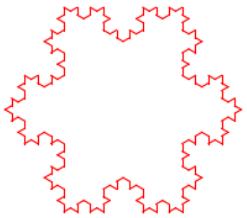
TP1 –FLOCON DE VON KOCH

OBJECTIFS :

Révision de listes, tracées de segments et appels de fonctions

... Tracer de fractales .



1. QUESTIONS PRELIMINAIRES

Même si python prévoit une manière de manipuler des nombres complexes, on définira dans tout le sujet un nombre complexe de la manière suivante :

```
>>> a = ( 3 . , 4 . )
```

Il faut ici comprendre que a = 3 + 4i.

Autrement dit, dans tout le sujet, un nombre complexe sera un « *tuple* » à deux éléments : la partie réelle et la partie imaginaire. On fera bien attention à utiliser les points, pour bien dire à python que l'on manipule des nombre flottants, et pas des entiers (sans quoi on aura des surprises en faisant des divisions !).

Q1. Programmer deux fonctions « partie réelle » et « partie imaginaire »

```
>>> def Re( z ) :
et
>>> def Im( z ) :
```

*Q2. Une fonction « **somme** » telle que*

```
>>> somme( a , a ) ;
( 6.0 , 8.0 )
```

Qui se lit mathématiquement a + a = 6 + 8i.

*Q3. Écrire une fonction **produit** qui calcule le produit de deux nombres complexes. On pensera à tester la fonction :*

```
>>> produit ( a , a )
( −7.0 ,24.0)
```

*Q4. Écrire de même une fonction **quotient** qui calcule le quotient de deux nombres complexes.*

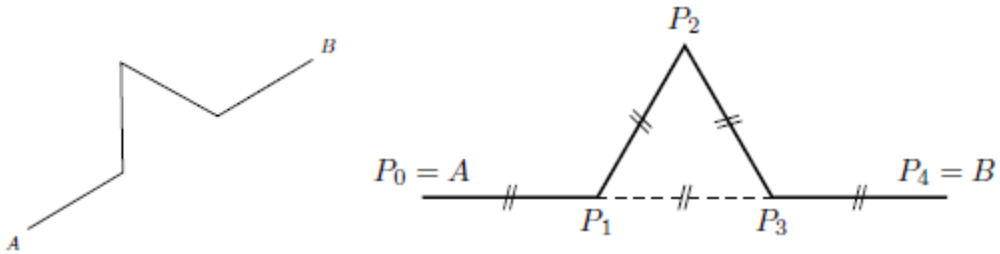
```
>>> quotient ( a , ( 2 . , 0 ) )
( 1.5 , 2.0 )
```

*Q5. Écrire une fonction **expo** qui calcule $e^{i\theta}$ pour n'importe quelle valeur de θ. Penser à utiliser le module math !*

```
>>> expo (math . pi /3)
(0.5, 0.866)
```

2. TRANSFORMER UN SEGMENT

On considère dans cette partie un segment $[AB]$ du plan complexe. Les points A et B ont pour affixes respectifs z_A et z_B L'objectif est de calculer, en fonction de z_A et z_B les coordonnées des points de la figure ci-dessous :



Ouvrir le fichier ***flocon_vide.py***. Observer les fonctions point1(z_A,z_B) qui renvoie l'affixe de P_1 et point3(z_A,z_B) qui renvoie l'affixe de P_3, toujours en fonction de celles de A et B.

Q6. Écrire une fonction point2(z_A,z_B) qui calcule l'affixe de P_2, en fonction des autres points et d'une rotation bien choisie. Et l'exprimer ensuite en fonction de de A et B

3. LES DESSINER

On souhaite enfin dessiner la figure précédente (et bien d'autres). On utilisera la bibliothèque pylab

Dans cette partie, on considère le programme suivant (il utilise des fonctions définies avant).

Q7. *Expliquer ce que fait le programme suivant. Et copier la fonction* ***drawseg()*** *dans le programme* ***flocon_vide.py*** ***Penser à remplacer les tirets et les apostrophes***

```
# Importer pylab
from pylab import *

# Fixer les bords du cadre du dessin
axes ( aspect=1)
ylim( −2.0 , 2.0 )
xlim( −2.0 , 2.0 )
origin = ( 0 , 0 )

def drawseg(seg,col):
    '"fonction drawseg: elle prend en argument un segment et une couleur et dessine ce segment."'
    zA=seg[0]
    zB=seg[1]
    plot((Re(zA),Re(zB)),(Im(zA),Im(zB)),col)

# On definit trois complexes
z1 = ( 0 , 0 )
z2 = expo ( pi / 3)
z3 = expo(−pi / 4)

# On dessine en rouge
col= 'r−'

# Dessiner deux segments
drawseg ( ( z1 , z2 ) , col )
drawseg ( ( z2 , z3 ) , col )

# Afficher le resultat
show ( )

def drawstar (n) :
   axes ( aspect=1)
   ylim( −2.0 , 2.0 )
   xlim( −2.0 , 2.0 )
   origin = ( 0 , 0 )
  for i in range ( 0 , n) :
    for j in range ( 2 , n−2) :
       drawseg ( ( expo (2* i* pi /n ) ,expo (2*( i+j )* pi /n ) ) , ' r−' )
   show ( )
```

4. LE FLOCON

En donnant au point *A* l'affixe −1 (ses coordonnées sont (−1, 0)), et au point *B* l'affixe 1, alors on peut dessiner cette figure avec le programme simple suivant :

```
axes ( aspect=1)
ylim( −2.0 ,2.0)
xlim( −2.0 ,2.0)
A=( −1. ,0.)
B=( 1 . , 0 . )
p1 = point1(A,B)
p2 = point2(A,B)
p3 = point3(A,B)
drawseg((A, p1) , ' r−' )
drawseg( (p1 , p2) , ' r−' )
drawseg( (p2 , p3) , ' r−' )
drawseg( (p3 ,B), ' r−' )
show ( )
```

Q8. Expliquer comment affecter à A, B et C les coordonnées de trois points de telle sorte que ABC soit un triangle équilatéral (on suggère de faire en sorte qu'ils soient tous les trois de module 1).

Q9. Écrire une fonction kochifie qui prend en argument un segment, et retourne une liste des 4 segments.

```
>>> k o c h i f i e ( ( ( 0 . , 0 . ) , ( 1 . , 0 . ) ) )
[ ( ( 0 . 0 , 0 . 0 ) , ( 0 . 3 3 3 , 0 . 0 ) ) ,
( ( 0 . 3 3 3 , 0 . 0 ) , ( 0 . 5 , 0 . 2 8 9 ) ) ,
( ( 0 . 5 , 0 . 2 8 9 ) , ( 0 . 6 6 7 , 0 . 0 ) ) ,
( ( 0 . 6 6 7 , 0 . 0 ) , ( 1 . 0 , 0 . 0 ) ) ]
```

Q10. Écrire une procédure dessineliste qui prend en argument une liste de segments et les affiche à l'écran.

Q11. Écrire une fonction vonkoch qui prend en argument une liste de segments et remplace chaque segment seg par les quatre segments de kochifie(seg), et retourne le résultat

Q12. En utilisant la liste associée à un triangle équilatéral, (on trace alors le flocon en appelant 4 fois la fonction vonkoch sur la liste de trois segments initiale), généraliser en un programme qui fait la même chose, mais en appliquant n fois la fonction vonkoch.

```
#définition de la liste des 3 segments de base (le triangle équilatéral):
liste=[(expo(0),expo(4*pi/3)),(expo(4*pi/3),expo(2*pi/3)),(expo(2*pi/3),expo(0))]
```

Q13. Peut-on avoir une idée du temps de calcul pour effectuer 10 itérations ? 20 itérations ? 100 itérations ? On pourra donner les réponses en secondes, en minutes, en années, en millénaires, etc.