Lets write a sodoku solver.

1. We will use numpy to define the sodiku grid
2. We'll implement a backtracking algorithm to recursively solve the Sudoku puzzle, considering the rules of Sudoku: each row, column, and 3x3 subgrid must contain all digits from 1 to 9 without repetition.
3. The solver will iterate through each empty cell in the Sudoku grid, attempting to place numbers from 1 to 9 that satisfy the rules.
4. If a valid number is found, the solver will proceed to recursively solve the remaining empty cells. If no valid number is found, it will backtrack and try a different number.
5. Once all cells are filled, the Sudoku puzzle will be solved, and the final solution will be printed.

```
#import libraries

import numpy as np

def solve(bo):
    """
    Solve the Sudoku puzzle using backtracking.

    Parameters:
        bo (list): Sudoku board as a 2D list.

    Returns:
        bool: True if the Sudoku is solved, False otherwise.
    """
    find = find_empty(bo)
    if not find:
        return True
    else:
        row, col = find

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

            if solve(bo):
                return True

            bo[row][col] = 0

    return False


def valid(bo, num, pos):
    """
    Check if it's valid to place the number in the given position.

    Parameters:
```

```
        bo (list): Sudoku board as a 2D list.
        num (int): Number to be placed.
        pos (tuple): Position (row, column) to check.

    Returns:
        bool: True if the placement is valid, False otherwise.
    """
    # 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 pos[0] != i:
            return False

    # Check 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


def print_board(bo):
    """
    Print the Sudoku board.

    Parameters:
        bo (list): Sudoku board as a 2D list.
    """
    for i in range(len(bo)):
        if i % 3 == 0 and i != 0:
            print("- - - - - - - - - - - - ")

        for j in range(len(bo[0])):
            if j % 3 == 0 and j != 0:
                print(" | ", end="")

            if j == 8:
                print(bo[i][j])
            else:
                print(str(bo[i][j]) + " ", end="")
```

```python
def find_empty(bo):
    """
    Find an empty cell in the Sudoku board.

    Parameters:
        bo (list): Sudoku board as a 2D list.

    Returns:
        tuple or None: Position (row, column) of the empty cell, or
None if no empty cell is found.
    """
    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

# Define the Sudoku boards
board = [
    [7, 8, 0, 4, 0, 0, 1, 2, 0],
    [6, 0, 0, 0, 7, 5, 0, 0, 9],
    [0, 0, 0, 6, 0, 1, 0, 7, 8],
    [0, 0, 7, 0, 4, 0, 2, 6, 0],
    [0, 0, 1, 0, 5, 0, 9, 3, 0],
    [9, 0, 4, 0, 6, 0, 0, 0, 5],
    [0, 7, 0, 3, 0, 0, 0, 1, 2],
    [1, 2, 0, 0, 0, 7, 4, 0, 0],
    [0, 4, 9, 2, 0, 6, 0, 0, 7]
]

board2 = [
    [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]
]

# Print the original Sudoku board
# print("Original Board:")
# print_board(board)

# # Solve the Sudoku puzzle
# solve(board)
```

```python
# # Print the solved Sudoku board
# print("\nSolved Board:")
# print_board(board)

# Print the original Sudoku board 2
print("\nOriginal Board 2:")
print_board(board2)

# Solve the Sudoku puzzle 2
solve(board2)

# Print the solved Sudoku board 2
print("\nSolved Board 2:")
print_board(board2)
```

```
Original Board 2:
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 2:
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
```