6.3 SUDOKU PUZZLE SOLVER

Question:
Write a program to solve a Sudoku puzzle by filling the empty cells.
A Sudoku solution must satisfy all of the following rules:
- Each of the digits 1–9 must occur exactly once in each row.
- Each of the digits 1–9 must occur exactly once in each column.
- Each of the digits 1–9 must occur exactly once in each of the 9 sub-boxes of the grid.

The '.' character indicates empty cells.

AIM
To implement a Sudoku solver in Python using the backtracking algorithm.

ALGORITHM
1. Traverse the board to find an empty cell (marked as '.').
2. For digits 1–9, check if placing the number is valid:
3. - Not already present in the current row.
4. - Not already present in the current column.
5. - Not already present in the 3×3 sub-grid.
6. If valid, place the number and recursively attempt to solve the rest of the board.
7. If no number is valid, backtrack by resetting the cell and trying another value.
8. Repeat until the board is completely filled with valid numbers.

PROGRAM

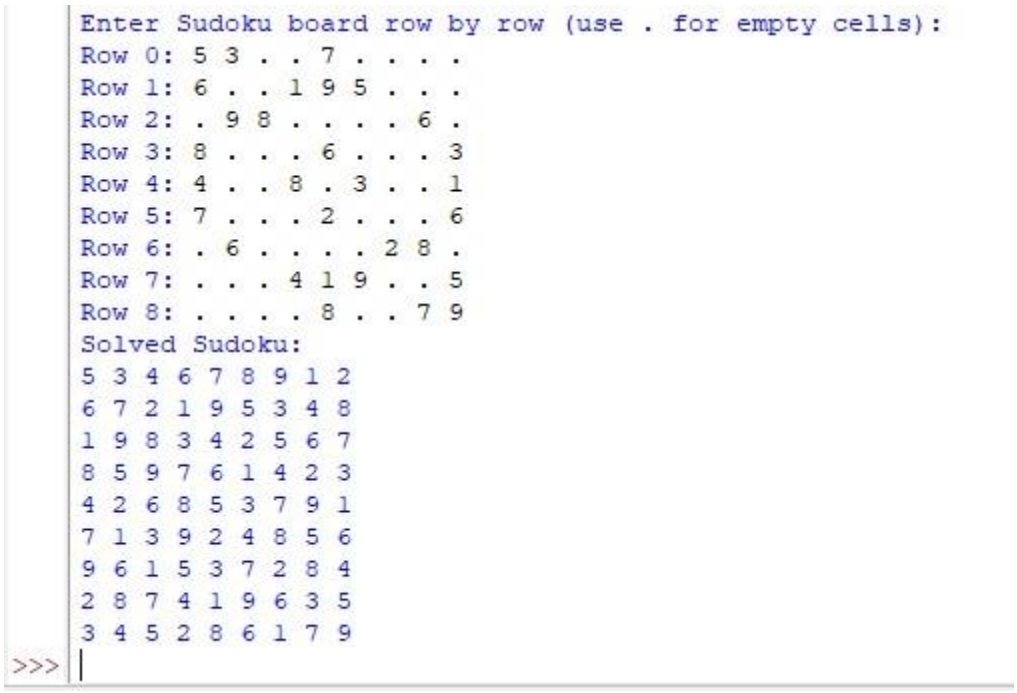
```
def solve_sudoku(board):
    def is_valid(r, c, ch):
        for i in range(9):
            if board[r][i] == ch or board[i][c] == ch or board[3*(r//3)+i//3][3*(c//3)+i%3] == ch:
                return False
        return True

    def backtrack():
        for r in range(9):
            for c in range(9):
                if board[r][c] == '.':
                    for ch in '123456789':
                        if is_valid(r, c, ch):
                            board[r][c] = ch
                            if backtrack():
                                return True
                            board[r][c] = '.'
                    return False
        return True

    backtrack()

board = []
print("Enter Sudoku board row by row (use . for empty cells):")
for i in range(9):
    row = input(f"Row {i}: ").split()
    board.append(row)

solve_sudoku(board)
print("Solved Sudoku:")
for row in board:
    print(' '.join(row))
```

Input:

Enter Sudoku board row by row (use . for empty cells):

Row 0:- 5 3 . . 7 . . . .
Row 1:- 6 . . 1 9 5 . . .
Row 2:- . 9 8 . . . . 6 .
Row 3:- 8 . . . 6 . . . 3
Row 4:- 4 . . 8 . 3 . . 1
Row 5:- 7 . . . 2 . . . 6
Row 6:- . 6 . . . . 2 8 .
Row 7:- . . . 4 1 9 . . 5
Row 8:- . . . . 8 . . 7 9

Output:



RESULT:

Thus, the program is successfully executed and the output is verified.

PERFORMANCE ANALYSIS:

- Time Complexity: Worst case exponential O(9^n), where n is the number of empty cells. However, with pruning and backtracking, it solves standard Sudoku efficiently.
- Space Complexity: O(1) (only the given board is modified in place; recursion depth at most 81).