1. Introduction

- This program is an implementation of a Sudoku solver using the backtracking algorithm. It solves a given sudoku board by recursively trying different numbers in the empty locations and backtracking when the numbers are no longer valid.

2. Design and Implementation

- This program is an implementation of a Sudoku Solver and takes a Sudoku board stored as multidimensional list and solves it using a backtracking algorithm.
- The solver function is the main recursive function that solves the board.
    - It initially calls the **find_empty** function which goes through the list looking for a location that is empty. If no location is empty the Sudoku is considered solved. If a location is found empty the location is returned in a tuple.
    - In the empty location the numbers 1-9 are tried and check if the number would be a valid solution.
        - If a number is valid, it will call the solver function again and continue with the next empty square.
        - If a valid number cannot be found to solve for the square it will reset the square to 0 and backtrack to the previous valid number.
        - This continues recursively until the whole board is solved.
    - Functions:
        - The **valid** function checks if a given number is a valid solution for the specified position on the board. It checks the row, column, and the 3x3 box containing the position to ensure that the number is not repeated.
        - The **print_board** function prints the current state of the Sudoku board in a visually appealing format.
        - The **find_empty** function searches for an empty square (represented by 0) on the board. It returns the position of the first empty square found, or None if no empty squares are left.
        - The **solver** function is the main recursive function that solves the Sudoku board. It starts by calling **find_empty** to find an empty square. If no empty squares are left, the board is solved, and the function returns True. Otherwise, it tries numbers from 1 to 9 in the empty square and checks if each number is a valid solution using the valid function. If a number is valid, it places the number in the square and recursively calls solver to solve the rest of the board. If the recursive call returns True, it means the board is solved, and the function returns True. If the recursive call returns False, it means the current number choice leads to an invalid solution, so it backtracks by resetting the square to 0 and tries the next number. If no number leads to a valid solution, the function returns False.

3. Conclusions

- Discuss what you personally learned from your project.
    - Recursion was a topic I needed extra practice on and the troubleshooting and coding for this program helped in making recursion a clearer concept. I also learned the backtracking algorithm and explored the different use cases for it.
- Discuss the best features and the shortcomings of the project.

  - Best Feature: It works, it solves the sudoku board and with the print_board function the user can see the board before and after it has been solved.
  - Short Coming: Some of the code could be a bit more compacted, it has no way for the user to input a sudoku board and has no validation in case the Sudoku board is not solvable.
- Discuss any choices that you might have made differently, in hindsight after completing the project.
  - I would love to add a GUI to this program and a way to intake a Sudoku board from a user prompt. I feel like I could have used the enumerate function in a few spots however, my understanding of the function is not extensive.
- Describe any additional features you may want to add in the future.
  - Definitely a GUI and a way for the user to input a Sudoku board for solving.