Récursivité

Définitions

Une fonction récursive comprend dans sa définition un ou plusieurs appels vers elle-même.
Mal écrite, une fonction récursive peut ne jamais s'arrêter.
Une fonction récursive doit par conséquent comprendre une condition d'arrêt.
Une fonction qui n'est pas récursive est dite itérative.

Exercice 1

La factorielle $n!$ d'un nombre entier $n \geq 1$ est définie par $n! = 1 \times 2 \times 3 \times \cdots \times n$.

1. Compléter le tableau ci-dessous (calculatrice interdite !).

n	1	2	3	4	5	6	7
$n!$							

2. Quelle est la relation de récurrence vérifiée par la factorielle ?
(En d'autres termes, exprimer $n!$ en fonction de $(n-1)!$)

3. On considère la fonction ci-dessous.

```python
def factorielle(n):
    return n*factorielle(n-1)
```

(a) Expliquer pourquoi cette fonction ne renvoie pas la valeur attendue pour l'appel `factorielle(4)`.

(b) Corriger la fonction pour qu'elle renvoie bien la valeur de $n!$ pour une valeur entière du paramètre n.

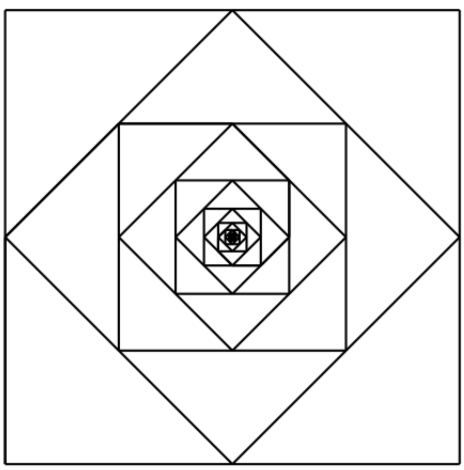
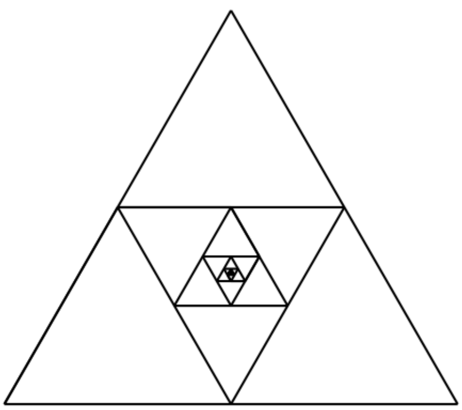
(c) Écrire un jeu de test permettant de vérifier les résultats (en se basant sur la première question).

(d) Ajouter la précondition sur le type de n.

(e) Combien d'appels récursifs sont réalisés lors de l'appel `factorielle(4)` ?

Exercice 2

Réaliser les figures suivantes en utilisant des fonctions récursives.


Exercice 3

On considère la fonction récursive `fibo` ci-dessous, qui prend en argument un nombre entier.

```python
def fibo(n):
    if n<=1: # condition d'arrêt
        return n
    return fibo(n-1)+fibo(n-2)
```

1. Quelle valeur renvoie l'appel `fibo(6)` ?
2. Combien d'appels récursifs ont été nécessaires ?
3. Proposer une version itérative de la fonction `fibo`.

Exercice 4

La suite logistique de paramètre $p \in [0\ ;\ 4]$ est la suite (u_n) définie par

$$\begin{cases} u_0 &= \frac{1}{2} \\ u_n &= p\, u_{n-1}\,(1 - u_{n-1}) \qquad \text{pour tout entier naturel } n > 0. \end{cases}$$

La fonction `logistique` ci-dessous renvoie une valeur approchée de u_n.

```python
def logistique(p, n):
    if n==0 : # condition d'arrêt
        return 0.5
    else :
        return p*logistique(p, n-1)*(1-logistique(p, n-1))
```

1. Déterminer le nombre d'appels récursifs effectués lors de l'appel `logistique(p, n)`.
2. Quelle amélioration peut-on apporter pour diminuer considérablement le nombre d'appels récursifs ?

Exercice 5

On considère un exemple de récursivité croisée : deux fonctions `foo` et `bar` qui s'appellent l'une l'autre.

```python
def foo(n):
    if (n==0) : # condition d'arrêt
        return True
    else :
        return bar(n-1)

def bar(n):
    if (n==0) : # condition d'arrêt
        return False
    else :
        return foo(n-1)
```

1. Quelle est la valeur renvoyée par l'appel `foo(2020)` ? Par l'appel `bar(2020)` ?
2. Combien d'appels récursifs sont effectués lors de l'appel `foo(2020)` ?
3. Proposer des versions itératives des fonctions `foo` et `bar`.

Exercice 6

La fonction d'Ackermann $A(m,n)$ est définie pour tous entiers naturels m et n par

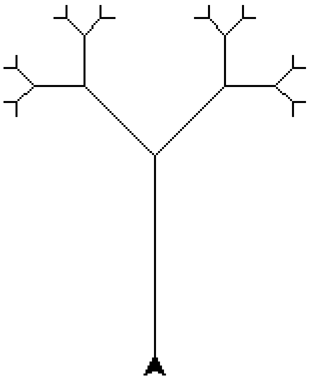
$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m, n-1)) & \text{si } m > 0 \text{ et } n > 0. \end{cases}$$

1. Écrire une fonction récursive `ackermann(m, n)` qui renvoie la valeur de $A(m,n)$.
   Vérifier que $A(3,3) = 61$ et $A(3,4) = 125$.
2. Quelle est la valeur de $A(4,1)$? de $A(4,2)$?

Exercice 7

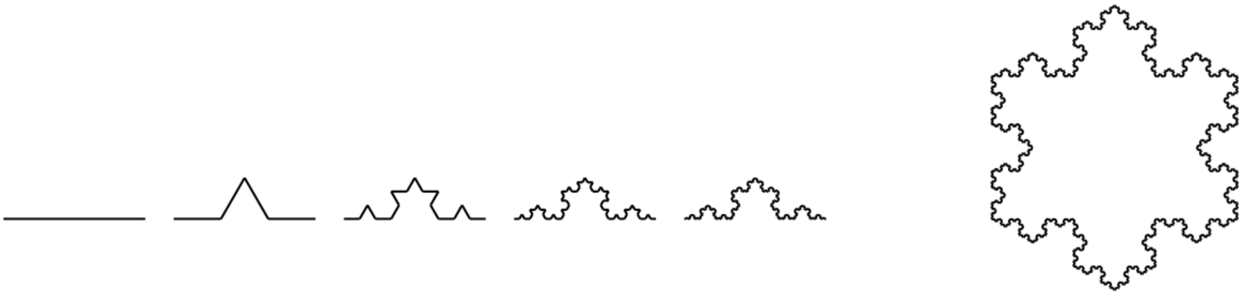
Réaliser la figure ci-dessous en utilisant une fonction récursive.

Chaque branche donne lieu à deux sous-branches dont la longueur est divisée par 2.



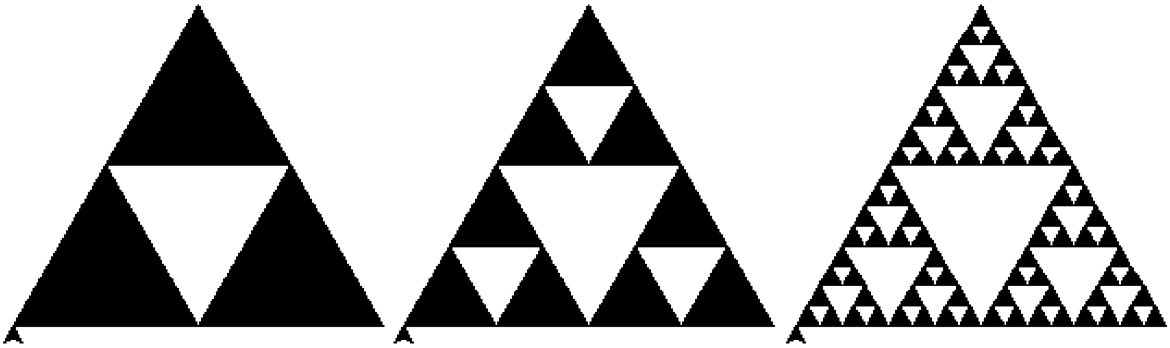
Exercice 8

Écrire une fonction récursive qui permet de réaliser le flocon de Von Koch.



Exercice 9

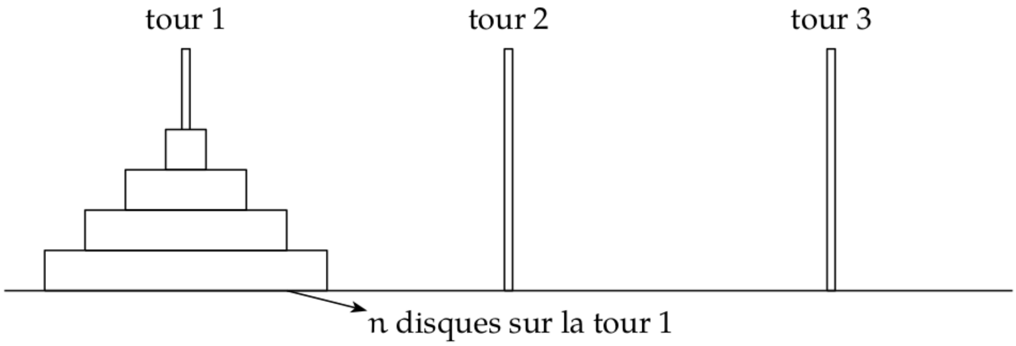
Utiliser une fonction récursive pour réaliser les triangles de Sierpinski ci-dessous.



Exercice 10

Le problème des tours de Hanoï consiste à déplacer tous les disques de la tour 1 sur la tour 3 en respectant les deux règles suivantes :

- on ne peut déplacer qu'un disque à la fois ;
- il n'est pas permis de déposer un disque sur un autre plus petit.



Ce problème admet une solution récursive simple :

- déplacer $n-1$ disques vers la tour 2 ;
- déplacer le dernier disque de la tour 1 vers la tour 3 ;
- déplacer les $n-1$ disques de la tour 2 vers la tour 3.

Écrire une fonction récursive `hanoi(n, dep, dest, inter)` qui affiche la liste des déplacements à effectuer pour déplacer n disques de la tour `dep` sur la tour `dest` en utilisant la tour `inter`.

Exercice 11

La valeur d'un nombre écrit en chiffres romains s'obtient selon le principe ci-dessous :

- si le nombre romain n'est constitué que d'un chiffre, sa valeur est donnée par la correspondance ci-dessous :

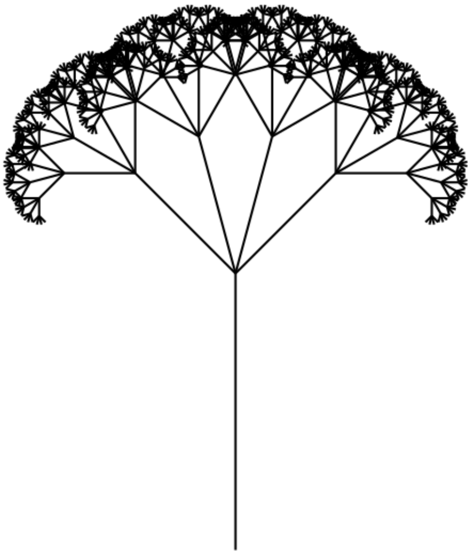
  $*$ M=1000 $*$ D=500 $*$ C=100 $*$ L=50 $*$ X=10 $*$ V=5 $*$ I=1
- Si le premier chiffre a une valeur inférieure au deuxième, alors on soustrait sa valeur à celle de tout le reste, sinon on additionne sa valeur à celle de tout le reste.

Écrire une fonction récursive `valeur_dec(s)` qui prend en argument une chaîne de caractères représentant un nombre écrit en chiffres romains, et renvoie sa valeur décimale.

Exercice 12

Réaliser la figure ci-dessous en utilisant une fonction récursive.

Chaque branche donne lieu à quatre sous-branches formant des angles de $30°$ entre elles, et dont la longueur est divisée par 2.



Exercice 13

Utiliser une fonction récursive pour réaliser l'arbre de Pythagore.