Le but de ce devoir maison est la réalisation d'un programme qui résout un sudoku.

I Mettre en place son environnement : travail n°1

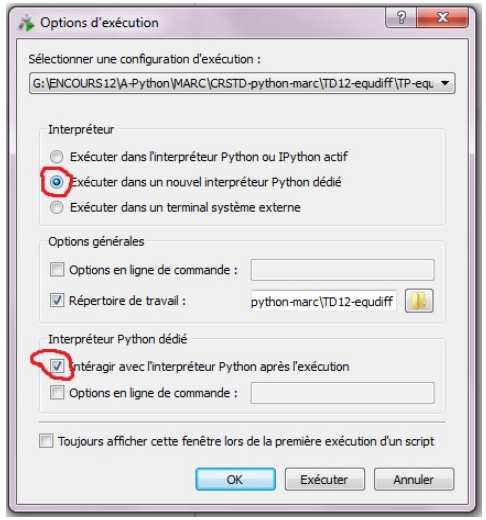
Le dossier `DM-sudoku` que vous venez de décompresser contient trois scripts Python.

`DM_sudoku.py`	le fichier principal quand tout sera terminé.
`sudoku_gui.py`	la boîte noire « **G**raphical **U**ser **I**nterface » (= interface graphique) en `tkinter`.
`sudoku_resolution.py`	le fichier de résolution du sudoku, c'est là que se concentreront vos efforts.

Plus précisément, vous ne modifierez que le fichier `sudoku_resolution.py`, et éventuellement la grille STEST du fichier `DM_sudoku.py` une fois le devoir fini... La procédure de rendu est la même que pour le DM précédent.

ASSUREZ vous de bien voir le bouton de rendu, pour cela il faut être connecté au site. Aucune excuse de perte de code ne sera acceptée au moment du rendu. Prenez vos précautions.

Tester le programme principal `DM_sudoku.py`, pour cela paramétrer `spyder` en cliquant sur  puis choisir



Un nouvel interpréteur python est lancé à chaque fois que l'on exécute le programme (touche `F5`) et l'on peut utiliser cet interpréteur à la fin du programme, cela est quasi-indispensable quand on utilise beaucoup `matplotlib` ou `tkinter`.

II Vérifier que le sudoku n'a pas de doublons

Pour l'instant, nous n'utiliserons que le fichier `sudoku_resolution.py`
Notre sudoku sera codé en python par une liste de listes qui représente un tableau 9×9.
Par exemple :

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

On accède à un élément par `s[i][j]` (`i`^{e} ligne et `j`^{e} colonne avec $(i, j) \in [\![0, 8]\!]$).
Chaque case du sudoku prend une valeur entre 0 et 9, 0 codant une case vide.
Avec la liste précédente, on obtient le sudoku incomplet suivant

Le fichier `sudoku_resolution.py` contient les fonctions de vérification et de résolution du sudoku.
Vous pouvez exécutez ce fichier, le programme principal (test de notre module), situé après `if __name__ == '__main__':` est alors interprété.
Lorsque nous voudrons utiliser ces fonctions dans un autre programme nous l'appellerons par un `import sudoku_resolution` et la partie de test ne sera pas exécutée.

III Travail n°2

Dans un premier temps, définissez les fonctions `test_ligne(s, i)` et `test_colonne(s, i)` qui prennent en entrée un sudoku `s` codé sous forme de matrice et un nombre `i` ou `j` et teste s'il y a un doublon à la ligne `i` ou la colonne `j`.
INDICATION
Vous pouvez par exemple définir une ligne de chiffres (`1`, non encore coché ou `0`, coché)

```
L = [1 for i in range(9)]
```

puis la fonction coche les chiffres utilisés en remplaçant `1` par `0` sur l'indice concerné à moins que le chiffre soit déjà coché ce qui est le signe d'un doublon.
Testez vos fonctions sur des exemples. En cas de doublons, affichez la ligne ou la colonne concernée. Vous avez des grilles déjà préparées dans les scripts. Faites en des copies puis modifiez les ! Vous devez obtenir des resultats positifs et négatifs pour chaque fonction pour être sûr...
Une idée peut être de créer une feuille séparée qui contient les différentes grilles de test.

IV Travail n°3

Ça fonctionne ?
- Oui : très bien, passez à la suite.
- Non : retour au travail précédent.

Maintenant définissez la fonction `test_carre(s, i, j)` qui teste les doublons pour le carré qui contient la case `(i, j)` du sudoku.
INDICATION `//` désigne la division entière (quotient dans la division euclidienne, pour les matheux du fond). Une premère étape indispensable est de pouvoir trouver et parcourir toutes les cases du carré qui contient la case d'indice `(i, j)`. Prenez un crayon, essayez...
Testez vos fonctions sur des exemples. En cas de doublons, affichez le carré concerné.

V Travail n°5

Définissez la fonction `fiable(s)` qui prend le sudoku `s` en entrée et renvoie `True` s'il n'y a pas de doublon et **False** sinon.
INDICATION il suffit d'utiliser les trois fonctions précédentes.

Travail n°6

Vous avez bien travaillé. Reposez-vous et amusez-vous en testant le programme principal `DM_sudoku.py` avec le bouton fiabilité en changeant des cases du sudoku, en rajoutant des doublons...
Si tout ne fonctionne pas correctement n'hésitez pas à poser des questions sur le site.

VI Travail n°7 – les choses sérieuses commencent

Nous allons reprendre les trois fonctions que vous avez construites au départ et les modifier légèrement.
• Définissez les fonctions `chiffre_barre_ligne(s, L, i)`, `chiffre_barre_colonne(s, L, j)` et `chiffre_barre_carre(s, L, i, j)` qui prend en plus en entrée une liste `L` de 9 entiers et modifient la liste `L`[1] toujours selon le schéma

`L[i]=0`⇔ le chiffre `i` est utilisé et `L[i]=1`⇔ le chiffre `i` est libre.

La modification consiste à effacer des 1, pas à en ajouter. En clair : on a déjà une liste de chiffres possibles, il faut continuer à éliminer des possibilités.
• Définissez alors la fonction `nombre_possible(s, i, j)` qui renvoie une liste représentant les chiffres encore disponibles dans le sudoku lorsqu'on examine la ligne `i`, la colonne `j`, le carré correspondant à la case `(i, j)` (bien sûr cela concerne une case `(i, j)` vide).

1. pas de problème pour modifier `L` qui est de type `list` donc `mutable`, `L` est une étiquette sur le tableau de valeurs.

Remarque importante Cette liste nous permettra de donner un score à la case vide `(i, j)`. En posant `score = sum(L)`, la variable `score` nous donne le nombre des chiffres disponibles par rapport à la case vide `(i, j)`.

VII Travail n°8 – on commence par quelle case ?

Nous allons chercher une case vide qui a un score minimal par rapport à toutes les cases vides disponibles (bien sûr pour une case vide ce score est $\geqslant 1$).
Compléter la fonction `score` puis écrire une fonction `meilleure_case(s)` qui renvoie les coordonnées `i` et `j` et la liste des chiffres disponibles ou non pour une case vide `(i, j)` de score minimal parmi toutes les cases vides.
S'il n'y a pas de case vide, la fonction renverra `-1, -1, []` par convention.

Travail n°9 – la résolution, enfin...

Tentez d'analyser la fonction **resolution**. Remarque : c'est une fonction *récursive* qui s'appelle elle-même pour tenter de résoudre la force brute notre sudoku. Le but est d'essayer toutes les possibilités.

VIII Travail n°10

La récompense ultime, on joue avec le programme principal (lancé avec F5) `DM_sudoku.py` et on épate ses parents ou ses frères et sœurs.