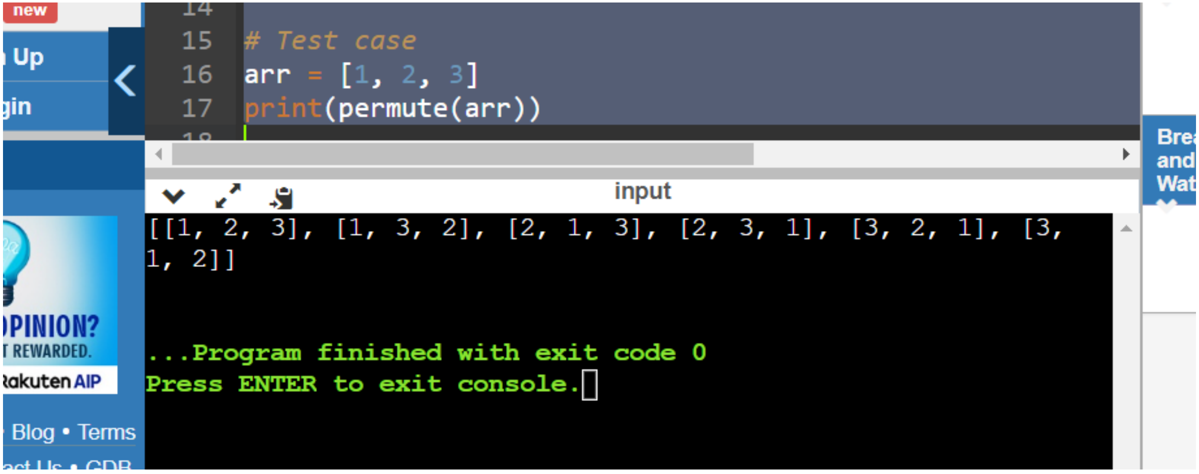
Tanish Mittal
IT-C
2000290130174

Day – 10
Recursion & Backtracking

Problem Statement: Given an array arr of distinct integers, print all permutations of String/Array.

```
def permute(nums):
    def backtrack(start):
        if start == len(nums):
            result.append(nums[:])
        else:
            for i in range(start, len(nums)):
                nums[start], nums[i] = nums[i], nums[start]
                backtrack(start + 1)
                nums[start], nums[i] = nums[i], nums[start]


    result = []
    backtrack(0)
    return result
arr = [1, 2, 3]
print(permute(arr))
```

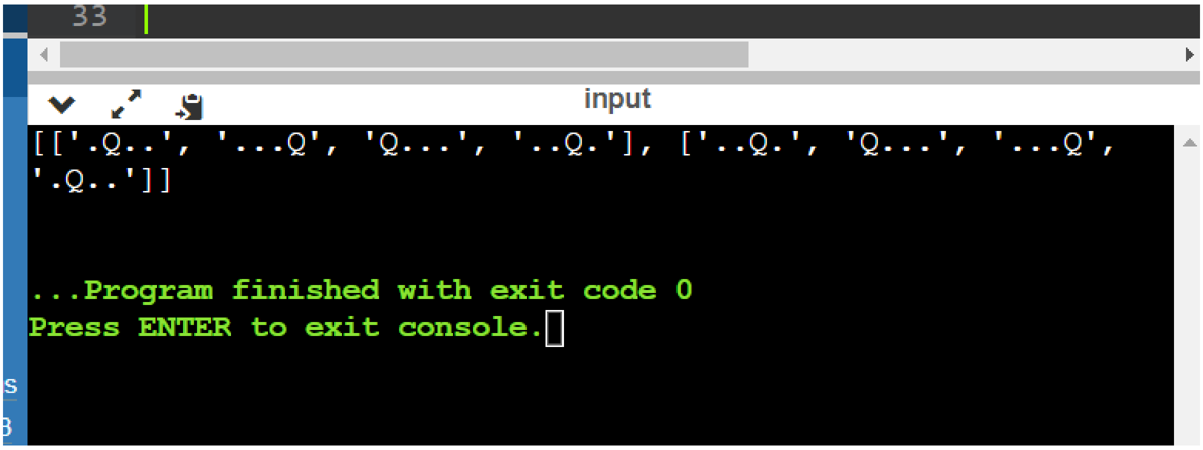


```
# Test case
arr = [1, 2, 3]
print(permute(arr))
```

```
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 2, 1], [3,
1, 2]]


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: The n-queens is the problem of placing n queens on n × n chessboard such that no two queens can attack each other. Given an integer n, return all distinct solutions to the n -queens puzzle. Each solution contains a distinct boards configuration of the queen's placement, where 'Q' and '.' indicate queen and empty space respectively.

```
def solveNQueens(n):
    def backtrack(row, col_placement, result):
        if row == n: # Base case: All queens have been placed
            result.append(generateBoard(col_placement))
        else:
            for col in range(n):
                if isValidPlacement(row, col, col_placement):
                    col_placement.append(col)
                    backtrack(row + 1, col_placement, result)
                    col_placement.pop()

    def isValidPlacement(row, col, col_placement):
        for i in range(row):
            if col == col_placement[i] or \
               row - i == abs(col - col_placement[i]):
                return False
        return True

    def generateBoard(col_placement):
        board = []
        for i in range(n):
            row = ['.'] * n
            row[col_placement[i]] = 'Q'
            board.append(''.join(row))
        return board

    result = []
    backtrack(0, [], result)
    return result
n = 4
solutions = solveNQueens(n)
print(solutions)
```



```
[['.Q..', '...Q', 'Q...', '..Q.'], ['..Q.', 'Q...', '...Q',
'.Q..']]


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement:

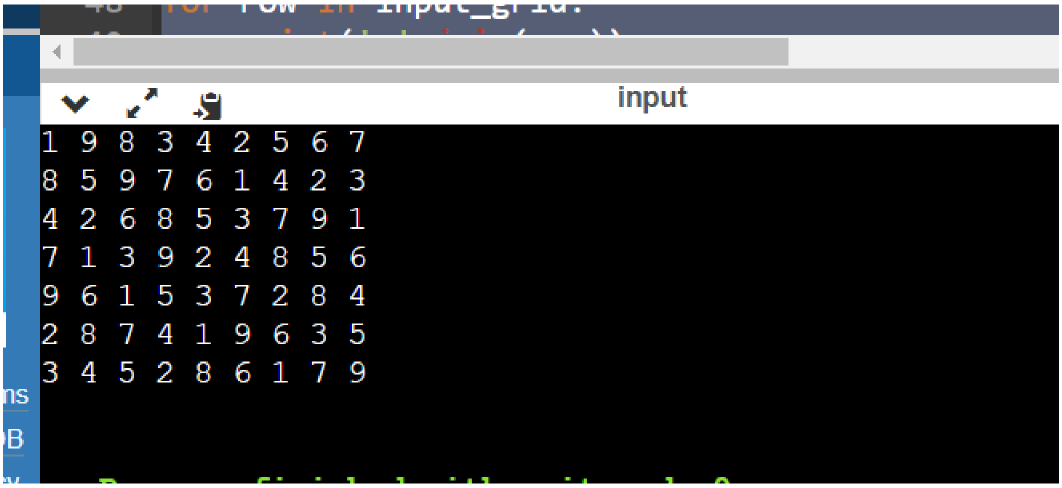
Given a 9×9 incomplete sudoku, solve it such that it becomes valid sudoku. Valid sudoku has the following properties.

1. All the rows should be filled with numbers(1 – 9) exactly once.

2. All the columns should be filled with numbers(1 – 9) exactly once.

3. Each 3×3 submatrix should be filled with numbers(1 – 9) exactly once.

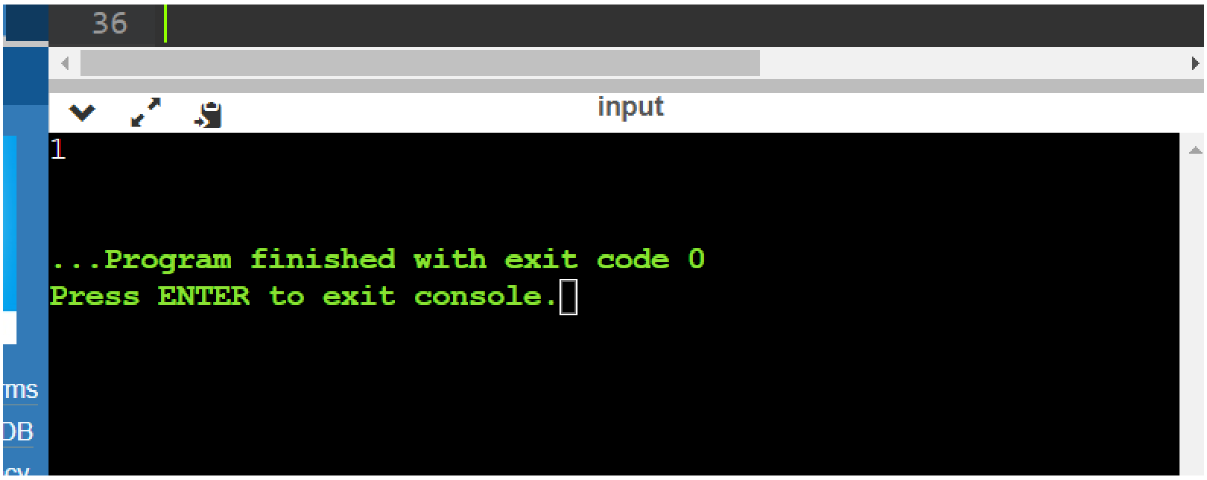
```
def is_valid(grid, row, col, num):
    for i in range(9):
        if grid[row][i] == num:
            return False
    for i in range(9):
        if grid[i][col] == num:
            return False
    start_row = 3 * (row // 3)
    start_col = 3 * (col // 3)
    for i in range(3):
        for j in range(3):
            if grid[start_row + i][start_col + j] == num:
                return False

    return True

def solve_sudoku(grid):
    for row in range(9):
        for col in range(9):
            if grid[row][col] == '.':
                for num in range(1, 10):
                    if is_valid(grid, row, col, str(num)):
                        grid[row][col] = str(num)
                        if solve_sudoku(grid):
                            return True
                        grid[row][col] = '.'

                return False

    return True


input_grid = [
    ['5', '3', '.', '.', '7', '.', '.', '.', '.'],
    ['6', '.', '.', '1', '9', '5', '.', '.', '.'],
    ['.', '9', '8', '.', '.', '.', '.', '6', '.'],
    ['8', '.', '.', '.', '6', '.', '.', '.', '3'],
    ['4', '.', '.', '8', '.', '3', '.', '.', '1'],
    ['7', '.', '.', '.', '2', '.', '.', '.', '6'],
    ['.', '6', '.', '.', '.', '.', '2', '8', '.'],
    ['.', '.', '.', '4', '1', '9', '.', '.', '5'],
    ['.', '.', '.', '.', '8', '.', '.', '7', '9']
]


solve_sudoku(input_grid)



for row in input_grid:
    print(' '.join(row))
```



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

Problem Statement: Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with the same color.

```
def graphColoringUtil(graph, m, colors, v):
    if v == len(graph):
        return True
    for c in range(1, m + 1):
        if isSafe(graph, colors, v, c):
            colors[v] = c
            if graphColoringUtil(graph, m, colors, v + 1):
                return True
            colors[v] = -1

    return False

def isSafe(graph, colors, v, c):
    for u in graph[v]:
        if colors[u] == c:
            return False
    return True

def graphColoring(N, M, Edges):
    graph = [[] for _ in range(N)]
    for u, v in Edges:
        graph[u].append(v)
        graph[v].append(u)

    colors = [-1] * N
    if graphColoringUtil(graph, M, colors, 0):
        return 1
    return 0

N = 4
M = 3
Edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]

print(graphColoring(N, M, Edges))
```

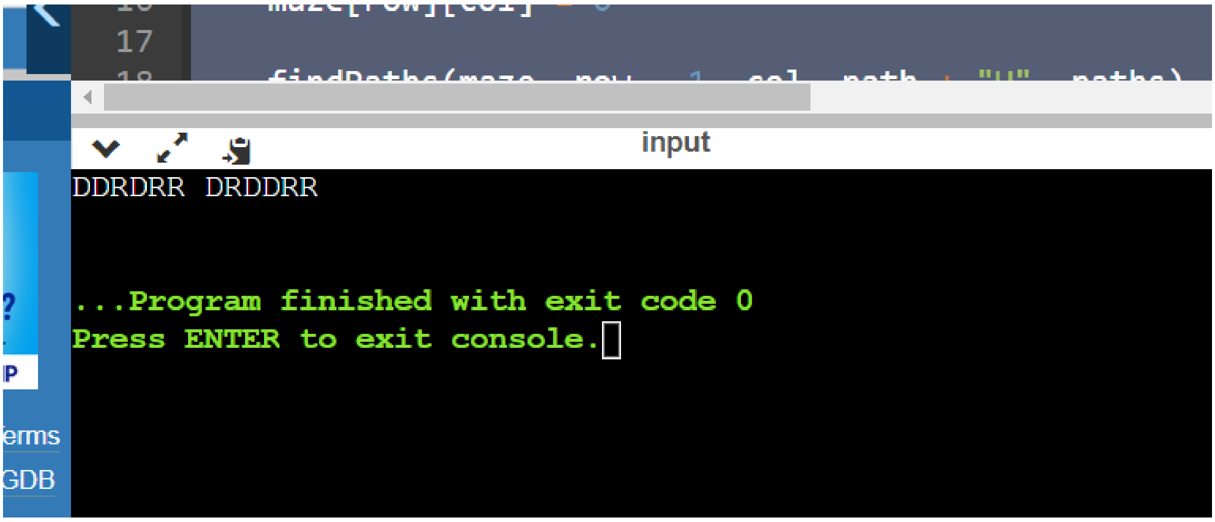


```
1


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Rat in a Maze

Consider a rat placed at **(0, 0)** in a square matrix of order **N * N**. It has to reach the destination at **(N - 1, N - 1)**. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are **'U'(up)**, **'D'(down)**, **'L' (left)**, **'R' (right)**. Value 0 at a cell in the matrix represents that it is blocked and the rat cannot move to it while value 1 at a cell in the matrix represents that rat can travel through it.

```
def findPaths(maze, row, col, path, paths):
    N = len(maze)
    if row == N - 1 and col == N - 1:
        paths.append(path)
        return

    if (
        row < 0
        or col < 0
        or row >= N
        or col >= N
        or maze[row][col] == 0
    ):
        return

    maze[row][col] = 0

    findPaths(maze, row - 1, col, path + "U", paths)
    findPaths(maze, row + 1, col, path + "D", paths)
    findPaths(maze, row, col - 1, path + "L", paths)
    findPaths(maze, row, col + 1, path + "R", paths)
    maze[row][col] = 1


def findMazePaths(N, m):
    paths = []
    findPaths(m, 0, 0, "", paths)
    paths.sort()
    return paths

N = 4
m = [
    [1, 0, 0, 0],
    [1, 1, 0, 1],
    [1, 1, 0, 0],
    [0, 1, 1, 1],
]

result = findMazePaths(N, m)
print(" ".join(result))
```



```
    maze[row][col] = 0

    findPaths(maze, row - 1, col, path + "U", paths)
```

```
DDRDRR DRDDRR


...Program finished with exit code 0
Press ENTER to exit console.
```