

1. Générer des listes

✍ Implémenter une fonction récursive `genere_liste(n)` qui génère une liste `n`.

```
>>> genere_liste(5)
[5, 4, 3, 2, 1, 0]
```

```
def genere_liste(n):
    if n == 0:
        return [0]
    else:
        return [n] + genere_liste(n-1)
```

✍ Implémenter une fonction récursive `genere_liste_cro(n)` qui génère une liste croissante jusqu'à `n`.

```
>>> genere_liste_cro(5)
[0, 1, 2, 3, 4, 5]
```

```
def genere_liste_cro(n):
    if n == 0:
        return [0]
    else:
        return genere_liste_cro(n-1) + [n]
```

2. Nettoyer des listes

✍ Implémenter une fonction récursive qui nettoie une liste triée en éliminant les éléments identiques.

```
>>> nettoie([0, 1, 1, 2, 3, 4, 4, 5, 6, 6])
[0, 1, 2, 3, 4, 5, 6]
```

```
def nettoie(liste):
    if len(liste) == 1:
        return [liste[0]]
    else:
        if liste[0] == liste[1]:
            liste = liste[1:]
            return nettoie(liste)
        else:
            return [liste[0]] + nettoie(liste[1:])
```

3. Convertir un nombre décimal en binaire

Pour cela, on utilisera la succession de divisions euclidiennes par 2.

Exemple : 29 en base 10. On effectue les divisions successives par 2 jusqu'à obtenir un quotient nul.

$$\begin{array}{r|l} 29 & 2 \\ \hline 1 & 14 \end{array} \quad
 \begin{array}{r|l} 14 & 2 \\ \hline 0 & 7 \end{array} \quad
 \begin{array}{r|l} 7 & 2 \\ \hline 1 & 3 \end{array} \quad
 \begin{array}{r|l} 3 & 2 \\ \hline 1 & 1 \end{array} \quad
 \begin{array}{r|l} 1 & 2 \\ \hline 1 & 0 \end{array}$$

En lisant de droite à gauche les restes obtenus, on obtient l'écriture en base 2 de l'entier. Ici, $29_{10} = 11101_2$

✍ Implémenter la fonction récursive `dec_vers_bin()` qui prend un entier naturel comme argument et renvoie son écriture binaire sous la forme d'une chaîne de caractères.

```
>>> dec_vers_bin(8)
'1000'
```

✍ Renseigner le docstring et fournir un jeu de tests pertinent.

```
def dec_vers_bin(n):
    """convertit un nombre décimal en binaire
    entrée : n (int)
    sortie (chaîne de caractères)
    >>> dec_vers_bin(10)
    '1010'
    >>> dec_vers_bin(0)
    '0'
    """

    if n <= 1:
        return str(n % 2)
    else:
        reste = n % 2
        n = n // 2
        return str(dec_vers_bin(n)) + str(reste)

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Une version plus compacte de la fonction `dec_vers_bin()` :

```
def dec_vers_bin(n):
    if n>1:
        return dec_vers_bin (n//2) + str (n %2)
    return str (n %2)
```

4. Convertir un nombre binaire en décimal

Si n est un entier naturel et a_0, a_1, \dots, a_p sont les chiffres de son écriture en base 2, on a :

$$n = 2^p a_p + 2^{p-1} a_{p-1} + \dots + a_0$$

✍ En remarquant que $n = 2 \times (2^{p-1} a_{p-1} + \dots + a_1) + a_0$ si $n \geq 2$, implémenter la fonction récursive `bin_vers_dec()` qui prend un chiffre binaire écrit sous forme de chaîne de caractère et renvoie son écriture décimal sous forme d'entier.

✍ Renseigner le docstring et fournir un jeu de tests pertinent.

```
def bin_vers_dec(chaine):
    """convertit un chiffre binaire en décimal
    chaine : String
    sortie : int
    >>> bin_vers_dec('1101')
    13
    >>> bin_vers_dec('0')
    0
    """

    if len(chaine) == 1:
        return int(chaine[0])
```

```
else:
    return 2*bin_vers_dec(chaine[:-1]) + int(chaine[-1])

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

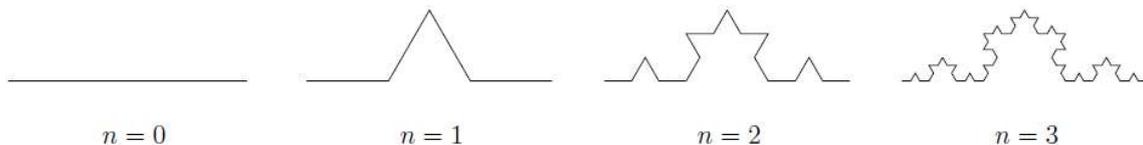
5. Flocon de Koch

Le **flocon de Koch** est l'une des premières courbes fractales à avoir été décrites. Elle a été inventée en 1904 par le mathématicien suédois **Helge von Koch**.

La courbe de Koch d'ordre n est définie de la manière suivante :

- si $n = 0$ c'est un segment de longueur 1 ;
- sinon :
 - 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.

Courbes de Koch d'ordre n pour $0 \leq n \leq 3$:

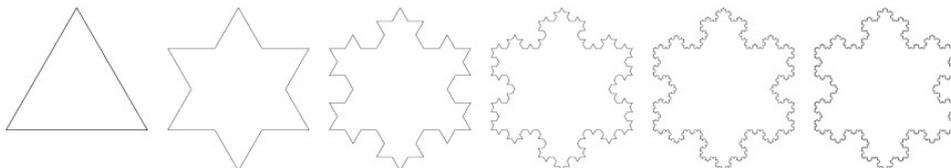


L'objectif du problème est de dessiner cette courbe à l'aide de la bibliothèque **Turtle** à l'aide d'une fonction récursive.

Pour cela, on remarquera que la courbe de Von Koch d'ordre $n + 1$ est constituée de 4 parties qui sont des courbes de Von Koch d'ordre n dont la taille a été divisée par 3.

✍ Implémenter une fonction récursive **courbe(n, long)** qui dessine cette courbe en utilisant **Turtle** avec n l'ordre de la courbe et **long** la longueur du segment à l'ordre 0.

✍ En utilisant la fonction précédente, écrire une fonction **flocon(n, long)** dessinant le flocon de Von Koch d'ordre n . Ci-dessous les flocons de l'ordre 0 à l'ordre 5 :



```
import turtle as tl

def courbe(n, long):
    if n == 0:
        tl.forward(long)
    else:
        courbe(n-1, long/3)
```

