Programmation et IA

Cours 3

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-ia-25

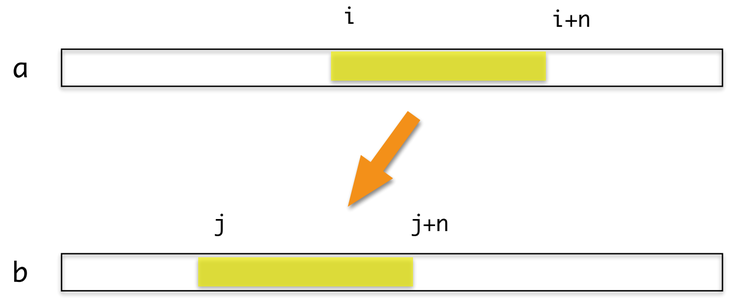
Plan

- récursivité
- réviser les classes et objets
- classes et objets
- structures de données (2)

Valeur d'un tableau — Alias

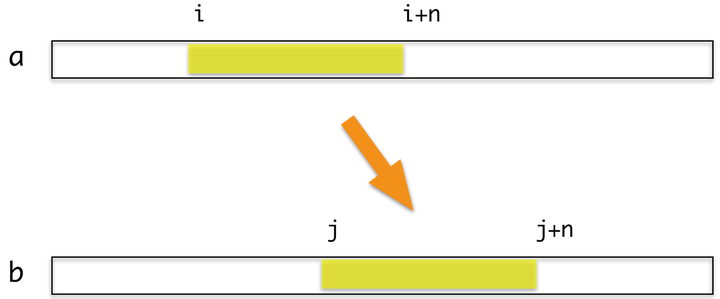
Exercice: le programme suivant est-il correct?

```
def copy (a, b, i, j, n) :
    for k in range (n) :
        b [j + k] = a [i + k]
```



Solution: Non! il faut écrire le programme suivant, correct, même quand les paramètres *a* et *b* sont des alias.

```
def copy (a, b, i, j, n) :
        if i > j :
            for k in range (n) :
                b [ j + k ] = a [i + k]
        else :
            for k in range (n-1, -1, -1) :
                b [ j + k ] = a [ i + k]
```



récursivité

Fonctions récursives

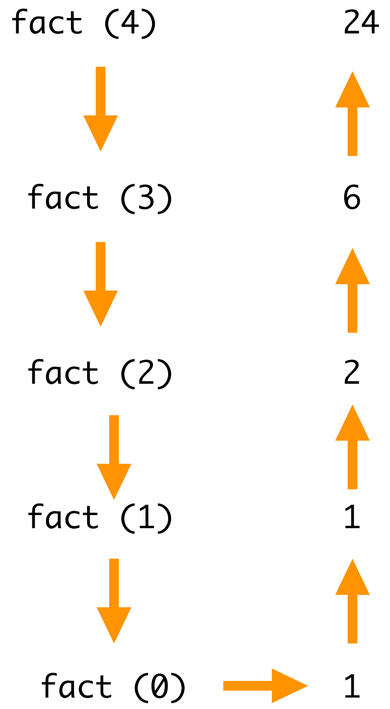
- une fonction qui se rappelle avec un argument plus petit

```
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
```

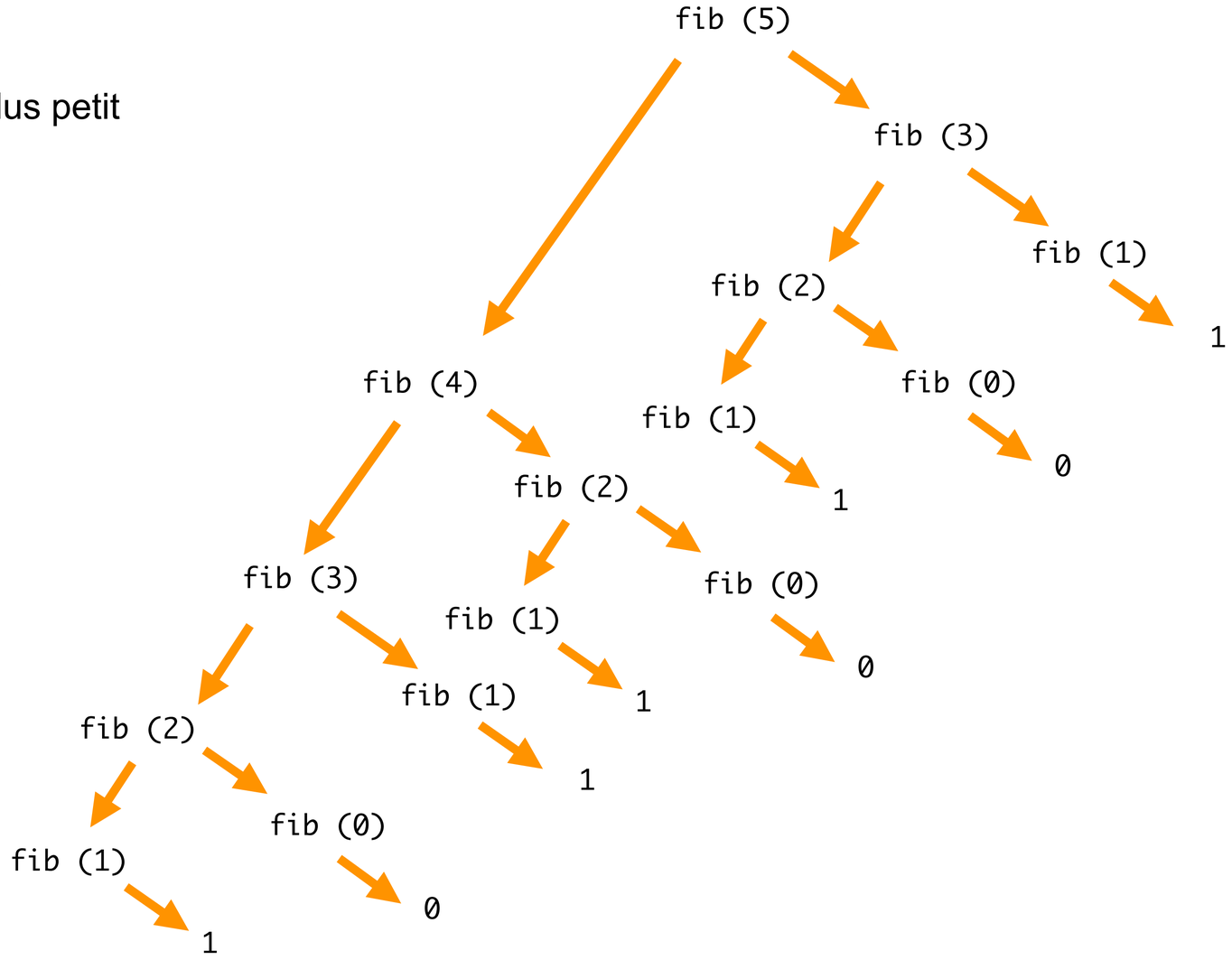
Fonctions récursives

- 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 ?

```
f (103) == 93
f (102) == 92
f (101) == 91
f (100) == f (f(111)) == f (101) == 91
f (99) == f (f(110)) == f (100) == .. 91
f (98) == f (f(109)) == f (99) == .. 91
f (97) == f (f(108)) == f (98) == .. 91
f (96) == f (f(107)) == f (97) == .. 91
...
f (91) == f (f(102)) == f(92) == .. 91
f (90) == f (f(101) == f (91) == .. 91
f (89) == f (f(100)) == .. f (91) == .. 91
f (88) == f (f(99)) == .. f (91) == .. 91
...
```

Fonctions récursives

- la fonction d'Ackermann croit très vite !

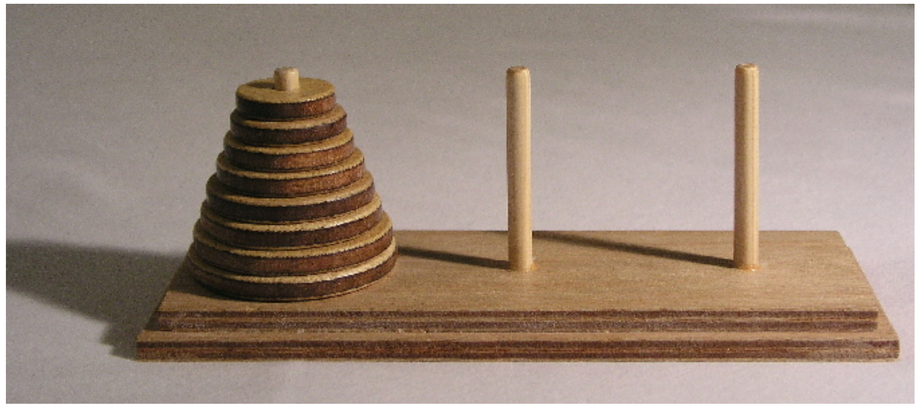
```
def A (m, n) :
    if m == 0 :
        return n + 1
    elif n == 0:
        return A (m-1, 1)
    else :
        return A (m-1, A (m, n-1))
```

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

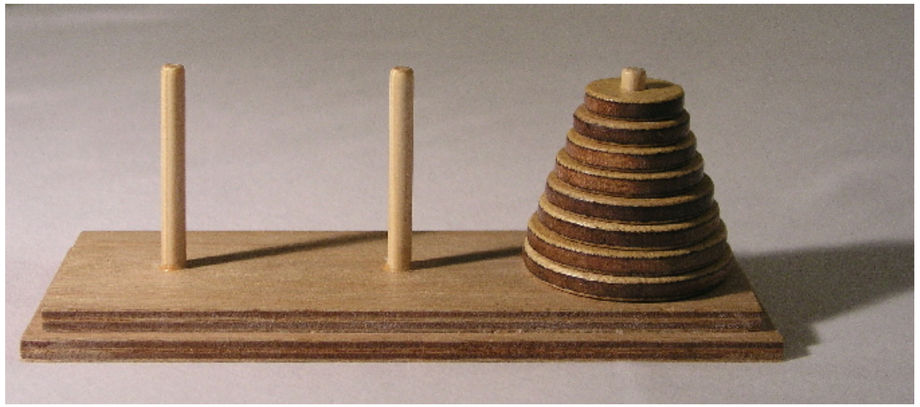
Les tours de Hanoi

- 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

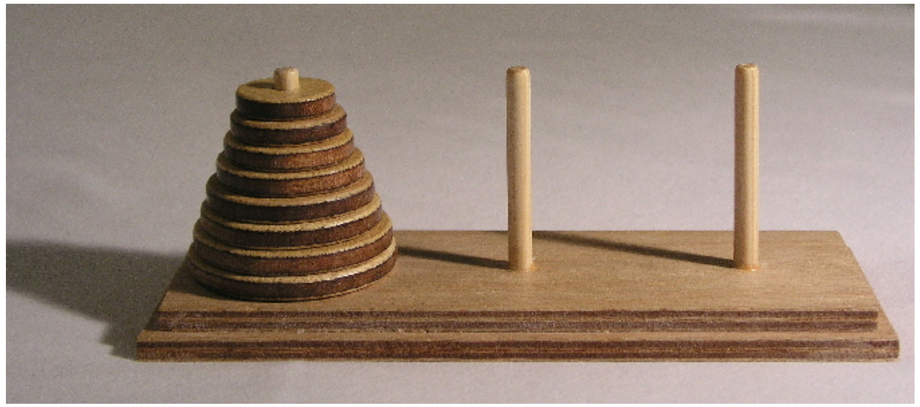
- il faut amener les n rondelles sur la pile 3
- on ne déplace qu'une seulle rondelle à la fois
- et on ne met jamais une rondelle au-dessus d'une plus petite



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



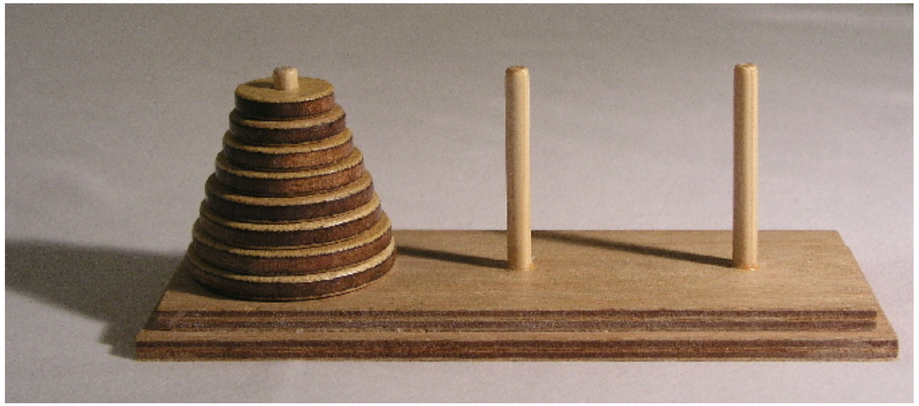
pile 1 **pile 2** **pile 3**

- 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

- 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 $6 - i - j$



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

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

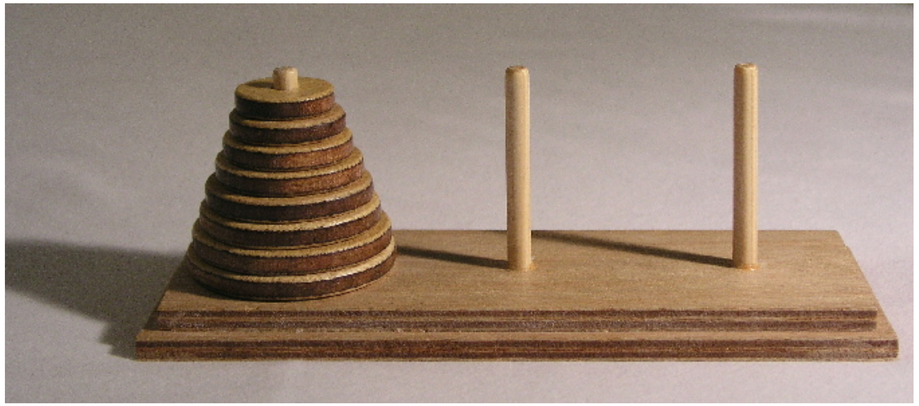
- j'amène les n-1 rondelles du dessus de la pile i sur la troisième pile
- j'amène la grosse rondelle de la pile i vers la pile j
- j'amène les n-1 rondelles de la troisième pile vers la pile j

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

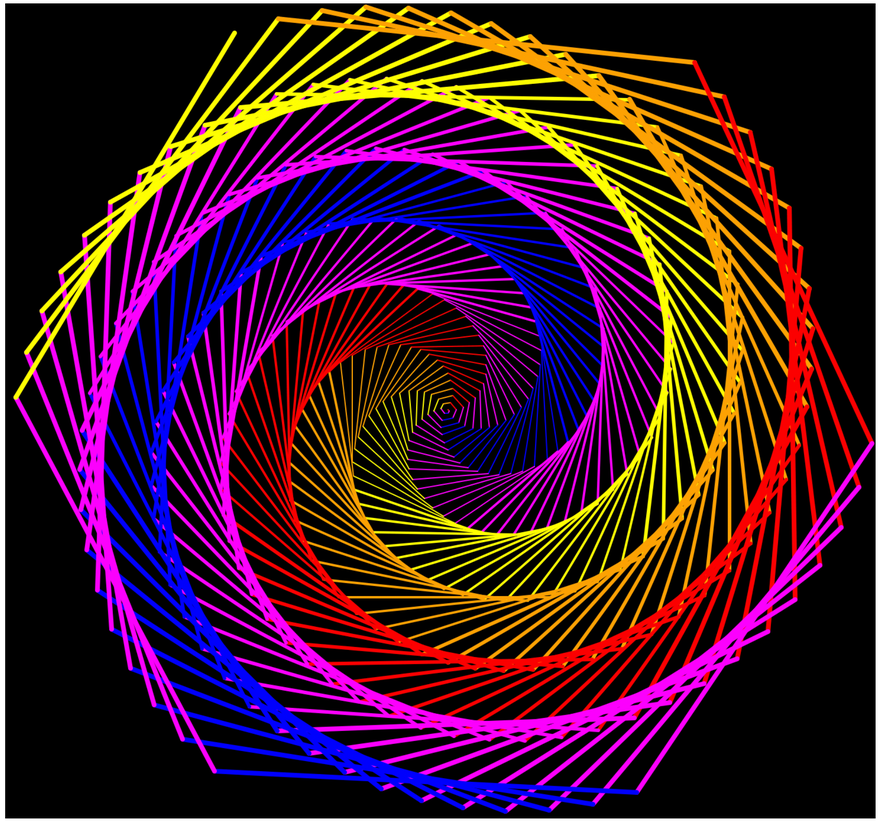
  (aussi appelé raisonnement par récurrence)

récursivité ⟷ raisonnement inductif

Graphique avec la tortue

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

```
import turtle as t

def spiralArt():
    colors = ['orange', 'red', 'magenta' , 'blue', 'magenta', \
              'yellow', 'green', 'cyan', 'purple']
    t.bgcolor('black')
    t.speed (0)
    for x in range (360):
        t.pencolor(colors [x % 6])
        t.pensize (x//100 + 1)
        t.forward (x)
        t.right (59)
    t.hideturtle()
    t.done()
```

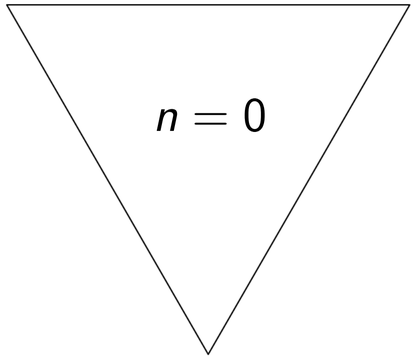


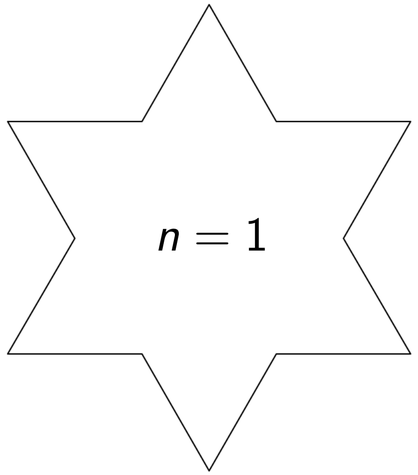
Graphique avec la tortue

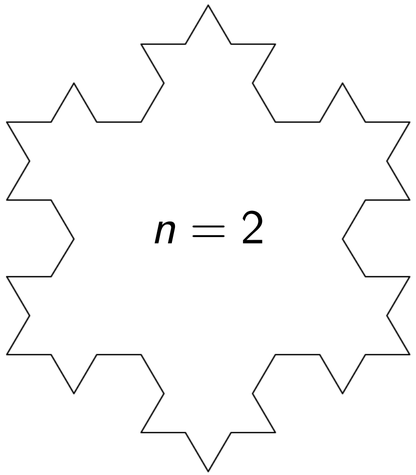
- le flocon de von Koch

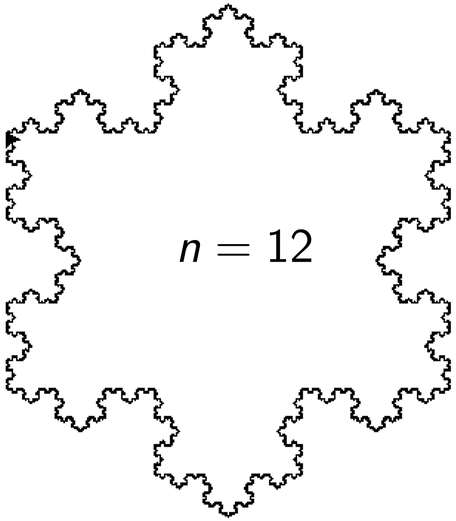
```
def koch (l, n) :
    if n <= 0 :
        t.forward (l)
    else:
        koch (l/3, n-1); t.left(60)
        koch (l/3, n-1); t.right (120)
        koch (l/3, n-1); t.left (60)
        koch (l/3, n-1)

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

```
def drawFlocon (length, order) :
    t.speed (100)
    flocon (length, order)
    t.hideturtle()
    t.done ()

drawFlocon (300, 4)
```







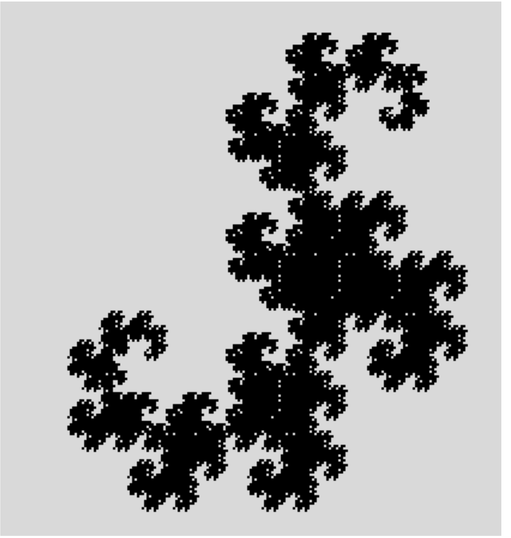


Graphique avec la tortue

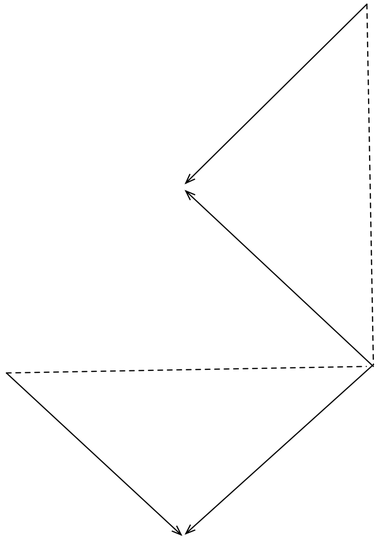
- la courbe du dragon

```
def dragon (n, l, s, color) :
    if n <= 1 :
       t.pencolor (color)
       t.forward (l)
    else :
       t.right (45*s)
       dragon (n-1, l/1.4142, 1, color)
       t.left (90*s)
       dragon (n-1, l/1.4142, -1, color)
       t.right (45*s)
```

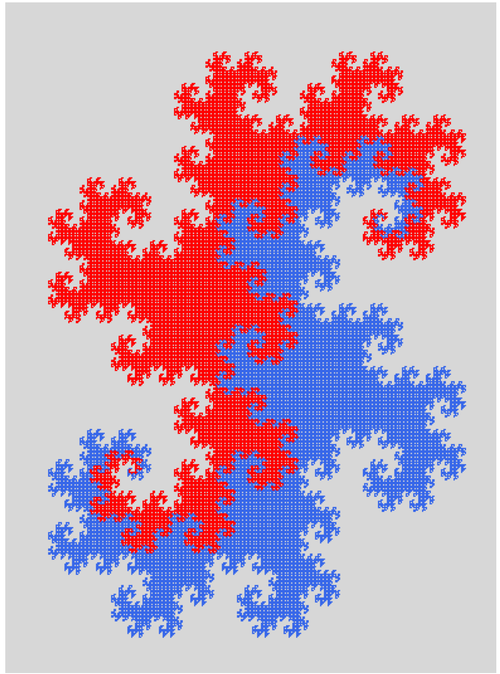
```
def drawDragon (order, length) :
    t.speed (100)
    dragon (order, length, 1, 'black')
    t.hideturtle()
    t.done ()

drawDragon (12, 200)
```



- on plie une feuille de papier n fois

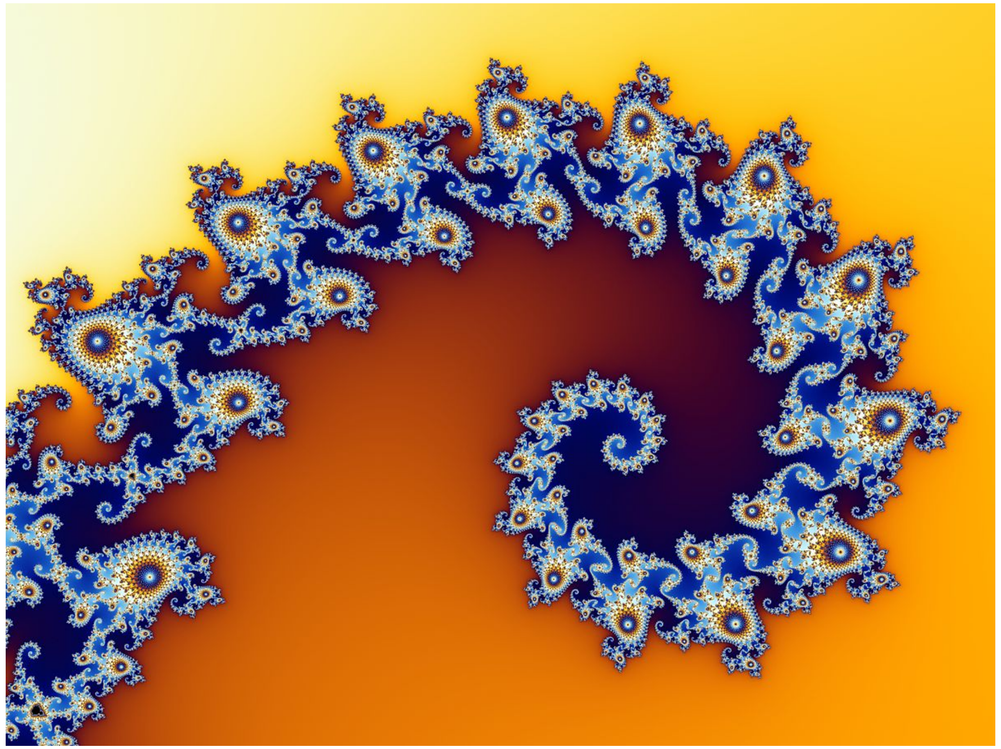


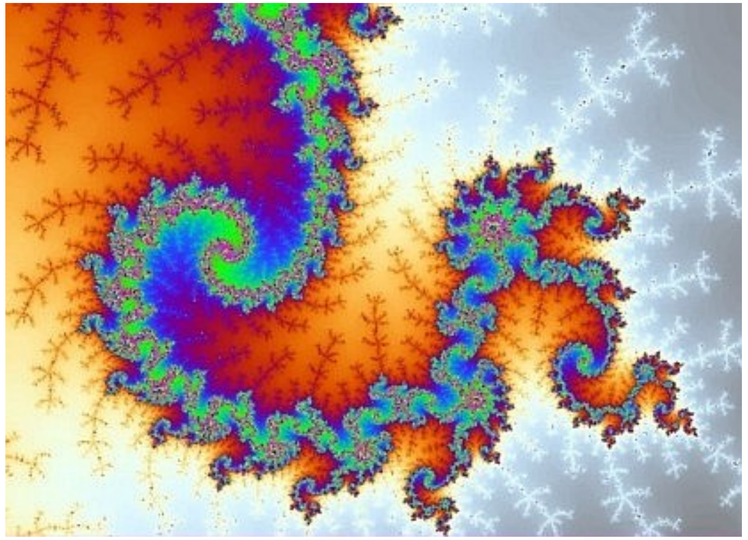
```
def drawTwinDragons (order, length) :
    t.speed (100)
    dragon (order, length, 1, 'blue')
    t.penup(); t.home (); t.pendown()
    dragon (order, length, -1, 'red')
    t.hideturtle()
    t.done ()

drawTwinDragons (13, 200)
```



Fractales

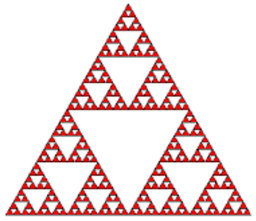
- les fractales de Benoît Mandelbrot

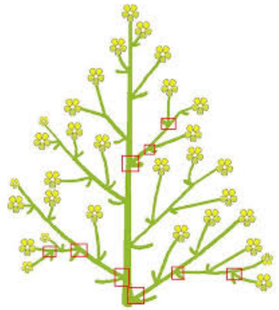


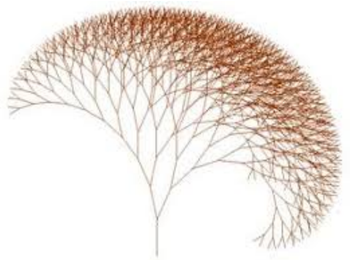


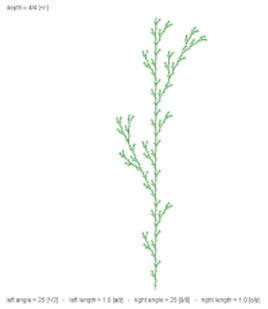
Fractales

- les fractales de Benoît Mandelbrot









les structures de données (2)

Classes et objets

• une classe décrit un ensemble d'objets tous de la même forme avec attributs et méthodes

```
class Point:
    def __init__ (self, x, y) :
        self.x = x
        self.y = y

    def __str__ (self) :
        return f'({self.x}, {self.y})'

    def __add__ (self, delta) :
        return Point (self.x + delta.x, self.y + delta.y)
```

constructeur d'un nouvel objet

`__str__` est appelé par `print`

`__add__` est appelé par `+`

• objets dans cette classe

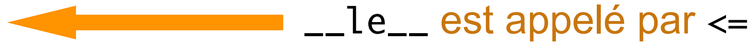
```
p1 = Point (10, 20)
print (p1)
(10, 20)

p2 = Point (40, 50)
print (p2)
(40, 50)

p3 = p1 + p2
print (p3)
(50, 70)
```

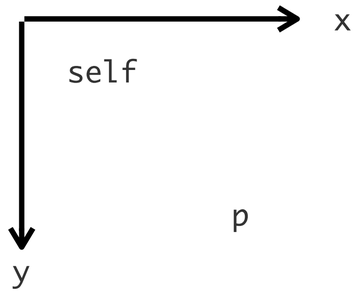
nouvel objet de la classe `Point`

Classes et objets

- les attributs d'un objet ont des valeurs quelconques (par exemple des références à d'autres objets)

```
class Point:
    # comme avant

    def __le__ (self, p) :
        return self.x <= p.x and self.y <= p.y
```





- le constructeur de la classe `Rectangle` utilise des objets `Point`

```
class Rectangle:
    def __init__ (self, p, q) :
        self.haut_gauche = p
        self.bas_droite = q
        if not p <= q :
            raise ValueError

    def __str__ (self) :
        return "Rectangle ({}, {})".format (self.haut_gauche, self.bas_droite)
```


```
r = Rectangle (p1, p3)
Rectangle (10, 20), (40, 50)
print (r)
```

Classes et héritage

- une classe peut être une sous-classe d'une classe plus générale

```
class Carre (Rectangle) :
    def __init__ (self, p, c) :
        super().__init__ (p, p + Point(c, c))
```

on appelle le constructeur de `Rectangle`

- un carré est un rectangle particulier

- le constructeur de `Carre` appelle le constructeur de la super classe `Rectangle`

Exercice écrire la classe `Polygone` qui construit des objets à partir d'une liste de `Point`

Exercice écrire la classe `Triangle`

Exercice écrire les méthodes `perimetre` et `surface`

Classes et objets

- les classes permettent d'**encapsuler** un ensemble de données (attributs) et de fonctions (méthodes)
- les classes représentent donc une forme de **modularité**
- à l'extérieur, le détail de l'implémentation des objets de cette classe est **opaque**
- on peut donc modifier la représentation sans changer leur interface et les fonctions qui utilisent cette classe
- **attention**: classes et modules sont deux notions différentes en Python
  [les modules sont importés et attachés à la notion de fichier]

les sources sont en `http://jeanjacqueslevy.net/prog-py-22/progs/c7a.py`

Prochain cours

- réviser les classes et objets
- arbres
- recherche en table
- arbres binaires de recherche
- arbres équilibrés