Environnement de travail

- Tutoriels :
  - Unix :
    - Initiation Unix
    - Environnement Unix
  - Python :
    - Python
    - Python et son environnement
  - Java :
    - Java et son environnement
  - Coder mieux :
    - Les tests unitaires
- Environnements de programmation:
  - Installer Python et Pycharm chez soi
  - Utiliser Pycharm

A savoir

- Si tu n'est pas à l'aise en Python: Python
- Installer et configurer PyCharm: PyCharm

On supposera ici que vous avez lu (et compris) les bases de Python et que vous pouvez lancer et utiliser un ide comme pyCharm.

Donc si tu n'est pas à l'aise en Python: Python

TP

Pycharm (l'ide) est installé par défaut dans les salles linux de l'ÉCM. Pour le lancer, ouvrez un terminal (cliquez sur la télé) et tapez : `pycharm` puis entrée. L'éditeur doit se lancer. Lors de la première utilisation il vous demandera plein de trucs, essayer de répondre intelligemment (vous ne pourrez de toute façon pas faire de raccourcis sur le bureau ou dans `/usr/local/bin`, lorsqu'il vous le demandera cliquez annuler).

Le but de ce TP est de vous dérouiller les méninges en algorithmie en essayant de résoudre le problème du sudoku. Nous nous contenterons de résoudre les sudokus très simples, libre à vous d'améliorer le procédé.

Lire un sudoku

Pour nous, un sudoku sera une liste de 9*9 nombres allant de 1 à 9 constituée de la concaténation de chaque ligne :

```
grille_1_complete = [5, 2, 9, 6, 7, 1, 8, 4, 3,
                     3, 1, 8, 4, 9, 2, 6, 5, 7,
                     6, 7, 4, 8, 5, 3, 2, 1, 9,
                     7, 4, 6, 3, 2, 5, 1, 9, 8,
                     2, 9, 3, 1, 8, 6, 4, 7, 5,
                     1, 8, 5, 7, 4, 9, 3, 6, 2,
                     4, 3, 2, 5, 6, 7, 9, 8, 1,
                     9, 6, 7, 2, 1, 8, 5, 3, 4,
                     8, 5, 1, 9, 3, 4, 7, 2, 6]
```

Un élément particulier

On vous demande de commencer par écrire une fonction (`element_sudoku`) permettant d'accéder un élément particulier du sudoku. On supposera en bon informaticien que les lignes et colonnes commencent à 0. La commande ci-après doit ainsi afficher : **`4, 5: 6`**

```
print("4, 5:", element_sudoku(4, 5, grille_1_complete))
```

Accéder aux lignes, colonnes et sous-matrices

Ecrivez 3 fonctions permettant de rendre une ligne, une colonne ou une sous-matrice particulière sous forme de liste. Les commandes ci-après :

```
print("ligne 4", ligne_sudoku(4, grille_1_complete))
print("colonne 5", colonne_sudoku(5, grille_1_complete))
print("sous-matrice contenant 4, 5", matrice_sudoku(4, 5,
grille_1_complete))
```

Afficheront :

```
ligne 4 [2, 9, 3, 1, 8, 6, 4, 7, 5]
colonne 5 [1, 2, 3, 5, 6, 9, 7, 8, 4]
sous-matrice contenant 4, 5 [3, 2, 5, 1, 8, 6, 7, 4, 9]
```

Sudoku correct ?

Utilisez les fonctions d'accès aux lignes, colonnes et sous-matrices que vous avez crées dans la partie précédente pour écrire une fonction permettant de savoir si un sudoku est correct ou pas.

Faites des tests avec des sudokus correct et des sudoku incorrect. Pour ma part :

```
print("sudoku correct ?", sudoku_correct(grille_1_complete))
```

répond :

```
sudoku correct ? True
```

Afficher un sudoku

Finalement, affichons le sudoku complet. Laissez libre court à votre imagination. Pour ma part, la commande :

```
affiche_sudoku(grille_1_complete)
```

affiche :

```
529 671 843
318 492 657
674 853 219

746 325 198
293 186 475
185 749 362

432 567 981
967 218 534
851 934 726
```

Sudoku Incomplet

On remplacera par des 0 les nombres inconnus. La variable ci-après est par exemple un sudoku incomplet de la grille précédente :

```
grille_1 = [0, 0, 0, 0, 0, 0, 0, 4, 0,
            0, 0, 8, 4, 0, 2, 0, 5, 7,
            0, 0, 4, 8, 0, 3, 2, 1, 0,
            7, 4, 0, 0, 0, 5, 0, 9, 0,
            2, 0, 3, 1, 0, 6, 4, 0, 5,
            0, 8, 0, 7, 0, 0, 0, 6, 2,
            0, 3, 2, 5, 0, 7, 9, 0, 0,
            9, 6, 0, 2, 0, 8, 5, 0, 0,
            0, 5, 0, 0, 0, 0, 0, 0, 0]
```

Ecrivez 3 méthodes permettant de rendre pour une ligne, une colonne ou une sous-matrice particulière un ensemble de valeurs manquantes.

Ainsi, les commandes :

```
print("valeurs manquantes ligne 5", valeurs_manquantes_ligne(5, grille_1))
print("valeurs manquantes colonne 5", valeurs_manquantes_colonne(5,
grille_1))
print("valeurs manquantes sous-matrice issue de 5, 5",
valeurs_manquantes_matrice(5, 5, grille_1))
```

Affichent chez moi :

```
valeurs manquantes ligne 5 {1, 3, 4, 5, 9}
valeurs manquantes colonne 5 {1, 9, 4}
valeurs manquantes sous-matrice issue de 5, 5 {8, 9, 2, 3, 4}
```

Valeurs manquantes

En déduire une fonction permettant de donner les valeurs possibles pour une case donnée (en ne prenant en compte que les valeurs manquantes). Par exemple :

```
print("possible en 5, 5:", possible_sudoku(5, 5, grille_1))
```

Affiche :

```
possible en 5, 5: {9, 4}
```

Complète ce qui est unique

Il n'y a qu'une seule possibilité pour la case (3, 3), Laquelle ? Ecrivez une fonction qui remplit toutes les cases à une unique possibilité. Cette fonction doit rendre un sudoku que l'on espère complet, ou au pire qui ne contient que des valeurs incomplètes ayant plus d'un choix.

Chez moi :

```
grille_1_complet_1_possible = rempli_sudoku_possible(grille_1)
affiche_sudoku(grille_1_complet_1_possible)

print("valeurs manquantes (3, 2)", possible_sudoku(3, 2,
grille_1_complet_1_possible))
print("valeurs manquantes (3, 6)", possible_sudoku(3, 6,
grille_1_complet_1_possible))
print("valeurs manquantes (3, 8)", possible_sudoku(3, 8,
grille_1_complet_1_possible))
```

Affiche :

```
020 000 040
018 402 057
074 803 210

740 325 090
```

```
293 186 475
080 700 062

032 507 980
960 208 530
050 000 020

valeurs manquantes (3, 2) {1, 6}
valeurs manquantes (3, 6) {8, 1}
valeurs manquantes (3, 8) {8, 1}
```

Ce n'est donc pas fini, mais on a un peu avancé.

Sudoku Incomplet v2

Améliorez votre remplisseur de sudoku. Je n'ai utilisé qu'une règle supplémentaire pour le résoudre complètement. En regardant la ligne 3 par exemple on voit qu'il y a 3 "trous", mais que 6 n'est disponible que pour la case (3, 2). C'est donc un nouveau type d'élément unique, et l'on est assuré que la case (3, 2) contient le nombre 6.

En ne rajoutant que ce test sur les lignes, le code suivant :

```
soluce = rempli_sudoku_possible_v2(grille_1)
affiche_sudoku(soluce)
print(soluce == grille_1_complete)
```

Affiche :

```
529 671 843
318 492 657
674 853 219

746 325 198
293 186 475
185 749 362

432 567 981
967 218 534
851 934 726

True
```

Résolution complète

Le code écrit précédemment ne permet que de résoudre des sudoku très simples. Pour résoudre tout les sudokus possibles, il faut mettre en place des techniques de backtracking.