Projet Sudoku

Présentation du jeu

Les règles

Un sudoku classique contient 9 lignes et 9 colonnes, soit 81 cases au total. Ces cases sont aussi groupées par carré 3x3 appelé **bloc** dans ce projet. Il y a donc 9 blocs dans une grille classique (en bleu dans la grille ci-dessous).

Le but du jeu est de remplir ces cases avec des chiffres allant de 1 à 9 en veillant toujours à ce qu'un même chiffre ne figure qu'une seule fois par colonne, une seule fois par ligne, et une seule fois par bloc (carré bleu de neuf cases).

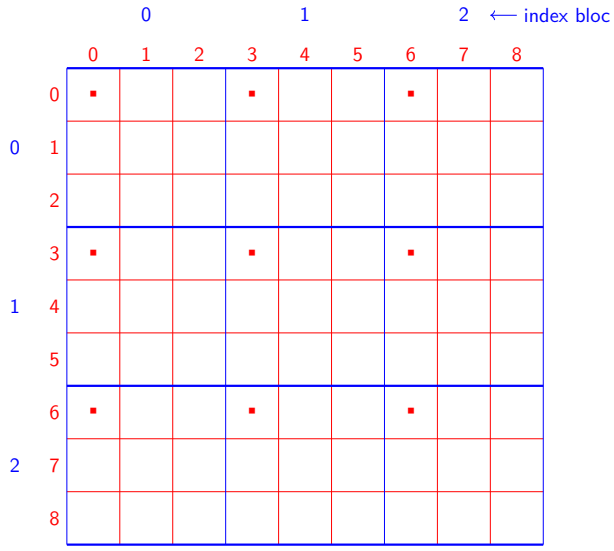
Voici un exemple de grille de sudoku. Elle contient déjà des valeurs et le but est de remplir toutes les autres cases. Cette grille sert de référence à plusieurs questions, c'est la `grille 0` :

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

Apprendre à faire des sudokus

En fin de document, vous avez une page 6 grilles de sudoku de niveau facile que vous pouvez résoudre. Entraînez-vous à faire des sudokus !

Comment représenter une grille de sudoku en python ?



On adopte les conventions de représentation utilisées dans la grille de gauche et le code python ci-dessous.

La grille ci-dessus peut être représentée en python par le code suivant :

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

Il a été choisi de remplacer un manque de valeur dans la grille de sudoku par un 0, puisque les *bonnes* valeurs sont comprises entre 1 et 9. On aurait aussi pu choisir `None`, mais comme on manipule des entiers, autant travailler avec un entier de plus.

Questions 1

On dispose d'une grille de sudoku en python via la variable `grille` comme dans l'exemple ci-dessus.
On désigne une case par un couple d'entier `ligne` et `col`.
Dans les questions qui suivent, il faut écrire le code python qui permet de répondre aux questions.

(1) Comment accéder au contenu de la case (`ligne`, `col`) ?
(2) Comment écrire une valeur dans une case donnée par `ligne` et `col` ?
(3) Quelles valeurs peuvent prendre les entiers `ligne` et `col` ?
(4) Comment savoir si la case est déjà remplie ou si elle est vide ?
(5) Comment parcourir la ligne d'index `ligne` de la grille ?
(6) Comment parcourir la colonne d'index `col` de la grille ?
(7) Comment afficher le contenu case par case de la grille entière ?

(8) Comment parcourir un bloc (carré bleu de 3x3 cases) ? On choisit le procédé suivant :
- je trouve les coordonnées (index `bloc_ligne` et `bloc_col`) de la case en haut à gauche du bloc (case représentée par un carré rouge ci-dessus)
- à partir de ces deux index, je parcours le bloc à l'aide de 2 boucles `for` : `for l in range(3)` et `for c in range(3)`

Comment déterminer les index `bloc_ligne` et `bloc_col` des cases rouges de chaque bloc à partir d'une case quelconque de la grille définie par les index `ligne` et `col` ?

(9) Appliquer le procédé présenté à la question précédente pour parcourir un bloc quelconque à partir d'une case quelconque de la grille.

Valeurs possibles d'une case et singleton

(10) Une compétence en informatique est de savoir découper un problème complexe en sous-problème plus simples. On considère le problème suivant : étant donné une case (via `ligne` et `col`) et une grille de sudoku, on veut déterminer quelles sont les valeurs possibles restantes qu'il est possible de mettre dans cette case en respectant les règles du sudoku (une valeur ne peut apparaître qu'une seule fois par ligne, colonne et bloc). Décomposer ce problème en une série de petits problèmes élémentaires sur papier.

(11) Coder les fonctions suivantes :

```
def valeurs_possibles_ligne(grille, ligne) :
    """
    Affiche toutes les valeurs possibles pour une case appartenant à la ligne ligne

    Entrées (pré-conditions) :
    - grille : tableau de 9 tableaux de 9 cases remplis d'entier [0-9]
    - ligne : index de ligne d'une case

    Sortie (post-conditions):
    - un tableau de toutes les valeurs possibles en tenant compte des contraintes
      en ligne
    """
    pass
```

```python
def valeurs_possibles_col(grille, col) :
    """
    Affiche toutes les valeurs possibles pour une case appartenant à la colonne col

    Entrées (pré-conditions) :
    - grille : tableau de 9 tableaux de 9 cases remplis d'entier [0-9]
    - col : index de colonne d'une case

    Sortie (post-conditions):
    - un tableau de toutes les valeurs possibles en tenant compte des contraintes
      en colonne
    """
    pass

def valeurs_possibles_bloc(grille, ligne, col) :
    """
    Affiche toutes les valeurs possibles pour une case appartenant à un bloc
    contenant la case (ligne, col)

    Entrées (pré-conditions) :
    - grille : tableau de 9 tableaux de 9 cases remplis d'entier [0-9]
    - ligne : index de ligne d'une case
    - col : index de colonne d'une case

    Sortie (post-conditions):
    - un tableau de toutes les valeurs possibles en tenant compte des contraintes
      en bloc
        """
    pass
```

Définition

S'il n'y a plus qu'une seule valeur possible dans une case du fait des contraintes en ligne, colonne et bloc, c'est un **singleton**.

(12) Dans la grille 0, montrer sur papier que la case (5,7) contient un singleton et donner sa valeur.

(13) Que peut-on faire si le tableau des valeurs possibles ne contient qu'une seule valeur ? Quelles conséquences cela a-t-il sur les valeurs possibles des autres cases dans la ligne ? dans la colonne ? dans le bloc ? C'est une question importante pour la compréhension du jeu.

(14) On applique le résultat de la question en appliquant le procédé suivant (pseudo-code) :

```
Pour chaque case de grille :
    Pour chaque zone parmi (ligne, colonne, bloc) :
        valeurs_possibles ← valeurs possibles de la zone
        s'il n'y a qu'une valeur dans valeurs_possibles :
            mettre cette valeur dans la case
            afficher un message (faire un copier-coller et adapter si besoin)

print(f"singleton de valeur {grille[ligne][col]} dans la case {ligne}x{col}")
```

(15) Une fois qu'on a parcouru une fois toute la grille et qu'une valeur a été écrite, que faut-il faire ? Est-ce qu'on arrête là ? Est-ce les conditions sur les ligne, colonne et bloc ont changé quelque part dans la grille ? C'est une question à laquelle vous avez déjà répondu à la question 13.

- si vous pensez qu'on doit continuer à appliquer le procédé ci-dessous : il faut savoir quand s'arrêter. Comment s'arrêter ?
- si vous pensez qu'on s'arrête là : il faut le justifier ! Vous pouvez par exemple tester sur plusieurs exemples : si vous pensez qu'on peut s'arrêter après un premier parcours de la grille, en faire un second ne devrait rien changer. Est-ce que vous observez cela ? Attention, le résultat à cette question dépend de votre grille initiale. Vous avez des grilles faciles en fin de document.

(16) Coder la fonction suivante :

```
def singletons(grille) :
    """
    Parcours le tableau grille à la recherche de singleton
    tant qu'il le faut et pas plus.

    Entrée (pré-condition) :
    - grille : tableau de 9 tableaux de 9 entiers [0-9]

    Sortie (post-condition) :
    Rien, le tableau grille est modifié (ou non) à l'intérieur de la fonction
    """
    pass
```

Tester son code

Une compétence fondamentale en informatique est d'apprendre à tester son code pour vérifier qu'il fait bien ce qu'il devrait. C'est une compétence difficile à acquérir car il faut apprendre à savoir comment tester dans des cas auxquels on n'avait pas penser. C'est là que le travail en équipe a une pertinence particulière quand tout le monde est pleinement impliqué à apporter son aide et ses réflexions !

```
.6.1.9.8.
.19...74.
2.......1
..79.26..
.3.....2.
..14.35..
1.......5
.75...23.
.9.5.6.7.
```

Ci-contre, le contenu du fichier `grille_0.txt` correspondant à la grille 0.

Pour utiliser facilement des grilles de sudoku, on va les stocker dans des fichiers et on va apprendre à lire des fichiers. Voici un exemple de grille où toutes les valeurs à deviner ont été remplacées par un point (on aurait pu choisir aussi de garder la convention du début et représenter la valeur à deviner par un 0, mais c'est moins lisible pour un être humain).

Quand on lit un fichier, il faut gérer les retours à la ligne qui sont codés par un caractère `'\n'` **que l'on ne voit pas** ! Il suffit en général de les éliminer via `ligne = ligne.strip('\n')`.

```
1  def get_sudoku_from_file(filename) :
2      """
3      lit la grille de sudoku contenue dans le fichier
4
5      Entrée (pré-condition) :
6      - filename : nom du fichier à lire (par exemple, grille_0.txt)
7
8      Sortie (post-conditions) :
9      - grille : tableau de 9 tableaux de 9 valeurs entières [0-9]
10     """
11     f = open(filename, 'r') # open ouvre le fichier
12     lignes = f.readlines() # on lit toutes les lignes d'un coup
13     f.close() # on ferme le fichier
```

```
14      grille = []
15      for ligne in lignes :
16          ligne = ligne.strip('\n')
17          # ... voir question ci-dessous
18      return grille
```

(17) Compléter la fonction `get_sudoku_from_file(filename)` par le pseudo-code :

```
Si la ligne ne contient pas 9 caractères,
        mettre un message d'erreur pertinent
        stopper la fonction avec return
t ← tableau vide
Pour chaque caractère de ligne
        si le caractère est un point
                ajouter 0 au tableau t
        sinon
                ajouter le caractère converti en entier au tableau t
ajouter le tableau t au tableau grille
```

(18) Il ne vous reste plus qu'à afficher la grille fournie par cette fonction et vérifiée qu'elle correspond bien à la grille 0 donnée dans l'énoncé. Pour cela, implémenter la fonction ci-dessous :

```
def affiche_grille(grille) :
    """
    affiche une grille de sudoku ligne par ligne

    Entrée (pré-condition) :
    - grille : tableau de tableau

    Sortie (post-condition) :
    Rien, un print() n'est pas un return!
    """
    for ligne in grille :
        print(ligne)
```

(19) Tester votre fonction `singletons(grille)`. Il faut vérifier que chaque singleton affiché par votre fonction existe. Mais il faut aussi vérifier qu'il n'existait pas de singleton oublié par la fonction. Donc, le plus simple, c'est de commencer sur papier. Trouver tous les singletons de la grille 0 à la main. Tester ensuite votre fonction.

(20) Question obligatoire pour les premiers ! Créer des grilles pour tester votre fonction dans les lignes, colonnes et bloc ! En faire un compte-rendu succinct mais représentatif de votre travail.

Singleton caché

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

On continue de travailler sur la grille 0 remise ci-contre pour éviter d'aller en début de fichier systématiquement.

Définition

Dans une zone donnée (ligne, colonne ou bloc), on considère les cases vides dont on a déterminé les valeurs possibles. Si une valeur apparaît une fois parmi toutes les valeurs possibles, c'est un **singleton caché**. Cette valeur va forcément dans sa case correspondante.

(21) On considère la grille ci-dessus à propos de la valeur `1`. Dans le bloc bleu contenant la case `(6,6)`, déterminer les valeurs possibles pour chaque case vide. Montrer alors qu'il n'y a qu'une seule case qui peut contenir la valeur `1` : c'est un exemple de singleton caché.

(22) Montrer qu'une fois placée la valeur `1` de la question précédente, il n'y a alors qu'une seule case du bloc bleu contenant la case `(3,6)` pouvant contenir la valeur `1`.

(23) Pour implémenter le singleton caché, on procède de la manière suivante (pseudo-code) :

```
possibles ← tableau vide
Pour chaque case vide de la zone considérée (ligne, colonne ou bloc) :
    trouver les valeurs possibles pour cette cases
    les mettre dans le tableau possibles
compteur ← tableau de 10 zéros
Pour chaque valeur du tableau possibles :
    incrémenter le tableau compteur à l'index valeur
singleton_cache ← tableau vide
Pour chaque valeur du tableau compteur égale à 1 :
    mettre dans le tableau singleton_cache l'index de la valeur
Pour chaque valeur du tableau singleton_cache :
    trouver la case correspondant à valeur
    mettre valeur dans cette case
```

Appliquer ce pseudo-code à la main au bloc en bas à droite de la grille 0. Vous avez déjà montré qu'il y avait un singleton caché à la question (21).

(24) A quelle(s) zone(s) peut s'appliquer le pseudo-code ci-dessus ?

(25) On veut détecter et placer dans une case les singletons cachés dans la grille de sudoku. On commence par travailler sur papier : décomposer ce problème en sous-problèmes plus simples. Prendre en photo ce papier et le rendre au professeur.

(26) Coder la fonction :

```
def singletons_caches(grille) :
    """
    détermine les singletons cachés de la grille

    Entrée (pré-confdition) :
    - grille : tableau de 9 tableaux de 9 entiers [0-9]

    Sortie (post-condition)
    Rien, on modifie en place le tableau grille
    """
    pass
```

Résoudre des sudokus

Défi

Peut-on résoudre des grilles de sudoku de niveau facile simplement en déterminant les singletons et les singletons cachés ?

(27) Tester sur les 6 grilles fournies en fin de document.

Page des indices

Indice question (8)

```
for ligne in range(9) :
    for col in range(9) :
        print(f"ligne={ligne} colonne={col} ({3*(ligne//3)},{3*(col//3)}) ")
    print()
```

6 grilles de niveau facile

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

Exemple 1

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

Example 2

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

Exemple 3

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

Example 4

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

Exemple 5

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

Example 6