Sudoku

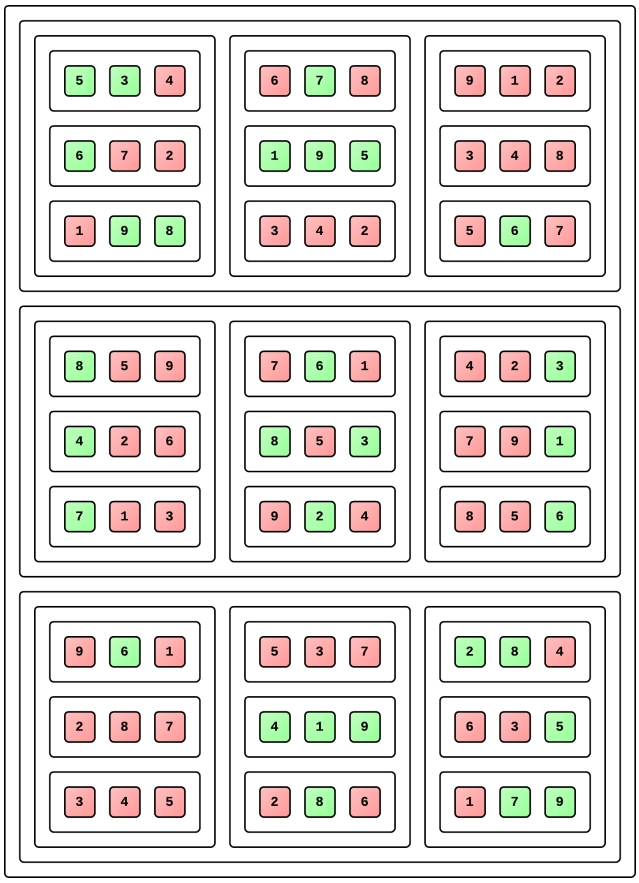
Introduction

Rappel : il est strictement INTERDIT d'utiliser l'opérateur @.

Le Sudoku

> *"Le sudoku (prononcé soudokou en français, est un jeu en forme de grille défini en 1979 par l'Américain Howard Garns.*
> *Le but du jeu est de remplir la grille avec une série de chiffres (ou de lettres ou de symboles) tous différents, qui ne se trouvent jamais plus d'une fois sur une même ligne, dans une même colonne ou dans une même sous-grille. La plupart du temps, les symboles sont des chiffres allant de 1 à 9, les sous-grilles étant alors des carrés de 3 × 3. Quelques symboles sont déjà disposés dans la grille, ce qui autorise une résolution progressive du problème complet."*
> Source : Wikipédia.

Pendant ce TP nous allons vous demander d'écrire un algorithme qui nous permettra de résoudre des grilles de sudoku simple (pas d'ambiguïté sur les valeurs pendant la résolution).
Afin de rendre les choses plus faciles (ou pas) le type de la grille vous est imposé !
On considérera dans un premier temps une grille de 9x9 contenant les valeurs de 1 à 9, on utilisera la valeur nulle (0) pour dire que la case est vide.



Référence :

```
# let grid_sample =
  [[[[5; 3; 0]; [6; 0; 0]; [0; 9; 8]];
    [[0; 7; 0]; [1; 9; 5]; [0; 0; 0]];
    [[0; 0; 0]; [0; 0; 0]; [0; 6; 0]]];
   [[[8; 0; 0]; [4; 0; 0]; [7; 0; 0]];
    [[0; 6; 0]; [8; 0; 3]; [0; 2; 0]];
    [[0; 0; 3]; [0; 0; 1]; [0; 0; 6]]];
   [[[0; 6; 0]; [0; 0; 0]; [0; 0; 0]];
    [[0; 0; 0]; [4; 1; 9]; [0; 8; 0]];
    [[2; 8; 0]; [0; 0; 5]; [0; 7; 9]]]];;
 val grid_sample : int list list list list = [...]
```

Level 0 : Les Must-do

0.1 Longueur

Écrire la fonction *length* qui retourne la longueur d'une liste.

```
val length : 'a list -> int = <fun>
```

0.2 Aplatir

Écrire la fonction *flatten* qui à partir d'une liste de liste, retourne une liste.

```
val flatten : 'a list list -> 'a list = <fun>
```

```
# flatten [[1;2;3];[4;5]];;
 - : int list = [1; 2; 3; 4; 5]
```

0.3 Présence

Écrire la fonction *check* qui nous indique si un élément est présent dans la liste.

```
val check : 'a -> 'a list -> bool = <fun>
```

0.4 Suppression

Écrire la fonction *remove* qui retire un élément dans la liste.

```
val remove : 'a -> 'a list -> 'a list = <fun>
```

0.5 Unicité

Écrire la fonction *list_uniq* qui va retirer les doublons de tous les élements et va retourner la liste filtrée. L'ordre des éléments n'a pas d'importance.

```
# list_uniq [1;2;3;5;6;8;0;0;0;2;0;3;0;5;6;0;1;4];;
 - : int list = [8; 2; 3; 5; 6; 0; 1; 4]
```

0.6 Unicité bis

Écrire la fonction *list_match* qui va retourner la liste des éléments (une seule occurence de chaque élément) présent dans les deux listes passées en paramètres. L'ordre des éléments n'a pas d'importance.

```
# list_match [1;2;3;5;6;8;0;0;0] [2;0;3;0;5;6;0;1;4];;
 - : int list = [8; 2; 3; 5; 6; 0; 1; 4]
```

1 Level 1 : Les Matrices ?

1.1 Rectangle

Écrire la fonction *grid_make_rectangle* qui retourne une liste contenant y listes contenant x valeurs nulles (0).

```
val grid_make_rectangle : int -> int -> int list list = <fun>
```

1.2 Carré

Écrire la fonction *grid_make_square* qui retourne une liste contenant x listes contenant x valeurs nulles (0).

```
val grid_make_square : int -> int list list = <fun>
```

1.3 Grille

Écrire la fonction *grid_make* qui retourne une liste de liste de liste de liste de valeurs nulles (0) conformement à la représentation imposée. Pour info pour une grille de $9x9$, il faut faire une grille de 3x3 (3 listes de 3 listes) dont les éléments sont des sous-grilles de 3x3 valeurs nulles (0).

```
val grid_make : int -> int list list list list = <fun>
```

```
# grid_make 9;;
 - : int -> int list list list list =
  [[[[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]; [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]];
    [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]];
   [[[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]; [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]];
    [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]];
   [[[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]; [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]];
    [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]]]
```

2 Level 2 : Les Extractions

2.1 Grille

Écrire la fonction *extract_square* qui retourne la sous-grille numéro n. Les sous grilles sont numérotées de 1 à 9, de haut en bas et de gauche à droite. La valeur de retour de la fonction doit être une liste de valeurs simples.

```
# extract_square grid_sample 9 values;;
 - : int list = [2; 8; 0; 0; 0; 5; 0; 7; 9]
```

2.2 Ligne

Écrire la fonction *extract_row* qui retourne la ligne numéro n. Les lignes sont numérotées de 1 à 9, de haut en bas. La valeur de retour de la fonction doit être une liste de valeurs simples.

```
# extract_row grid_sample 5 values;;
 - : int list = [4; 0; 0; 8; 0; 3; 0; 0; 1]
```

2.3 Colonne

Écrire la fonction *extract_column* qui retourne la colonne numéro n. Les colonnes sont numérotées de 1 à 9, de gauche à droite. La valeur de retour de la fonction doit être une liste de valeurs simples.

```
# extract_column grid_sample 6 values;;
 - : int list = [0; 5; 0; 0; 3; 0; 0; 9; 0]
```

2.4 Affichage

Écrire la fonction *grid_print* qui affiche une grille sur la sortie standard.

```
# grid_print grid_sample;;
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
 - : unit = ()
```

3 Level 3 : Les vérifications

3.1 Présence et Unicité

Écrire la fonction *list_validate* qui permet de vérifier que les elements d'une liste sont présent dans une autre liste, cela une seule et unique fois. Attention au cas particulier de la valeur nulle (0).

```
val list_validate : int list -> int list -> bool = <fun>

# list_validate [1;2;3;4;5;6;7;8;9] [1;2;3;4;5;6;7;8;9];;
 - : bool = true

# list_validate [0;1;3;4;0;0;0;1;9] [1;2;3;4;5;6;7;8;9];;
 - : bool = false

# list_validate [1;3;5;0;4;6;0;0;2] [1;2;3;4;5;6;7;8;9];;
 - : bool = true
```

3.2 Grille valide

Écrire la fonction *grid_validate* qui permet de vérifier qu'une grille respecte bien les règles du sudoku. C'est à dire pas deux éléments de même valeur sur une ligne ou colonne ou sous-grille. Attention toujours aux valeurs nulles (0).

```
val grid_validate : int list list list list -> bool = <fun>
```

3.3 Grille pleine

Écrire la fonction *grid_isfull* qui permet de vérifier qu'une grille est pleine ou pas. C'est à dire qu'elle ne contient plus de valeurs nulles (0).

```
val grid_isfull : int list list list list -> bool = <fun>
```

4 Level 4 : La résolution

4.1 Élement manquant

Écrire la fonction *find_missing* qui retourne la liste des élements d'une liste non présents dans la première.

```
# find_missing [8;2;3;5;6;0;1;4] [1;2;3;4;5;6;7;8;9];;
 - int list = [7; 9]
```

4.2 Les choix

Écrire la fonction *grid_find* qui liste les possibilités de valeurs pour une case de notre grille de sudoku.

```
val grid_find : int list list list list -> int -> int -> int list = <fun>

# grid_find grid_sample 9 1;;
  - : int list = [2; 4; 8]

# grid_find grid_sample 1 9;;
  - : int list = [1; 2; 3]

# grid_find grid_sample 1 7;;
  - : int list = [1; 3; 9]
```

4.3 La solution au rang suivant

Écrire la fonction *grid_nsolve* qui donne la grille suivante dans le processus de résolution :
On se propose de parcourir toute la grille et pour chaque case :

- Si la case contient une valeur non nulle, on met cette valeur dans la nouvelle grille.
- Si la case contient une valeur nulle, on va regarder la liste des solutions possibles pour cette case
  - Si la liste des solutions contient plus d'une possibilité, on ignore la case et on passe a la suivante.
  - Si la liste des solutions ne contient qu'une seule valeur, on met cette valeur dans la nouvelle grille

Une fois l'ensemble des cases vues, on retourne la nouvelle grille.

```
# grid_print (grid_nsolve grid_sample);;
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 5 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 7 2 8 4
0 0 0 4 1 9 0 3 5
0 0 0 0 8 0 0 7 9
 - : unit = ()
```

4.4 Résolution complète

Écrire la fonction *solve* qui donne la solution pour une grille de sudoku.

```
# grid_print (solve grid_sample)
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
 - : unit = ()
```

5 Level 5 : Les bonus - obligatoire ou presque :)

5.1 Et si on changeait le type

On recommence la même chose mais en changeant les valeurs utilisables. Par exemples au lieu d'utiliser les chiffres de 1 à 9, on se propose d'utiliser des lettres A à I.

5.2 Grilles de tailles différentes

On recommence la même chose mais avec des grilles de tailles différentes. Ceci implique plus de valeurs possible. On ne considèrera que les ensembles de valeurs dont la racine carrée de la taille est entière (4, 9, 16, 25, ...).

6 Level 6 : Les bonus

6.1 La vraie résolution complète

Dans la partie obligatoire, on ne propose de résoudre que des grilles simple, ou il n'y a jamais d'ambiguïté sur les valeurs à choisir. Cepdendant ce cas n'est pas représentatif de l'ensemble des grilles de sudoku. On se propose de gérer dans cette partie la résolution des grilles avec ambiguïté sur le choix des valeurs. Attention certains choix vous emmènerons donc sur des grilles non valides.

6.2 La génération de grille

Maintenant que nous sommes capables de résoudre toutes les grilles, nous allons donc passer à la création de celles-ci. Il nous suffit de prendre une grille complète et valide, et de remplacer certaines valeurs par la valeur nulle. La complexité de la grille dépendra des positions et quantités de valeurs que vous allez retirer.

7 Level 7 : Bonus : j'ai vraiment rien d'autre a faire

Puisque apparement le reste était tellement trivial (et je vous comprend) que vous vous ennuyer sur le reste du tp, je vous invite à aller sur wikipedia et d'implementer les autres types de grille de sudoku :

- Nonomino
- Killer Sudoku
- Hyper Sudoku
- ...