Exercices sur la récursivité

(1) Ecrire une fonction récursive qui calcule la factorielle d'un entier naturel n. On rappelle que $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$, si $n \geq 1$ et $0! = 1$.

(2) Ecrire une fonction récursive qui calcule le pgcd de deux entiers naturels a et b par la méthode d'Euclide.

(3) Ecrire une fonction récursive qui calcule le n^e nombre de Fibonacci. On rappelle que la suite de Fibonacci est définie par : $F_0 = 1$, $F_1 = 1$ et $F_n = F_{n-1} + F_{n-2}$, pour tout entier $n \geq 2$.

(4) a) Ecrire une fonction récursive qui calcule le n^e terme d'une suite arithmétique de premier terme a et de raison r données. b) Même exercice en prenant une suite géométrique.

(5) a) Ecrire une fonction récursive qui calcule x^n où x est un nombre réel (de type `float`) et n est un entier naturel. b) Améliorer la fonction en utilisant l'algorithme rapide vu en classe de 2^e. c) Etendre la fonction aux exposants négatifs.

(6) Ecrire deux fonctions récursives qui calculent respectivement : a) le nombre de chiffres et b) la somme des chiffres d'un entier naturel donné.

(7) a) Ecrire une fonction qui calcule récursivement le produit de deux entiers naturels. b) Etendre la fonction au cas d'entiers relatifs.

(8) Ecrire deux fonctions récursives : la première compte le nombre de lettres d'une une chaîne de caractères et la deuxième inverse une chaîne de caractères.

(9) Ecrire une fonction booléenne et récursive qui teste si une chaîne de caractères donnée est une anagramme d'une autre chaîne de caractères donnée. Par exemple : `'algorithme'` est une anagramme de `'logarithme'`.

(10) Ecrire une fonction récursive qui retourne le maximum (resp. le minimum) d'une liste de nombres.

(11) Ecrire une fonction qui prend en entrée une chaîne de caractères sous la forme d'une somme de réels, par ex. $'-1.37 + 40 - 10.08 + 7 - 244'$ et qui évalue récursivement cette somme.

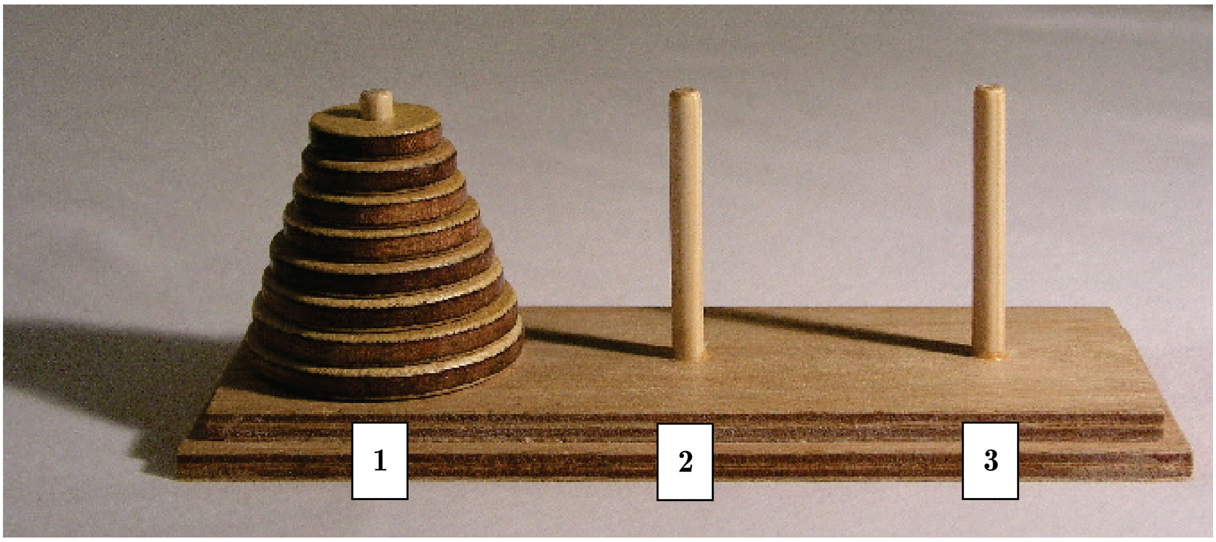
(12) Ecrire une fonction récursive qui trie de façon récursive une liste du plus petit au plus grand élément par la méthode du tri par sélection, vue en 2^e.

(13) a) Ecrire une fonction récursive qui retourne la notation binaire d'un entier naturel. Par exemple : $25\ (= \mathbf{1} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0) \rightarrow 11001$.

b) Ecrire une fonction récursive qui retourne la notation décimale d'un nombre binaire. Par exemple : 11001 $\to$ 25.

(14) Ecrire une version récursive de l'algorithme de Horner permettant d'évaluer un polynôme donné $p(x)$ en un réel donné x_0. Le polynôme est transmis à la fonction sous forme de la liste de ses coefficients, suivant les puissances croissantes. Par exemple, le polynôme $p(x) = 4x^3 - 5x + 7$ est représenté par la liste [7,–5,0,4]

(15) Le jeu des *Tours de Hanoï* est constitué de trois piquets verticaux, notés 1, 2 et 3 et de n disques superposés de tailles strictement décroissantes avec un trou au centre et enfilés autour du piquet 1 ; ces disques forment les tours.



Le but du jeu consiste à déplacer l'ensemble des disques pour que ceux-ci se retrouvent enfilés autour du piquet 3 en respectant les règles suivantes :

- les disques sont déplacés un par un ;
- un disque ne doit pas se retrouver au-dessus d'un disque plus petit.

Le problème se résout de manière récursive. En effet, supposons le problème résolu pour $n-1$ disques c.-à-d. que l'on sache transférer $n-1$ disques depuis le piquet $i \in \{1,2,3\}$ jusqu'au piquet $j \in \{1,2,3\} \setminus \{i\}$ en respectant les règles du jeu. Pour transférer n disques du piquet i vers le piquet j :

- on amène les $n-1$ disques du haut du piquet i sur le piquet intermédiaire, qui a le numéro $6-i-j$ (ici il faut faire l'appel récursif);
- on prend le dernier disque du piquet i et on le met seul en j ;
- on ramène les n-1 disques de $6-i-j$ en j (encore un appel récursif).

On demande d'écrire une procédure récursive **`Hanoi`** qui permet d'afficher dans une liste les mouvements élémentaires à accomplir pour déplacer n disques du piquet i au piquet j. Par exemple : 1→2,1→3,2→3 lorsque $n = 2$.

(16) Voici un algorithme récursif pour résoudre une grille de SUDOKU. Le but de l'exercice est de compléter l'algorithme par les fonctions manquantes et bien sûr de comprendre son fonctionnement.

```python
def solveSudoku(grid, i=0, j=0):
    i, j = findNextCellToFill(grid)
    if i == -1:
        return True
    for e in range(1, 10):
        if isValid(grid, i, j, e):
            grid[i][j] = e
            if solveSudoku(grid, i, j):
                return True
            # Undo the current cell for backtracking
            grid[i][j] = 0
    return False
```

a) **`grid`** est une matrice 9x9 contenant les chiffres de 0 à 9. Le 0 signifie que la cellule n'est pas encore remplie.

b) Ecrire la fonction **`findNextCellToFill`**, qui cherche et retourne les coordonnées d'une cellule vide, donc contenant le chiffre 0, sinon elle retourne **`-1, -1`**.

c) Ecrire la fonction **`rowOK(grid,i,e)`** qui retourne **`True`** si le chiffre **`e`** ne se trouve pas encore dans la ligne numéro **`i`**, **`False`** sinon.

d) Ecrire de même la fonction **`columnOk(grid,j,e)`** qui retourne **`True`** si le chiffre **`e`** ne se trouve pas encore dans la colonne numéro **`j`**, **`False`** sinon.

e) Ecrire de même la fonction **`sectorOk(grid,i,j,e)`** qui retourne **`True`** si le chiffre **`e`** ne se trouve pas encore dans le « secteur » contenant la cellule aux coordonnées **`(i,j`**, **`False`** sinon. Exemple de « secteur » :

```
-------------------------------------
¦  8  0  0 ¦  0  0  0 ¦  0  0  0 ¦
¦  0  0  3 ¦  6  0  0 ¦  0  0  0 ¦
¦  0  7  0 ¦  0  9  0 ¦  2  0  0 ¦
-------------------------------------
¦  0  5  0 ¦  0  0  7 ¦  0  0  0 ¦
¦  0  0  0 ¦  0  4  5 ¦  7  0  0 ¦
¦  0  0  0 ¦  1  0  0 ¦  0  3  0 ¦
-------------------------------------
¦  0  0  1 ¦  0  0  0 ¦  0  6  8 ¦
¦  0  0  8 ¦  5  0  0 ¦  0  1  0 ¦
¦  0  9  0 ¦  0  0  0 ¦  4  0  0 ¦
-------------------------------------
```

Secteur défini par la cellule aux coordonnées (4,8).

f) Déduire de c), d) et e) la fonction **isValid(grid,i,j,e)** qui retourne **True** si on peut placer le chiffre **e** dans la cellule **(i,j)** de la grille sans violer les 3 règles du Sudoku, **False** sinon.

g) Ecrire une fonction **printSudoku(grid)** qui affiche à l'écran une grille de Sudoku dans le format de la question e)

h) Tester le programme sur la grille de Sudoku suivante (affichée aussi ci-dessus) :

```
hardest_sudoku = [
    [8,0,0,0,0,0,0,0,0],
    [0,0,3,6,0,0,0,0,0],
    [0,7,0,0,9,0,2,0,0],
    [0,5,0,0,0,7,0,0,0],
    [0,0,0,0,4,5,7,0,0],
    [0,0,0,1,0,0,0,3,0],
    [0,0,1,0,0,0,0,6,8],
    [0,0,8,5,0,0,0,1,0],
    [0,9,0,0,0,0,4,0,0]
]
```

i) Expliquer le fonctionnement de la fonction récursive **solveSudoku**.

(17) Voici une fonction **f** d'un ancien examen de fin d'études secondaires :

```
def f(s):
    if len(s) <= 1:
        return s
    if len(s) == 2:
        return s[1] + s[0]
    return s[-1] + f(s[1:len(s) - 1]) + s[0]
```

a) Exécutez cette fonction en prenant comme argument **'recursive'**. (Ecrire toutes les étapes !)

b) Que retourne cette fonction en général ?

c) On peut simplifier cette fonction sans que le résultat soit différent ! Quelles lignes sont superflues ?

d) Ecrire une version itérative de cette fonction.

(18) Voici une fonction **g** un peu plus compliquée que la fonction **f** de l'exercice précédent :

```
def g(s):
    if len(s) <= 1:
        return s
    if len(s) // 2 % 2 == 1:
        return s[-1] + g(s[1:len(s) - 1]) + s[0]
    return s[0] + g(s[1:len(s) - 1]) + s[-1]
```

a) Exécutez cette fonction en prenant successivement comme arguments **''**, **'a'**, **'ab'**, **'abc'** etc., jusqu'à ce que vous ayez compris le mécanisme.

b) Prévoyez (sans tricher ☺) les résultats de :

- **`g('exam')`**
- **`g('function')`**
- **`g('recursive')`**
- **`g('recursively')`**

Vérifiez ensuite !

c) Un informaticien veut utiliser la fonction **`g`** pour coder un à un tous les mots d'un texte. Ecrire une fonction **`code(text)`** qui permet de coder automatiquement chaque mot d'un texte passé en paramètre, en supposant que les différents mots de ce texte sont séparés par un espace.

d) Ecrire ensuite une fonction **`decode(text)`** qui permet de décoder un texte dont tous les mots ont été codés à l'aide de la fonction **`g`**.

(19) On donne la fonction booléenne récursive suivante :

```
def mystery(a, b):
    if a == '':
        return True
    if a[0] not in b:
        return False
    return mystery(a[1:], b)
```

a) Calculer en précisant toutes les étapes :

`mystery('musique','mathematiques')`

`mystery('musicien','instrument')`

b) Donner trois exemples de strings **`x`** avec des longueurs différentes tels que la condition **`mystery(x,'abc') and mystery('abc',x)`** soit égale à **`True`**.

c) Expliquer ce que calcule la fonction **`mystery`** en général.

d) Ecrire une version itérative de cette fonction.