TP n°4 : Récursivité

CORRECTION

I. Figures alphanumériques

1) Tester le programme suivant :

`def figure(n):`
`  if n>0:`
`  print(n*"*")`
`  figure(n-1)`
	Quel est l'affichage produit dans la console lorsqu'on appelle `figure(5)` ?
	*****
	****
	***
	**
	*

2) Programmer une fonction récursive figure2(n) qui pour $n = 5$ affichera dans la console :
*
**
***
****
*****

```
def figure2(n):
    if 0<n:
        figure2(n-1)
        print(n*"*")
```

II. Dichotomie version récursive

On rappelle qu'il est possible de déterminer si un élément est présent ou non dans une liste triée dans l'ordre croissant à l'aide d'une recherche dichotomique. Nous avions précédemment codé la version itérative de cet algorithme en utilisant une boucle `while`. Maintenant nous allons programmer la version récursive.

1) Aidez-vous de l'algorithme fourni ci-dessous pour recopier et compléter le programme de recherche dichotomique.

Algorithme de recherche dichotomique	Recherche dichotomique récursive

Si la liste L contient au moins un élément.
On la coupe en deux en calculant l'indice m au milieu.
On compare L[m] et l'élément recherché x.
S'ils sont égaux
On renvoie Vrai
Si x<L[m]
On cherche dans la première moitié de la liste
Sinon
On cherche dans la deuxième moitié de la liste
	`def dicho(x,L):`
	`  if len(L)==0:`
	`  return False`
	`  m = len(L)//2`
	`  if x == L[m]:`
	`  return True`
	`  elif x < L[m]:`
	`  return dicho(x,L[0:m])`
	`  else:`
	`  return dicho(x,L[m+1:])`

III. Tri rapide

Le tri rapide est une méthode inventée par Tony Hoare pour trier une liste L dans l'ordre croissant. Pour cela l'idée est de choisir un pivot (le premier élément de la liste par exemple), de parcourir la liste L et de la séparer en 3 : les éléments inférieurs ou égaux au pivot (1), le pivot (2) et les éléments strictement supérieurs au pivot (3).
Une solution est de créer deux sous-listes Lp (contenant les éléments plus petits) et Lg (contenant les éléments plus grands), de les trier récursivement puis de reconstituer l'ensemble par concaténation.

Pour simplifier les choses nous allons procéder en deux étapes :

1) Écrire une fonction `separe(L)` qui prend en argument une liste supposée non vide et qui renvoie deux listes Lp et Lg telles que définies ci-dessus.

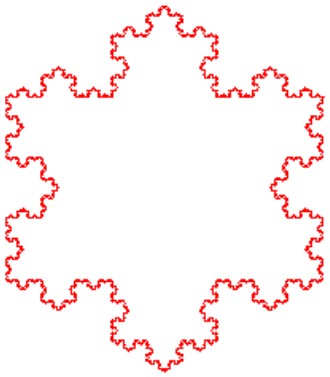
```
def separe(L):
    Lp, Lg = [],[]
    for x in L[1:]:
        if x <= L[0]:
            Lp.append(x)
        else:
            Lg.append(x)
    return(Lp,Lg)
```

Vérifier votre fonction en appelant `separe([6,7,1,6,9,2])` qui doit renvoyer `([1,6,2],[7,9])`

2) Écrire une fonction récursive `Tri_rapide(L)` qui renvoie la liste L triée dans l'ordre croissant.

```
def Tri_rapide(L):
    if len(L)<=1:
        return L
    else:
        return Tri_rapide(separe(L)[0]) + [L[0]] + Tri_rapide(separe(L)[1])
```

IV. Fractales

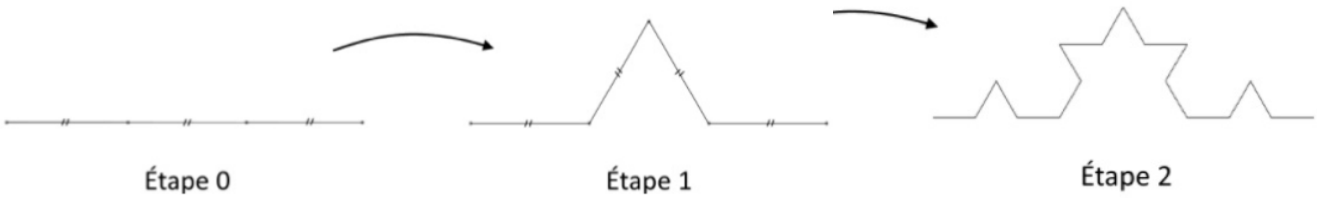


Le flocon de Von Koch, représenté à droite et imaginé en 1904 par le mathématicien suédois Helge Von Koch est l'une des premières courbes fractales à avoir été décrite. Les fractales permettent entre autres de modéliser le développement de certaines bactéries.

Pour construire ce flocon nous allons commencer par construire une courbe de Van Koch à l'aide du module turtle de Python. Dans ce module, la tortue est un outil de dessin pouvant tourner et avancer par segments successifs. Vous trouverez aisément l'aide de fonctionnement en ligne mais voici un rappel des fonctions les plus utiles.
Après avoir importé en module à l'aide de la commande **`import`** `turtle` **`as`** `tl` on peut utiliser :

- `tl.forward(cote)` pour faire avancer la tortue de `cote` pixels tout en traçant.
- `tl.left(angle)` pour faire tourner la tortue de `angle` degrés. Le sens dépend du paramétrage.
- `tl.setheading(angle)` pour déterminer l'orientation initiale `angle` en degrés.
- `tl.hideturtle()` pour cacher la tortue.
- `tl.speed(vitesse)` pour accélérer ou ralentir la tortue.
- `tl.color('couleur')` pour choisir la couleur du tracé.
- `tl.exitonclick()` pour que la fenêtre de dessin reste ouverte lorsque le tracé est terminé.

Une courbe de Van Koch de rang n=0 est un simple segment. Ensuite, à chaque incrémentation de n, tous les segments présents au rang précédent sont divisés en trois parts égales. Parmi ces trois parts, celle au centre est alors transformée en un triangle équilatéral dont on efface la base. La figure ci-dessous illustre ce principe :



1. Recopier et compléter la fonction récursive `courbeVonKoch(n,cote)` prenant en paramètre le rang `n` souhaité et la taille `cote` souhaitée pour le segment initial permettant de tracer la courbe de Von Koch de rang n. Attention tous les paramètres d'initialisation et de réglage sont à insérer hors de la fonction que l'on appellera ensuite dans l'éditeur. Après avoir recopié ces paramètres appeler votre fonction pour n = 0,1 et 2 pour vérifier si elle reproduit bien les dessins ci-dessus.

2. Recopier et compléter la fonction `flocon` pour obtenir finalement le flocon de Von Koch de rang n. Appeler votre fonction pour obtenir la figure de la page précédente.

```
import turtle as tl


def courbeVonKoch(  n, cote  ) :
      if n == 0 :
            tl.forward(cote)
      else :
            courbeVonKoch(n-1, cote/3)
            tl.left(60)
            courbeVonKoch(n-1, cote/3)
            tl.left(-120)
            courbeVonKoch(n-1, cote/3)
            tl.left(60)
            courbeVonKoch(n-1, cote/3)


def flocon(n, cote) :
      for i in range(3) :
            courbeVonKoch(  n, cote  )
            tl.left(-120)
            i = i+1

tl.setheading(0) # orientation intiale de la tête : vers la droite de l'écran
tl.hideturtle() # on cache la tortue
tl.speed(0)  # on accélère la tortue
tl.color('green')
flocon(  n = 3, cote = 200  )
tl.exitonclick() # pour  garder ouverte la fenêtre
```