Exercice 1. — Écrire une fonction récursive `compte_a_rebours(n)` qui affiche les entiers de `n` (supposé positif) à 0.
Par exemple, `compte_a_rebours(5)` doit afficher 5 -> 4 -> 3 -> 2 -> 1 -> 0.

Exercice 2. — Écrire une fonction récursive `boucle(i, k)` qui affiche les entiers entre `i` et `k`.
Par exemple, `boucle(0, 3)` doit afficher 0 1 2 3.

Exercice 3. — Le Plus Grand Diviseur Commun (PGCD) de deux entiers naturels `a` et `b` est calculable à l'aide de l'algorithme d'Euclide qui repose sur la propriété : `pgcd(a, b) = pgcd(b, r)` où `r` est le reste de la division euclidienne de `a` par `b`. En voici une version itérative.

```
def pgcd(a, b):
    while a%b != 0:
        a, b = b, a%b
    return b
```

Écrire une version récursive de cette fonction.

Exercice 4. — Écrire une fonction récursive `inverse(st)` qui prend en paramètre une chaîne de caractères `st` renvoyant la chaîne obtenue en inversant l'ordre des caractères.
Par exemple, `inverse('azerty')` renvoie `'ytreza'`

Exercice 5. —

1. Écrire une fonction récursive `somme(t)` qui prend en argument une liste `t` de nombres et renvoie la somme des termes de cette liste.
2. Expliquer avec l'exemple `somme([4, 7, 2])` le déroulement de l'exécution de la fonction.

Exercice 6. — Écrire une fonction récursive `appartient(v, t, i)` prenant en paramètres une valeur `v`, une liste `t` un entier `i` et renvoyant `True` si `v` apparaît dans `t` entre l'indice `i` (inclus) et `len(t)` (exclu), et `False` sinon.
On supposera que `i` est toujours compris entre 0 et `len(t)`.

Exercice 7. —

1. Écrire une version itérative de la fonction `expo_rapide(x, n)` qui prend en paramètres un nombre `x` et un entier positif `n` et renvoyant la valeur de x^n.

   Pour ce faire, on note que $x^0 = 1$ et on remarque que :

   - si $n = 2k$ (c'est-à-dire si n est pair), alors $x^n = x^{2k} = \left(x^2\right)^k$;
   - si $n = 2k+1$ (c'est-à-dire si n est impair), alors $x^n = x^{2k+1} = x \times \left(x^2\right)^k$.

2. Écrire une version récursive non terminale de cette fonction.
3. Écrire une version récursive terminale de cette fonction.

Exercice 8. — Écrire une fonction récursive `nombre_de_chiffres(n)` qui prend un entier positif ou nul `n` en argument et renvoie son nombre de chiffres.
Par exemple, `nombre_de_chiffres(34126)` doit renvoyer 5.

Exercice 9. — En s'inspirant de l'exercice précédent, écrire une fonction récursive `nombre_de_bits_1(n)` qui prend un entier positif ou nul et renvoie le nombre de bits valant 1 dans la représentation binaire de `n`.
Par exemple, `nombre_de_bits_1(255)` doit renvoyer 8.

Exercice 10. —

1. Expliquer quel est le résultat renvoyé par la fonction `mystere`.

```
def mystere(n):
    if n < 2:
        return str(n)
    else:
        return mystere(n//2) + str(n%2)
```

2. Écrire une fonction récursive `binaire(r, n)` qui prend en arguments un entier relatif `r` et un entier naturel non nul `n` et qui renvoie la représentation machine de `r` sur `n` bits en utilisant la méthode du complément à deux (si `r` est négatif).

   Pour ce faire,

   - on suppose que l'entier `r` peut être représenté sur `n` bits, soit :

$$-2^{n-1} \leqslant r \leqslant 2^{n-1} - 1 ;$$

- on n'écrit pas les 0 inutiles.

3. Compléter le code de la fonction avec un compteur passé en paramètre afin d'obtenir un résultat composé d'exactement n caractères.

Exercice 11. — On dispose d'une fonction `nettoie` qui prend en argument une liste triée d'entiers et élimine les éléments identiques.
Par exemple, si `lst = [1, 1, 2, 6, 6, 6, 8, 8, 9, 10]`, après l'instruction `nettoie(lst)`, `lst` est modifiée et a pour valeur `[1, 2, 6, 8, 9, 10]`.

```
def nettoie(t):
    n = len(t)
    k = 0
    while k < n - 1:
        if t[k] != t[k+1]:
            k = k + 1
        else:
            del t[k]
            n = len(t)
```

Écrire une version récursive de cette fonction.

Exercice 12. — Tester le programme ci-dessous puis en écrire une version récursive de la fonction `dessin`.

```
from turtle import *

couleurs = ['blue', 'green', 'yellow', 'orange', 'red', 'purple']

def dessin():
    for i in range(180):
        color(couleurs[i%6])
        forward(i)
        right(59)

bgcolor('black')
dessin()
```

Exercice 13. — La courbe de Koch est une figure qui s'obtient de manière récursive. Le cas à l'ordre 0 de la récurrence est simplement le dessin d'un segment d'une certaine longueur l, comme ci-dessous (figure de gauche).

Le cas récursif d'ordre n s'obtient en divisant le segment en trois morceaux de même longueur $l/3$, puis en dessinant un triangle équilatéral dont la base est le morceau du milieu, en prenant soin de ne pas dessiner cette base. Cela forme une sorte de chapeau dessiné sur la figure de droite ci-dessus. On réitère ce processus à l'ordre $n-1$ pour chaque segment de ce chapeau (qui sont tous de longueur $l/3$). Par exemple, les courbes obtenues à l'ordre 2 et 3 sont données ci-dessous (à gauche et à droite respectivement).

Écrire une fonction `koch(l, n)` qui dessine avec `Turtle` un flocon de Koch de profondeur `n` à partir d'un segment de longueur `l`.