DS Informatique
Sudoku

Notations
Pour m et n deux entiers naturels, $[\![m, n]\!]$ désigne l'ensemble des entiers k tels que $m \leq k \leq n$.
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 9×9, découpée en 9 carrés de taille 3×3. Le but est de la remplir avec des chiffres de $[\![1, 9]\!]$, de sorte que chaque ligne, chaque colonne et chacun des 9 carrés de taille 3×3 contienne une et une seule fois chaque entier de $[\![1, 9]\!]$.

— Lorsqu'aucun chiffre n'apparaît deux fois sur une même ligne, une même colonne ou un même carré, on dit que la grille est **correcte**. Les cases vides ne comptent pas : une grille vide est donc toujours correcte.

— Lorsque toutes les cases sont remplies, on dit que la grille est **complète**.

— Une grille est dite **bien posée** si on peut la compléter (en rajoutant des chiffres) pour en faire une grille correcte et complète, et ce de manière unique. En particulier, toute grille bien posée est nécessairement correcte.

En pratique, certaines cases sont déjà remplies et on fera l'hypothèse que la grille de Sudoku qui nous intéresse est bien posée. En Python, on représente un Sudoku par une liste de taille 9×9, c'est-à-dire une liste de 9 listes de taille 9, dans laquelle les cases vides 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 = [[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]]
```

De plus, on assigne un numéro aux 9 carrés de taille 3×3, de 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],...,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

Dans tout ce qui suit, on suppose qu'on dispose d'une grille `L` qui est correcte. Dans les algorithmes, on ne cherchera donc pas à vérifier que `L` (donc ses lignes, ses colonnes, ses carrés) sont corrects.

1. 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 (l'ordre importe peu) :

```
>>> 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)*.

   Pour toute case (i, j), on remarque que la case en haut à gauche du carré 3×3 qui contient (i, j) a pour coordonnées

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

2. Recopier et compléter la fonction `carre(L, i, j)`, qui renvoie la liste des nombres compris entre 1 et 9 qui apparaissent dans le carré 3×3 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 (l'ordre importe peu) :

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

```
>>> carre(L, 4, 5)
[5, 4]
```

3. Écrire 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]
```

4. Écrire une fonction `complet(L)` qui prend une liste Sudoku `L` comme argument, et qui renvoie `True` si la grille est complète, `False` sinon.
5. Rappeler la définition d'opération élémentaire (celle vue en TP).
6. Quel est le pire cas (ou les pires cas) pour la fonction `complet` ? Combien d'opérations élémentaires sont effectuées dans ce(s) cas ?

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 la grille `M` ci-dessous :

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

```
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. À partir des fonctions précédentes, écrire une fonction `nb_possible(L, i, j)`, indiquant le nombre de chiffres possibles à la case `(i, j)`.

   On souhaite disposer d'une fonction `un_tour(L)` qui parcourt une fois 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],
 ...
 [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
```

10. Recopier le code ci-desssus en corrigeant les erreurs.
11. Écrire une fonction `naif(L)` qui exécute la fonction `un_tour` tant qu'elle modifie la liste, puis qui renvoie `True` si la grille est complète, et `False` sinon.
12. Étant donné une grille complète, on veut déterminer si elle a été correctement remplie.
    Écrire une fonction `ligne_correcte(L, i)` qui a pour paramètres une grille Sudoku `L` complète 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 $[\![1, 9]\!]$.

    On définit de la même manière les fonctions `colonne_correcte(L, i)` pour la colonne `i` et `carre_correct(L, i)` pour le carré `i`. On ne demande pas d'écrire ces deux fonctions.
13. Écrire une fonction `correct(L)` qui prend une liste Sudoku `L` complète comme argument, et qui renvoie `True` si la grille est correcte, et `False` sinon.