SCT_SD_3

TASK 3:

Create a program that solves sudoku puzzles automatically. The program should take an input grid representing an unsolved sudoku puzzle and use an algorithm to fill in the missing numbers.

Source Code: (PYTHON)

```
board = []
#Sudoku puzzle solver
def board_input(bo):
    bo.clear()  # Clear the list to ensure it's empty before taking input
    print("Enter the Sudoku grid row by row (9 rows of 9 space-separated numbers, use
0 for empty cells):")
    for i in range(9):
        row = list(map(int, input().split()))
        if len(row) != 9:
            print("Each row must have exactly 9 numbers!")
            return None
        bo.append(row)

def solve(bo):
    find = find_empty(bo)
    if not find:
        return True  # Board is completely filled
    else:
        row, col = find

    for i in range(1, 10):
        if valid(bo, i, (row, col)):
            bo[row][col] = i

            if solve(bo):  # Recursive call
                return True

            bo[row][col] = 0  # Backtrack

    return False
```

```python
def valid(bo, num, pos):
    # Check row
    for i in range(len(bo[0])):
        if bo[pos[0]][i] == num and pos[1] != i:
            return False

    # Check column
    for i in range(len(bo)):
        if bo[i][pos[1]] == num and i != pos[0]:
            return False

    # Check 3x3 box
    box_x = pos[1] // 3
    box_y = pos[0] // 3
    for i in range(box_y * 3, box_y * 3 + 3):
        for j in range(box_x * 3, box_x * 3 + 3):
            if bo[i][j] == num and (i, j) != pos:
                return False

    return True  # If all checks pass

def printb(bo):
    for i in range(len(bo)):
        if i % 3 == 0 and i != 0:
            print("_ _ _ _ _ _ _ _ _ _ _ _")  # Horizontal separator for 3x3 blocks

        for j in range(len(bo[i])):
            if j % 3 == 0 and j != 0:
                print(" | ", end="")  # Vertical separator for 3x3 blocks

            print(bo[i][j], end=" ")  # Print the number with space

        print()  # Move to the next row

def find_empty(bo):
    for i in range(len(bo)):
        for j in range(len(bo[0])):
            if bo[i][j] == 0:
                return (i, j)  # row, col
    return None

# Get input from user
board_input(board)

# Print the Sudoku grid
print("\nUnsolved Sudoku Grid:\n")
printb(board)
```

```python
# Solve and print the solved board
if solve(board):
    print("\nSolved Board:")
    printb(board)
else:
    print("\nNo solution exists for the given Sudoku grid.")
```

Output:

```
/usr/bin/python3 /Users/ayushghosh/Desktop/SCT_SD/SCT_SD_3/SCT_SD_3.py
ayushghosh@Ayushs-MacBook-Air SCT_SD_3 % /usr/bin/python3 /Users/ayushghosh/Desktop/SCT_SD/SCT_SD_3/SCT_SD_3.py
Enter the Sudoku grid row by row (9 rows of 9 space-separated numbers, use 0 for empty cells):
0 0 0 2 6 0 7 0 1
6 8 0 0 7 0 0 9 0
1 9 0 0 0 4 5 0 0
8 2 0 1 0 0 0 4 0
0 0 4 6 0 2 9 0 0
0 5 0 0 0 3 0 2 8
0 0 9 3 0 0 0 7 4
0 4 0 0 5 0 0 3 6
7 0 3 0 1 8 0 0 0

Unsolved Sudoku Grid:

0 0 0  | 2 6 0  | 7 0 1
6 8 0  | 0 7 0  | 0 9 0
1 9 0  | 0 0 4  | 5 0 0
_ _ _ _ _ _ _ _ _ _ _ _
8 2 0  | 1 0 0  | 0 4 0
0 0 4  | 6 0 2  | 9 0 0
0 5 0  | 0 0 3  | 0 2 8
_ _ _ _ _ _ _ _ _ _ _ _
0 0 9  | 3 0 0  | 0 7 4
0 4 0  | 0 5 0  | 0 3 6
7 0 3  | 0 1 8  | 0 0 0

Solved Board:
4 3 5  | 2 6 9  | 7 8 1
6 8 2  | 5 7 1  | 4 9 3
1 9 7  | 8 3 4  | 5 6 2
_ _ _ _ _ _ _ _ _ _ _ _
8 2 6  | 1 9 5  | 3 4 7
3 7 4  | 6 8 2  | 9 1 5
9 5 1  | 7 4 3  | 6 2 8
_ _ _ _ _ _ _ _ _ _ _ _
5 1 9  | 3 2 6  | 8 7 4
2 4 8  | 9 5 7  | 1 3 6
7 6 3  | 4 1 8  | 2 5 9
ayushghosh@Ayushs-MacBook-Air SCT_SD_3 %
```