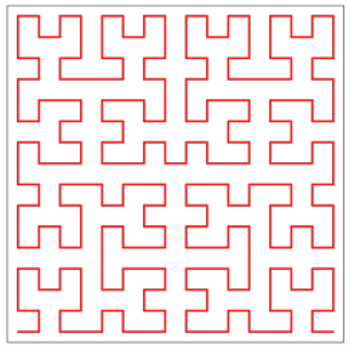
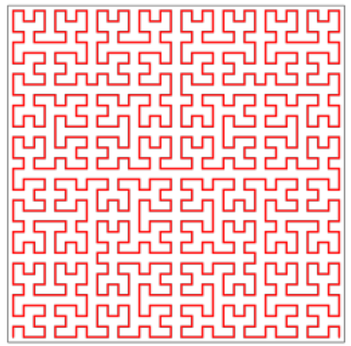
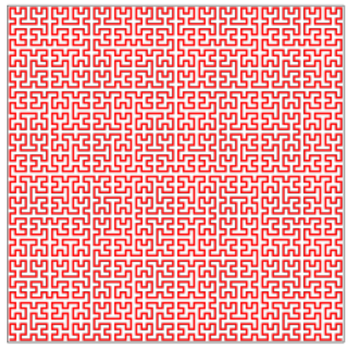


Fig. 3.



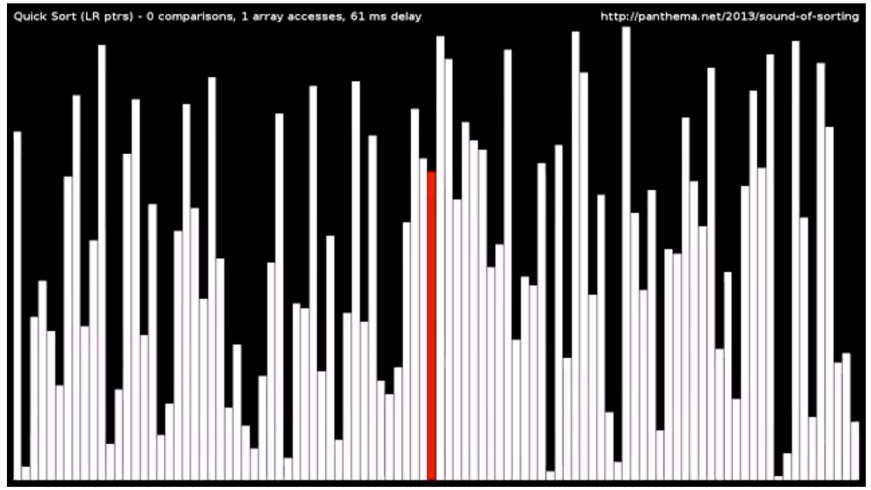




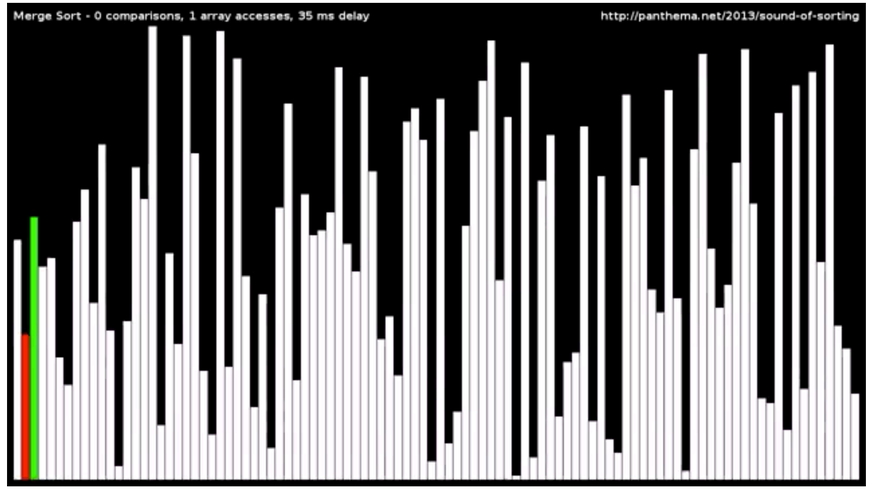
Exercice : écrire le programme qui la dessine

Tri rapide — Tri fusion

- tri et récursivité: quicksort



- tri et récursivité: mergesort



Conclusion

VU:

- récusivité
- raisonnement inductif
- fractales

TODO list

- tri récursif
- classes
- objets