TD 13 : Fractales

Une **figure fractale** est un objet qui présente une **structure similaire à toutes les échelles.** Beaucoup d'objets fractals possèdent une invariance par dilatation (structure identique si l'on grossit une partie de la figure), cette propriété s'énonce en disant que la figure est **auto-similaire**, cette symétrie constitue **l'invariance d'échelle.** Un objet fractal est facilement défini comme la limite d'une suite récurrente convergente, ce qui permet d'en obtenir une représentation approchée par un terme de rang suffisamment élevé de la suite. Les modifications de la figure sont de plus en plus fines au fil des itérations et l'œil ne distingue plus l'approximation de la limite. On envisage la construction impérative, puis récursive, du **flocon de Koch** et du **triangle de Sierpinski**.

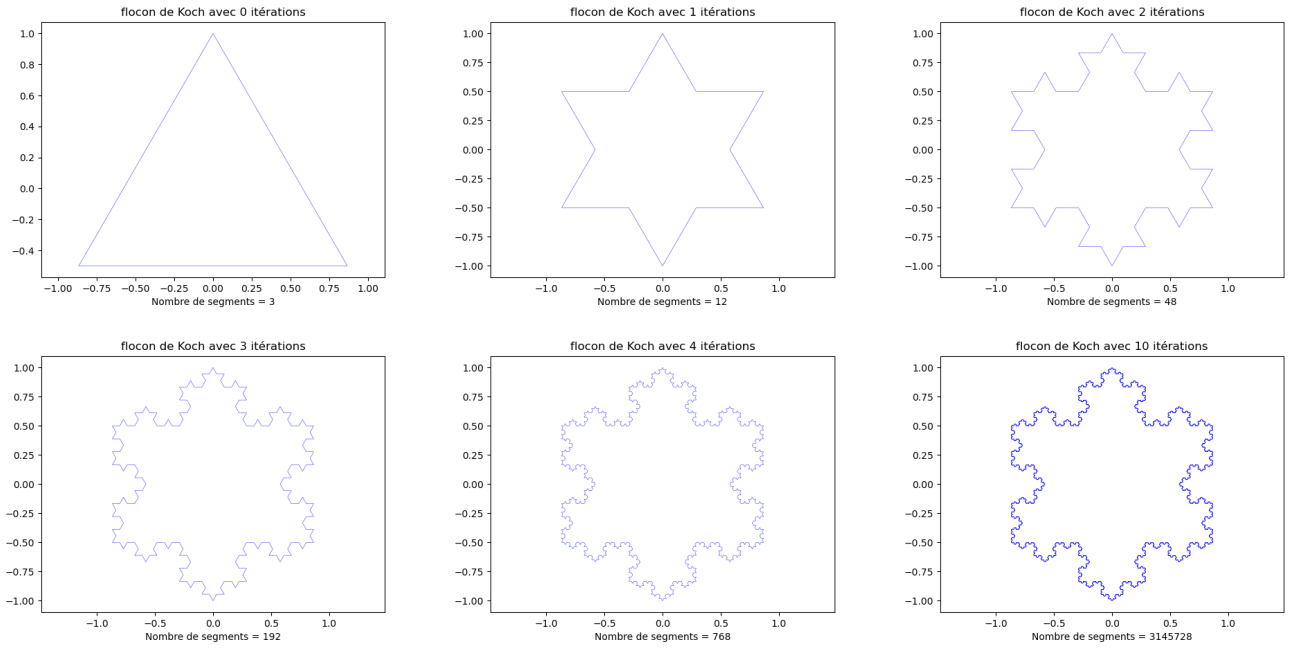
Point méthode

Les deux figures étudiées s'appuient sur un triangle équilatéral de base (A,B,C) de centre O, tel que $A(0,1)$. Les points du plan sont repérés par leur affixe, par exemple $A = j$ (avec $j^2 = -1$). On utilise le module `cmath` de Python, notamment pour la fonction `rect` qui permet de définir un complexe par son module et son argument. On récupère les parties réelle et imaginaire d'un complexe `z` par `z.real` et `z.imag`. Le nombre `pi` est défini dans le module `cmath`. Ex : `z=2+1j`.

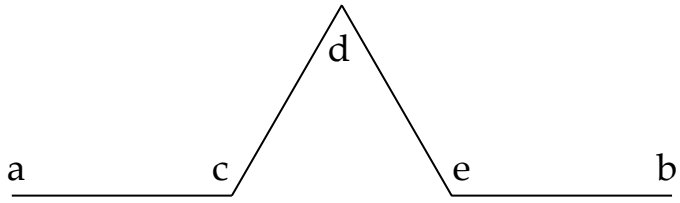
Exercice 1 (Travail préliminaire)

1. Rappeler comment trouver l'affixe du milieu I d'un bipoint (A, B) à partir des affixes de A et B.
2. Rappeler comment trouver l'affixe d'un vecteur $\overrightarrow{AB}$ à partir des affixes de A et B.
3. Rappeler comment obtenir l'affixe z' du vecteur $\overrightarrow{OM'}$ à partir de l'affixe z du vecteur $\overrightarrow{OM}$ si $\overrightarrow{OM'}$ s'obtient à partir de $\overrightarrow{OM}$ par une rotation d'angle α.
4. Trouver les représentations trigonométriques des affixes de B et C du triangle de base.

Flocon de Koch



À chaque itération, tout segment $[a, b]$ est remplacé par 4 segments $[a, c]$, $[c, d]$, $[d, e]$ et $[e, b]$.



Le code suivant est utilisé en préambule ; la fonction courbe est appelée pour tracer le flocon de Koch. On a déclaré $n = 8$ mais ce nombre d'itérations peut être choisi par l'utilisateur.

```
import matplotlib.pyplot as plt
from cmath import rect, pi
A, B, C = 1j, rect(1, -pi/6), rect(1, -5*pi/6)
r, n = rect(1, pi/3), 8
def courbe(x,y):
    plt.plot(x, y, 'b', linewidth = 0.2)
    plt.axis('equal')
    plt.show()
```

Exercice 2 (Flocon de Koch - version impérative)

Un point est représenté par son affixe.

1. Donner les affixes des points c, d et e en fonction des affixes de a et b.
2. Écrire une version impérative de tracé du flocon de Koch qui construit un tableau (liste Python) des affixes des points anguleux de la courbe. Lors de chaque itération, on parcourt l'ensemble `P` des points (affixes) obtenus lors de l'itération précédente et l'on insère 3 points entre deux points consécutifs pour constituer le nouveau tableau `Q`. On peut se contenter de compléter le programme suivant :

```
P = [B, C, A]
if n:
    a = A
    for _ in range(n):
        Q = []
        for b in P:
            s = a + b
            c, e =  (s + a) / 3, .....
            a, d = ....., c + r * .....
            Q += [....., ....., ....., .....]
        P = Q
x = .....
y = .....
courbe(x,y)
```

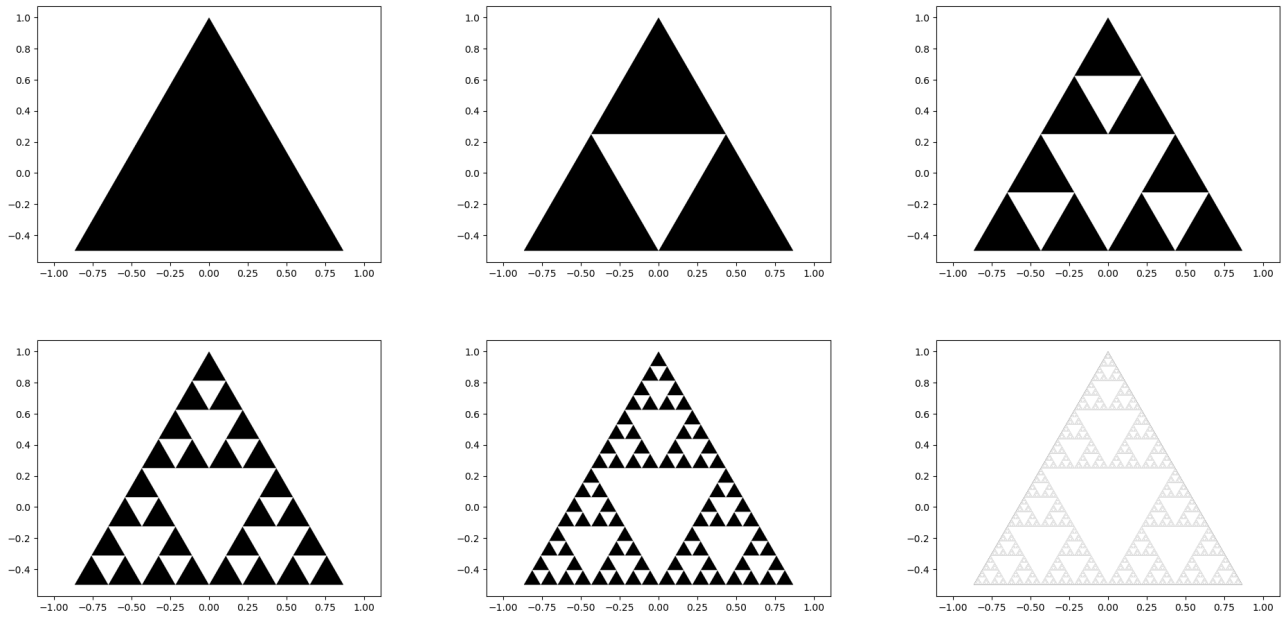
Exercice 3 (Flocon de Koch - version récursive)

La nature récursive d'une courbe fractale incite à privilégier l'usage d'une fonction récursive de manière à n'utiliser aucune structure de données autre que le tableau de points définitifs.

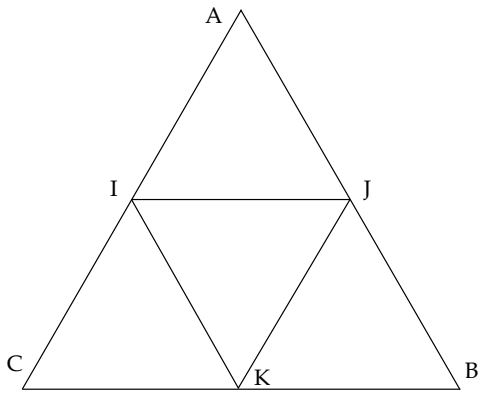
1. Écrire une version récursive de tracé du flocon de Koch qui construit un tableau (liste Python) des affixes des points anguleux de la courbe. Chaque nouveau segment fait l'objet d'un appel récursif. On peut se contenter de compléter le programme suivant :

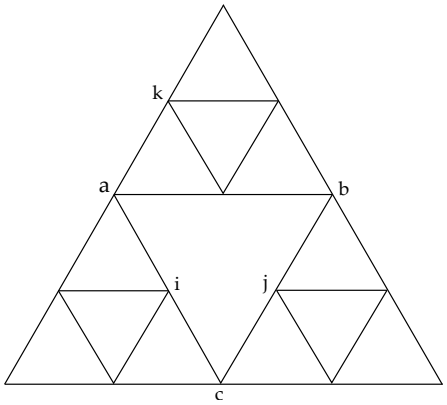
```
x, y = [0], [1]
def koch(a, b, n):
    if n:
        s = a + b
        c, e = (s + a) / 3, .....
        d = c + r * .....
        koch(a, c, n - 1)
        koch(....., ....., .....)
        koch(....., ....., .....)
        koch(....., ....., .....)
    else:
        x.append(b.real)
        y.append(.....)
koch(A, B, n) ; koch(....., ....., n) ; koch(....., ....., n)
courbe(x,y)
```

Triangle de Sierpinski



À chaque itération, on ajoute trois triangles blancs autour des triangles blancs ajoutés à l'itération précédente. Le code suivant est utilisé en préambule ; on utilise la fonction `Polygon` du module `matplotlib.patches` pour colorier un triangle (en blanc ou noir). On a déclaré $n = 8$ mais ce nombre d'itérations pourra être choisi par l'utilisateur.






```
from matplotlib.patches import Polygon
import matplotlib.pyplot as plt
from cmath import rect, pi
def xy(A): return A.real, A.imag
def triangle(A, B, C, c):
    ax.add_patch(Polygon([xy(A), xy(B), xy(C)],
                 color = c, fill = True, linewidth = 0.2))
n = 8
A, B, C = 1j, rect(1, -pi/6), rect(1, -5*pi/6)
```

Exercice 4 (Triangle de Sierpinski - version impérative)

Utiliser les schémas précédents pour réfléchir.

1. Donner les affixes des points i, j et k à partir des affixes a, b et c.
2. Écrire une version impérative de tracé du triangle de Sierpinski qui construit un tableau dynamique (liste Python) de triplets (3-tuples Python) représentant les sommets des triangles blancs. Lors de chaque itération, on considère les triangles ajoutés lors de l'itération précédente et pour chacun, on ajoute trois triplets en fin de tableau. On peut se contenter de compléter le programme suivant :

```
ax = plt.figure().add_subplot()
triangle(A, B, C, 'black')
if n :
    d = (B - C) / 2
    T = [((A + C) / 2, ....., C + d)]
    for k in range(n - 1):
        d /= 2
        for a, b, c in T[-3**k:]:
            i, j = ....., .....
            k = .....
            T.append((k, k + d, a + d))
            T.append((....., ....., .....))
            T.append((....., ....., .....))
    for t in T:
        triangle(*t, 'white')
plt.axis('equal')
plt.show()
```

Exercice 5 (Triangle de Sierpinski - version récursive)

1. Écrire une version récursive de tracé du triangle de Sierpinski. Colorier le triangle central (I,J,K) en blanc et réaliser trois appels récursifs sur les triangles noirs qui l'entourent. On peut se contenter de compléter le programme suivant :

```
def sierpe(A, B, C, n):
    if n:
        I, J, K = ....., ....., .....
        triangle(....., ....., ..... 'white')
        sierpe(....., ....., n - 1)
        sierpe(....., ....., .....)
        sierpe(....., ....., .....)
ax = plt.figure().add_subplot()
triangle(....., ....., ....., 'black')
sierpe(....., ....., ....., n)
plt.axis('equal')
plt.show()
```

Pour s'entraîner

Reprendre les méthodes précédentes pour dessiner un tapis de Sierpinski :