Sudoku Solver

```
import tkinter as tk
from tkinter import messagebox

# Define global variables
solutions = []
current_solution_index = 0
grid = [[0 for _ in range(9)] for _ in range(9)]
solution_count = 0  # Track the number of solutions found
user_input_cells = set()

# Function to find an empty location in the Sudoku grid
def find_empty_location(grid):
    for row in range(9):
        for col in range(9):
            if grid[row][col] == 0:
                return row, col
    return None, None

# Function to check if a number is used in a row
def used_in_row(grid, row, num):
    return num in grid[row]

# Function to check if a number is used in a column
def used_in_col(grid, col, num):
    return num in [grid[row][col] for row in range(9)]

# Function to check if a number is used in a 3x3 box
def used_in_box(grid, row, col, num):
    for i in range(3):
        for j in range(3):
            if grid[i + row][j + col] == num:
                return True
    return False

# Function to check if it's safe to place a number at a given location
def is_safe(grid, row, col, num):
    return not used_in_row(grid, row, num) and not used_in_col(grid,
col, num) and not used_in_box(grid, row - row % 3, col - col % 3, num)

# Function to solve the Sudoku puzzle recursively
def solve_sudoku(grid):
    global solutions, solution_count
    solutions = []
    solution_count = 0
    solve_sudoku_helper(grid)

# Helper function for solving Sudoku recursively
def solve_sudoku_helper(grid):
```

```
    global solutions, solution_count
    if solution_count >= 10:  # Stop when 10 solutions are found
        return

    row, col = find_empty_location(grid)
    if row is None or col is None:
        solutions.append([row[:] for row in grid])
        solution_count += 1
        return

    # Try numbers 1-9
    for num in range(1, 10):
        if is_safe(grid, row, col, num):
            grid[row][col] = num
            solve_sudoku_helper(grid)
            grid[row][col] = 0

# Function to display the next solution
def next_solution():
    global current_solution_index
    if current_solution_index < len(solutions) - 1:
        current_solution_index += 1
        display_solution()

# Function to display the previous solution
def prev_solution():
    global current_solution_index
    if current_solution_index > 0:
        current_solution_index -= 1
        display_solution()

# Function to display the current solution
def display_solution():
    global solutions, current_solution_index
    if solutions and current_solution_index < len(solutions):
        grid = solutions[current_solution_index]
        for i in range(9):
            for j in range(9):
                entry = entries[i][j]
                entry.delete(0, tk.END)
                entry.insert(0, str(grid[i][j]))
                if (i, j) in user_input_cells:
                    entry.config(bg="lightsteelblue")  # Color for
user-input cells
                else:
                    entry.config(bg="mistyrose")    # Color for
solver-filled cells
    else:
        messagebox.showinfo("No Solution", "No solution exists or all
solutions have been displayed.")
```

```
# Function to solve the Sudoku puzzle when the "Solve" button is
clicked
def solve_button_clicked():
    global solutions, grid, user_input_cells
    user_input_cells = set()  # Reset user input cells
    for i in range(9):
        for j in range(9):
            value = entries[i][j].get()
            if value:
                grid[i][j] = int(value)
                user_input_cells.add((i, j))  # Record user input
cells
            else:
                grid[i][j] = 0
    solve_sudoku(grid)
    if solutions:
        display_solution()
    else:
        messagebox.showinfo("No Solution", "No solution exists for
this Sudoku puzzle.")

# Initialize Tkinter
root = tk.Tk()
root.title("Sudoku Solver")

# Create grid of entry widgets
entries = []
for i in range(9):
    row = []
    for j in range(9):
        entry = tk.Entry(root, width=3, bg="beige", justify="center",
font=("Arial", 12))
        entry.grid(row=i, column=j)
        row.append(entry)
    entries.append(row)

# Create frame for buttons
button_frame = tk.Frame(root)
button_frame.grid(row=10, column=0, columnspan=9, pady=10)

# Create Previous Solution button
prev_button = tk.Button(button_frame, text="Previous Solution",
command=prev_solution, bg="lightpink")
prev_button.pack(side=tk.LEFT, padx=5)

# Create Solve button
solve_button = tk.Button(button_frame, text="Solve",
command=solve_button_clicked, bg="powderblue")
solve_button.pack(side=tk.LEFT, padx=5)
```

```
# Create Next Solution button
next_button = tk.Button(button_frame, text="Next Solution",
command=next_solution, bg="lightgreen")
next_button.pack(side=tk.RIGHT, padx=5)

# Main loop
root.mainloop()
```

Sources:

- **Backtracking (Think Like a Programmer)**

Link: https://www.youtube.com/watch?v=gBC_Fd8EE8A

- **Tkinter Course - Create Graphic User Interfaces in Python Tutorial**

Link: https://youtu.be/YXPyB4XeYLA

- **solving Sudoku using backtracking algorithm**

Link: https://youtu.be/VH85GjJhp98