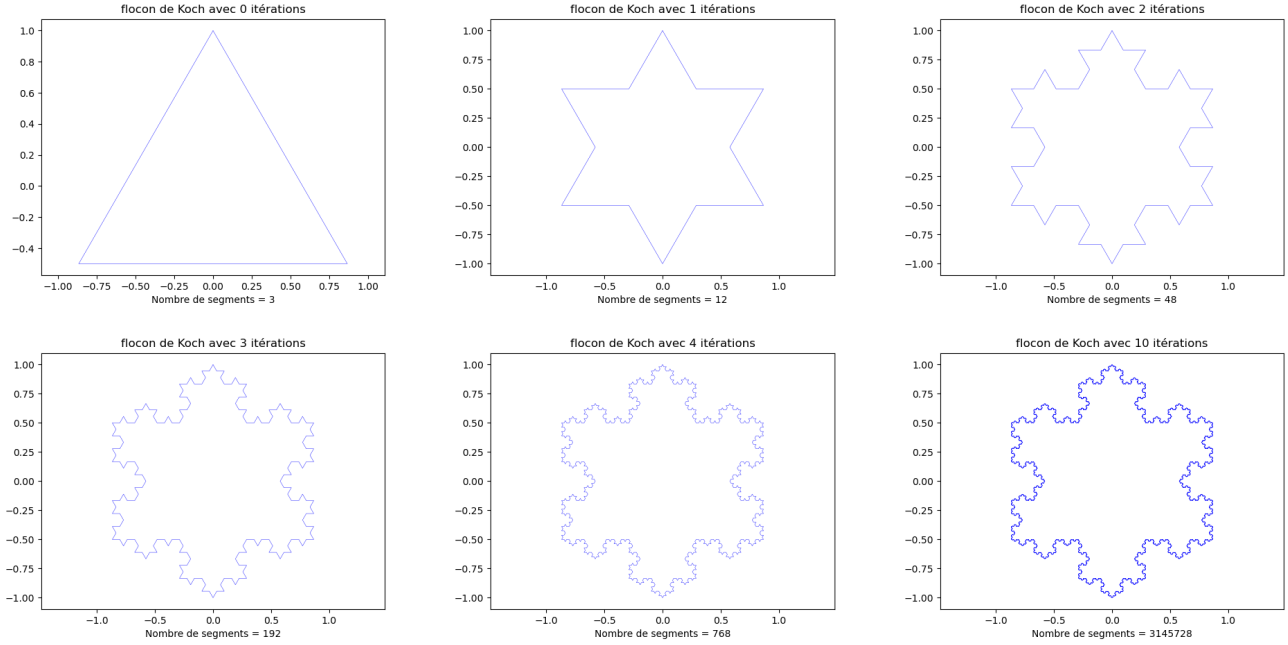
MPSI - ITC - TD 14 - Fractales

Une **fractale** est une figure qui se répète à l'intérieur d'elle-même, dit autrement, il y a une invariance par dilatation (structure identique si l'on grossit une partie de la figure). On parle alors de **figure auto-similaire**, ou encore **d'invariance d'échelle.** Il faudrait une précision infinie pour tracer une vraie figure fractale (ce qui est donc impossible). Cependant, on peut définir des suites de figures qui se rapprochent de plus en plus d'une fractale, et à partir d'un certain rang dans la suite l'image qu'on a construit devient indistinguable à l'œil nu de la vraie fractale. C'est ce que nous allons faire dans ce TD. On s'intéressera à deux fractales : le **flocon de Koch** et le **triangle de Sierpinski**.

Flocon de Koch



Conseils de tracé

Si vous voulez obtenir des courbes comme celles ci-dessus, on vous conseille de

- Imposer la même échelle sur votre axe vertical et horizontal, pour éviter un étirement horizontal ou vertical
- Utiliser une épaisseur de tracé de 0.2
- Tracer en bleu

Si vous avez utilisé la ligne d'import `import matplotlib.pyplot as plt`, les conseils précédents peuvent être appliqués en

- Réalisant l'appel `plt.axis('equal')` avant d'appeler `plt.show`
- En passant l'argument nommé `linewidth = 0.2` quand vous appellerez `plt.plot`
- En passant l'argument nommé `color = 'blue'` quand vous appellerez `plt.plot`

La ligne où vous appelez `plt.plot` peut donc ressembler à :

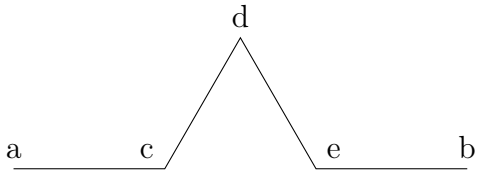
```
plt.plot(x, y, color='blue', linewith=0.2)
```

Exercice 1 (Tracé)

Définir une fonction `trace(x, y)` qui prend en entrée une liste d'abscisses `x` et une liste d'ordonnées correspondantes `y`, et qui trace la courbe qui relie dans l'ordre les points décrits par `x` et `y`.

La suite qui approxime le flocon de Koch est définie comme suit :

- À l'ordre 0, il y a juste un triangle équilatéral (A, B, C) de centre $O(0, 0)$ et sommet $A(0, 1)$.
- Pour passer de l'ordre n à l'ordre $n + 1$, on remplace chaque segment $[a, b]$ apparaissant dans le flocon de Koch d'ordre n par 4 segments $[a, c]$, $[c, d]$, $[d, e]$ et $[e, b]$, définis par



Les nombres complexes en python

Les deux figures étudiées s'appuient sur un triangle équilatéral (A,B,C) de centre $O(0, 0)$ et sommet $A(0, 1)$. Ici on repérera les points du plan par leur affixe. Par exemple, si le point A a pour coordonnées $(0, 1)$, son affixe est i (avec $i^2 = -1$). Si vous voulez définir un complexe par sa partie réelle et sa partie imaginaire, vous pouvez utiliser la notation *partie réelle* + *partie imaginaire* `j`, dans laquelle vous pouvez omettre la partie réelle si elle vaut 0. Par exemple, l'expression python `1 + 2j` s'évalue vers le complexe $1 + 2i$; et pour parler de l'affiche i en python, on pourra écrire `1j`. On récupère les parties réelle et imaginaire d'un complexe `z` par `z.real` et `z.imag`. On utilise le module `cmath` de Python pour la manipulation de nombres complexes. Notamment la fonction `rect` du module `cmath` qui permet de définir un complexe par son module et son argument (si vous voulez comprendre pourquoi cette fonction s'appelle `rect`, pensez à regarder sa documentation, avec `help(cmath.rect)` dans un contexte où vous aurez importé le module `cmath`).

Exercice 2 (Questions préliminaires sur les affixes)

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. Si (A, B, C) est un triangle équilatéral de centre l'origine du repère $O(0, 0)$, trouver les représentations trigonométriques des affixes de B et C.

Exercice 3 (Flocon de Koch - version récursive)

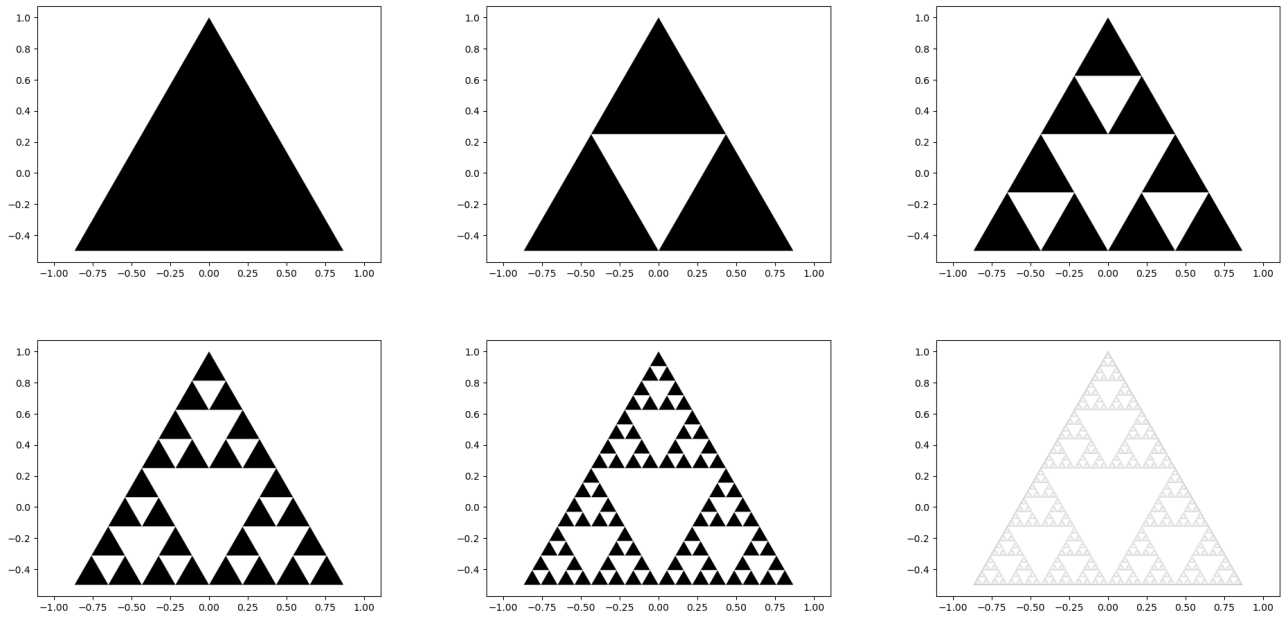
La nature récursive d'une courbe fractale incite à privilégier l'usage d'une fonction récursive.

1. Un point est représenté par son affixe. Donner les affixes des points c, d et e en fonction des affixes de a et b.
2. Écrire un code qui utilise de la récursivité pour trouver les affixes des points anguleux du flocon de Koch. 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 > 0:
        c = ...
        d = ...
        e = ...
        koch(a, c, n - 1)
        koch(....., ....., .....)
        koch(....., ....., .....)
        koch(....., ....., .....)
    else:
        x.append(b.real)
        y.append(.....)

A = ....
B = .....
C = .....
koch(A, B, n)
koch(....., ....., n)
koch(....., .....,  n)
trace(x,y)
```

Triangle de Sierpinski



La suite qui approxime le triangle de Sierpinski est définie comme suit :

— À l'ordre 0, on commence par un triangle équilatéral noir

— Pour passer de l'ordre n à l'ordre $n+1$, on remplace chaque triangle noir de l'ordre n par un triangle blanc et trois triangles noirs, comme suit :

Exercice 4 (Triangle de Sierpinski - version récursive)

Le fichier `mpsi_itc_td_14_code_fourni.py` contient des fonctions `initialise_figure()` et `triangle(A, B, C, couleur)` qui vous aideront à tracer le triangle de Sierpinski.

1. Écrire une version récursive de tracé du triangle de Sierpinski. On peut se contenter de compléter le code suivant :

```
def sierpi_aux(A, B, C, n):
    if n > 0:
        I = ...
        J = ...
        K = ...
        triangle(... , ...., ...., 'white')
        sierpi_aux(.... , ....., ....., n - 1)
        sierpi_aux(....., ....., ......, .....)
        sierpi_aux(....., ....., ......, .....)
    else:
        triangle(...., ......, ......., 'black')

def sierpi(n):
    A = .....
    B = .....
    C = .....
    initialise_figure()
    sierpi_aux(A, B, C, n)
    plt.axis('equal')
    plt.show()
```

Exercice 5 (Bonus : version itératives)

1. Écrire du code qui trace le flocon de koch et le triangle de Sierpinski sans utiliser de fonctions récursives.
2. Comparer les versions récursives aux versions itératives. Lesquelles sont plus simples ?

Exercice 6 (Bonus : tapis de Sierpinski)

Écrire un code permettant de tracer le tapis de Sierpinski.