Chapitre 4 - Exercices sur la récursivité

1. Prouver la correction et la terminaison de la fonction suivante avec $(n, m) \in \mathbb{N}^2$

```
def somme(m,n):
    if n == 0:
        return m
    else:
        return 1+somme(m,n-1)
```

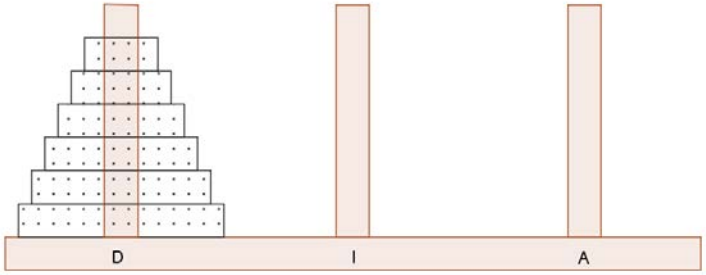
En écrire une version récursive terminale.

2. On considère la suite de Fibonacci définie par $F_0 = 0$ et $F_1 = 1$, puis pour tout entier $n \geq 2$, $F_n = F_{n-1} + F_{n-2}$.

Ecrire deux fonctions (l'une itérative, l'autre récursive) prenant en argument un entier n et retournant F_n, le terme de rang n de la suite de Fibonacci. Comparer les deux. Améliorer si nécessaire la version récursive.

3. Le problème des tours de Hanoi consiste à déplacer tous les disques d'une tour de départ D à une tour d'arrivée A en s'aidant d'une tour intermédiaire I et en respectant les règles suivantes :

- On ne peut déplacer qu'un disque à la fois
- On ne peut pas placer un disque sur un disque plus petit



Ecrire une fonction d'entête **def** hanoi(n, D='D', A='A', I='I'): affichant, pour tout entier n, la solution du problème des tours de Hanoi à n disques en indiquant la liste de déplacements (Déplacement de D sur A, de D sur I, de A sur I…).

Prouver la correction et la terminaison de votre fonction et étudier sa complexité (nombre de déplacements).

4. *Fonction de McCarthy, $n \in \mathbb{N}$ (vu dans un sujet de l'X)*

```
def f(n):
    if n > 100:
        return n-10
    else:
        return f(f(n+11))
```

Cet algorithme termine-t-il ? Que fait-il ?

5. *Fonction d'Ackermann*

Soit la fonction A définie pour tout $(m, n) \in \mathbb{N}^2$ par :

$A(0, n) = n + 1$
$A(m+1, 0) = A(m, 1)$
$A(m+1, n+1) = A(m, A(m+1, n))$

a) Démontrer que la fonction A est bien définie sur $\mathbb{N}^2$.
b) Calculer $A(m,n)$ pour $(m, n) \in [[0 ; 2]] \times [[0 ; 3]]$.
c) Programmer la fonction d'Ackermann en Python.
d) Exprimer $A(1,n)$ et $A(2,n)$ en fonction de n, pour $n \in \mathbb{N}$.
e) Montrer que $A(3, n) = 2^{n+3} - 3$ pour tout $n \in \mathbb{N}$.

La fonction d'Ackermann, fournit un exemple de fonction récursive mais non primitive récursive (notions totalement hors programme – pour simplifier, les fonctions récursives sont toutes les fonctions calculables, et les fonctions primitives récursives sont les fonctions pouvant être programmées avec des boucles for mais sans boucle while ou récursivité).

6. *Anagrammes*

Écrire une fonction récursive affichant tous les anagrammes d'un mot.

Indications au verso

Indication 1
Étant donné un mot $m = l_1 l_2 \ldots l_n$ où l_1, l_2, …, l_n sont les lettres du mot. Alors l'ensemble des anagrammes du mot *m* est l'ensemble des mots commençant par une lettre l_i suivi d'un anagramme du mot $l_1 l_2 .. l_{i-1} l_{i+1} \ldots l_n$ (donc du mot *m* privé de la lettre l_i).

Indication 2
On pourra écrire une fonction d'entête **def** anagramme(mot, ana='').
À chaque appel récursif, mot diminue d'une lettre tandis que la longueur des anagrammes augmente de 1.

6bis. Écrire une fonction récursive renvoyant la liste de tous les anagrammes d'un mot.

7. *Conversion décimal → binaire*
Un algorithme de conversion de l'écriture décimale vers l'écriture binaire s'appuie sur la méthode de l'algorithme d'Hörner.

Exemple :

$13 = 6 \times 2 + 1$
$= (3 \times 2 + 0) \times 2 + 1$
$= ((\mathbf{1} \times 2 + \mathbf{1}) \times 2 + \mathbf{0}) \times 2 + \mathbf{1}$
$= \mathbf{1} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2 + \mathbf{1}$
$13 = (1101)_2$

En pratique, on obtient l'écriture binaire de 13 en effectuant des divisions euclidiennes successives par 2.
Les chiffres de l'écriture binaire de 13 sont les restes de ces divisions euclidiennes.

On présente généralement ainsi :

13	1
6	0
3	1
1	1
0	

a) Convertir de même 105 en binaire.
b) Écrire une fonction récursive binaire renvoyant une liste contenant les chiffres de l'écriture en binaire d'un entier n non nul. Ainsi, binaire(13) renverra [1, 1, 0, 1].