# Récursivité 2

**Exercice 1 : Numérotation de grille sudoku**

Pour écrire un programme de résolution d'un sudoku, on utilise un tableau de nombres à deux dimensions pour définir la grille. Par exemple :

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

Les éléments de la grille sont accessibles par leurs coordonnées dans le tableau :
- ligne 0 : G[0][0], G[0][1], G[0][2], ..., G[0][8]
- ligne 1 : G[1][0], G[1][1], G[1][2], ..., G[1][8]
- ...
- ligne 8 : G[8][0], G[8][1], G[8][2], ..., G[8][8]

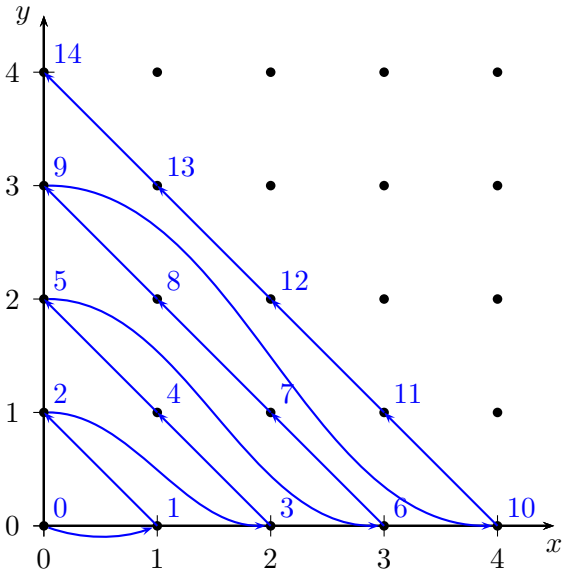
On décide de repérer chaque cellule par un entier. La cellule de coordonnées (i ; j) (avec $0 \leq i \leq 8, 0 \leq j \leq 8$) est repérée par numero(i, j) avec la définition ci-dessous :

```
def numero(i, j):
    if i == 0 and j == 0: return 0
    elif j > 0: return numero(i, j - 1) + 1
    else: return numero(i - 1, 8) + 1
```

Dessiner une grille de sudoku vide dont on remplira chaque cellule avec son numéro. En déduire alors une fonction numeroBis plus simple, non récursive, prenant en entrée deux entiers i et j et renvoyant le numéro de la cellule de coordonnées (i ; j).

**Exercice 2 : La diagonale de Cantor**

On numérote chaque point du plan de coordonnées (x ; y) (où x et y sont des entiers naturels) par le procédé suggéré sur la figure ci-dessous :




Écrire une fonction cantor, prenant en entrée deux entiers x et y, qui renvoie le numéro du point de coordonnées (x ; y). Cette fonction sera définie de manière récursive.

**Exercice 3 : Comptage de chiffres**

On rappelle que le quotient de la division euclidienne d'un entier n par 10 donne le nombre de dizaines de cet entier. Le quotient de la division euclidienne de n = 5478 par 10 est par exemple 547.

En déduire une fonction nbChiffres, prenant en entrée un entier naturel n (écrit en décimal), qui renvoie le nombre de chiffres de cet entier n écrit en base 10. Cette fonction sera définie de manière récursive.

**Exercice 4 : Un peu de complexité**

On rappelle que la suite de Fibonacci est définie de manière récursive par :

$$\begin{cases} fibo_0 = fibo_1 = 1 \\ \forall n \geq 2, fibo_n = fibo_{n-1} + fibo_{n-2} \end{cases}$$

On donne deux fonctions qui calculent toutes les deux le terme de rang n de cette suite :

**Fonction fibo_naive**

Entrée :
$n$ est un entier naturel.

Corps de l'algorithme :
1 Si $n \leq 1$, alors
2 Renvoyer 1
3 Sinon
4 Renvoyer $fibo\_naive(n-1) + fibo\_naive(n-2)$
5 Fin Si

**Fonction fibo_acc**

Entrée :
$n$ est un entier naturel ; $res1$ et $res2$ sont deux entiers naturel qui valent 1 par défaut.

Corps de l'algorithme :
1 Si $n \leq 1$, alors
2 Renvoyer $res2$
3 Sinon
4 Renvoyer $fibo\_acc(n-1, res2, res1 + res2)$
5 Fin Si

1. Combien y a-t-il d'appels à la fonction fibo_naive quand on lance fibo_naive(5) ?
2. Combien y a-t-il d'appels à la fonction fibo_acc quand on lance fibo_acc(5) ?
3. Programmer fibo_naive et fibo_acc. Essayer différentes valeurs de n pour comparer la rapidité de ces deux fonctions.