

Dans cette feuille d'exercices ,et sauf indication contraire , toutes les fonctions demandées sont des fonctions qui utilisent la récursivité.

Pour chacune des fonctions , on écrira une spécification.

Exercice 1

Ecrire une fonction *Etoiles*(*n*) qui a pour paramètre un entier naturel *n* non nul et qui renvoie la chaîne `'**...**'` où on répète *n* fois le caractère `'*'`

Exercice 2(palindrome)

Un mot est un palindrome s'il s'écrit de la même façon à l'endroit et à l'envers.

Ainsi les mots 'patap' et 'kayak' sont des palindromes.

Un mot formé d'une seule lettre est également un palindrome.

Complétez la fonction *Est_palindrome* qui renvoie True si le mot est un palindrome , False sinon.

```
def Est_palindrome(mot):
    '''
    renvoie Vrai si mot est un palindrome
    Faux sinon
    '''
    #le cas de base
    if len(mot)<=1:
        return True
    if .....
        return False

    else:
        return .....
```

Exercice 3

(Dans cet exercice , on s'interdira l'utilisation des fonctions natives **int** et **bin**)

1. Ecrire une fonction non récursive *dec_vers_bin*(*n*), qui renvoie l'écriture binaire d'un entier *n* positif (écriture en base 2).
2. Ecrire une version récursive *dec_vers_bin_recuratif*(*n*) de la fonction précédente.
3. Ecrire une fonction récursive *bin_vers_dec*(chaîne) qui renvoie l'entier représenté en binaire par chaîne.
4. Ecrire une fonction *affiche*(*n*,*b*) qui affiche l'écriture de *n* en base *b* où $2 \leq b \leq n$.

Exercice 4

L'algorithme d'Euclide consiste à effectuer des divisions successives afin de déterminer le plus grand commun diviseur de deux entiers positifs a ($a > 0$) et b , noté $\text{PGCD}(a,b)$.

Ce nombre correspond au **dernier reste non nul** dans ces divisions. Pour déterminer par exemple le $\text{PGCD}(72,27)$, on effectue:

$$72 = 27 \times 2 + 18$$

$$27 = 18 \times 1 + 9$$

$$18 = 9 \times 2 + 0$$

D'où $\text{PGCD}(72,27) = 9$.

On notera que $\text{PGCD}(a,0) = a$ pour tout entier a strictement positif.

1. Complétez le programme suivant écrit en python qui renvoie le PGCD de deux entiers a et b .

```
def PGCD(a,b):
    '''
    a est un entier naturel non nul
    b est un entier naturel
    '''
    if b==0:
        return
    while b!=0:
        r= a % b #reste de la division euclidienne de a par b
        .....
        .....
    return a
```

2. On souhaite écrire une fonction `pgcd_recuratif` qui renvoie le $\text{PGCD}(a,b)$, $a > 0$ et b positif ou nul.
Quel serait le cas de base?
3. En remarquant que $\text{PGCD}(a,b) = \text{PGCD}(b, a \% b)$, écrire la fonction `pgcd_recuratif`.

Exercice 5(triangle de Pascal)

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Dans le triangle ci-dessus chaque nombre d'une ligne numéro n ($n \geq 0$) et d'une colonne p est la somme des deux nombres situés sur la ligne $n - 1$ qui appartiennent à la même colonne p et à la colonne précédente $p - 1$. ($0 \leq p \leq n - 1$).

Par exemple, $6 = 3 + 3$ et $4 = 3 + 1$.

Ces nombres sont égaux aux coefficients binomiaux en mathématiques très utiles en probabilités.

1. Ecrire une fonction récursive `coeff_binom(n,p)` où $0 \leq p \leq n - 1$, qui calcule le coefficient binomial(n,p).
2. Ecrire une fonction `triangle_Pascal(n)` qui affiche le triangle de Pascal (on obtient $n+1$ lignes à l'affichage de 0 à n)

Exercice 6

L'anagramme d'un mot est un mot s'écrivant avec les mêmes lettres que le mot initial. Par exemple : ironique et onirique ; baignade et badinage ; estival et vitales.

Ici, nous ne demandons pas que les mots envisagés soient des mots du dictionnaire. Ainsi 'abc' et 'bca' sont deux anagrammes.

On souhaite écrire une fonction récursive **anagramme** qui renvoie la liste des anagrammes d'un mot.

Principe .

- Cas de base :
 - si le mot est une chaîne vide, la liste des anagrammes est alors constituée de l'unique chaîne vide
 - si le mot n'a qu'une seule lettre, la liste de ses anagrammes ne contient qu'un seul élément : le mot lui-même.
- Dans les autres cas :
 - on peut définir la liste des anagrammes d'un mot à partir de la liste des anagrammes du mot obtenu en enlevant la première lettre, en plaçant cette première lettre successivement dans toutes les positions dans chaque élément de la liste.

Un exemple : prenons le cas d'un mot de trois lettres : 'abc'. La première lettre est 'a'.

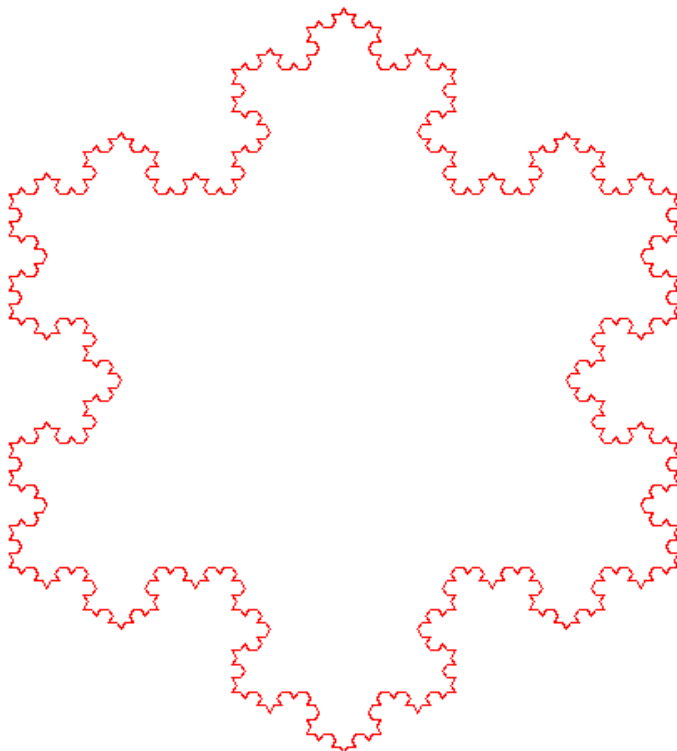
Le mot obtenu en enlevant la première lettre est 'bc' et la liste des anagrammes de 'bc' est ['bc', 'cb'].

Pour constituer la liste des anagrammes de 'abc', il suffit de placer la lettre 'a' dans toutes les positions possibles des anagrammes de 'bc'. On obtient la liste ['abc', 'bac', 'bca', 'acb', 'cab', 'cba'].

De la même façon, on obtiendra la liste de toutes les anagrammes du mot 'dabc' en glissant la lettre 'd' dans toutes les positions possibles de toutes les anagrammes de 'abc'

Exercice 7

Une image qui a une apparence similaire quelle que soit l'échelle à laquelle on l'observe est appelée une fractale (il y a d'autres types de fractales). Un exemple simple de fractale est le flocon de Von Koch, dont voici une représentation (pour une profondeur de 4).



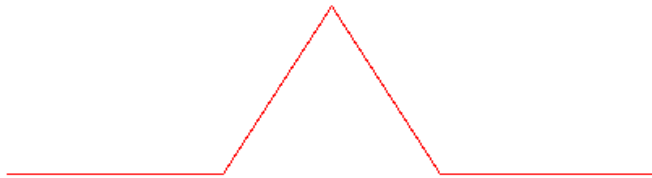
Il faut d'abord construire la courbe de Von Koch. On peut la créer à partir d'un segment de droite, en modifiant récursivement chaque segment de droite de la façon suivante :

- on divise le segment de droite en trois segments de longueurs égales ;
- on construit un triangle équilatéral ayant pour base le segment médian de la première étape ;
- on supprime le segment de droite qui était la base du triangle de la deuxième étape.

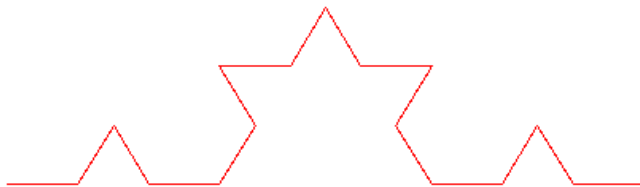
En partant d'un segment à la première étape:



on aboutit à la figure suivante à l'étape 3:



Si on recommence la même construction en partant de chaque segment de la dernière figure , on obtient



Pour tracer la courbe avec une certaine profondeur n et en partant d'un segment de longueur l , on va utiliser le module `turtle` de `Python` dont voici un exemple :

```
import turtle as t
t.color("red") #pour utiliser la couleur rouge
t.fd(100) #pour tracer un segment de longueur 100
```

Voici quelques commandes de base utilisant le module `turtle` (on pourra consulter le lien suivant pour des informations approfondies sur ce module)

<https://docs.python.org/fr/3.8/library/turtle.html>

- `goto(x, y)` : Aller à l'endroit de coordonnées x, y
- `forward(d)` ou simplement `fd` : Avancer d'une distance donnée d .
- `backward(d)` ou simplement `bk` : Reculer d'une distance donnée d .
- `up()` : Relever le crayon (pour pouvoir avancer sans dessiner)
- `down()` : Abaisser le crayon (pour recommencer à dessiner)

- `color(couleur)` : Changer la couleur
- `left(angle)` : Tourner à gauche d'un angle donné (exprimé en degrés)
- `right(angle)` : Tourner à droite d'un angle donné (exprimé en degrés)
- `width(épaisseur)` : Choisir l'épaisseur du tracé

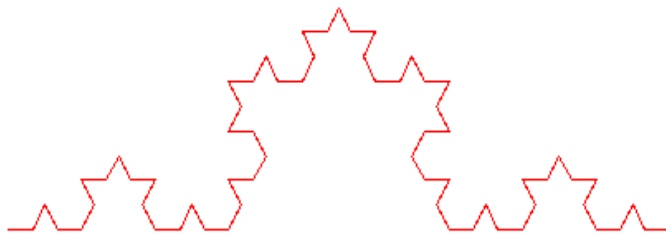
Ecrire une fonction `koch(n,1)` qui prend pour paramètres un entier n désignant la profondeur de la récursivité et a la longueur du segment initial.

On complètera les lignes vides de la fonction ci-dessous.

```
def Koch(n,a):
    if n==0: #le cas de base
        .....
    else:
        Koch(n-1,a/3)
        t.left(60)
        .....
        .....
        .....
        .....

    return None
```

l'appel `koch(3,400)` donne le résultat



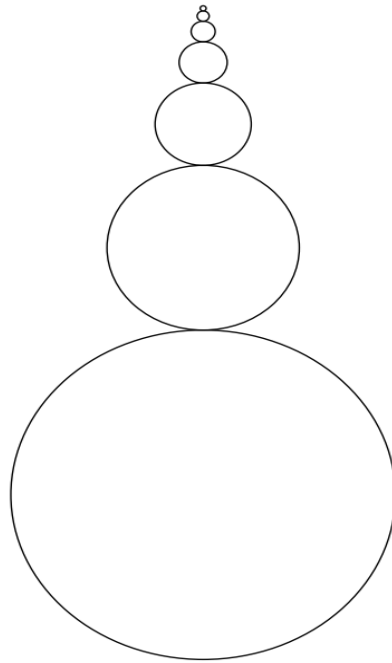
Ecrire la fonction `flocon(n,a)` qui dessine le flocon de Von Koch, a étant la longueur du segment de départ et n la profondeur de la récursivité utilisée dans la fonction `koch(n,a)`.

Exercice 8

Pour tracer un cercle , on donne la fonction suivante

```
def dessine_cercle(x,y,r):
    t.color("blue")
    t.hideturtle()
    t.up()
    t.goto(x,y-r)
    t.down()
    t.circle(r)
    return None
```

1. Tester cette fonction sur plusieurs exemples.
2. On souhaite écrire un programme en Python pour obtenir la figure suivante où plusieurs cercles sont tangents intérieurement (au même point) et dont les rayons respectifs diminuent de moitié en partant du plus grand cercle comme montré ci-dessous



3. Comment faire pour obtenir le dessin suivant?

