Informatique et Programmation

Appendice 2

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

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

Plan

- correction des devoirs maisons [dm1 et dm3]
- listes (tableaux), ensembles, dictionnaires
- notation par compréhension
- classes et objets
- classes et héritage
- un autre paquetage graphique

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

un cours Python en `http://www.w3schools.com/python/default.asp`

Valeur d'un tableau — Alias

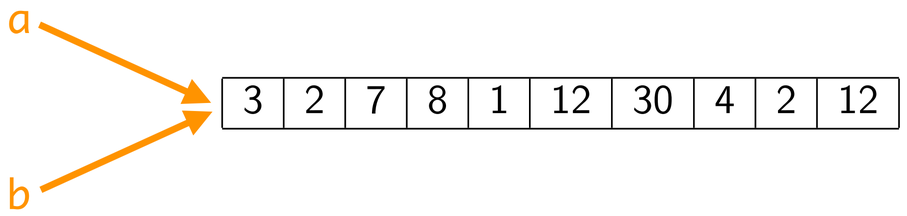
- soient 2 tableaux a et b

```
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]
```



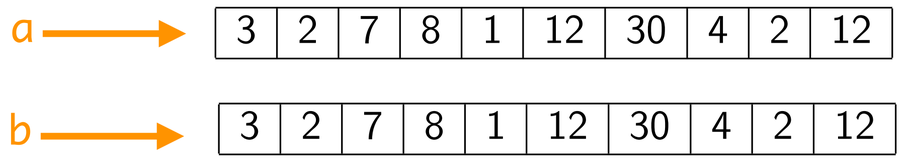
- les variables a et b sont 2 alias d'un même tableau

```
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

- la valeur de a ou de b est l'adresse mémoire de son premier élément

données modifiables et alias sont sources de bugs

Valeur d'un tableau — Alias

- égalité et identité sont deux notions différentes

```
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]

print (a is b)
True

print (a == b)
True
```

identitié

égalité

```
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, 7, 8, 1, 12, 30, 4, 2, 12]

print (b)
[3, 2, 7, 8, 1, 12, 30, 4, 2, 12]

print (a is b)
False

print (a == b)
True
```

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

[random.randint (0, 3) for i in range(20)]

```
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}
```

Exercice Que fait l'expression `[random.randint (0, 3) for i in range(20)]` ??

Fractales

- la courbe de Hilbert

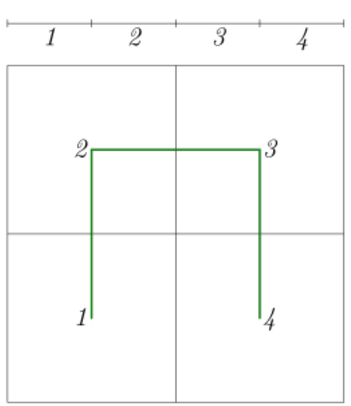


Fig. 1.

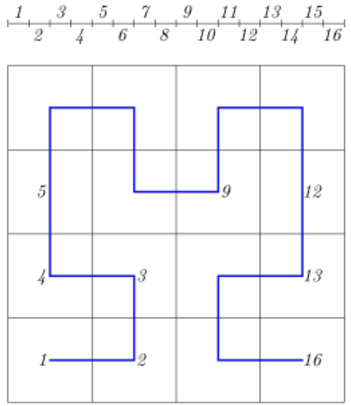


Fig. 2.

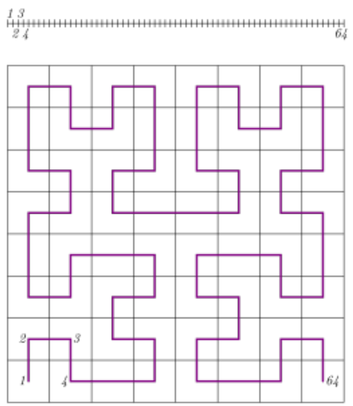
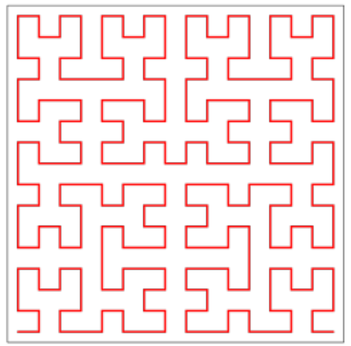
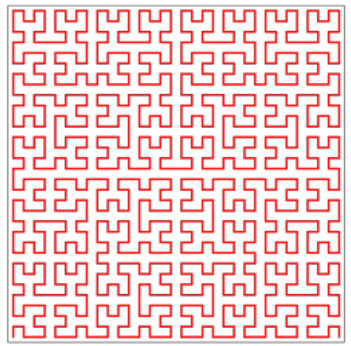
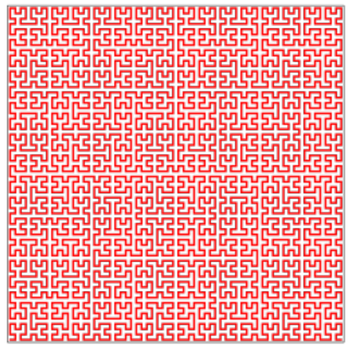


Fig. 3.







Exercice : écrire le programme qui la dessine

Fractales

- la courbe de Hilbert

```
c = 30
```

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

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

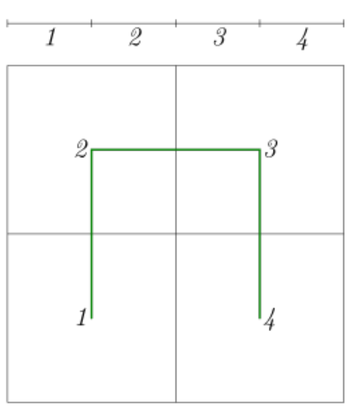


Fig. 1.

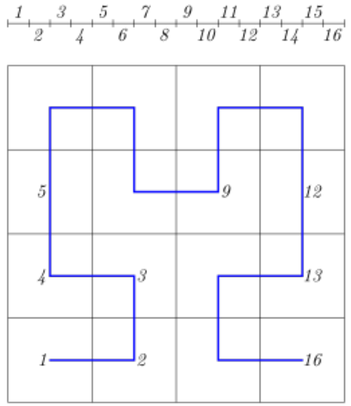


Fig. 2.

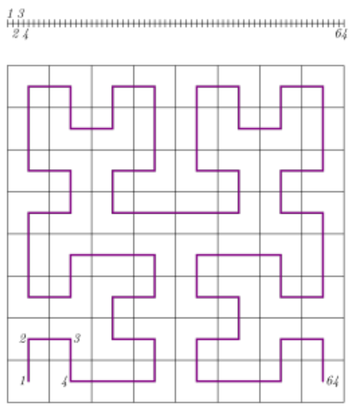
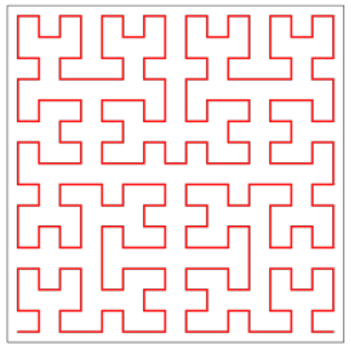
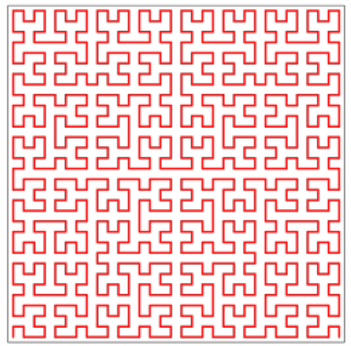
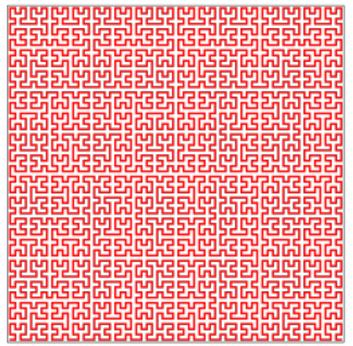


Fig. 3.

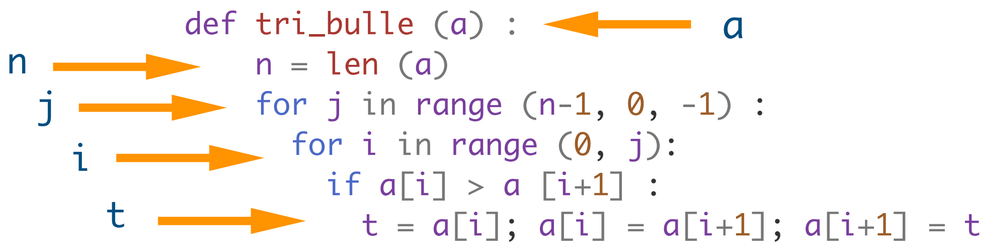




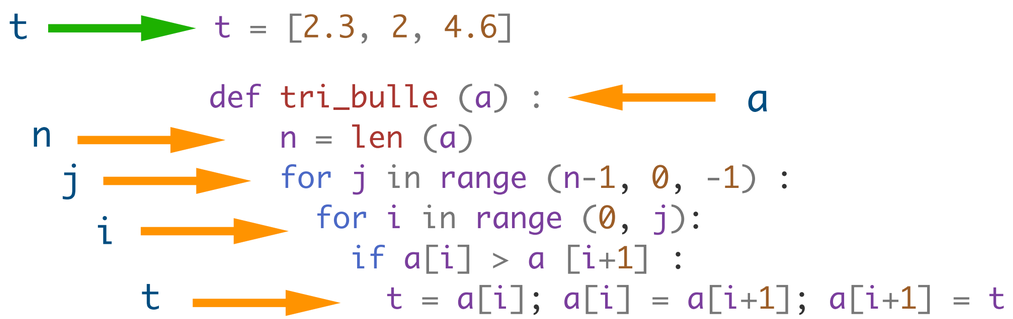


Quelques remarques

- les variables déclarées dans une fonction n'existent que dans le code de cette fonction



- les variables `n`, `j`, `i`, `t` et *a* sont **locales** à la fonction `tri_bulle`



- la variable `t` **globale** est distincte de la variable `t` locale

Quelques remarques

- les fonctions ou données (de librairie..) sont regroupées en modules
- par exemple, on charge le module `random` avec

```
import random
```

- notation qualifiée avec nom de module `random.sample`

```
def rand_array (n, p) :
    return random.sample (range(p), n)
```

- on peut raccourcir la notation qualifiée `rd.sample` le nom du module avec

```
import random as rd
```

- on peut utiliser la notation simple `sample` sans le nom du module avec

```
from random import *
```

- la liste des modules disponibles s'obtient avec

```
help()
modules
. . .
quit
```

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 "(%d, %d)" %(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.x)
10
print (p1.y)
20
```

nouvel objet de la classe `Point`

```
print (p1.__str__())
Point(10, 20)

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

```
p2 = Point (30, 40)
print (p1.__add__ (p2))
Point(40, 60)

print (p1 + p2)
Point(40, 60)
```

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

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__` est appelé par `<=`

- 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 "({}, {})".format (self.haut_gauche, self.bas_droite)
```

on vérifie que q est dans le quadrant inférieur droit

```
r = Rectangle (p1, p3)
((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`

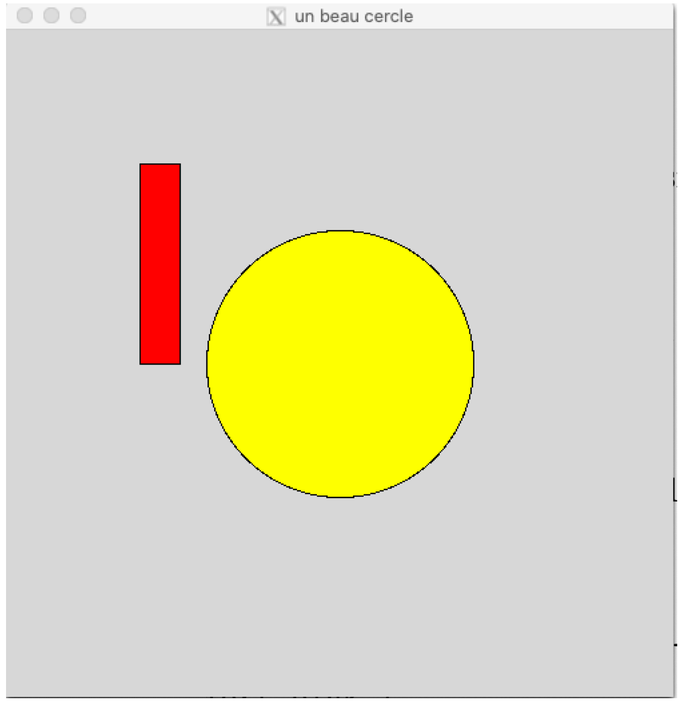
Graphique

- 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
```

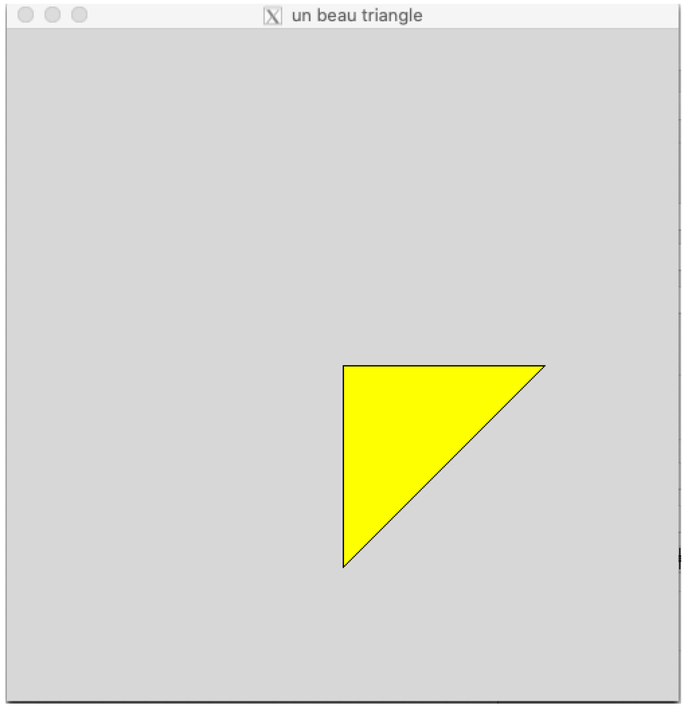


Graphique

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

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

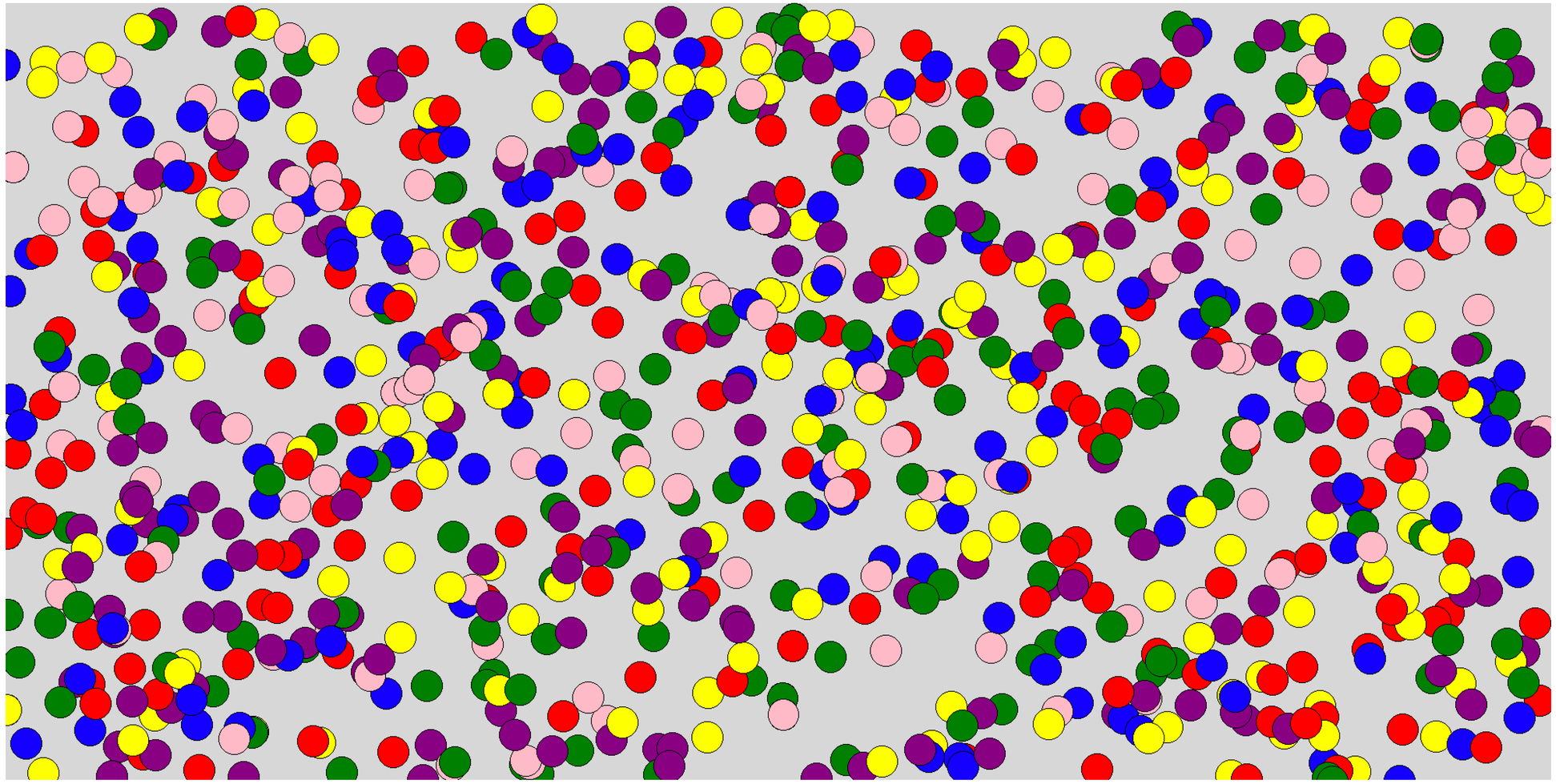
```
def dessin2 ():
    win = GraphWin("un beau triangle", 500, 500)
    p1 = Point (250,250)
    p2 = Point (400,250)
    p3 = Point (250,400)
    pol = Polygon(p1, p2, p3)
    pol.setFill("yellow")
    pol.draw(win)
    win.getMouse() # Pause to view result
    win.close()    # Close window when done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



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`)

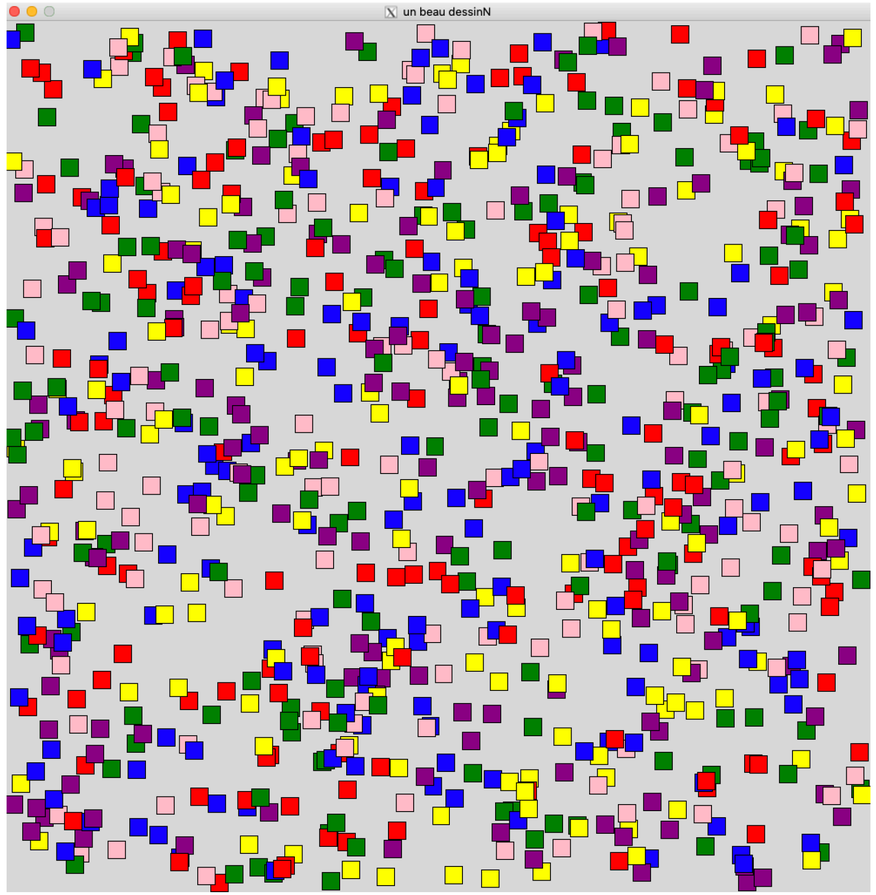
```
def dessinM ():
    winx = 2000; winy = 1000
    win = GraphWin("un beau dessinM", winx, winy, autoflush=False)
    win.setCoords (0, 0, winx, winy)
    cols = ("red", "yellow", "green", "blue", "purple", "pink")
    n = 600
    a = random.sample(range(winx-20),n)
    b = random.sample(range(winy-20),n)
    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`)

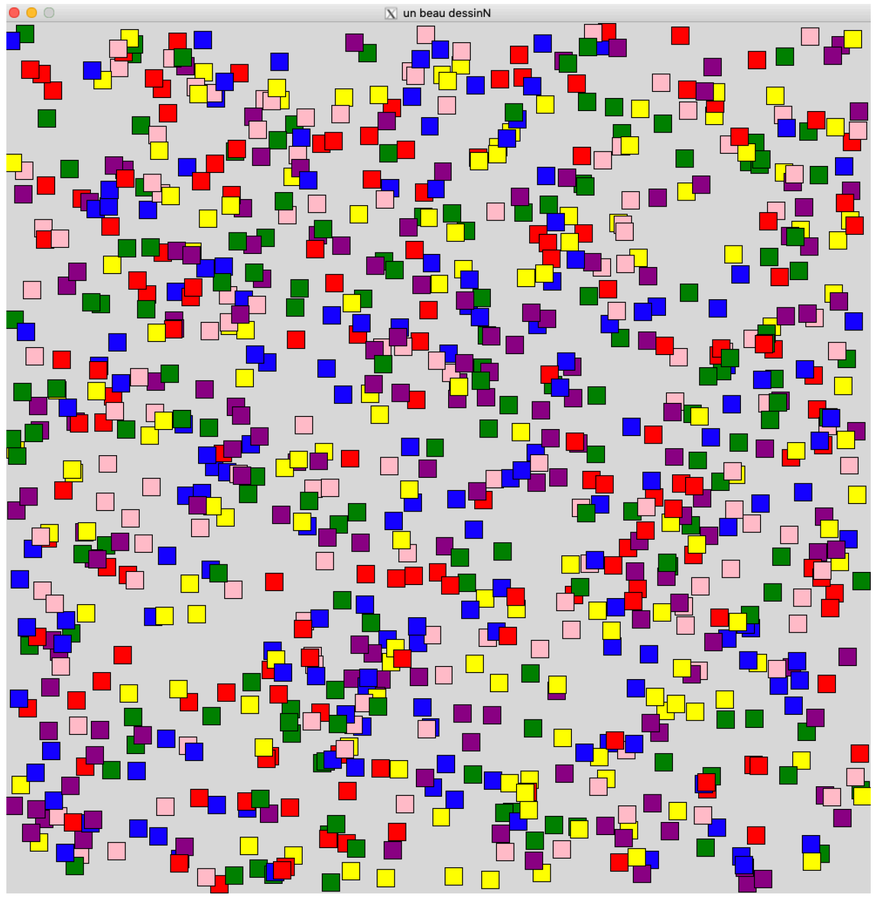
```
def dessinR (n, largeur):
    winx = 1000; winy = 1000
    win = GraphWin("un beau dessinM", winx, winy, autoflush=False)
    win.setCoords (0, 0, winx, winy)
    cols = ("red", "yellow", "green", "blue", "purple", "pink")
    a = random.sample(range(winx-largeur),n)
    b = random.sample(range(winy-largeur),n)
    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()
```

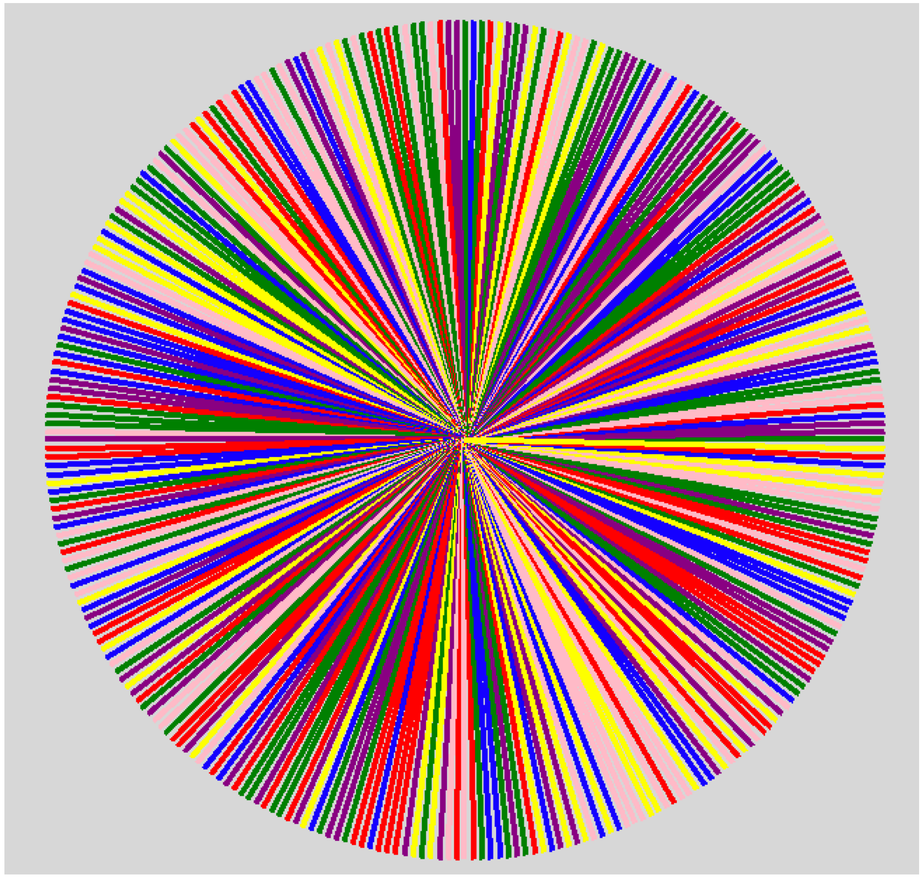


Graphique

- 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()
```