Informatique et Programmation

Cours 4

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

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

Plan

- tris élémentaires
- typage dynamique
- portée des variables
- exemple de paquetage graphique

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

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

Recap

- mots clés en Python (déjà vus en rouge)

```
>>> help()
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

les sources sont en http://jeanjacqueslevy.net/prog-py-22/progs/c4a.py

Tri

- fonctions utiles sur les tableaux

```
import random

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

générer un tableau de n entiers aléatoires entre 0 et p-1

```
>>> rand_array (10, 100)
[48, 7, 59, 99, 45, 88, 56, 15, 61, 35]

>>> rand_array (10, 100)
[44, 86, 29, 30, 72, 67, 12, 15, 75, 45]
```

on suppose len(a) > 0

```
def index_min_of (a) :
    imin = 0
    for i in range (1, len(a)):
        if a[i] < a[imin] :
            imin = i
    return imin
```

```
def print_array (a) :
    for elt in a :
        print (elt, end = ' ')
    print ()
```

Tri

- on cherche le minimum et on le met en tête..
  et on recommence à partir du deuxième élément, etc…

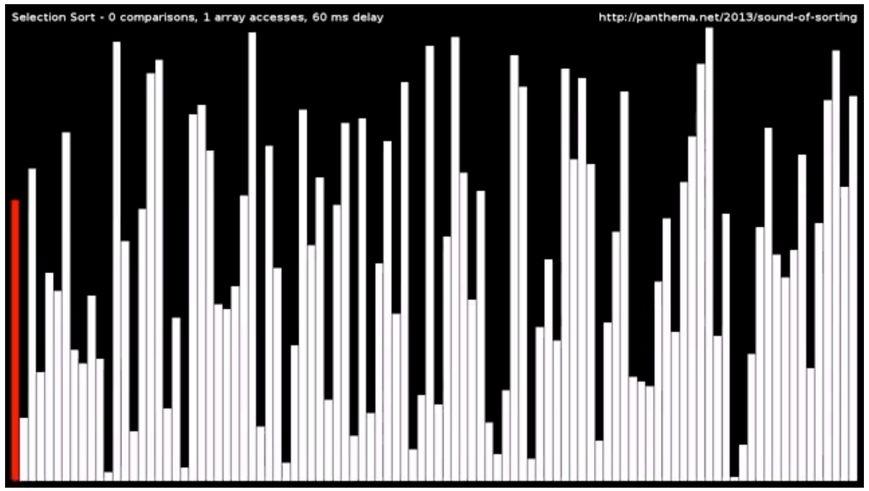
on suppose `len(a) > 0`

```
def index_min_of (a) :
    imin = 0
    for i in range (1, len(a)):
        if a[i] < a[imin] :
            imin = i
    return imin
```

```
def tri_selection (a) :
    n = len (a)
    for i in range (n-1) :
      jmin = i;
      for j in range(i+1, n) :
          if a[j] < a[jmin] :
              jmin = j
      t = a[i]; a[i] = a[jmin]; a[jmin] = t
```

échanger les valeurs de `a[i]` et `a[jmin]`

http://visualgo.net/en/sorting

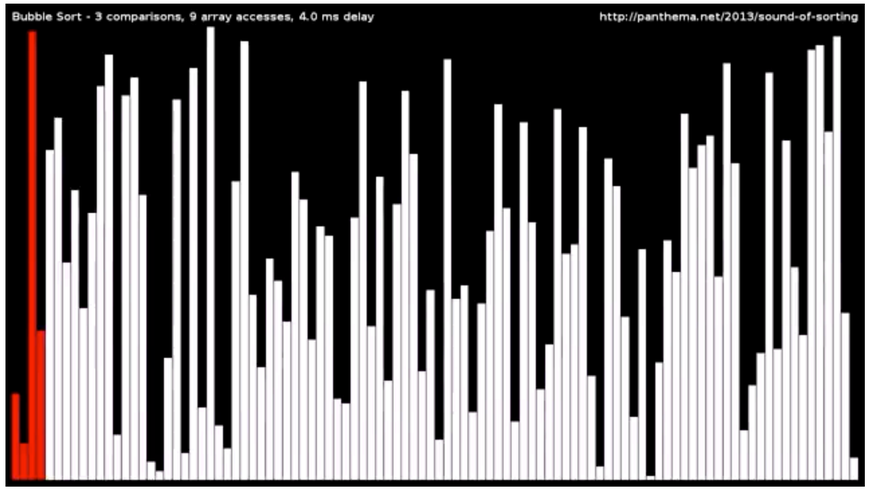


Tri

- un bulle pousse le maximum vers la fin.. et
  on recommence jusqu'à l'avant-dernier élément, etc……

```
def tri_bulle (a) :
    n = len (a)
    for j in range (n-1, 0, -1) :
      for i in range (0, j):
        if a[i] > a [i+1] :
          t = a[i]; a[i] = a[i+1]; a[i+1] = t
```

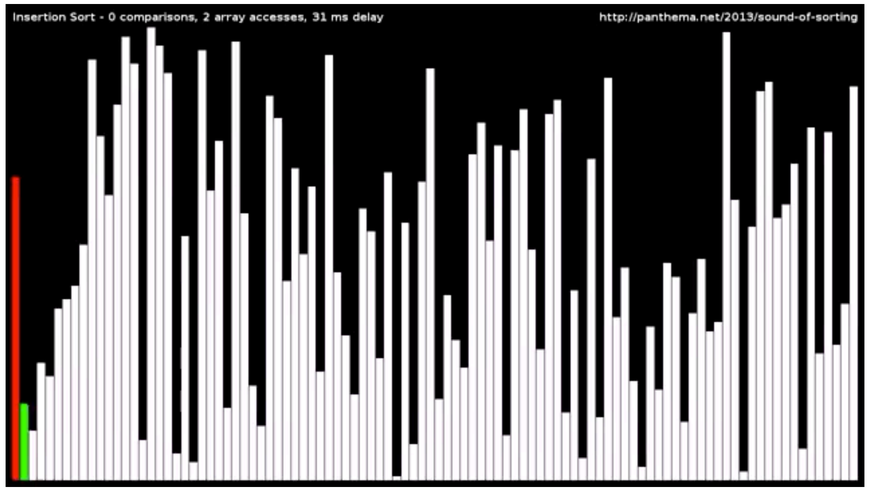
échanger les valeurs de `a[i]` et `a[i+1]`

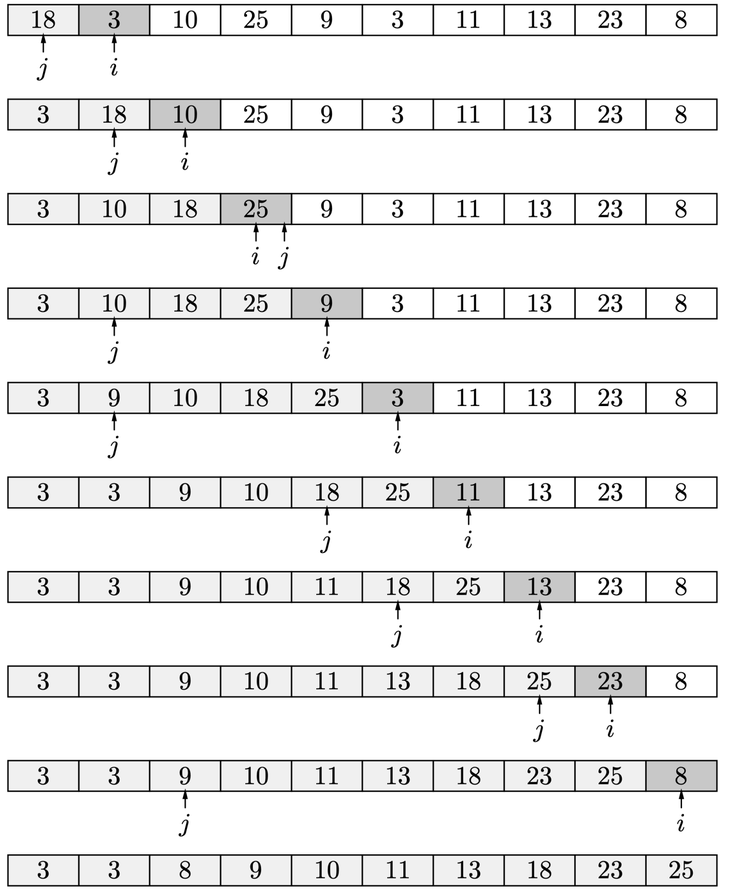


Tri

- on insère les éléments dans la partie gauche déjà triée, etc…… [comme dans un jeu de cartes]

```
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
```





Quelques remarques

- on peut trier des tableaux d'entiers, de réels, de chaînes de caractères

```
a = [44, 127, 24, 15, 60, 149, 147, 72, 36, 34]
b = [2.3, 2, 4.6]
c = [ 'camille', 'jean-jacques', 'paul', ‘axel']

d = [ 'camille', 28, 'paul', 'axel']
```

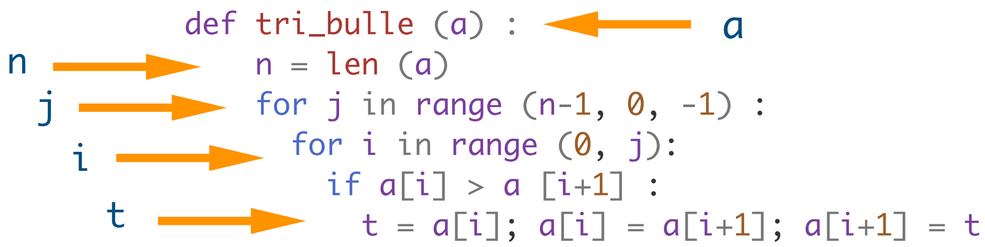
```
tri_selection (a)   →   [15, 24, 34, 36, 44, 60, 72, 127, 147, 149]
tri_selection (b)   →   [2, 2.3, 4.6]
tri_selection (c)   →   ['axel', 'camille', 'jean-jacques', 'paul']
```

```
tri_selection (d)   →   Traceback (most recent call last):
                          File "<stdin>", line 1, in <module>
                          File "<stdin>", line 6, in tri_selection
                        TypeError: '<' not supported between instances of 'int' and 'str'
```

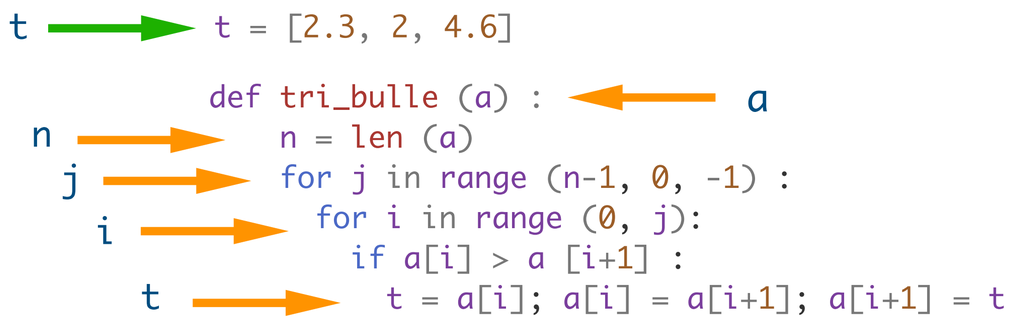
- en Python, les **types sont dynamiques**
- et on ne voit les erreurs de types qu'à l'exécution

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
```

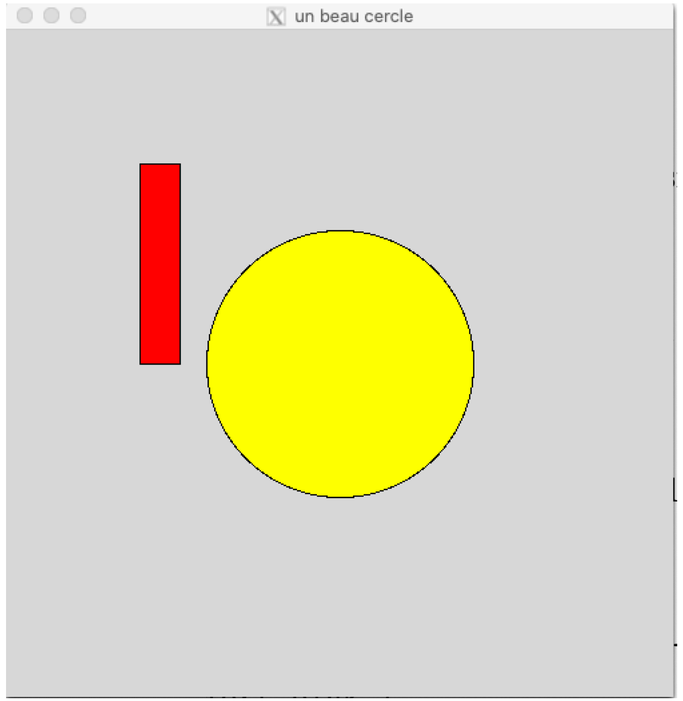
Graphique

- 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
```

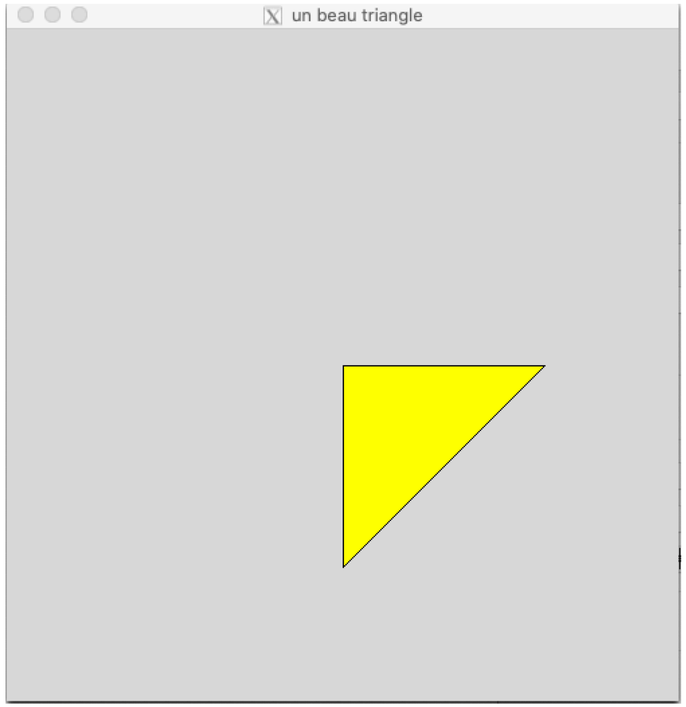


Graphique

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

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

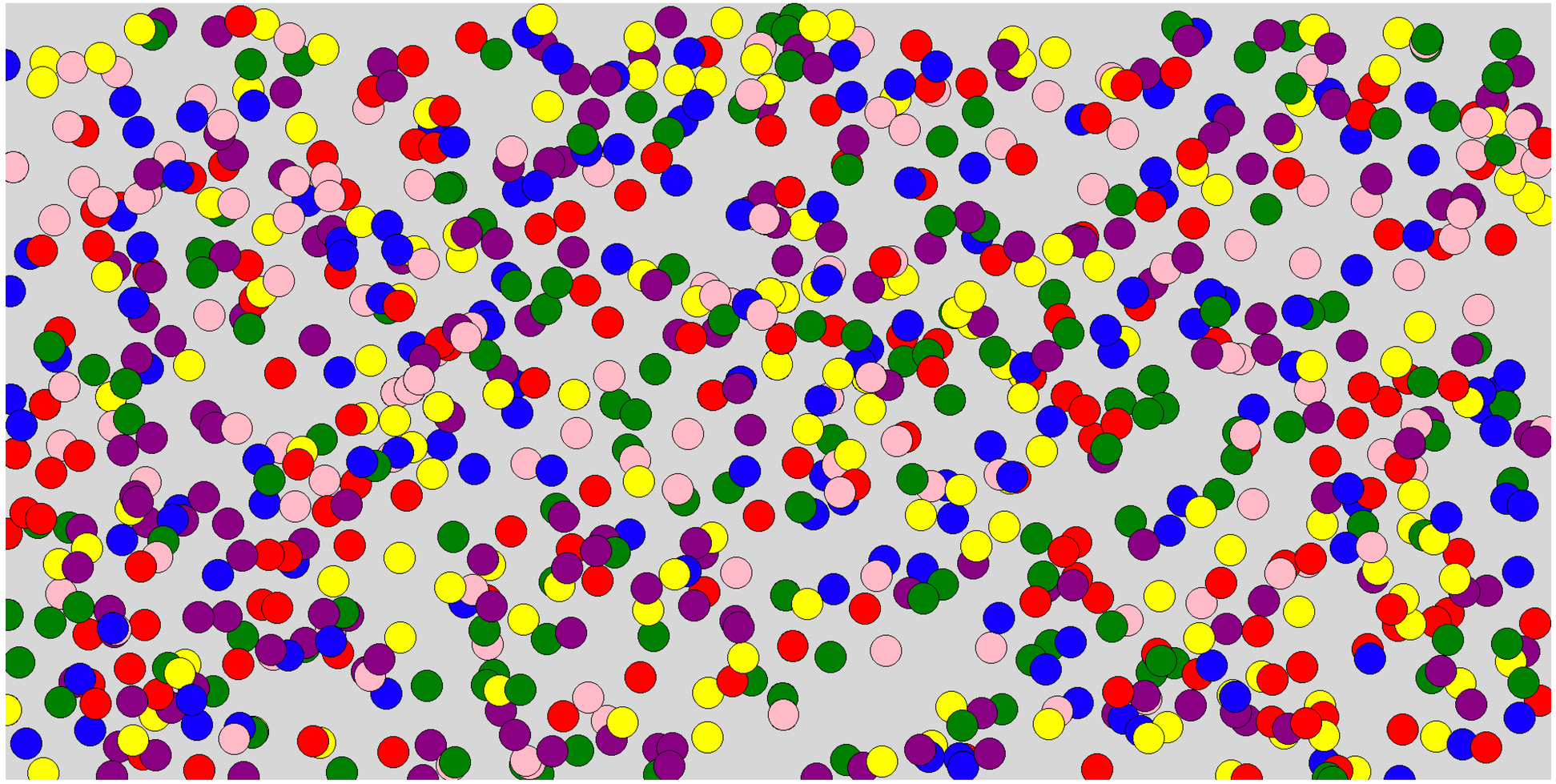
```
def dessin2 ():
    win = GraphWin("un beau triangle", 500, 500)
    p1 = Point (250,250)
    p2 = Point (400,250)
    p3 = Point (250,400)
    pol = Polygon(p1, p2, p3)
    pol.setFill("yellow")
    pol.draw(win)
    win.getMouse() # Pause to view result
    win.close()    # Close window when done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



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`)

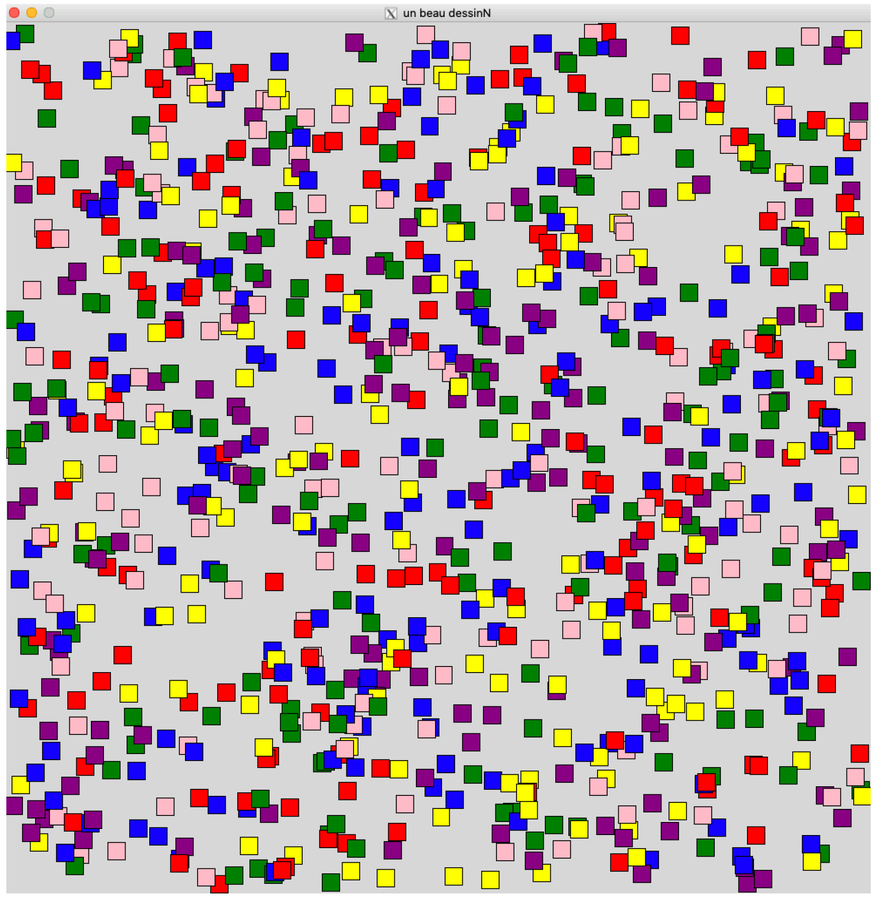
  (cf. `http://mcsp.wartburg.edu/zelle/python/graphics`)

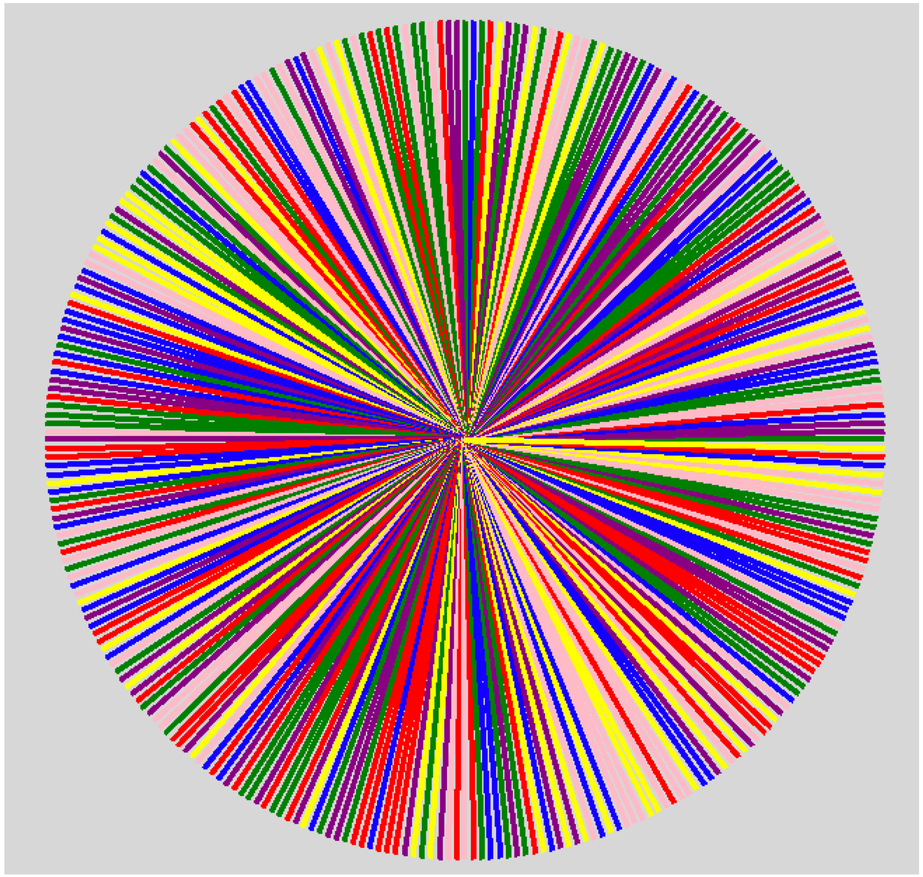
```
def dessinM ():
    winx = 2000; winy = 1000
    win = GraphWin("un beau dessinM", winx, winy, autoflush=False)
    win.setCoords (0, 0, winx, winy)
    cols = ("red", "yellow", "green", "blue", "purple", "pink")
    n = 600
    a = random.sample(range(winx-20),n)
    b = random.sample(range(winy-20),n)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