TP 4 : Flocon de Koch

1 - Introduction

L'objectif de ce TP est d'étudier la fonction la convolution et de l'appliquer pour générer un Flocon de Koch

Le TP sera à réaliser en python 3. Les librairies utilisées sont installées sur les machines de l'université, vous pouvez néanmoins les installer sur vos propres machines à l'aide de l'utilitaire pip présent par défaut avec python.

N'hésitez pas à regarder régulièrement la documentation de ces librairies, des exemples d'utilisation accompagnent généralement l'explication de chaque fonction.

- Python 3: https://docs.python.org/3/
- Numpy: https://docs.scipy.org/doc/numpy/reference/
- Scipy: https://docs.scipy.org/doc/scipy/reference/
- Matplotilb: https://matplotlib.org/contents.html

À part si cela est précisé, vous ne devez pas utiliser directement de boucle (f or , $whi \leq$) ou de branchement conditionnel (if) durant ce TP..

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sc
import scipy.signal
```

Afin de vous guider dans la détection d'erreur dans votre code, nous avons introduit des blocs de tests. Il n'est pas nécessaire que vous compreniez en détail le code de ces blocs. Vous devez uniquement les exécuter et corriger les erreurs de votre code si un des tests n'est pas valide. Il est important de noter que le fait de valider le test ne garantit pas que votre code ne contient pas d'erreur. Par contre un test non validé implique nécessairement que votre code contient une erreur.

- Si tous les tests sont valides, vous aurez un message écrit en vert indiquant : Ok - Tous les tests sont validés.
- Si un des tests n'est pas valide, vous aurez un message écrit en rouge indiquant : Au moins un test n'est pas validé.
- Pour les tests non valides, vous aurez des éléments d'information sur le test non valide. En particulier, un message écrit en jaune vous détaillera la nature du test échoué.

In [2]:
```
# YOUR CODE HERE
a =  42
```

In [3]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(a,42,err_msg="\033[93m {}\033[00m" .format('Te
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

2 - Outils introductifs

2.1 - Étude de la convolution sur plusieurs filtres

Nous allons dans cette partie étudier l'effet de la convolution avec différents types de filtres sur des fonctions sinusoïdales.

Définissez une variable $f = 1$Hz correspondant à la fréquence d'un signal sinusoïdale.

In [4]:
```
# YOUR CODE HERE
f = 1
```

Construisez un tableau `x` contenant 2000 valeurs de -5 à 5.

In [5]:
```
# YOUR CODE HERE
x = np.linspace(-5, 5, 2000)
```

In [6]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x.shape,(2000,),err_msg="\033[93m {}\033[00m"
    np.testing.assert_equal(x[0],-5,err_msg="\033[93m {}\033[00m" .format(
    np.testing.assert_equal(x[-1],5,err_msg="\033[93m {}\033[00m" .format(
    np.testing.assert_almost_equal(x[42],-4.789894947473737,err_msg="\033[9
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Construisez un tableau `y1` correspondant à l'équation $\sin(2\pi f x)$.

In [7]:
```
# YOUR CODE HERE
y1 = np.sin(2 * np.pi * x)
```

In [8]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(y1.shape,(2000,),err_msg="\033[93m {}\033[00m"
    np.testing.assert_almost_equal(y1[0],1.22464680e-15,err_msg="\033[93m
    np.testing.assert_almost_equal(y1[-1],1.22464680e-15,err_msg="\033[93m
    np.testing.assert_almost_equal(y1[64],0.9043983406927987,err_msg="\033
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

2.1.1 Filtrage par un filtre porte

Définissez un vecteur aléatoire e tiré selon une loi normale, de même taille que le tableau x.
Définissez et affichez la variable $y1e = y1 + 0.1e$.

In [9]:
```
# YOUR CODE HERE
e = np.random.standard_normal(size=np.shape(x))
y1e = y1 + 0.1 * e
```

In [10]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(y1e.shape,(2000,),err_msg="\033[93m {}\033[00m
    np.testing.assert_array_less(np.mean(np.abs(y1-y1e)),1e-1,err_msg="\03
    np.testing.assert_array_less(np.abs(0.1-np.std(np.abs(y1-y1e))),1e-1,e
    np.testing.assert_array_less(1e-1,np.max(np.abs(y1-y1e)),err_msg="\033
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Construisez et affichez un filtre `filtrePorte` défini par:

$$\begin{cases} 1 & \text{si } |x| < s \\ 0 & \text{sinon} \end{cases}$$

Vous pouvez fixer le seuil s à 10^{-1}.

In [11]:
```
# YOUR CODE HERE
s =  10 ** -1
filtrePorte = np.zeros_like(x)
filtrePorte[np.abs(x) < s] = 1
```

In [12]:

```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(filtrePorte.shape,(2000,),err_msg="\033[93m {}
    np.testing.assert_equal(np.sum(filtrePorte!=1)+np.sum(filtrePorte==1),
    np.testing.assert_equal(np.sum(filtrePorte),40,err_msg="\033[93m {}\03
    np.testing.assert_equal(np.sum(filtrePorte[980:(980+40)]),40,err_msg="
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

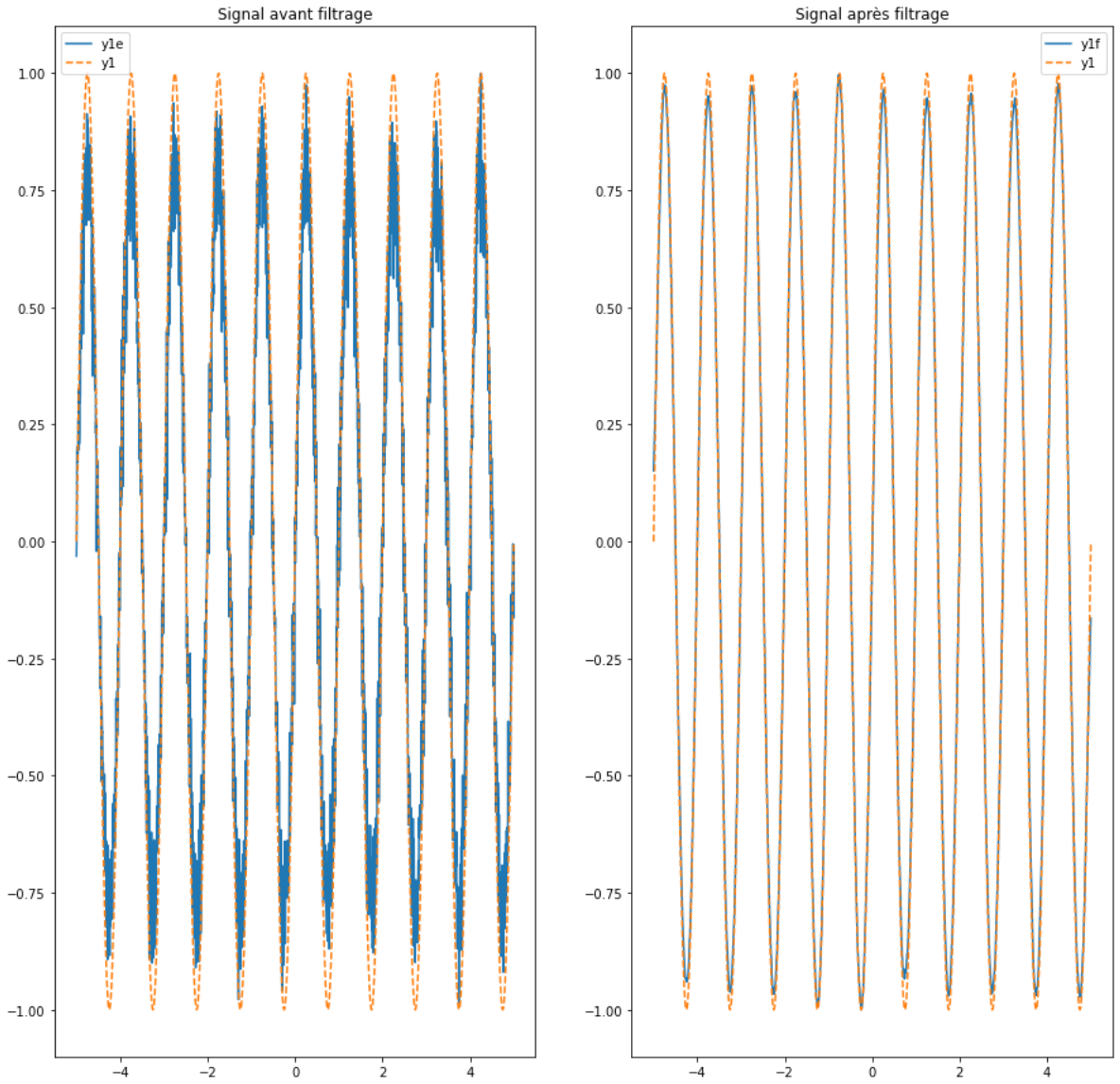
```
Ok - Tous les tests sont validés.
```

Il est possible de supprimer le bruit du signal `y1e` en utilisant un filtre moyenneur tel que la porte précédemment définie.

En utilisant la fonction `sc.signal.convolve` de scipy (avec le paramètre `'same'`), convoluez le signal `y1e` avec `filtrePorte` pour donner le vecteur `y1f`. Divisez le résultat par le maximum de la valeur absolue de ses valeurs (afin d'avoir des valeurs dans l'intervalle $[-1, 1]$). Affichez le résultat ainsi normalisé sur une figure. Que constatez-vous par rapport au signal `y1` ?

In [13]:

```
# YOUR CODE HERE
y1f = sc.signal.convolve(y1e, filtrePorte, mode="same")
fig, axs = plt.subplots(1,2,figsize=(15,15))
axs[0].plot(x,y1e/np.max(np.abs(y1e)))
axs[0].plot(x,y1,'--')
axs[0].legend(('y1e','y1'))
axs[0].set_title('Signal avant filtrage')

axs[1].plot(x,y1f/np.max(np.abs(y1f)))
axs[1].plot(x,y1,'--')
axs[1].legend(('y1f','y1'))
axs[1].set_title('Signal après filtrage')
plt.show()
```



In [14]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(y1f.shape,(2000,),err_msg="\033[93m {}\033[00m
    np.testing.assert_array_less(np.mean(np.abs(y1-y1f/np.max(y1f))),5e-2,
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

2.1.2 Filtrage par un filtre sinus cardinal

Le filtrage par un sinus cardinal permet de faire un filtre passe-bas qui ne laisse passer que les fréquences inférieures à une fréquence de coupure. On va illustrer ce phénomène ici en combinant deux signaux sinusoïdaux de fréquences différentes puis en appliquant un filtrage avec une fréquence de coupure bien choisie pour ne laisser passer qu'un des deux signaux d'origine.

Construisez un tableau `y2` correspondant à l'équation $\sin(2\pi 2fx)$.

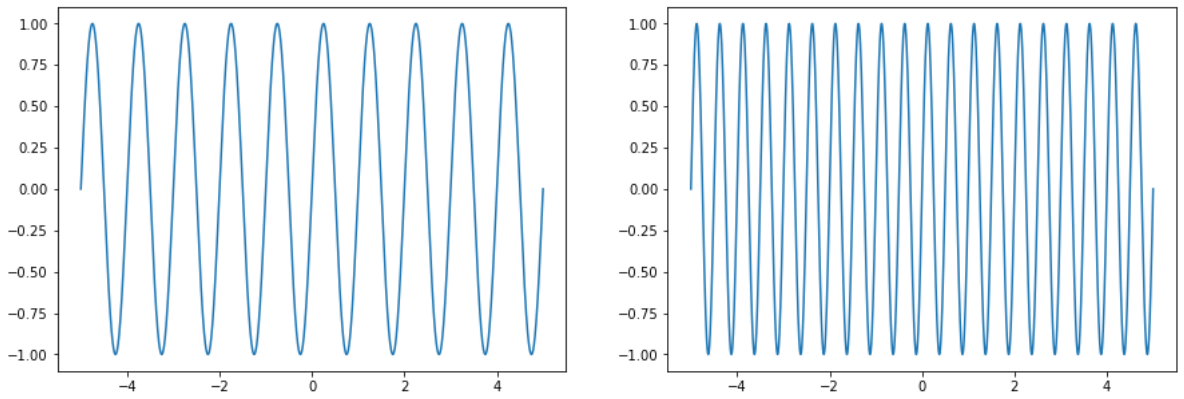
In [15]:
```
# YOUR CODE HERE
y2 = np.sin(2 * np.pi * 2 * f * x)
```

In [16]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(y2.shape,(2000,),err_msg="\033[93m {}\033[00m"
    np.testing.assert_almost_equal(y2[0],1.22464680e-15,err_msg="\033[93m
    np.testing.assert_almost_equal(y2[-1],1.22464680e-15,err_msg="\033[93m
    np.testing.assert_almost_equal(np.sum(np.abs(y2)),1272.6026630324313,e
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Affichez `y1` et `y2` sur deux figures différentes.

In [17]:
```
# YOUR CODE HERE
fig, axs = plt.subplots(1,2,figsize=(15,5))
axs[0].plot(x, y1)
#axs[0].legend(('y1e','y1'))
#axs[0].set_title('Signal avant filtrage')

axs[1].plot(x,y2)
#axs[1].plot(x,y1,'--')
#axs[1].legend(('y1f','y1'))
#axs[1].set_title('Signal après filtrage')
plt.show()
```



Définissez et affichez la fonction $y12 = \frac{y1+y2}{\max(y1+y2)}$. Ce signal correspond à la combinaison des deux signaux sinusoïdaux `y1` et `y2` .

In [18]:
```
# YOUR CODE HERE
somme = y1 + y2
y12 = somme/ np.max(somme)
```

In [19]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(y12.shape,(2000,),err_msg="\033[93m {}\033[00m
    np.testing.assert_almost_equal(y12[0],2.0872629244849403e-15,err_msg="
    np.testing.assert_almost_equal(y12[-1],-2.0872629244849403e-15,err_msg=
    np.testing.assert_almost_equal(np.sum(np.abs(y12)),903.748871217134,er
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas 
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Construisez un filtre `filtreSinc` défini par:

$$\mathrm{sinc}(2f_s x)$$

sinc est le sinus cardinal défini dans numpy. Vous pouvez fixer la fréquence f_s à 1.5. La taille de ce vecteur sera la même que `x` .

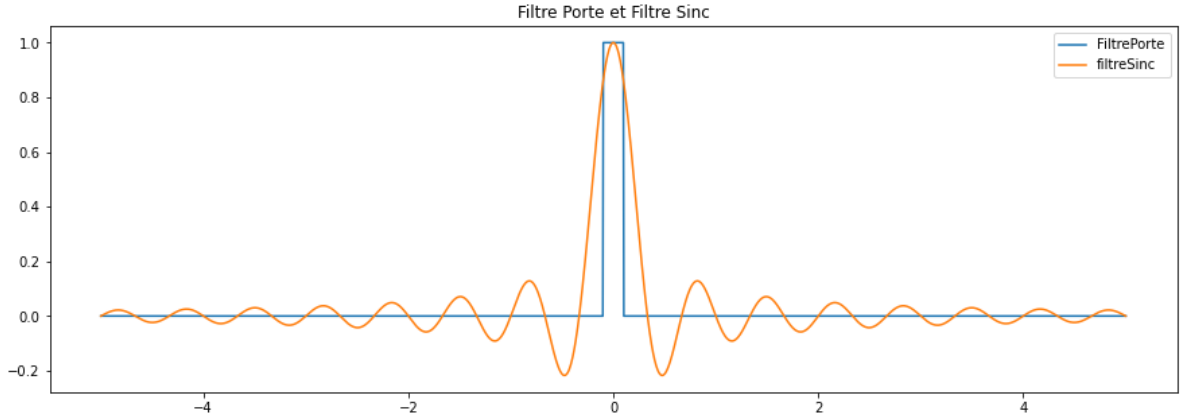
In [20]:
```
# YOUR CODE HERE
fs = 1.5
filtreSinc = np.sinc(2 * fs * x)
```

In [21]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(filtreSinc.shape,x.shape,err_msg="\033[93m {}\(
    np.testing.assert_almost_equal(filtreSinc[0],1.1437264410803267e-16,er
    np.testing.assert_almost_equal(filtreSinc[1000],0.9999073824352694,err_
    np.testing.assert_almost_equal(np.sum(np.abs(filtreSinc)),151.257372088
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas 
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

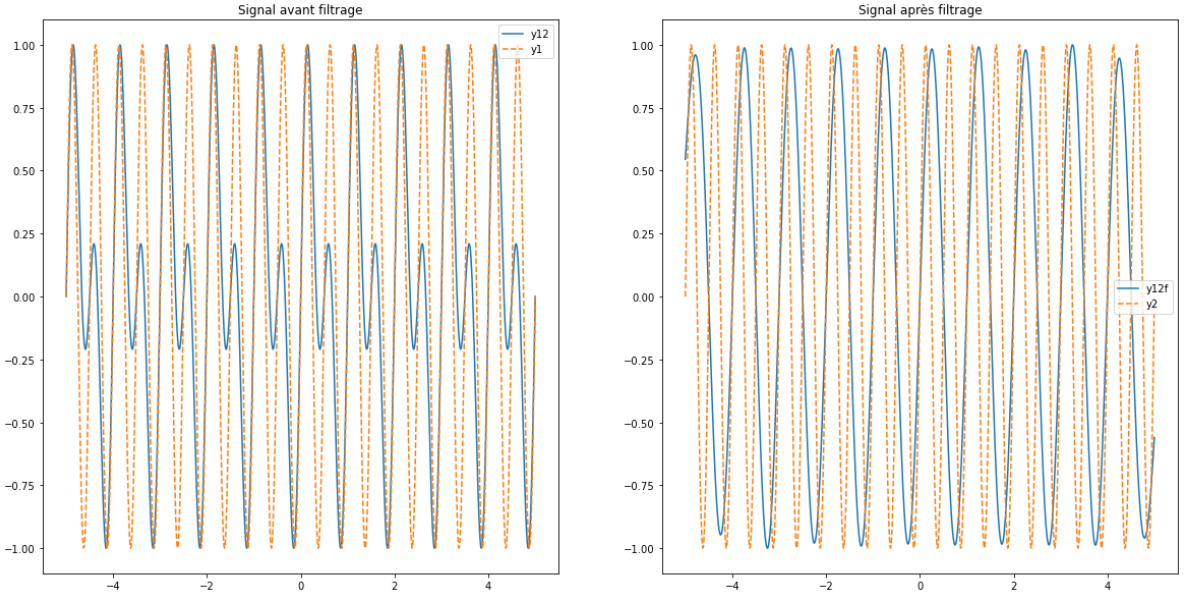
Affichez dans une même figure les filtres `filtrePorte` et `filtreSinc` .

In [22]:
```
# YOUR CODE HERE
fig, axs = plt.subplots(1,1,figsize=(15,5))
axs.plot(x, filtrePorte, label="FiltrePorte")
axs.plot(x,filtreSinc, label="filtreSinc")
axs.legend()
axs.set_title('Filtre Porte et Filtre Sinc')

plt.show()
```



En utilisant la fonction `sc.signal.convolve` de scipy (avec le paramètre `'same'`), convoluez le signal `y12` avec `filtreSinc`. Divisez le résultat par la valeur maximale et affichez le sur une figure. Que constatez-vous par rapport au signal `y1` ?

In [23]:
```
# YOUR CODE HERE
fig, axs = plt.subplots(1, 2, figsize=(20, 10))
y12f = sc.signal.convolve(y12, filtreSinc, mode="same")

axs[0].plot(x,y12/np.max(np.abs(y12)))
axs[0].plot(x,y2,'--')
axs[0].legend(('y12','y1'))
axs[0].set_title('Signal avant filtrage')

axs[1].plot(x,y12f/np.max(np.abs(y12f)))
axs[1].plot(x,y2,'--')
axs[1].legend(('y12f','y2'))
axs[1].set_title('Signal après filtrage')

plt.show()
```



2.2 Rotation

Nous allons dans cette partie étudier comment faire des rotations de points dans le plan 2D. Cela nous sera utile pour construire le flocon de Koch dans la suite du TP.

Déclarez un tableau `x` de taille 2 par 2 contenant les points de coordonnées (0,0) et (1,1).

Une ligne du tableau correspondra à un point.

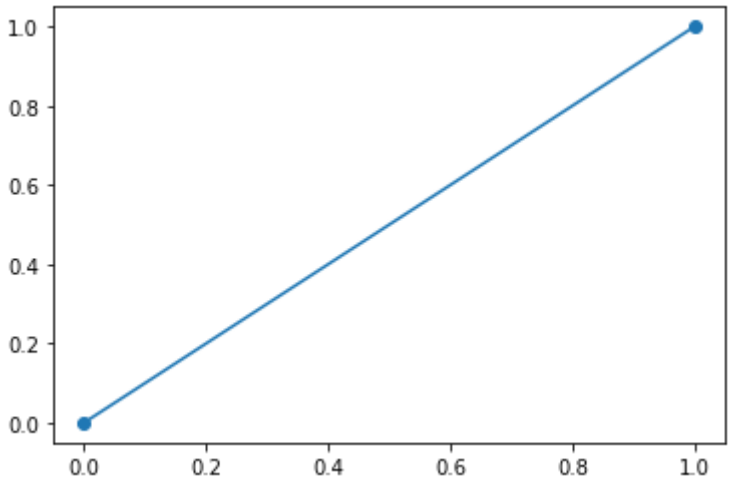
In [24]:
```
x = np.array([[0, 0],[1, 1]])
```

In [25]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x.shape,(2,2),err_msg="\033[93m {}\033[00m" .f
    np.testing.assert_equal(x.reshape(-1),[0,0,1,1],err_msg="\033[93m {}\0
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Affichez les deux points précédents reliés par une droite.

In [26]:
```
# YOUR CODE HERE
plt.scatter(x[:, 0], x[:, 1])
plt.plot(x[:,0], x[:,1])
plt.show()
```



Définissez une fonction `get_rotation_matrix` qui prend en argument un angle `t` et qui retourne la matrice $\begin{bmatrix} \cos(t) & \sin(t) \\ -sin(t) & cos(t) \end{bmatrix}$.

In [27]:
```
def get_rotation_matrix(t):
    return np.array([[np.cos(t), np.sin(t)], [- np.sin(t), np.cos(t)]])
```

In [28]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    _x_ = np.random.rand(int(1e3))*2*np.pi
    for i in range(len(_x_)):
        tmp = get_rotation_matrix(_x_[i])
        tmp2 = tmp**2
        np.testing.assert_almost_equal(np.sum(np.diag(tmp2))+np.sum(np.diag
        np.testing.assert_almost_equal(np.prod(np.diag(tmp))-np.prod(np.dia
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.

del _x_,tmp,tmp2
```

```
Ok - Tous les tests sont validés.
```

Faire la rotation de centre (0,0) d'un point `a` par un angle t revient à multiplier le vecteur ligne contenant les coordonnées de `a` par la matrice de rotation de la question précédente. Attention, on parle ici de produit matriciel et non d'un produit terme à terme.

En utilisant la fonction `get_rotation_matrix`, faites une rotation de $\frac{\pi}{3}$ des points (0,0) et (1,1) que vous aviez stockés dans la variable `x`. Vous stockerez le résultat dans une variable `xr`.

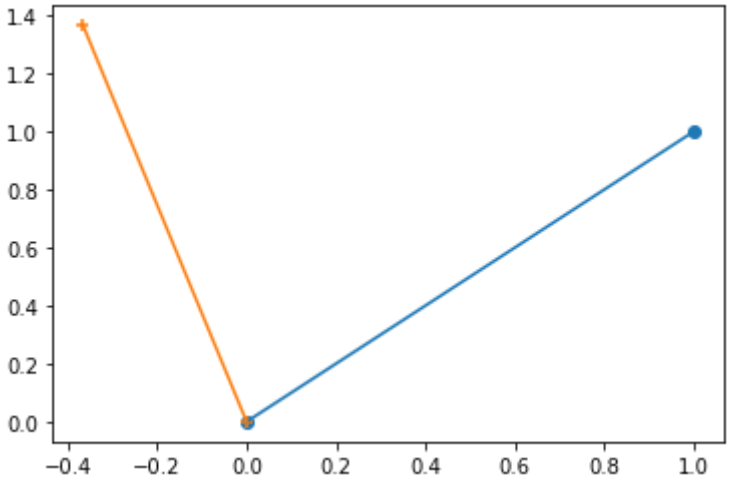
In [29]:
```
# YOUR CODE HERE
xr = np.dot(x, get_rotation_matrix(np.pi/3))
```

In [30]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_almost_equal(xr,np.array([[0,0],[-0.3660254,  1.3660
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Affichez le segment allant du point (0,0) au point (1,1) ainsi que la rotation de ce segment d'un angle de $\frac{\pi}{3}$.

In [31]:
```
# YOUR CODE HERE
plt.scatter(x[:, 0], x[:, 1])
plt.plot(x[:,0], x[:,1])

plt.scatter(xr[:, 0], xr[:, 1], marker="+")
plt.plot(xr[:,0], xr[:,1])

plt.show()
```

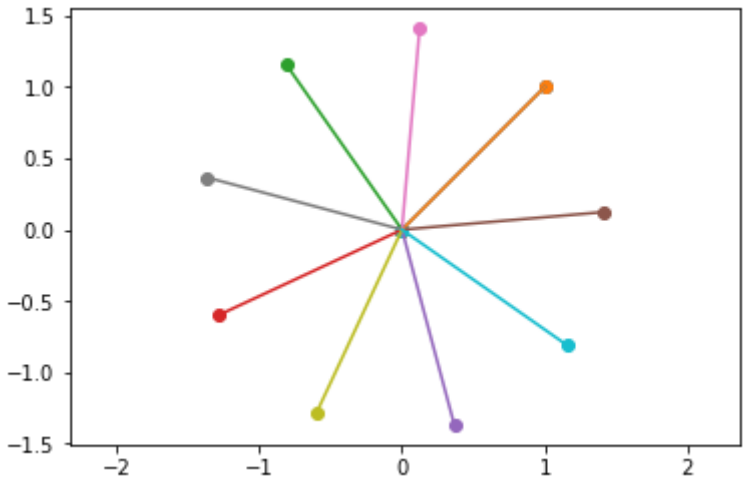


En utilisant une boucle `for` affichez 10 rotations successives partant du segment allant du point (0,0) au point (1,1) tel que les angles entre chaque rotation soient les mêmes et que la dernière rotation vous ramène sur le segment initial.

In [32]:

```
# YOUR CODE HERE
X = np.array([[0, 0],[1, 1]])
angles = np.linspace(-2 * np.pi, 2 * np.pi, 10)
print(angles)
for angle in angles:
    plt.scatter(X[:, 0], X[:, 1])
    plt.plot(X[:,0], X[:,1])
    X = np.dot(x, get_rotation_matrix(angle))

plt.axis("equal")
plt.show()
```

```
[-6.28318531 -4.88692191 -3.4906585  -2.0943951  -0.6981317   0.6981317
  2.0943951   3.4906585   4.88692191  6.28318531]
```



Dans le cas où la rotation n'est pas centrée sur l'origine mais correspond à un centre de coordonnée `c`, il faut calculer $(x-c)R+c$ au lieu de xR. Faites une rotation du segment (0,0)-(1,1) d'un angle de $\frac{\pi}{3}$ par rapport au centre de coordonnée (2,1). Vous placerez le résultat dans la variable `xr2`.

In [33]:
```
# YOUR CODE HERE
c = np.array([[2, 1]])
R = get_rotation_matrix(np.pi / 3)
xr2 = np.dot((x - c), R) + c
```

In [34]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_almost_equal(xr2,np.array([[ 1.8660254, -1.2320508],
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas 
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

2.2 - Division en trois d'un segment

Soit un segment défini par deux points p_1 et p_2 alors le point se trouvant à 1/3 du segment $[p_1, p_2]$ est le point $p_{1/3} = p_1 + \frac{p_2 - p_1}{3} = 2\frac{p_1}{3} + \frac{p_2}{3}$. En utilisant `x` (qui stocke les points $(0, 0)$ et $(1, 1)$), calculez ce point pour le segment $(0, 0) - (1, 1)$. Vous nommerez le résultat `x1_3` .

In [35]:
```
# YOUR CODE HERE
x1_3 = (2 * x[0, :] + x[1, :]) / 3
```

In [36]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_almost_equal(x1_3,np.ones(2)/3,err_msg="\033[93m {}\(
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas 
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Faites de même pour le point se trouvant au 2/3 du segment que vous nommerez `x2_3` .

In [37]:
```
# YOUR CODE HERE
p1 = x[0, :]
p2 = x[1, :]
x2_3 = p1 + 2 * (p2 - p1) / 3
```

In [38]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_almost_equal(x2_3,np.ones(2)*2/3,err_msg="\033[93m {}
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

En utilisant un opérateur de concaténation sur les lignes, concaténez la valeur d'un tableau ligne contenant (0,0) avec `x1_3` , `x2_3` et la dernière ligne de `x` afin d'avoir une matrice de taille 4 par 2 que vous nommerez `x_end` .

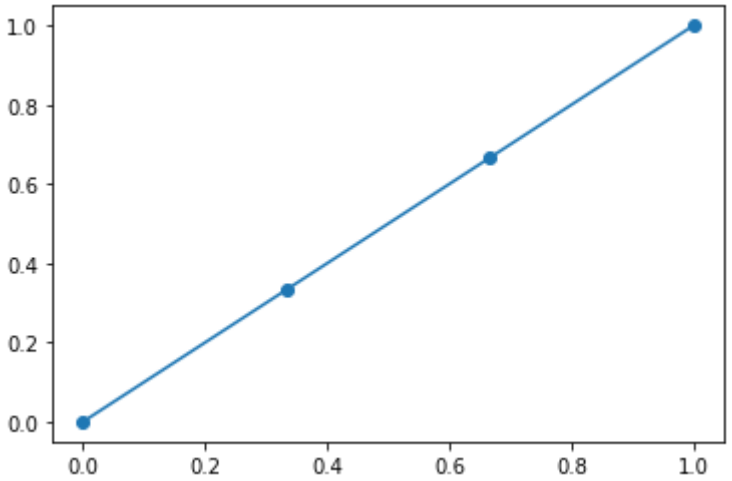
In [39]:
```
# YOUR CODE HERE
x_end = np.stack((np.zeros_like(x1_3), x1_3, x2_3, x[-1, :]), axis=0)
```

In [40]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x_end.shape,(4,2),err_msg="\033[93m {}\033[00m
    np.testing.assert_almost_equal(x_end[0,:],np.zeros(2),err_msg="\033[93
    np.testing.assert_almost_equal(x_end[1,:],x1_3,err_msg="\033[93m {}\03
    np.testing.assert_almost_equal(x_end[2,:],x2_3,err_msg="\033[93m {}\03
    np.testing.assert_almost_equal(x_end[3,:],np.ones(2),err_msg="\033[93m
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Affichez les points de la question précédente

In [41]:
```
# YOUR CODE HERE
plt.scatter(x_end[:, 0], x_end[:, 1])
plt.plot(x_end[:, 0], x_end[:, 1])
plt.show()
```

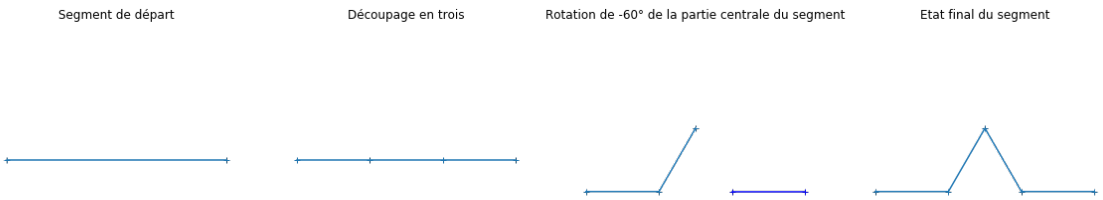


3 - Construction d'un flocon de Koch

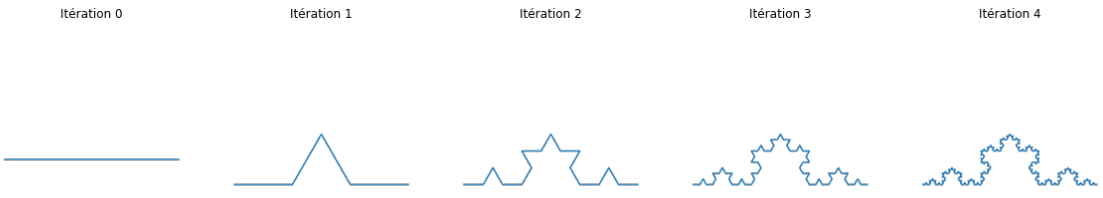
Le flocon de Koch est une des premières figures fractales de l'histoire. Il a été proposé par le mathématicien suédois Helge von Koch en 1904.

Sa construction consiste à répéter un processus itératif simple afin de construire une certaine figure. À chaque itération, on applique à tous les segments les opérations suivantes:

- on découpe les segments en trois parties égales
- on fait une rotation de -60° de la partie centrale par rapport à son sommet gauche.
- on relie le point créé par la rotation et le point à 2/3 du segment.



Exemples des figures obtenues après plusieurs itérations:



3.1 - Première étape de la construction d'un Flocon de Koch

Calculez le point `xm2` correspondant à la rotation du point `x2_3` par rapport au point x1_3 et d'angle $\frac{\pi}{3}$.

In [42]:
```
# YOUR CODE HERE
# @ produit matriciel aussi
xm2 = np.dot(x2_3 - x1_3, get_rotation_matrix(np.pi / 3)) + x1_3
```

In [43]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(xm2.shape,(2,),err_msg="\033[93m {}\033[00m" .
    np.testing.assert_almost_equal(xm2,np.array([0.2113249, 0.7886751]),er
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m".format('Ok - Tous les tests sont validés.'
```

```
Ok - Tous les tests sont validés.
```

En reprenant le code que vous avez utilisé pour construire `x_end`, faites maintenant en sorte d'avoir une matrice contenant le point (0,0), x1_3, xm2, x2_3 et la dernière ligne de x.

In [44]:
```
# YOUR CODE HERE
x_end = np.stack((np.array([0,0]), x1_3, xm2, x2_3, x[-1, :]), axis=0)
```

In [45]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x_end.shape,(5,2),err_msg="\033[93m {}\033[00m
    np.testing.assert_almost_equal(x_end[0,:],np.zeros(2),err_msg="\033[93
    np.testing.assert_almost_equal(x_end[1,:],x1_3,err_msg="\033[93m {}\03
    np.testing.assert_almost_equal(x_end[2,:],xm2,err_msg="\033[93m {}\033
    np.testing.assert_almost_equal(x_end[3,:],x2_3,err_msg="\033[93m {}\03
    np.testing.assert_almost_equal(x_end[4,:],np.ones(2),err_msg="\033[93m
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Nous allons maintenant généraliser les étapes que nous venons de faire pour construire les différentes itérations d'un flocon de Koch.

Renommez la matrice `x_end` en `x1`. Cette matrice correspond à un flocon de Koch après une itération de construction.

In [46]:
```
# YOUR CODE HERE
x1 = np.stack((np.array([0,0]), x1_3, xm2, x2_3, x[-1, :]), axis=0)
x1
```

Out[46]:
```
array([[0.        , 0.        ],
       [0.33333333, 0.33333333],
       [0.21132487, 0.78867513],
       [0.66666667, 0.66666667],
       [1.        , 1.        ]])
```

In [47]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x1.shape,(5,2),err_msg="\033[93m {}\033[00m" .
    np.testing.assert_equal(np.sum(x1),5,err_msg="\033[93m {}\033[00m" .fo
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

La première étape que nous avions faite pour l'itération 1, consistait à calculer le point se trouvant à 1/3 du segment. Nous allons maintenant généraliser cela à tous les segments. Afin de réaliser cela, vous calculerez `x1_3` en utilisant une convolution 2D entre la matrice `x1` et la matrice colonne $\begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \end{bmatrix}$ de taille $(2, 1)$. Vous ajouterez l'argument `'same'` à la convolution.

In [48]:
```
# YOUR CODE HERE
x1_3 = sc.signal.convolve2d(x1, np.array([[1/3], [2/3]]), mode="same")
x1_3
```

Out[48]:
```
array([[0.        , 0.        ],
       [0.11111111, 0.11111111],
       [0.29266384, 0.48511393],
       [0.36310547, 0.74800565],
       [0.77777778, 0.77777778]])
```

In [49]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x1_3.shape,(5,2),err_msg="\033[93m {}\033[00m"
    np.testing.assert_almost_equal(x1_3,np.array([[0.        , 0.        ]
       [0.11111111, 0.11111111],
       [0.29266384, 0.48511393],
       [0.36310547, 0.74800565],
       [0.77777778, 0.77777778]]),
    err_msg="\033[93m {}\033[00m" .format('Test 2 : La nouvelle valeur de x
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

En utilisant un mécanisme similaire à la question précédente, calculez `x2_3` correspondant à la liste des points se trouvant à 2/3 de chaque segment.

In [50]:
```
# YOUR CODE HERE
x2_3 = sc.signal.convolve2d(x1, np.array([[2/3], [1/3]]), mode="same")
x2_3
```

Out[50]:
```
array([[0.        , 0.        ],
       [0.22222222, 0.22222222],
       [0.25199435, 0.63689453],
       [0.51488607, 0.70733616],
       [0.88888889, 0.88888889]])
```

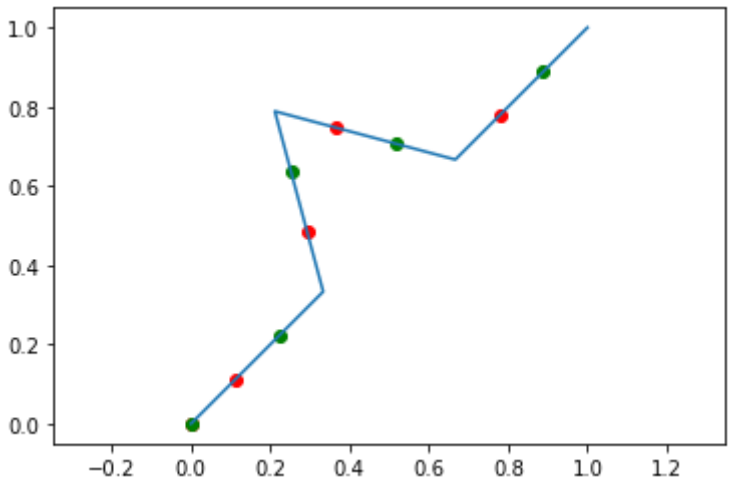
In [51]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x2_3.shape,(5,2),err_msg="\033[93m {}\033[00m"
    np.testing.assert_almost_equal(np.sum(x2_3),4+1/3,err_msg="\033[93m {}
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Affichez sur la même figure le flocon de Koch à l'étape 1, les points `x1_3` en rouge et les points `x2_3` en vert.

In [52]:
```
# YOUR CODE HERE
plt.plot(x1[:, 0], x1[:, 1])
plt.scatter(x1_3[:, 0], x1_3[:, 1], color="red")
plt.scatter(x2_3[:, 0], x2_3[:,1], color="green")
plt.axis("equal")
```

Out[52]:
```
(-0.05, 1.05, -0.05, 1.05)
```



Calculez `x1bis` correspondant à `x1` sans sa dernière ligne.

In [53]:
```
# YOUR CODE HERE
x1bis = x1[:-1, :]
```

In [54]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x1bis.shape,(x1.shape[0]-1,x1.shape[1]),err_msg
    np.testing.assert_almost_equal(x1bis,np.array([[0.       , 0.       ],
       [0.3333333, 0.3333333],
       [0.2113249, 0.7886751],
       [0.6666667, 0.6666667]]),err_msg="\033[93m {}\033[00m" .format('Test
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Calculez `x1_3bis` correspondant à `x1_3` sans sa première ligne et `x2_3bis` correspondant à `x2_3` sans sa première ligne.

In [55]:
```
# YOUR CODE HERE
x1_3bis = x1_3[1:, :]
x2_3bis = x2_3[1:, :]
```

In [56]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x1_3bis.shape,(x1_3.shape[0]-1,x1_3.shape[1]),e
    np.testing.assert_almost_equal(x1_3bis,np.array([[0.11111111, 0.111111
       [0.29266384, 0.48511393],
       [0.36310547, 0.74800565],
       [0.77777778, 0.77777778]]),err_msg="\033[93m {}\033[00m" .format('Te
    np.testing.assert_almost_equal(x2_3bis,np.array([[0.22222222, 0.222222
       [0.25199435, 0.63689453],
       [0.51488607, 0.70733616],
       [0.88888889, 0.88888889]]),err_msg="\033[93m {}\033[00m" .format('Te
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

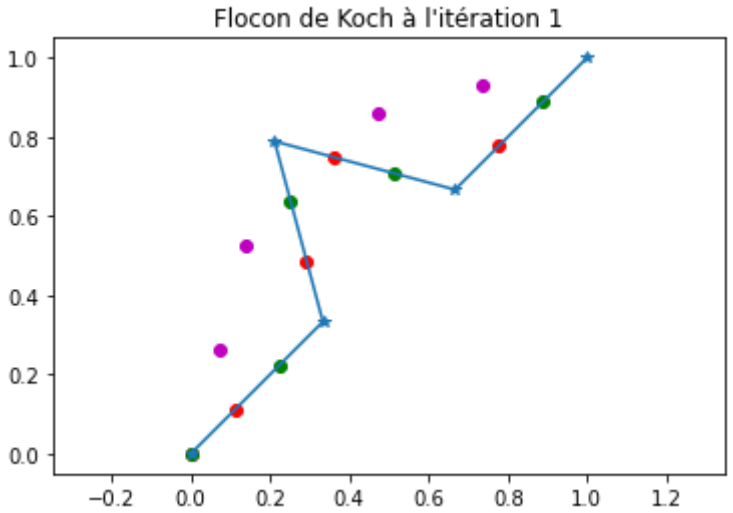
Calculez les rotations des points de `x2_3bis` par rapport au centre de `x1_3bis` pour un angle de $\frac{\pi}{3}$. Pour le point à la ligne i dans `x2_3bis` vous utiliserez le centre à la ligne i de `x1_3bis` . Le résultat sera stocké dans la variable `xm2` .

In [57]:
```
# YOUR CODE HERE
xm2 =  np.dot(x2_3bis - x1_3bis, get_rotation_matrix(np.pi / 3)) + x1_3bis
```

In [58]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(xm2.shape,(x1_3.shape[0]-1,x1_3.shape[1]),err_
    np.testing.assert_almost_equal(xm2,np.array([[0.07044162, 0.26289171],
       [0.14088324, 0.52578342],
       [0.47421658, 0.85911676],
       [0.73710829, 0.92955838]]),err_msg="\033[93m {}\033[00m" .format('T
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
 Ok - Tous les tests sont validés.
```

Affichez sur la même figure le flocon de Koch à l'étape 1, les points `x1_3` en rouge, les points `x2_3` en vert et les points `x2m` en magenta. Vous devriez obtenir le résultat suivant:



In [59]:
```
# YOUR CODE HERE
plt.plot(x1[:, 0], x1[:, 1])
plt.scatter(x1_3[:, 0], x1_3[:, 1], color="red")
plt.scatter(x2_3[:, 0], x2_3[:,1], color="green")
plt.scatter(xm2[:, 0], xm2[:, 1], color="magenta")
plt.axis("equal")
```

Out[59]: `(-0.05, 1.05, -0.05, 1.05)`

Les étapes qui vont suivre permette de fusionner les différents points que nous avons calculé en une seule figure tel que les points soient correctement ordonnés pour être affiché correctement.

Commencez par concaténer selon les lignes les matrices `x1bis, x1_3bis, xm2` et `x2_3bis` . Respectez l'ordre donné. Le résultat sera stocké dans `x_end` .

In [60]:
```
# YOUR CODE HERE
x_end = np.vstack([x1bis, x1_3bis, xm2, x2_3bis])
x_end.shape
```

Out[60]: (16, 2)

In [61]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x_end.shape,(16,2),err_msg="\033[93m {}\033[00
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Redimensionnez `x_end` pour avoir un tenseur de 3 dimensions ayant 4 lignes et 2 de profondeur. Le nombre de colonnes devra être calculé automatiquement par numpy.

In [62]:
```
# YOUR CODE HERE
x_end = np.reshape(x_end, (4, -1, 2))
x_end.shape
```

Out[62]: (4, 4, 2)

In [63]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x_end.shape,(4,4,2),err_msg="\033[93m {}\033[0
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Échangez les lignes et les colonnes de `x_end` .

In [64]:
```
# YOUR CODE HERE
x_end = np.transpose(x_end, (1,0,2))
```

In [65]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x_end.shape,(4,4,2),err_msg="\033[93m {}\033[0(
    np.testing.assert_almost_equal(x_end,np.array( [[[0.    ,      0.
  [0.11111111, 0.11111111],
  [0.07044162, 0.26289171],
  [0.22222222, 0.22222222]],
 [[0.33333333, 0.33333333],
  [0.29266384 ,0.48511393],
  [0.14088324, 0.52578342],
  [0.25199435, 0.63689453]],
 [[0.21132487, 0.78867513],
  [0.36310547, 0.74800565],
  [0.47421658, 0.85911676],
  [0.51488607, 0.70733616]],
 [[0.66666667, 0.66666667],
  [0.77777778, 0.77777778],
  [0.73710829, 0.92955838],
  [0.88888889, 0.88888889]]]

),err_msg="\033[93m {}\033[00m" .format('Test 2 : Les valeurs de x_end ne s
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Redimensionnez `x_end` pour avoir une matrice de 2 colonnes.

In [66]:
```
# YOUR CODE HERE
x_end = np.reshape(x_end, (-1,2))
x_end.shape
```

Out[66]:
```
(16, 2)
```

In [67]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x_end.shape,(4*4,2),err_msg="\033[93m {}\033[0(
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Insérez à la fin de `x_end` la dernière ligne de `x1` .

In [68]:
```
# YOUR CODE HERE
print(x_end.shape)
x_end = np.vstack((x_end, x1[-1]))
x_end.shape
```

```
(16, 2)
```

Out[68]:
```
(17, 2)
```

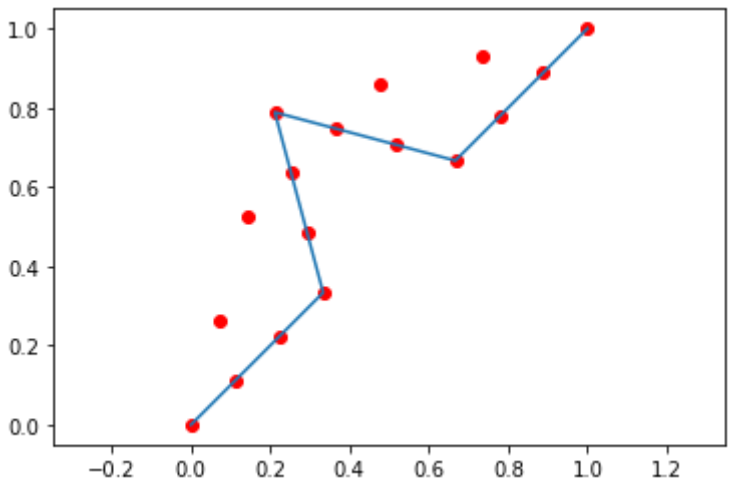
In [69]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(x_end.shape,(17,2),err_msg="\033[93m {}\033[00r
    np.testing.assert_almost_equal(x_end,np.array( [[0.    ,      0.
  [0.11111111, 0.11111111],
  [0.07044162, 0.26289171],
  [0.22222222, 0.22222222],
  [0.33333333, 0.33333333],
  [0.29266384 ,0.48511393],
  [0.14088324, 0.52578342],
  [0.25199435, 0.63689453],
  [0.21132487, 0.78867513],
  [0.36310547, 0.74800565],
  [0.47421658, 0.85911676],
  [0.51488607, 0.70733616],
  [0.66666667, 0.66666667],
  [0.77777778, 0.77777778],
  [0.73710829, 0.92955838],
  [0.88888889, 0.88888889],
  [1,1]]),err_msg="\033[93m {}\033[00m" .format('Test 2 : Les valeurs de x_
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Affichez les points de `x_end` . Nous avons maintenant la deuxième itération d'un flocon de Koch.

In [70]:
```
plt.plot(x1[:, 0], x1[:, 1])
plt.scatter(x_end[:,0], x_end[:, 1], color="red")
plt.axis("equal")
```

Out[70]:
```
(-0.05, 1.05, -0.05, 1.05)
```



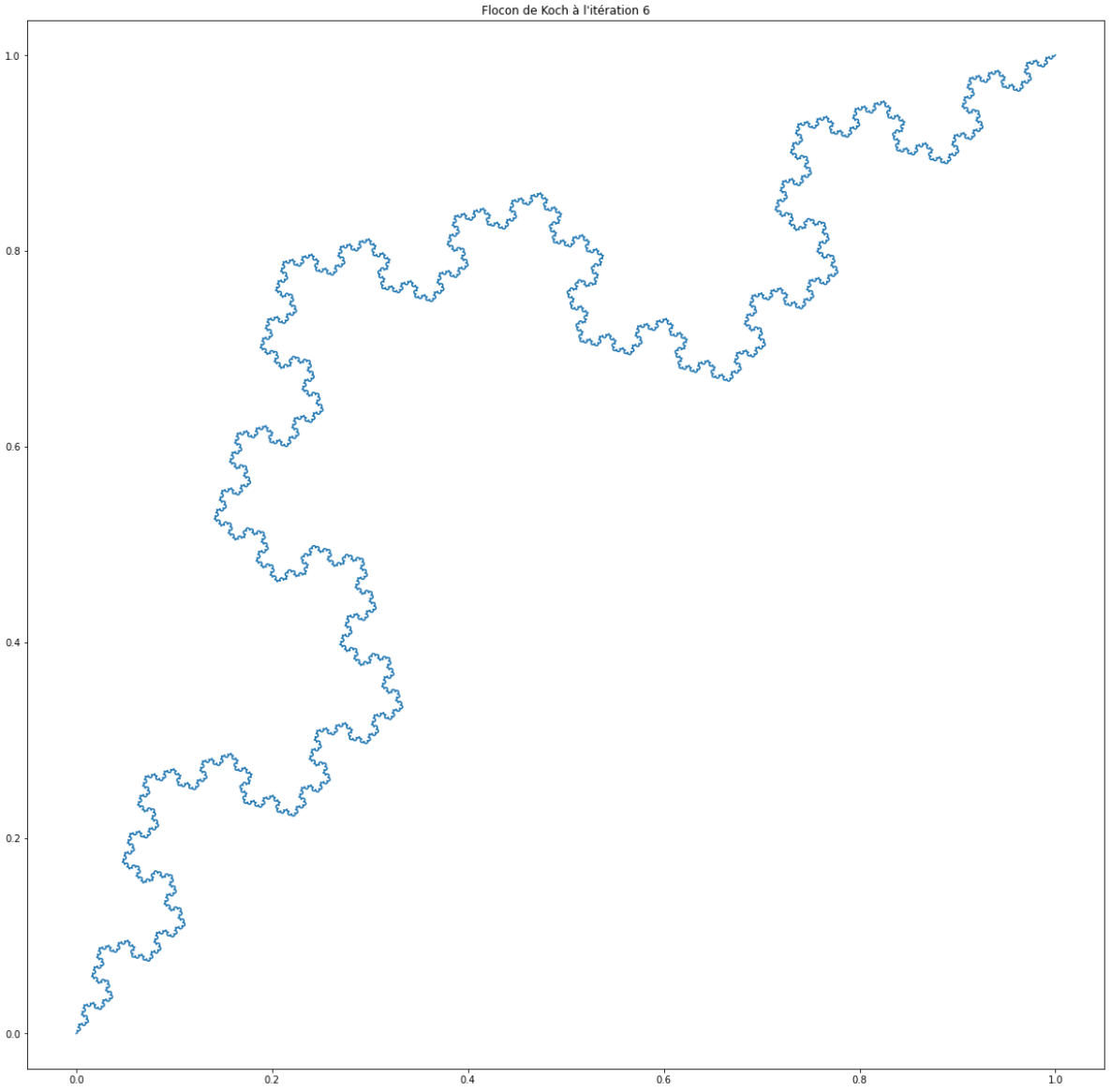
Écrivez une fonction permettant de calculer un flocon de Koch après `n` itération. Testez avec `n=6`

In [71]:

```
def koch_n(x=np.array([[0,0],[1,1]]),t = np.pi/3,n=6):
    # YOUR CODE HERE
    for i in range(n):

        x1_3 = sc.signal.convolve2d(x, np.array([[1/3], [2/3]]), "same")[1
        x2_3 = sc.signal.convolve2d(x, np.array([[2/3], [1/3]]), "same")[1

        xm2 =  np.dot(x2_3 - x1_3, get_rotation_matrix(t)) + x1_3
        x_end = np.vstack((x[:-1], x1_3, xm2, x2_3))
        x_end= np.reshape(x_end, (4, -1, 2))
        x_end = np.transpose(x_end, (1,0,2))
        x_end = np.reshape(x_end, (-1,2))
        x_end = np.vstack((x_end, x[-1]))
        x= x_end
    return x_end
```

In [72]:

```
f = koch_n()

plt.figure(figsize=(20,20))
plt.plot(f[:,0],f[:,1])
plt.axis('equal')
plt.title('Flocon de Koch à l\'itération 6')
plt.show()
```



In [ ]: