

## DS Informatique Sudoku

### Notations

Pour  $m$  et  $n$  deux entiers naturels,  $\llbracket m; n \rrbracket$  désigne l'ensemble des entiers  $k$  tels que  $m \leq k \leq n$ .

*La résolution d'une grille de Sudoku est une gymnastique du cerveau qui peut être assimilée à un décodage correcteur d'effacement. En effet, à partir d'une grille presque vide, il est possible (pour une grille bien faite) de la compléter d'une unique manière.*

*L'objectif de cet exercice est de mettre en œuvre une méthode naïve permettant de compléter une grille de Sudoku.*

Une grille de Sudoku est une grille de taille  $9 \times 9$ , découpée en 9 carrés de taille  $3 \times 3$ . Le but est de la remplir avec des chiffres de  $\llbracket 1; 9 \rrbracket$ , de sorte que chaque ligne, chaque colonne et chacun des 9 carrés de taille  $3 \times 3$  contienne une et une seule fois chaque entier de  $\llbracket 1; 9 \rrbracket$ . Lorsqu'un chiffre n'apparaît jamais deux fois sur une même ligne, une même colonne ni un même carré, on dit que la grille est **correcte**. Lorsque toutes les cases sont occupées, on dit que la grille est **complète**. En pratique, certaines cases sont déjà remplies et on fera l'hypothèse que le Sudoku qui nous intéresse est à la fois correct et bien écrit, c'est à dire qu'il possède une unique solution.

On représente en Python une grille de Sudoku par une liste de taille  $9 \times 9$ , c'est-à-dire une liste de 9 listes de taille 9, dans laquelle les cases non remplies sont associées au chiffre 0. Ainsi, la grille suivante est représentée par la liste ci-contre :

	6					2		5
4			9	2	1			
	7				8			1
					5			9
6	4						7	3
1			4					
3			7				6	
			1	4	6			2
2		6					1	

$L = \llbracket [0, 6, 0, 0, 0, 0, 2, 0, 5],$   
 $[4, 0, 0, 9, 2, 1, 0, 0, 0],$   
 $[0, 7, 0, 0, 0, 8, 0, 0, 1],$   
 $[0, 0, 0, 0, 0, 5, 0, 0, 9],$   
 $[6, 4, 0, 0, 0, 0, 0, 7, 3],$   
 $[1, 0, 0, 4, 0, 0, 0, 0, 0],$   
 $[3, 0, 0, 7, 0, 0, 0, 6, 0],$   
 $[0, 0, 0, 1, 4, 6, 0, 0, 2],$   
 $[2, 0, 6, 0, 0, 0, 0, 1, 0] \rrbracket$

Les 9 carrés de taille  $3 \times 3$  sont numérotés du haut à gauche jusqu'en bas à droite. Ainsi, sur cette grille, le carré 0, en haut et à gauche, contient les chiffres 6, 4 et 7; le carré 1, en haut et au milieu, contient les chiffres 9, 2, 1 et 8; le carré 8, en bas et à droite, contient les chiffres 6, 2 et 1.

On rappelle que les lignes du Sudoku sont alors les éléments de  $L$  accessibles par  $L[0], \dots, L[8]$ . L'élément de la case  $(i, j)$  est accessible par  $L[i][j]$ .

**Remarque :** on fera bien attention, dans l'ensemble de ce sujet, aux indices des tableaux. Les lignes, ainsi que les colonnes, sont indicées de 0 à 8.

## Partie A : Généralités

On montre aisément que si une grille de Sudoku est complète, alors pour chacune des lignes, chacune des colonnes et chacun des carrés de taille  $3 \times 3$ , la somme des chiffres fait 45.

1. Ecrire une fonction `ligne_complete(L, i)` qui prend une liste Sudoku `L` et un entier `i` entre 0 et 8, et renvoie `True` si la ligne `i` du Sudoku `L` vérifie les conditions de remplissage d'un Sudoku, c'est à dire que la somme fait 45, et `False` sinon. On ne cherchera pas à déterminer si la ligne est correcte.
2. Ecrire OBLIGATOIREMENT avec un `while` (même si vous pouvez le faire avec un `for`!) la fonction `colonne_complete(L, i)` qui vérifie la même chose pour la colonne `i`.
3. Recopier et compléter la fonction suivante `carre_complet(L, i)` pour le carré `i` qui utilise **2 boucles imbriquées** pour vérifier si la somme de toutes les cases d'un carré de 3 cases est égale à 45. Encore une fois, on ne cherchera pas à déterminer si le carré est correct.

```
def carre_complet(L, i):
    ligne = 3 * (i // 3)
    colonne = 3 * (i % 3)
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
```

4. Ecrire une fonction `complet(L)` qui prend une liste Sudoku `L` comme argument, et qui renvoie `True` si la grille est complète, `False` sinon.
5. Recopier et compléter la fonction suivante `ligne(L, i)`, qui renvoie la liste des nombres compris entre 1 et 9 qui apparaissent sur la ligne d'indice `i`.

```
def ligne(L, i):
    chiffre = []
    ...
    ...
    ...
    return chiffre
```

Avec la grille donnée dans l'énoncé, on doit obtenir :

```
>>> ligne(L, 0)
[6, 2, 5]
```

On définit alors, de la même manière, la fonction `colonne(L, j)` qui renvoie la liste des nombres compris entre 1 et 9 qui apparaissent dans la colonne `j` (*on ne demande pas d'écrire son code*).

6. On se donne une case  $(i, j)$ , avec  $(i, j) \in \llbracket 0; 8 \rrbracket^2$ .

Montrer que la case en haut à gauche du carré  $3 \times 3$  auquel appartient la case  $(i, j)$  a pour coordonnées

$$\left( 3 \times \left\lfloor \frac{i}{3} \right\rfloor, 3 \times \left\lfloor \frac{j}{3} \right\rfloor \right)$$

où  $\lfloor x \rfloor$  représente la partie entière de  $x$ .

7. Recopier et compléter alors la fonction `carre(L, i, j)`, qui renvoie la liste des nombres compris entre 1 et 9 qui apparaissent dans le carré  $3 \times 3$  auquel appartient la case  $(i, j)$ .

```
def carre(L, i, j):
    icoin = 3 * (i // 3)
    jcoin = 3 * (j // 3)
    ...
    ...
    ...
    ...
    ...
    return chiffre
```

Avec la grille donnée dans l'énoncé, on doit obtenir :

```
>>> carre(L, 4, 6)
[9, 7, 3]
>>> carre(L, 4, 5)
[5, 4]
```

8. Ecrire une fonction `chiffres_ok(L, i, j)` qui renvoie la liste des chiffres que l'on peut écrire en case  $(i, j)$ . Si la case  $(i, j)$  est déjà remplie, la fonction renvoie une liste vide.

Par exemple, avec la grille initiale :

```
>>> chiffres_ok(L, 4, 2)
[2, 5, 8, 9]
```

On pourra, dans la suite du sujet, utiliser les fonctions définies précédemment.

## Partie B : Algorithme naïf

Naïvement, on commence par compléter les cases n'ayant qu'une seule possibilité.

Nous prendrons dans la suite comme Sudoku :

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

9. A partir des fonctions précédentes, écrire une fonction `nb_possible(L, i, j)`, indiquant le nombre de chiffres possibles à la case  $(i, j)$ .
10. On souhaite disposer de la fonction `un_tour(L)` qui parcourt l'ensemble des cases du Sudoku et qui complète les cases dans le cas où il n'y a qu'un chiffre possible, et renvoie `True` s'il y a eu un changement, et `False` sinon. La liste `L` est alors modifiée.

Par exemple, en partant de la grille initiale `M` :

```
>>> un_tour(M)
True
>>> M
[[2, 0, 0, 0, 9, 0, 3, 0, 0], [0, 1, 9, 0, 8, 0, 5, 7, 4],
 [0, 0, 8, 4, 0, 0, 6, 2, 9], [5, 9, 0, 6, 2, 1, 4, 8, 7],
 [0, 2, 7, 0, 3, 8, 1, 6, 5], [0, 6, 1, 5, 7, 4, 2, 9, 3],
 [0, 8, 5, 0, 0, 9, 7, 3, 0], [9, 3, 6, 0, 5, 0, 8, 4, 2],
 [0, 0, 2, 0, 6, 0, 9, 5, 1]]
```

On propose la fonction suivante :

```
def un_tour(L):
    changement = False
    for i in range(0, 9):
        for j in range(0, 9):
            if L[i][j] = 0:
                if nb_possible(l, i, j) = 1:
                    L[i][j] = chiffres_ok(L, i, j)[1]
    return changement
```

Recopier ce code en en corrigeant les erreurs.

11. Ecrire une fonction `complete(L)` qui exécute la fonction `un_tour` tant qu'elle modifie la liste, et renvoie `True` si la grille est complétée, et `False` sinon.

### Partie C : Vérification de la grille

12. Etant donné une grille entièrement complétée, on veut déterminer si elle a été correctement remplie.

Ecrire une fonction `ligne_correcte(L, i)` qui a pour paramètres une liste Sudoku `L` entièrement complétée et un entier `i` entre 0 et 8, et qui renvoie `True` si la ligne `i` du Sudoku `L` contient une et une seule fois chaque entier de  $\llbracket 1, 9 \rrbracket$ .

On définit de la même manière les fonctions `colonne_correcte(L, i)` pour la colonne `i` et `carre_correcte(L, i)` pour le carré `i`. On ne demande pas d'écrire ces deux fonctions.

13. Ecrire une fonction `correct(L)` qui prend une liste Sudoku `L` entièrement complétée comme argument, et qui renvoie `True` si la grille est correcte, et `False` sinon."