



SUDOKU

NAÏS MAXIME AHMED



Sommaire

- L'idée du Sudoku
- Intérêt du projet Sudoku
- Comment on s'est organisés
- Notion
- Git : network
- Présentation des 3 checkers
- Démarche Solver

Principes du Sudoku





Principes du SUDOKU

- Une valeur unique par ligne
- Une valeur unique par colonne
- Une valeur unique par sous-grille de 3x3

Principes du SUDOKU

SUDOKU

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

ANSWER:

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

Intérêt du projet





Intérêt du projet

- Abstraction
- Algorithmie
- Travail en groupe sur Git Hub
- S'organiser dans la durée en groupe avec des objectifs
- Réutiliser les fonctions, les boucles, les bases de programmation en Python

Organisation

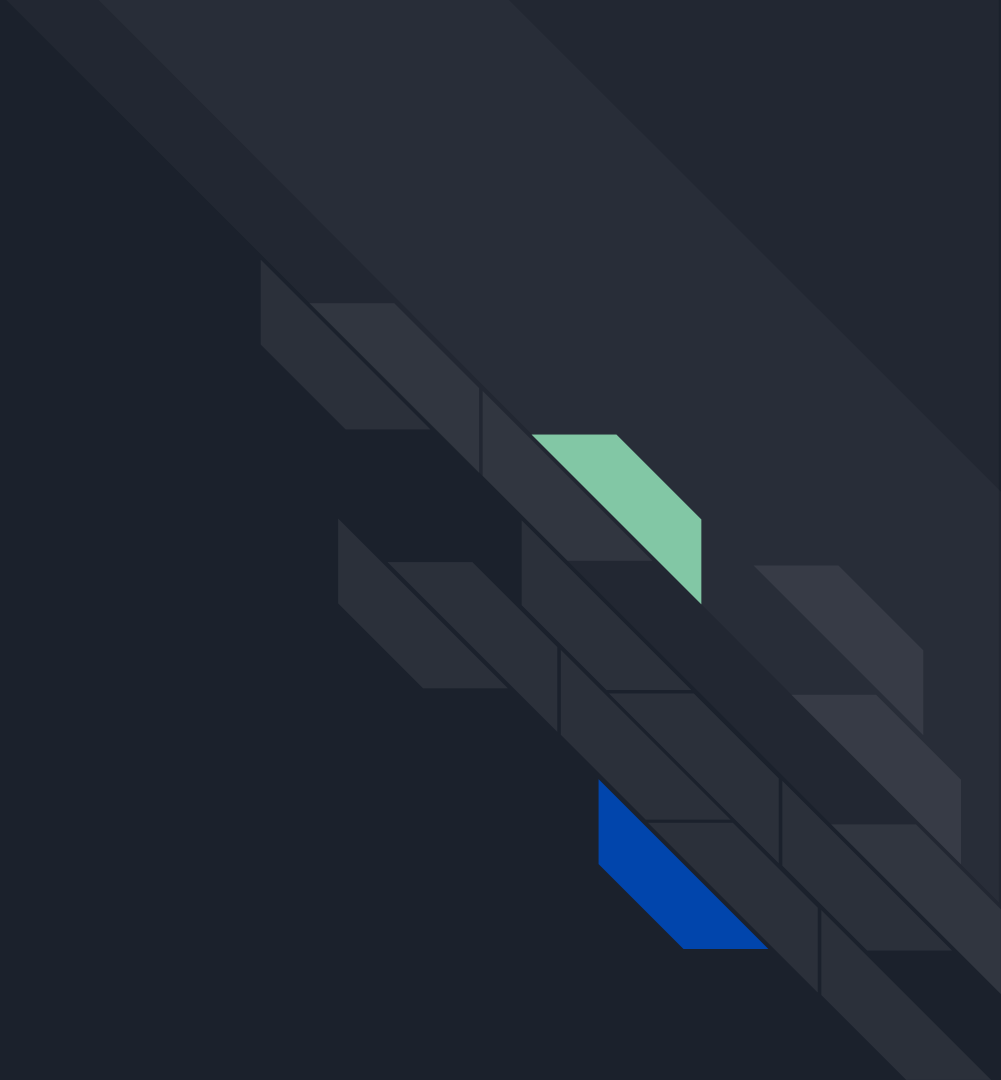




Organisation

- Conceptualisation de la problématique
- Expérimentation de différentes méthodes
- Confrontation des résultats
- Discussion sur les axes d'améliorations
- Résultat final

Notion



Notion

To Do 0

+ Nouvelle page

Doing 3

<https://www.pythoniste.fr/python/undes-fonctions-recursives-en-python/>

Veille algorithme récursivité 19/11 + 23/11

Doing Naïs

Interface : Tkinter ? Web ? Kivy

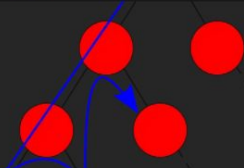
Doing

To drop duplicated by column 23/11

Doing Naïs

+ Nouvelle page

Done 7



Veille Backtracking - 23/11

Maxime Done 🙌

Création du repository GitHub - 23/11

Ahmed Done 🙌

premiers essais sur jupyter-notebook

Done 🙌

Sudoku checker

Ahmed Done 🙌

Sudoku checker

Maxime Done 🙌

Sudoku checker Naïs 23/11

Naïs Done 🙌

To find 0 values 23/11

Naïs Done 🙌

Groupes masqués

Naïs 4

Maxime 2

Ahmed 2

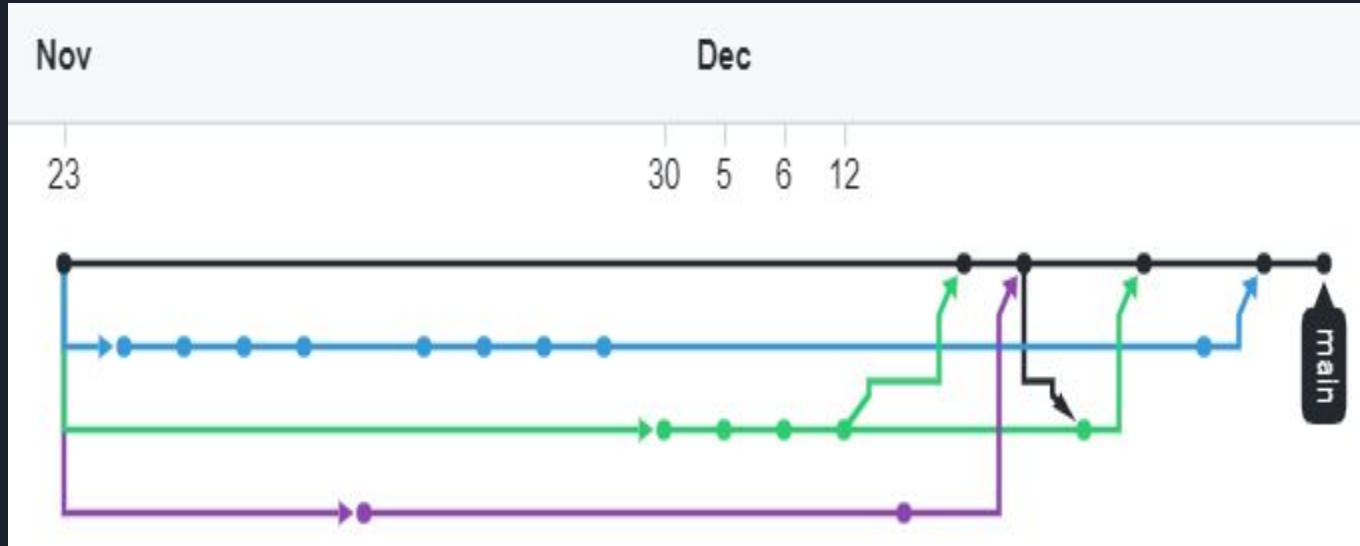
Moyen 0

Sans « Status » 4

Git : Network



Git : Network



Les checkers



Checker d'Ahmed


```
def verif (grid) :  
  
    for i in range (9) :  
        x = grid[i,:]   
        y = grid[:,i]   
        X = (i//3)*3   
        Y = (i//3)*3   
        if len(set(x)) != 9 or len(set(y)) != 9 or len(set(grid[X:X+3, Y:Y+3].ravel())) != 9 :  
            return False   
    return True
```

```
verif(grille_v)
```



Checker de Maxime

```
def checker_max(sudoku):  
  
    grid = np.asarray(sudoku)  
  
    def row(grid):  
        row = grid[i,:]  
        return len(set(row)) == len(row)  
  
    def column(grid):  
        column = grid[:,i]  
        return len(set(column)) == len(column)
```

```
def case(grid):
    #le premier reshape capture les lignes dans des cases 3x3
    #le swapaxes change l'axe du reshape et prend alors les 3 premiere ligne et colonne
    #Le 2eme remet l'array en forme
    case = grid.reshape(3, 3,-1, 3).swapaxes(1,2).reshape(-1, 3, 3)
    for i in range (3):
        if not row(grid):
            return False
        if not column(grid):
            return False
        else:
            return True

for i in range (9):
    if not row(grid):
        if not column(grid):
            if not case(grid):
                return False
    else:
        return True
```



Checker de Naïs

```
# Creation of DataFrames
a=np.random.randint(1,10, size=(3,3))
a_df = pd.DataFrame(a)
b=np.random.randint(1,10, size=(3,3))
b_df = pd.DataFrame(b)
c=np.random.randint(1,10, size=(3,3))
c_df = pd.DataFrame(c)
d=np.random.randint(1,10, size=(3,3))
d_df = pd.DataFrame(d)
e=np.random.randint(1,10, size=(3,3))
e_df = pd.DataFrame(e)
f=np.random.randint(1,10, size=(3,3))
f_df = pd.DataFrame(f)
g=np.random.randint(1,10, size=(3,3))
g_df = pd.DataFrame(g)
h=np.random.randint(1,10, size=(3,3))
h_df = pd.DataFrame(h)
i=np.random.randint(1,10, size=(3,3))
i_df = pd.DataFrame(i)
```

```
concat_a_b = pd.concat([a_df,b_df],axis=1,ignore_index = True)
concat_a_b_c = pd.concat([concat_a_b,c_df],axis=1,ignore_index=True)
concat_a_b_c_d = pd.concat([d_df,e_df,f_df],axis=1,ignore_index=True)
concat_a_b_c_d=concat_a_b_c_d.reset_index(drop=True)
concat_a_b_c_d_1 = pd.concat([concat_a_b_c,concat_a_b_c_d],axis=0,ignore_index=True)
concat_g_h_i = pd.concat([g_df,h_df,i_df],axis=1,ignore_index=True)

concat_a_b_c_d = concat_a_b_c_d.loc[~concat_a_b_c_d.index.duplicated(keep='first')]
concat_g_h_i = concat_g_h_i.loc[~concat_g_h_i.index.duplicated(keep='first')]

concat_final = pd.concat([concat_a_b_c_d_1,concat_g_h_i],axis=0,ignore_index=True)
print(concat_final)

df_compact = [a_df,b_df,c_df,d_df,e_df,f_df,g_df,h_df,i_df]
```

```

def check_column(concat_final) :
    for i in range(9):
        return concat_final[i].is_unique

def check_row(concat_final) :
    for j in range(9):
        return concat_final.loc[j].is_unique

def arr_tolist(arr):
    arr_list = arr.tolist()
    return arr_list

def set_list(arr_list):
    arr_set = set(arr_list)
    return arr_set

def test_arr(arr_set):
    return len(arr_set) == 9

arr = []
arr = np.array(arr)

```

```

def check_df():
    mi = 0
    for k in df_compact :
        arr = pd.unique(k.loc[0])
        arr = np.append(arr,pd.unique(k.loc[1]))
        arr = np.append(arr,pd.unique(k.loc[2]))
        pouet = test_arr(set_list(arr_tolist(arr)))
        if pouet == True :
            mi = mi + 1
        else : continue
    print(mi)
    return mi == 9

check_df()

def sudoku_checker(concat_final) :
    if check_column(concat_final) :
        if check_row(concat_final):
            return check_df(df_compact) == True
        else : return False

sudoku_checker(concat_final)

```

Démarche du Solver





Démarche du solver

- Parcourir la grille par deux boucles for imbriquées
- S'arrêter quand la case est égale à 0
- Prendre un nombre entre 1 et 9 (boucle for)
- Vérifier si dans ligne, colonne et 3x3 le chiffre n'existe pas sinon passer au chiffre suivant de la boucle for
- Une fois que c'est valide continuer sur la grille en renvoyant la fonction
- Lorsqu'il se bloque est qu'il retourne False (pas de solution), recommencer (probablement avec un autre return)

Merci !

