l




Project by : Aman Pathan

: N Vallabh

gride game

**Sudoku**

Sudoku is a logic-based, combinatorial number-placement puzzle. In classic Sudoku, the objective is to fill a 9 × 9 grid with digits so that each column, each row, and each of the nine 3 × 3 sub grids that compose the grid contain all of the digits from 1 to 9

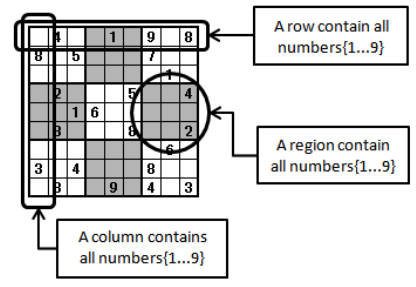
**What is the purpose of the game Sudoku?**

Sudoku is a logic based number placement puzzle. The objective is to fill a 9×9 grid so that each column, each row, and each of the nine 3×3 boxes (also called blocks or regions) contains the digits from 1 to 9, only one time each (that is, exclusively). The puzzle setter provides a partially completed grid

**Sudoku Solver Algorithm**

Your Sudoku Generator algorithm may need to use a Sudoku Solver Algorithm in order to test whether a generated grid is solvable and to check that it only gives a single solution.

The most common type of Sudoku Solver Algorithm is based on a backtracking algorithm used to investigate all possible solutions of a given grid.



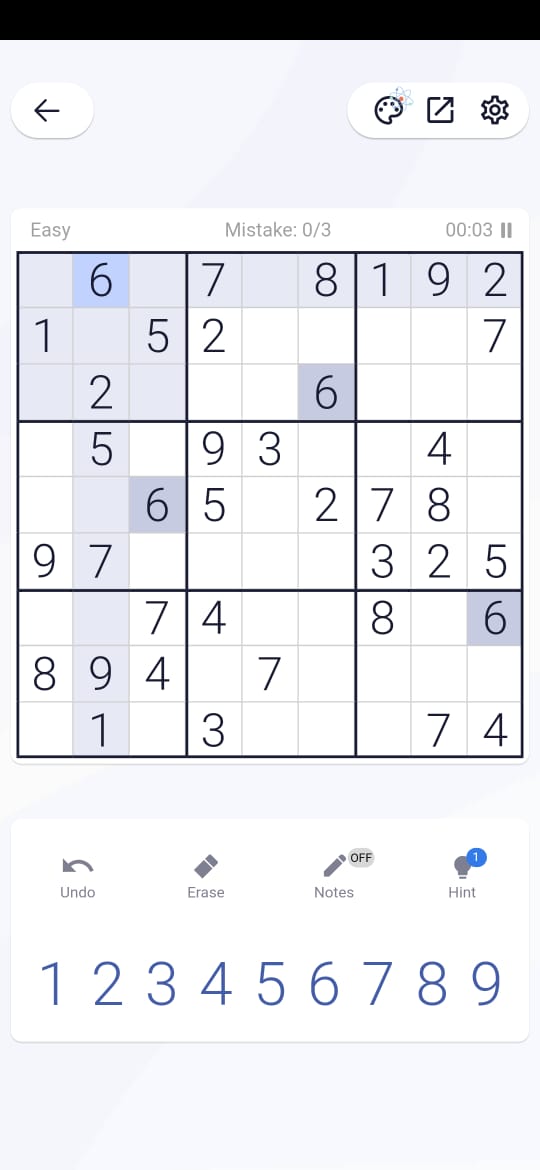
### Backtracking algorithm

Backtracking is a type of [depth-first search](https://en.wikipedia.org/wiki/Depth-first_search) algorithm that can be used to find some or all solutions to a problem, and is often used for [constraint satisfaction problems](https://en.wikipedia.org/wiki/Constraint_satisfaction_problem) such as Sudoku or crossword puzzles, or the [eight queens puzzle](https://en.wikipedia.org/wiki/Eight_queens_puzzle).

In Sudoku the constraints are:

* each square must contain a number from 1-9
* each number 1-9 can only occur once in a column, row, or 3x3 box

Backtracking is a brute force search algorithm that incrementally builds a solution, and backtracks when it takes a direction in the search path that will not lead to a solution.



M **=** 9

**def** puzzle(a):

**for** i **in** range(M):

**for** j **in** range(M):

            print(a[i][j],end **=** " ")

        print()

**def** solve(grid, row, col, num):

**for** x **in** range(9):

**if** grid[row][x] **==** num:

**return** False

**for** x **in** range(9):

**if** grid[x][col] **==** num:

**return** False

    startRow **=** row **-** row **%** 3

    startCol **=** col **-** col **%** 3

**for** i **in** range(3):

**for** j **in** range(3):

**if** grid[i **+** startRow][j **+** startCol] **==** num:

**return** False

**return** True

**def** Suduko(grid, row, col):

**if** (row **==** M **-** 1 **and** col **==** M):

**return** True

**if** col **==** M:

        row **+=** 1

        col **=** 0

**if** grid[row][col] > 0:

**return** Suduko(grid, row, col **+** 1)

**for** num **in** range(1, M **+** 1, 1):

**if** solve(grid, row, col, num):

            grid[row][col] **=** num

**if** Suduko(grid, row, col **+** 1):

**return** True

        grid[row][col] **=** 0

**return** False

'''0 means the cells where no value is assigned'''

grid **=** [[0, 6, 0, 7, 0, 8, 1, 9, 2],

[1, 0, 5, 2, 0, 0, 0, 0, 7],

[0, 2, 0, 0, 0, 6, 0, 0, 0],

[0, 5, 0, 9, 3, 0, 0, 4, 0],

[0, 0, 6, 5, 0, 2, 7, 8, 0],

[9, 7, 0, 0, 0, 0, 3, 2, 5],

[0, 0, 7, 4, 0, 0, 8, 0, 6],

[8, 9, 4, 0, 7, 0, 0, 0, 0],

[0, 1, 0, 3, 0, 0, 0, 7, 4]

]

**if** (Suduko(grid, 0, 0)):

    puzzle(grid)

**else**:

    print("Solution does not exist:(")

Solution :

4 6 3 7 5 8 1 9 2

1 8 5 2 9 3 4 6 7

7 2 9 1 4 6 5 3 8

2 5 8 9 3 7 6 4 1

3 4 6 5 1 2 7 8 9

9 7 1 8 6 4 3 2 5

5 3 7 4 2 9 8 1 6

8 9 4 6 7 1 2 5 3

6 1 2 3 8 5 9 7 4

## **Conclusion**

That’s all for building a sudoku solver in Python! I hope you had fun reading through the article and learning how we implemented the code The conclusion of a Sudoku puzzle is when all the cells of the grid are filled with numbers in such a way that each column, each row, and each of the nine 3×3 sub-grids that compose the grid contains all of the digits from 1 to 9. This is done without repeating any numbers in a row, column or sub-grid. That’s all for building a sudoku solver in Python! I hope you had fun reading through the article and learning how we implemented the code