**Experiment 8**

**Implementation Of Knowledge Representation Schemes**

**Problem taken:** Solving Soduko using Knowledge Representation.

**Aim:**

To solve Soduko using Knowledge Representation & implement a solution for it.

**Language:** Python 3.7

**Problem Description:**

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.

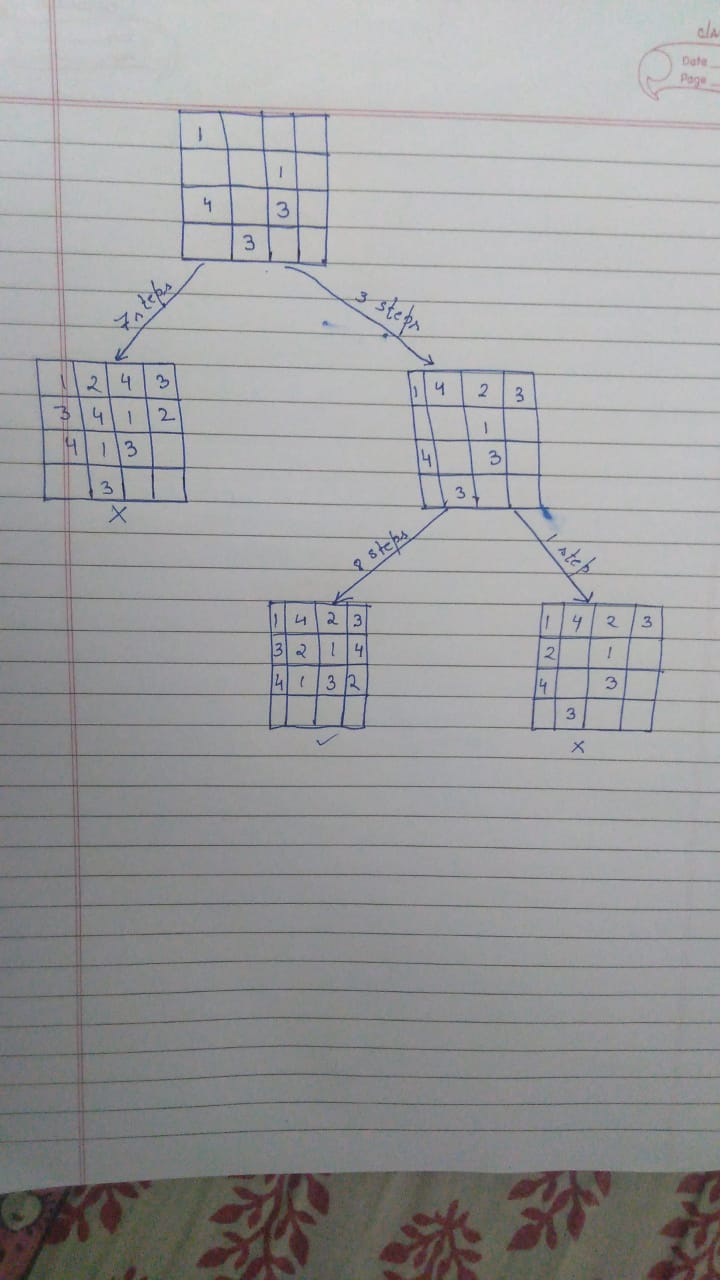
**Problem Formulation:**

* Create a function that checks after assigning the current index the grid becomes unsafe or not. Keep Hashmap for a row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true; hashMap can be avoided by using loops.
* Create a recursive function that takes a grid.
* Check for any unassigned location. If present then assign a number from 1 to 9, check if assigning the number to current index makes the grid unsafe or not, if safe then recursively call the function for all safe cases from 0 to 9. if any recursive call returns true, end the loop and return true. If no recursive call returns true then return false.
* If there is no unassigned location then return true.

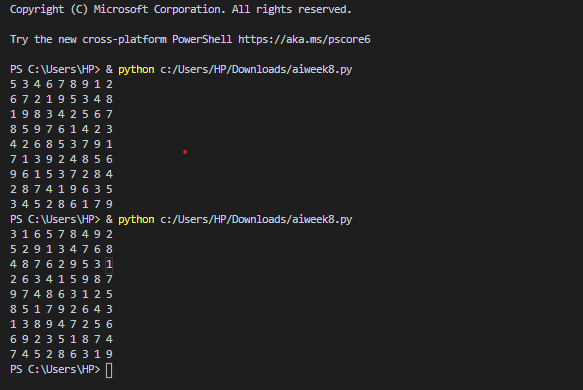
**Source code:**

size = 9  
# empty cells have value zero  
grid1 = [  
 [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]]  
  
grid2 = [[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]]  
  
# two grids are used as sample, either one can be used.  
matrix = grid1.copy()  
# print sudoku  
def print\_sudoku():  
 for i in matrix:  
 for j in i:  
 print(j, end=' ')  
 print('')  
  
  
# assign cells and check  
def number\_unassigned(row, col):  
 num\_unassign = 0  
 for i in range(0, size):  
 for j in range(0, size):  
 # cell is unassigned  
 if matrix[i][j] == 0:  
 row = i  
 col = j  
 num\_unassign = 1  
 a = [row, col, num\_unassign]  
 return a  
 a = [-1, -1, num\_unassign]  
 return a  
  
  
# check validity of number  
def is\_safe(n, r, c):  
 # checking in row  
 for i in range(0, size):  
 # there is a cell with same value  
 if matrix[r][i] == n:  
 return False  
 # checking in column  
 for i in range(0, size):  
 # there is a cell with same value  
 if matrix[i][c] == n:  
 return False  
 row\_start = (r // 3) \* 3  
 col\_start = (c // 3) \* 3  
 # checking submatrix  
 for i in range(row\_start, row\_start + 3):  
 for j in range(col\_start, col\_start + 3):  
 if matrix[i][j] == n:  
 return False  
 return True  
  
  
# check validity of number  
def solve\_sudoku():  
 row = 0  
 col = 0  
 # if all cells are assigned then the sudoku is already solved  
 # pass by reference because number\_unassigned will change the values of row and col  
 a = number\_unassigned(row, col)  
 if a[2] == 0:  
 return True  
 row = a[0]  
 col = a[1]  
 # number between 1 to 9  
 for i in range(1, 10):  
 # if we can assign i to the cell or not  
 # the cell is matrix[row][col]  
 if is\_safe(i, row, col):  
 matrix[row][col] = i  
 # backtracking  
 if solve\_sudoku():  
 return True  
 # f we can't proceed with this solution  
 # reassign the cell  
 matrix[row][col] = 0  
 return False  
  
  
if solve\_sudoku():  
 print\_sudoku()  
else:  
 print("No solution")

**Test Cases:**



**Verification:**



**Result:**

The Soduko using Knowledge Representation & Backtracking was analyzed and an optimal solution was devised. This solution was the coded & tested against various test cases and documented.