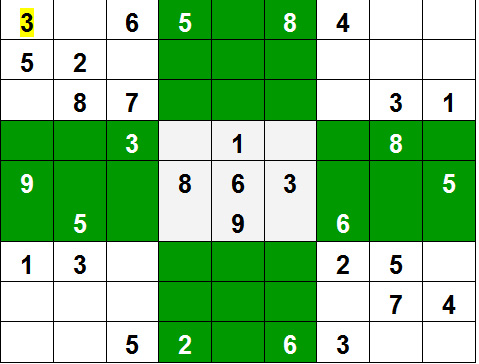
**Sudoku**

Given a partially filled 9×9 2D array ‘grid[9][9]’, the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.



**Approach:** The naive approach is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found, i.e. for every unassigned position fill the position with a number from 1 to 9. After filling all the unassigned position check if the matrix is safe or not. If safe print else recurs for other cases.  
**Algorithm:**

1. Create a function that checks if the given matrix is valid sudoku or not. Keep Hashmap for the row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true;
2. Create a recursive function that takes a grid and the current row and column index.
3. Check some base cases. If the index is at the end of the matrix, i.e. i=N-1 and j=N then check if the grid is safe or not, if safe print the grid and return true else return false. The other base case is when the value of column is N, i.e j = N, then move to next row, i.e. i++ and j = 0.
4. if the current index is not assigned then fill the element from 1 to 9 and recur for all 9 cases with the index of next element, i.e. i, j+1. if the recursive call returns true then break the loop and return true.
5. if the current index is assigned then call the recursive function with index of next element, i.e. i, j+1

Code –

# N is the size of the 2D matrix N\*N

N = 9

# A utility function to print grid

def printing(arr):

for i in range(N):

for j in range(N):

print(arr[i][j], end = " ")

print()

# Checks whether it will be

# legal to assign num to the

# given row, col

def isSafe(grid, row, col, num):

# Check if we find the same num

# in the similar row , we

# return false

for x in range(9):

if grid[row][x] == num:

return False

# Check if we find the same num in

# the similar column , we

# return false

for x in range(9):

if grid[x][col] == num:

return False

# Check if we find the same num in

# the particular 3\*3 matrix,

# we return false

startRow = row - row % 3

startCol = col - col % 3

for i in range(3):

for j in range(3):

if grid[i + startRow][j + startCol] == num:

return False

return True

# Takes a partially filled-in grid and attempts

# to assign values to all unassigned locations in

# such a way to meet the requirements for

# Sudoku solution (non-duplication across rows,

# columns, and boxes) \*/

def solveSuduko(grid, row, col):

# Check if we have reached the 8th

# row and 9th column (0

# indexed matrix) , we are

# returning true to avoid

# further backtracking

if (row == N - 1 and col == N):

return True

# Check if column value becomes 9 ,

# we move to next row and

# column start from 0

if col == N:

row += 1

col = 0

# Check if the current position of

# the grid already contains

# value >0, we iterate for next column

if grid[row][col] > 0:

return solveSuduko(grid, row, col + 1)

for num in range(1, N + 1, 1):

# Check if it is safe to place

# the num (1-9) in the

# given row ,col ->we

# move to next column

if isSafe(grid, row, col, num):

# Assigning the num in

# the current (row,col)

# position of the grid

# and assuming our assined

# num in the position

# is correct

grid[row][col] = num

# Checking for next possibility with next

# column

if solveSuduko(grid, row, col + 1):

return True

# Removing the assigned num ,

# since our assumption

# was wrong , and we go for

# next assumption with

# diff num value

grid[row][col] = 0

return False

# Driver Code

# 0 means unassigned cells

grid = [[3, 0, 6, 5, 0, 8, 4, 0, 0],

[5, 2, 0, 0, 0, 0, 0, 0, 0],

[0, 8, 7, 0, 0, 0, 0, 3, 1],

[0, 0, 3, 0, 1, 0, 0, 8, 0],

[9, 0, 0, 8, 6, 3, 0, 0, 5],

[0, 5, 0, 0, 9, 0, 6, 0, 0],

[1, 3, 0, 0, 0, 0, 2, 5, 0],

[0, 0, 0, 0, 0, 0, 0, 7, 4],

[0, 0, 5, 2, 0, 6, 3, 0, 0]]

if (solveSuduko(grid, 0, 0)):

printing(grid)

else:

print("no solution exists ")

OUTPUT –

3 1 6 5 7 8 4 9 2

5 2 9 1 3 4 7 6 8

4 8 7 6 2 9 5 3 1

2 6 3 4 1 5 9 8 7

9 7 4 8 6 3 1 2 5

8 5 1 7 9 2 6 4 3

1 3 8 9 4 7 2 5 6

6 9 2 3 5 1 8 7 4

7 4 5 2 8 6 3 1 9