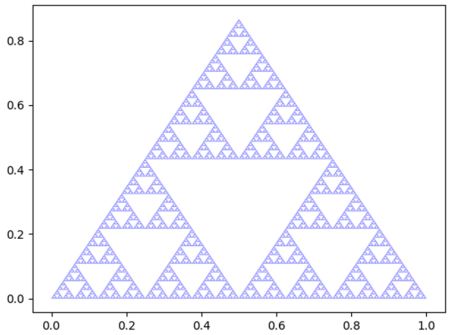
TP - Récursivité

Exercice 1. Triangle de Sierpinski

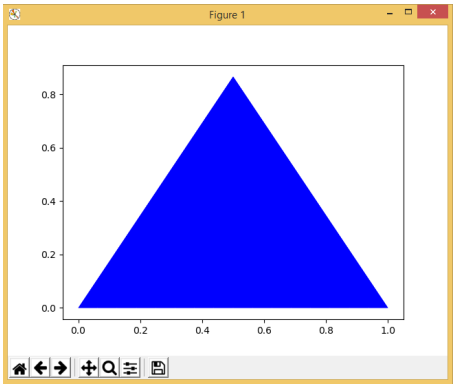


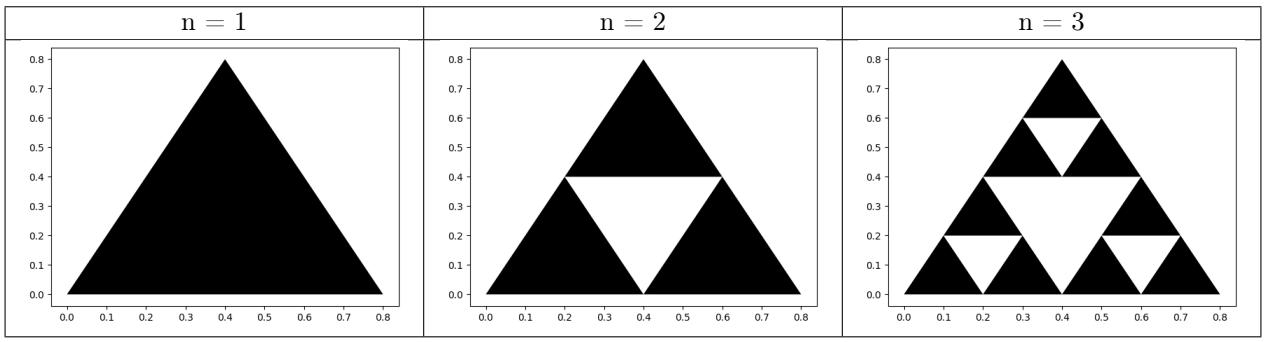
Le code suivant permet de tracer sur une figure un polygone dont on définit les n sommets en argument sous la forme d'une liste de couples de coordonnées $[[X_A, Y_A], [X_B, Y_B], ..., [X_n, Y_n]]$

```
import numpy as np
import matplotlib.pyplot as plt

def polygon(liste_points, color=0) :
  '''
  Parametre optionnel : color prend 2 valeurs 0 ou 1. Par defaut vaut 0.
  '''
  # Test le parametre couleur
  assert color in [0, 1]

  x, y = [], []

  # Choix de la couleur du polygone
  # Couleur possible : 'b':bleu, 'g':green, 'w':blanc, 'm':violet, 'p': aleatoire,...
  if color == 0: ink = 'k' # Noir
  else: ink = 'w'        # Blanc

  for point in liste_points :
    x.append(point[0])
    y.append(point[1])
    plt.fill(x, y, ink)

A = np.array([0,0])
B = np.array([0.4, 0.8])
C = np.array([0.8,0])
polygon([A,B,C])
plt.show()
```

Remarque : l'utilisation de liste au lieu de numpy est parfaitement possible, mais numpy peut être pratique.

L'exécution du code renvoie :



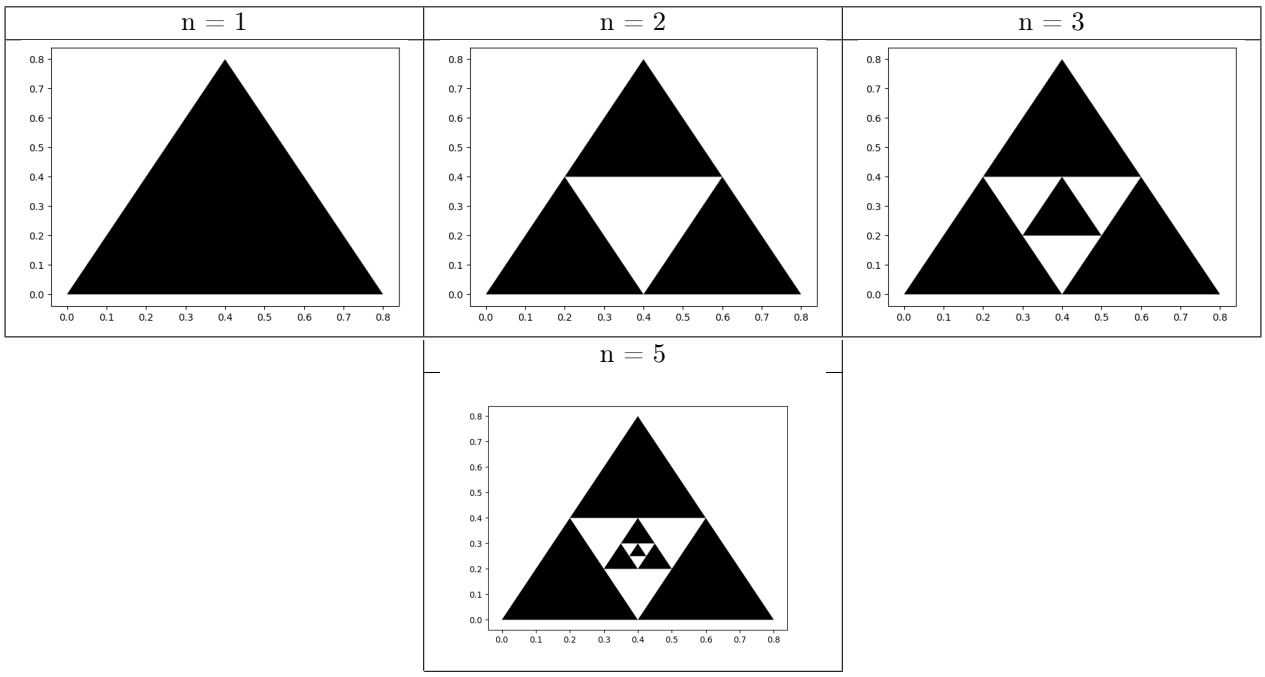


Question 1 : En vous aidant des figures ci-dessus, déterminer les coordonnées des triangles noirs pour n $\in$ [1, 2, 3].

Question 2 : Combien y-a-t-il de triangles pour n $\in$ [1, 2, 3, 4]. En déduire le nombre de triangles noirs pour n=p, p $\in \mathbb{N}^*$.

Question 3 : Écrire une fonction récursive sierpinski traçant le résultat affiché ci-dessus (triangles équilatéraux). On n'utilisera pas le paramètre `color` lors de l'appel de la fonction `polygon`.

Voici maintenant les images suivantes :

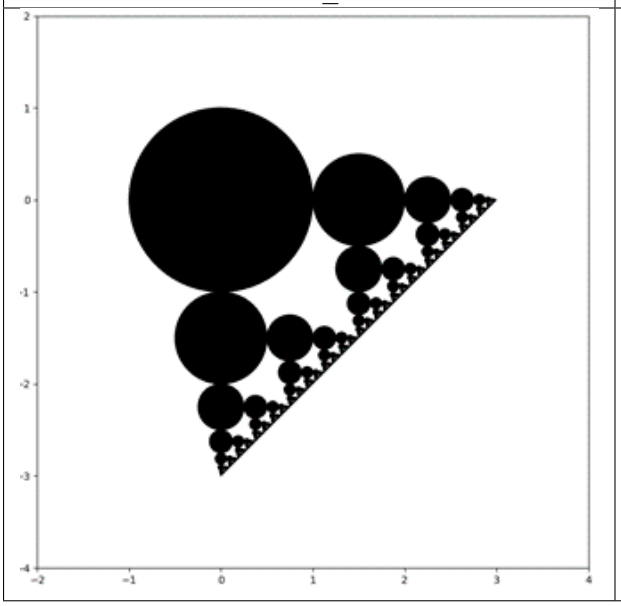
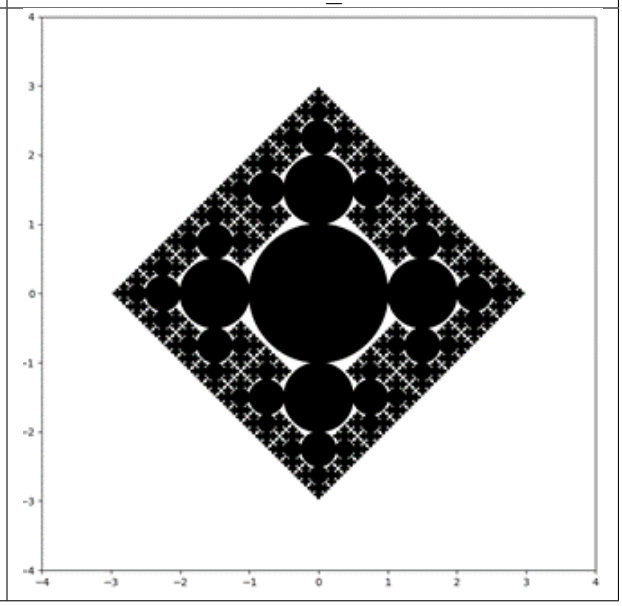


Question 4 : Donner les coordonnées du triangle blanc pour n = 2 et du plus petit triangle pour n $\in$ [3, 4]. Que constatez vous ?

Question 5 : Écrire une fonction récursive triangles_infinis traçant le résultat affiché ci-dessous (triangles équilatéraux). On pourra se servir du paramètre `color` de la fonction `polygon`.

Exercice 2. Cercles récursifs

Soient les figures ci-dessous :

bulle_1	bulle_2
	

On souhaite les obtenir par récursivité.

Soit le code suivant :

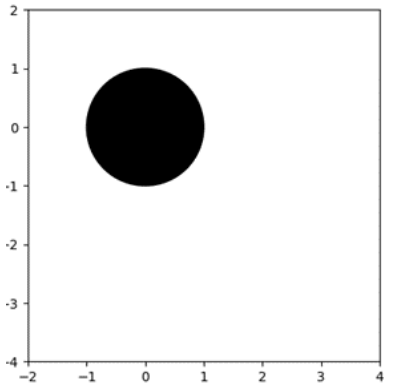
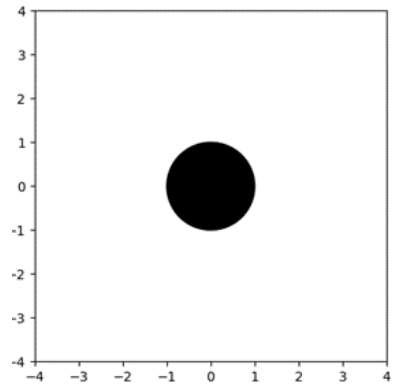
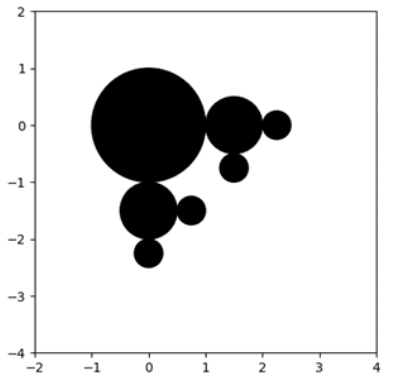
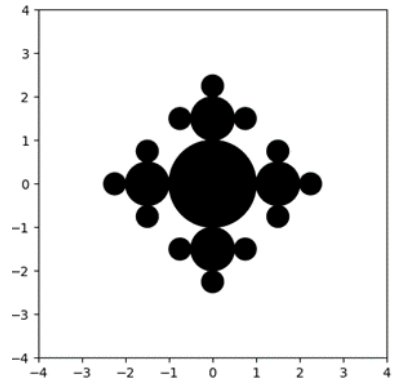
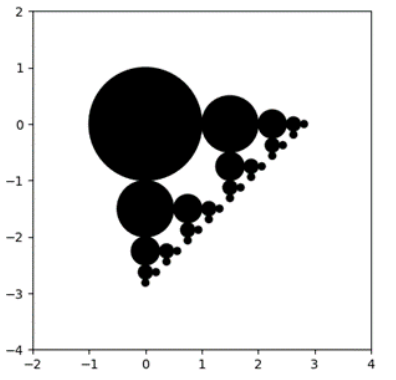
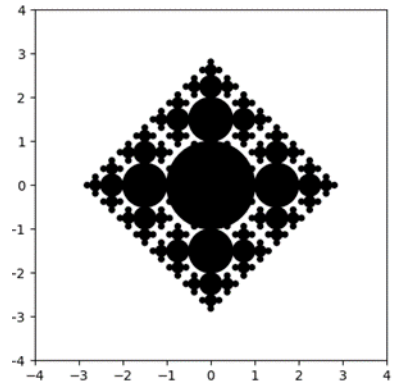
```
import matplotlib.pyplot as plt

plt.close('all')

def cercle(x,y,r):
    circle = plt.Circle((x,y),r,color='k')
    plt.gca().add_artist(circle)

def affiche(x_min,x_max,y_min,y_max):
    plt.xlim(x_min,x_max)
    plt.ylim(y_min,y_max)
    plt.show()
    plt.gca().set_aspect('equal', adjustable='box')

def bulles_1(n):
    def ...
        ...
    ...
    affiche(-2,4,-4,2)
    plt.figure()

def bulles_2(n):
    def ...
        ...
    ...
    affiche(-4,4,-4,4)
    plt.figure()

n = 5
bulles_1(n)
bulles_2(n)
```

Après avoir créé une figure vide avec « plt.figure() », les appels successifs de la fonction « cercle(x,y,r) » ajoutent à la figure les cercles voulus sans les afficher.

Enfin, l'appel de la fonction « affiche(x_min,x_max,y_min,y_max) » trace la figure dans l'intervalle demandé et remet le repère orthonormé avec equal.

Question 1 : compléter la fonction bulles_1 afin de réaliser l'image associée de manière récursive.

Question 2 : compléter la fonction bulles_2 afin de réaliser l'image associée de manière récursive.

n	bulle_1	bulle_2
1		
3		
5		

Remarque : pour bulle_2, la méthode programmée, pour obtenir les figures du tableau ci-dessus, crée les 4 cercles à chaque sous cercle. Dans le cas de bulle_2 pour n = 5, on voit apparaître des sous cercles proches du cercle de rayon unité.

Question 3 : si vous avez fait comme dans l'exemple, programmez une méthode qui évite cela.

D'après: D. DEFAUCHY