IFT211/776
Programmation scientifique en Python
Conclusion

Gabriel Girard

Département d'informatique
UNIVERSITÉ DE SHERBROOKE

9 février 2017

UNIVERSITÉ DE SHERBROOKE

Thème 11 — Conclusion

1. Programmation
2. Le langage Python
3. Reste à voir ...
   - Classes en Python
   - Récursivité

UNIVERSITÉ DE SHERBROOKE

Concepts généraux de programmation

- Concept d'abstraction
- Programmation structurée
  - séquence
  - sélection
  - itération
- Programmation modulaire
  - un programme est composé de petits modules courts, avec un degré élevé de cohésion et un seul point d'entrée et de sortie.
  - chaque module est conçu pour être codé et testé séparément.

Langage Python

- Structure d'un programme Python
- Types de base : définition et utilisation
- Expressions arithmétiques et logiques
- Énoncés
  - affectation ;
  - sélection : `if`
  - itération : `while` et `for`
- Les fonctions

Reste à voir : abstraction de données

- On veut créer des nouveaux types de données
- On peut regrouper des données pour que :
  - les données soient vues comme un tout ;
  - les opérations soient intégrées.

Exemple 1 : Type « float »

- Les attributs (données membres) :
  - La mantisse.
  - La caractéristique (exposant)
  - Le signe de la mantisse
- Les opérations :
  - $+, -, *, /$
  - $=$
  - $==, <=, >=, <, >, !=$

Reste à voir : classes en Python

- En Python, on crée de nouveaux types grâce aux classes (`class`).
- Une opération d'une classe s'appelle une **méthode.**
- Une variable d'une classe s'appelle un attribut (membre)
- Une variable définie à partir d'un de ces nouveaux types s'appelle un **objet** (ou instance).

Syntaxe des classes

```
class Nom_de_la Classe:
    def __init__(self, ...)
    def fct1(self, ...)
    ...
    def fct_n(self, ...)
```

Exemple 3 : Type « Compteur »

Exemple 3

On veut définir un type de donnée abstrait qui est un compteur.

Type Compteur – Implantation

```
class Compteur:
    def __init__(self, val=0):
        self.MAX = 100
        self.MIN = -100
        self.valeur = val
    def incremente(self):
        if self.valeur < self.MAX:
            self. valeur += 1
        else:
            print("Debordement du compteur. ")
    def decremente(self):
        if self.valeur > self.MIN:
            self.valeur -= 1
        else:
            print("Debordement du compteur. ")
    def valeur_courante(self):
        return self.valeur
```

UNIVERSITÉ DE SHERBROOKE

Type Compteur : utilisation

```
cpt1 = Compteur()
cpt2 = Compteur(25)
for i in range(10):
    cpt1.incremente()
    cpt2.decremente()
    print("cpt1 =", cpt1.valeur_courante(),
          " , cpt2 =", cpt2.valeur_courante())
```

Utilisation de la classe Compteur

```
cpt1 = 1  , cpt2 = 24
cpt1 = 2  , cpt2 = 23
cpt1 = 3  , cpt2 = 22
cpt1 = 4  , cpt2 = 21
cpt1 = 5  , cpt2 = 20
cpt1 = 6  , cpt2 = 19
cpt1 = 7  , cpt2 = 18
cpt1 = 8  , cpt2 = 17
cpt1 = 9  , cpt2 = 16
cpt1 = 10  , cpt2 = 15
```

Exemple : Type Point

```
class Point:

    instances = 0

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
        Point.instances += 1

    def get_val_x(self): return self.x
    def get_val_y(self): return self.y

    def lecture(self):
        self.x = int(input("Valeur en x: "))
        self.y = int(input("Valeur en y: "))
```

```
class Point:
    ...
    def evaluer_distance(self, point2):
        dis_x = point2.x - self.x
        dis_y = point2.y - self.y
        return math.sqrt(dis_x ** 2 + dis_y ** 2)

    def __eq__(self, p):
        return self.x == p.x and self.y == p.y

    def __str__(self):
        return "(" + str(self.x) + ", " +
            str(self.y) + ")"
```

```
pt1 = Point(1,2)
pt2 = Point()
pt2.lecture()
distance = pt1.evaluer_distance(pt2)
pt4 = Point()
if pt2 == pt4: print("Egaux")
else : print("Différent")
print("Distance entre :", str(pt1), str(pt2), "=",
    distance)
print("Le nombre de points crees est",
    pt1.instances)
--------------------
Valeur en x: 2
Valeur en y: 3
Différent
Distance entre : (1, 2) (2, 3) = 1.414213562373095
Le nombre de points créés est 3
```

UNIVERSITÉ DE SHERBROOKE

Exemple 6 : Type Cercle

```
from point import *
import math

class Cercle:
  def __init__(self, centre=Point(0, 0), rayon=0):
    self.centre = centre
    self.rayon = rayon

  def val_centre(self):
    return self.centre

  def val_rayon(self):
    return self.rayon
```

```
class Cercle:
  ...
  def calculer_surface(self):
    return math.pi * (self.rayon ** 2)

  def __eq__(self, c1):
     egal = False
     if self.centre == c1.centre and self.rayon ==
        c1.rayon:
        egal = True
     return egal

  def __str__(self):
     return "centre = " + str(self.centre) +
            ", rayon = " + str(self.rayon)
```

```
def main():
    c1 = Cercle()
    c2 = Cercle(Point(1,2), 10)
    c3 = Cercle(Point(1,2), 10)
    if c2 == c3 : print(str(c2))
    print(c2.calculer_surface())

if __name__ == "__main__":
     main()
====================================

centre = (1, 2), rayon = 10
314.1592653589793
```

Exemple 7 : Type Rationnel

```
class Rationnel:

  def __init__(self, num=0, denom=1):
    self.num = num
    if denom != 0:
       self.denom = denom
    else:
       print("denominateur nul")
       self.num = 0
       self.denom = 1

  def __str__(self):
    return str(self.num) + "/" + str(self.denom)
```

```
def simplifier(self):
  if self.num == 0: self.denom = 1
  else:
    i = self.num
    if i < 0: i = -i
    j = self.denom
    # recherche du plus grand commun diviseur
    while i != j:
        if i > j: i -= j
        else: j -= i
    self.num = int(self.num / i)
    self.denom = int(self.denom / i)

def m_deno(self, r):
    if self.denom != r.denom:
        self.num *= r.denom
        r.num *= self.denom
        self.denom *= r.denom
        r.denom = self.denom
```

```
    def __add__(self, r):
        t = Rationnel()
        t.num = self.num
        t.denom = self.denom
        t.m_deno(r)
        t.num += r.num
        t.simplifier()
        return t
    def __mul__(self, r):
        t = Rationnel()
        t.num = self.num * r.num
        t.denom = self.denom*r.denom
        t.simplifier()
        return t
    def __eq__(self, r):
        t1 = Rationnel(); t2 = r
        t1.num = self.num
        t1.denom = self.denom
        t1.m_deno(t2)
        return t1.num == t2.num
```

```
def main():
    nb1 = Rationnel(2, 3); nb2 = Rationnel(-4, 0)
    nb3 = Rationnel(2, 3); a = Rationnel(2, 3)
    b = Rationnel(1, 3);   c = Rationnel(3, 4)

    print("Le rationnel nb1 vaut: ", str(nb1))
    print("Le rationnel nb2 vaut: ", str(nb2))
    nb2 = nb1 * nb3
    print("Le rationnel nb2 vaut: ", str(nb2))
    nb2 = nb1 + b
    print("Le rationnel nb2 vaut: ", str(nb2))
    nb2 = nb1 + c
    print("Le rationnel nb2 vaut: ", str(nb2))
    if a == c: print("Égaux")
    else: print("Différents")
    if a == nb3: print("Égaux")
    else: print("Différents")

if __name__ == "__main__":
    main()
```

```
Dénominateur nul
Le rationnel nb1 vaut:  2/3
Le rationnel nb2 vaut:  0/1
Le rationnel nb3 vaut:  2/3
Le rationnel nb2 vaut:  4/9
Le rationnel nb2 vaut:  1/1
Le rationnel nb2 vaut:  17/12
Différents
Égaux
```

Reste à voir : récursivité

- Récursivité.

Définition

La résursivité consiste à implanter une solution comme une fonction qui s'appelle elle-même afin de répéter un certain nombre de fois un traitement avec des données différentes et ce jusqu'à ce que la solution soit trouvée.

- **Exemple :**
  Écrire un programme qui trouve le factoriel d'un nombre.

UNIVERSITÉ DE SHERBROOKE

Factoriel : algorithme

- fact(0) = 1 (cas d'arrêt)
- fact(n) = n * fact(n-1) (décomposition)

Factoriel : code

```
def fact(nb):
    if nb == 1:
        resultat = 1
    else:
        resultat = nb * fact(nb-1)
    return resultat

nombre = int(input("Entrez le nombre : "))
factoriel = fact(nombre)
print("Le factoriel  est :", factoriel)
```

Forme d'un algorithme récursif

-> Si c'est un cas d'arrêt
  - alors on résout le problème

-> sinon
  - on le réduit grâce à la récursivité

Factoriel : code

```
def fact(nb):
    if nb == 1:
        resultat = 1
    else:
        resultat = nb * fact(nb-1)
    return resultat
```

Tri sélection

```
def tri(liste, liste_triee):
    if len(liste) > 0 :
        petit = min(liste)
        liste_triee.append(petit)
        liste.remove(petit)
        tri(liste, liste_triee)
    return(liste_triee)

l = [6,3,8,1,4,7]
l1 = []
tri(l,l1)
print(l1)
```

Blob : description

- On vous demande d'écrire un programme qui compte le nombre de points dans une figure dessinée sur un tableau en 2D.
- Les cellules pleines peuvent être connectées pour former une figure.
- Deux cellules sont connectées si elles se touchent horizontalement, verticalement ou diagonalement.

Blob : description

1	1	0	0	0	1	1	1
1	1	0	0	0	0	0	1
1	0	0	1	0	0	0	1
1	0	0	1	1	0	0	0
1	0	1	1	1	1	0	0
1	0	0	1	1	0	0	0
1	0	0	1	0	0	1	1
1	1	0	0	0	1	0	1

Blob : algorithme

- Algorithme
  - Cas d'arrêt :
    1. la cellule est vide — on retourne 0
    2. la cellule n'est pas dans la grille — on retourne 0
  - Décomposition :
    Si une cellule n'est pas vide, on retourne 1 plus le nombre de cellule associées aux voisins de la cellule courante.
- Contraintes
  ** Pour s'assurer que l'on ne compte pas deux fois la même cellule, lorsqu'on la compte on la marque comme vide ou avec une valeur égale à -1 (si on veut retrouver la figure originale).

Blob : algorithme

On reçoit les coordonnées en x et en y
Si (x,y) est vide retourne 0
Si (x,y) est hors de la grille retourne 0
Si (x,y) est remplie
    retourne 1 + compte des cellules de ses voisins

Blob : code

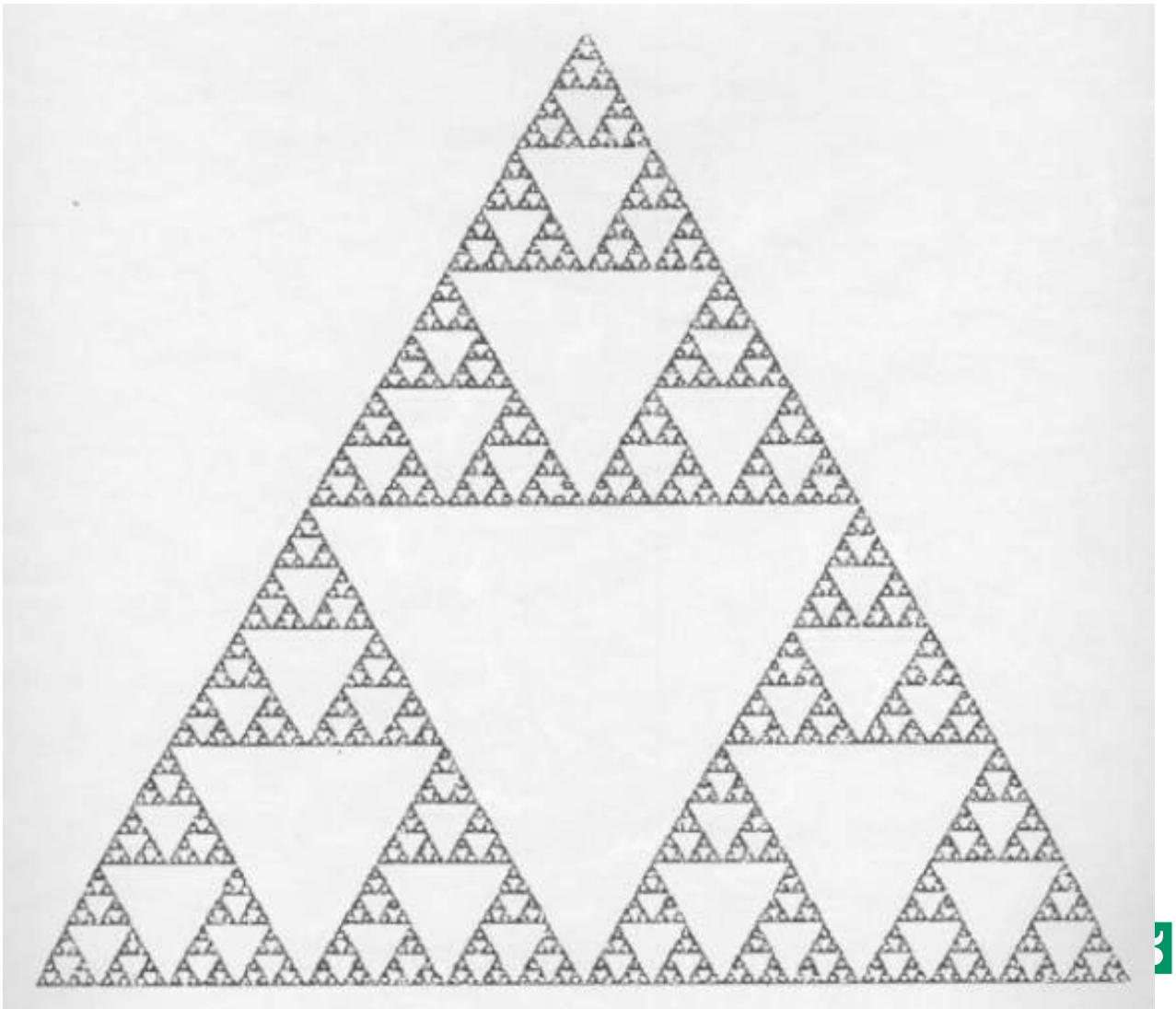
```
MAX_Y = 8
MAX_X = 8
grille = np.array([1,1,0,0,0,1,1,1,
                   1,1,0,0,0,0,0,1,
                   1,0,0,1,0,0,0,1,
                   1,0,0,1,1,0,0,0,
                   1,0,1,1,1,1,0,0,
                   1,0,0,1,1,0,0,0,
                   1,0,0,1,0,0,1,1,
                   1,1,0,0,0,1,0,1])
grille = grille.reshape((MAX_X, MAX_Y))

x, y = map(int, input("x , y =").split(","))
cpt = blob(grille, x, y);
print("Pts ds figure =", cpt)
```

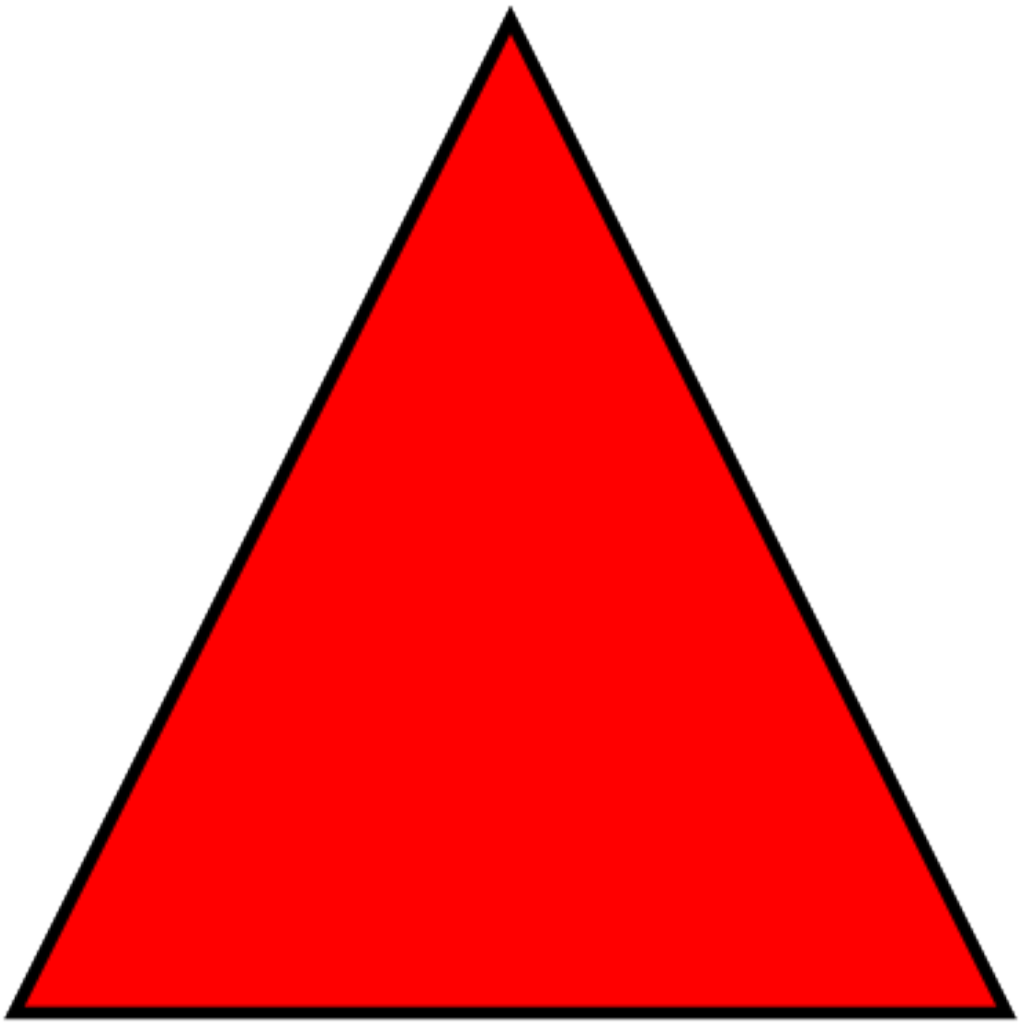
Blob : implémentation

```
def blob(grille, x, y):
    if x<0 or (x > MAX_X -1) or y < 0 or (y >
        MAX_Y -1):
         compte = 0;
    elif grille[x][y] == 0 : compte = 0;
    else :
         grille[x][y] = 0
         compte = 1 + blob(grille, x-1, y) + \
                  blob(grille, x-1, y+1) + \
                  blob(grille, x-1, y-1) + \
                  blob(grille, x, y+1) + \
                  blob(grille, x, y-1) + \
                  blob(grille, x+1, y+1) + \
                  blob(grille, x+1, y) + \
                  blob(grille, x+1, y-1)
    return compte;
```

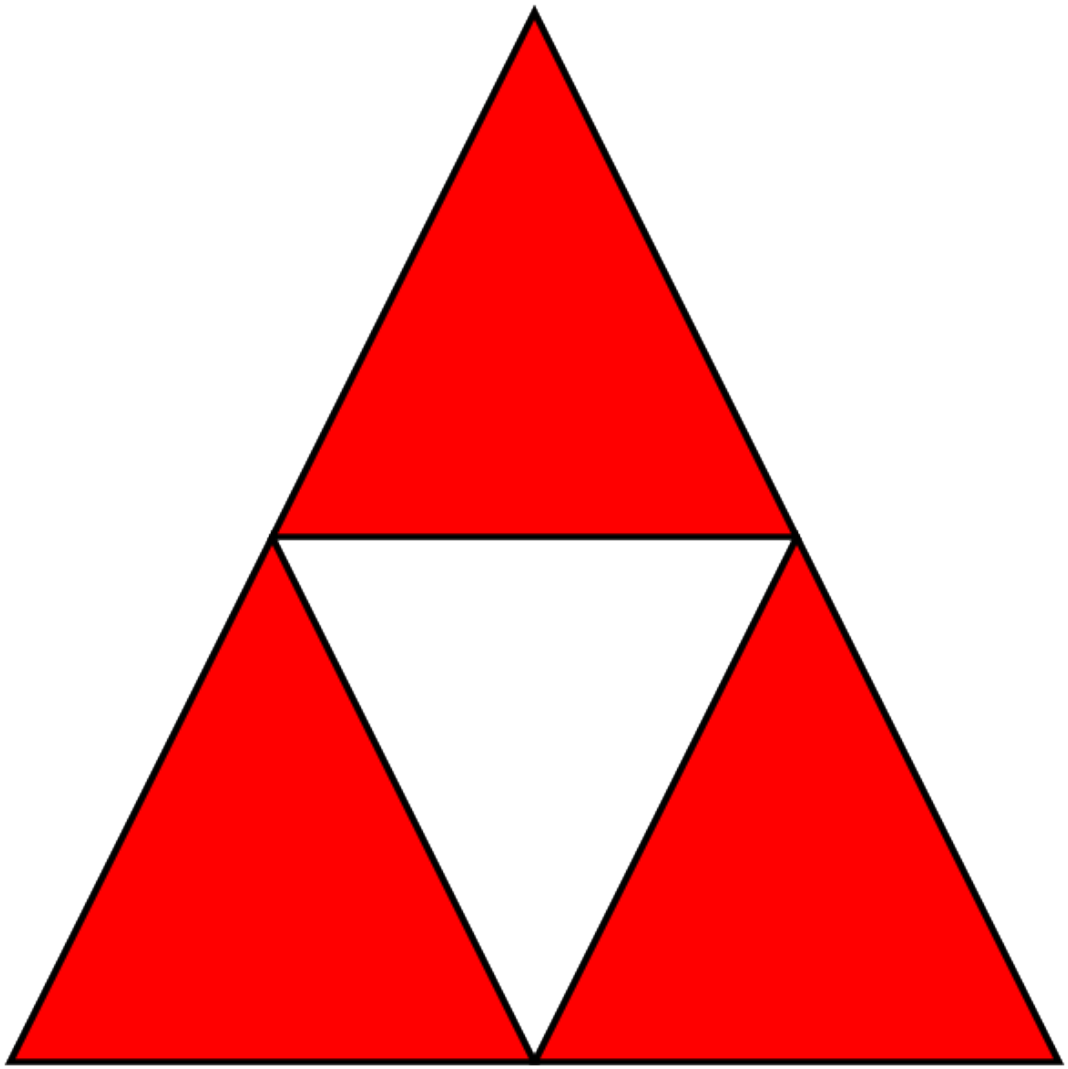
Sierpinski Gasket : description



Sierpinski Gasket : description

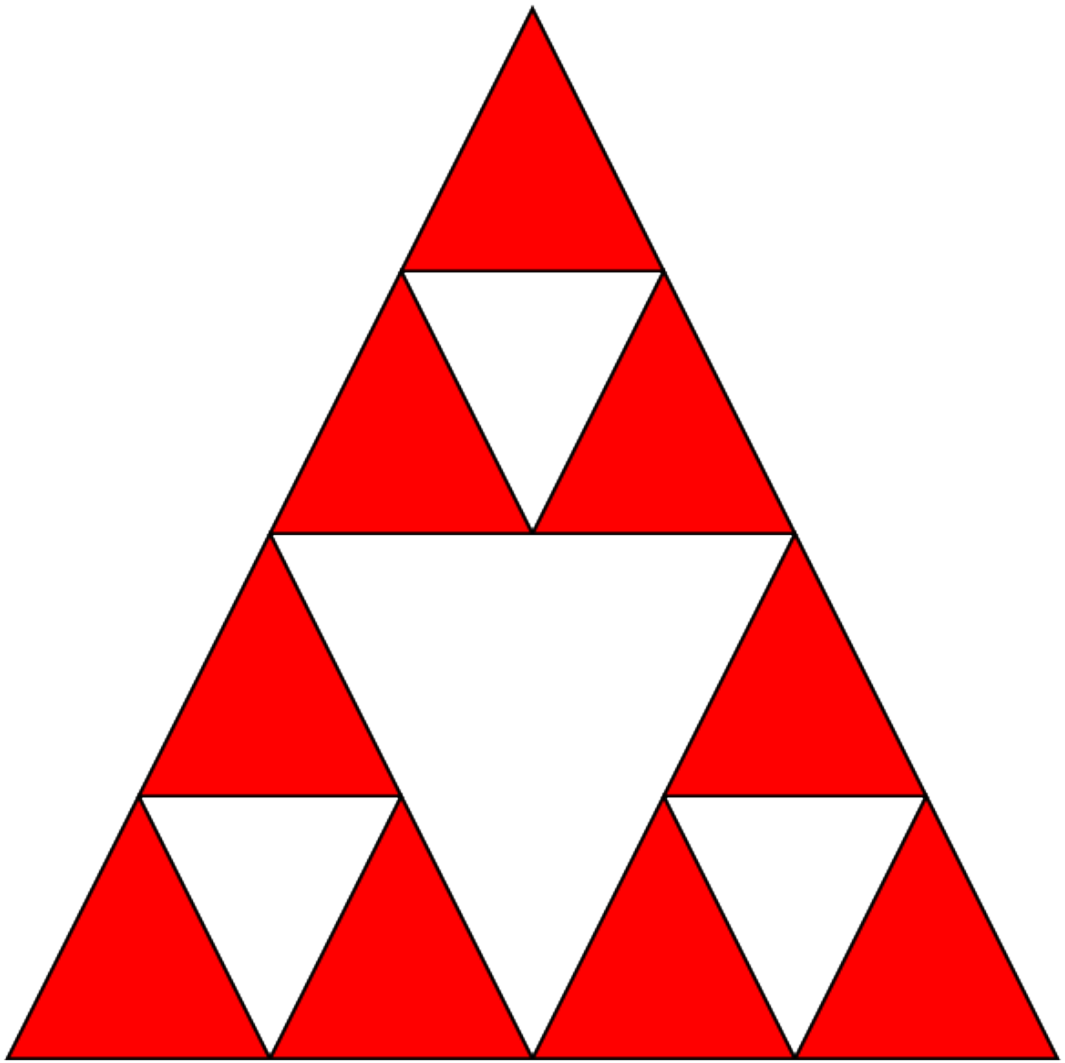


Sierpinski Gasket : description



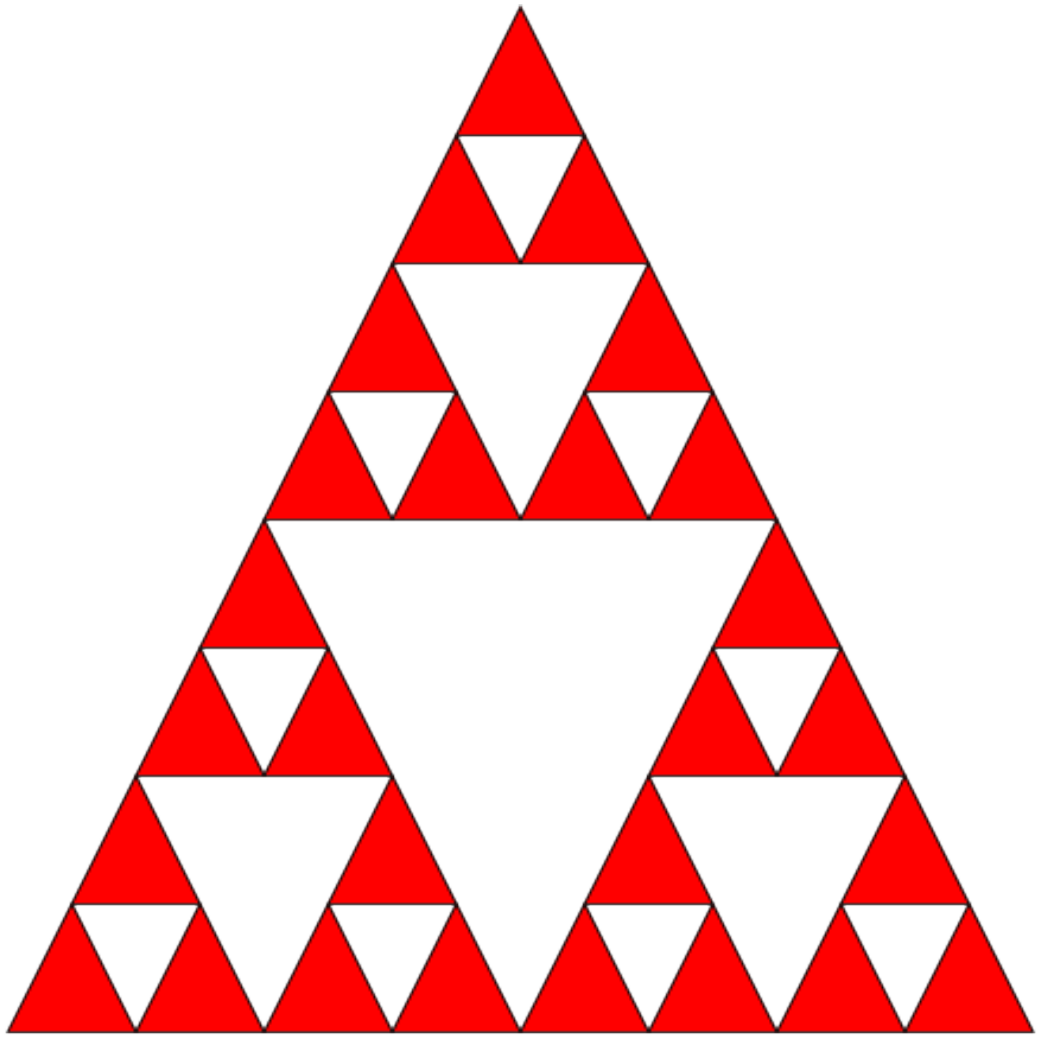
UNIVERSITÉ DE SHERBROOKE

Sierpinski Gasket : description



UNIVERSITÉ DE SHERBROOKE

Sierpinski Gasket : description



Sierpinski Gasket : implémentation

```
def sier(ecran, p1, p2, p3, niveau):
    if niveau == 0:
        # dessiner un triangle plein
        dessinerTriangle(ecran, p1, p2, p3)
    else:
        # faire les trois sous-paniers de
           Sierpinski
        niveau -= 1
        sier(p1, p1.mi(p2), p1.mi(p3), niveau)
        sier(p2.mi(p1), p2, p2.mi(p3), niveau)
        sier(p3.mi(p1), p3.mi(p2), p3, niveau)
```

Arbre fractal

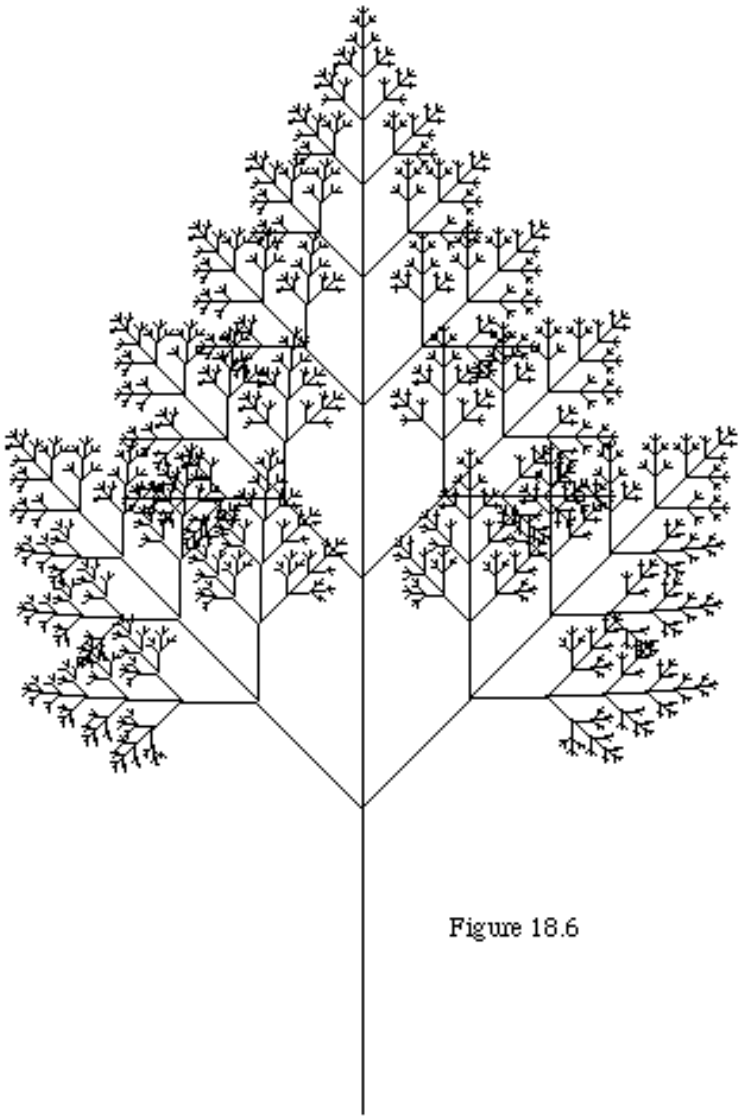


Figure 18.6

Arbre fractal

```
def dessinerA(e, p1, p2, lgA, a) :
  if lgA > 2 :
    dessinerLi(e, p1,  p2)  # le tronc
    p3 = calculerPt(p2, lgA//2, a+45) # gauche
    dessinerA(e, p2, p3,lgA//2, a+45)
    p3 = calculerPt(p2, lgA//2, a-45) # droite
    dessinerA(e, p2, p3,lgA//2, a-45)
    p3 = calculerPt(p2, int(lgA*3/4), a)
    dessinerA(e, p2, p3, int(lgA*3/4), a) # sommet

e = Canevas()
p1 = Point()
p1.lecture()
lgTronc = int(input( "Longueur de l'arbre : "))
p2 = Point(p1.valX(), p1.valY() + longueurTronc)
dessinerArbre(e, p1, p2, lgTronc, 90)
```

Reste à voir ...

- Notions avancées de programmation et meilleure maîtrise du langage.
  - programmation système et les opérateurs binaires ;
  - l'héritage ;
  - le polymorphisme ;
  - programmation générique
  - le traitement des erreurs (try, except)
  - la programmation fonctionnelle (lambda)
- La programmation par événements
- La programmation orientée objet.

UNIVERSITÉ DE SHERBROOKE

Reste à voir

- D'autre bibliothèques
- Outils de mise au point (debugger interactif ou trace)
- Les structures de données
- Les algorithmes et leur complexité