

main.py



Run

Output

```
1 def print_board(solution):
2     N = len(solution)
3     board = [['.' for _ in range(N)] for _ in
               range(N)]
4     for row in range(N):
5         board[row][solution[row]] = 'Q'
6     for row in board:
7         print(' '.join(row))
8     print("\n")
9 def solve_n_queens(N):
10     def is_safe(queens, row, col):
11         for i in range(row):
12             if queens[i] == col or abs
               (queens[i] - col) == abs(i -
               row):
13                 return False
14             return True
15     def solve(queens, row):
16         if row == N:
17             solutions.append(queens[:])
18             return
19         for col in range(N):
20             if is_safe(queens, row, col):
21                 queens[row] = col
22                 solve(queens, row + 1)
23     solutions = []
24     queens = [-1] * N
25     solve(queens, 0)
26     return solutions
27 N = 4
28 solutions = solve_n_queens(N)
29 for solution in solutions:
30     print_board(solution)
31
```

```
. Q . .
. . . Q
Q . . .
. . Q .

. . Q .
Q . . .
. . . Q
. Q . .
```

```
=== Code Execution Successful ===
```



main.py



Run

Output

```
1 def print_board_with_obstacles(solution, N,
  board):
2     board = [['.' for _ in range(N)] for _ in
      range(N)]
3     for row in range(N):
4         if solution[row] != -1:
5             board[row][solution[row]] = 'Q'
6
7     for (row, col) in obstacles:
8         board[row][col] = 'X'
9
10
11    for row in board:
12        print(' '.join(row))
13    print("\n")
14
15 def solve_n_queens_with_obstacles(N, obstacles
  ):
16     def is_safe(queens, row, col):
17
18         if (row, col) in obstacles:
19             return False
20         for i in range(row):
21
22             if queens[i] == col or abs
              (queens[i] - col) == abs(i -
                row):
23                 return False
24         return True
25
26     def solve(queens, row):
27         if row == N:
28             solutions.append(queens[:])
29             return
30         for col in range(N):
31             if is_safe(queens, row, col):
32                 queens[row] = col
33                 solve(queens, row + 1)
34                 queens[row] = -1
35
36     solutions = []
37     queens = [-1] * N
38     solve(queens, 0)
39     return solutions
40 N = 5
41 obstacles = [(2, 2), (4, 4)]
42 solutions = solve_n_queens_with_obstacles(N,
  obstacles)
43 for solution in solutions:
44     print_board_with_obstacles(solution, N,
      obstacles)
45
```

```
Q . . . .
. . Q . .
. . X . Q
. Q . . .
. . . Q X
```

=== Code Execution Successful ===

main.py



Run

Output



```

1 def solve_sudoku(board):
2     def is_valid(board, row, col, num):
3
4         for i in range(9):
5             if board[row][i] == num:
6                 return False
7
8
9         for i in range(9):
10            if board[i][col] == num:
11                return False
12
13
14            start_row, start_col = 3 * (row // 3),
15                3 * (col // 3)
16            for i in range(3):
17                for j in range(3):
18                    if board[start_row +
19                        i][start_col + j] == num:
20                        return False
21
22            return True
23
24        def solve(board):
25            for row in range(9):
26                for col in range(9):
27                    if board[row][col] == '.':
28
29                        for num in map(str, range(1
30                            , 10)):
31                            if is_valid(board, row,
32                                col, num):
33                                board[row][col] =
34                                    num
35                                if solve(board):
36                                    return True
37                                board[row][col] = '
38                                    .'
39                                return False
40
41            return True
42            solve(board)
43        return board
44
45 board = [
46     ["5", "3", ".", ".", "7", ".", ".", ".", "."],
47     ["6", ".", ".", "1", "9", "5", ".", ".", "."],
48     [".", "9", "8", ".", ".", ".", ".", "6", "."],
49     ["8", ".", ".", ".", "6", ".", ".", ".", "3"],
50     ["4", ".", ".", "8", ".", "3", ".", ".", "1"],
51     ["7", ".", ".", ".", "2", ".", ".", ".", "6"],
52     [".", "6", ".", ".", ".", ".", "2", "8", "."],
53     [".", ".", ".", "4", "1", "9", ".", ".", "5"],
54     [".", ".", ".", "8", ".", ".", "7", "9"]
55 ]
56
57 solved_board = solve_sudoku(board)
58 for row in solved_board:
59     print(row)
60

```

```

['5', '3', '.', '.', '7', '.', '.', '.', '.']
['6', '.', '.', '1', '9', '5', '.', '.', '.']
['.', '9', '8', '.', '.', '.', '.', '6', '.']
['8', '.', '.', '.', '6', '.', '.', '.', '3']
['4', '.', '.', '8', '.', '3', '.', '.', '1']
['7', '.', '.', '.', '2', '.', '.', '.', '6']
['.', '6', '.', '.', '.', '.', '2', '8', '.']
['.', '.', '.', '4', '1', '9', '.', '.', '5']
['.', '.', '.', '.', '8', '.', '.', '7', '9']

```

```

=== Code Execution Successful ===

```

main.py



Run

Output

Clear



JS

GO

php



```
1 def solve_sudoku(board):
2     def is_valid(board, row, col, num):
3
4         for i in range(9):
5             if board[row][i] == num:
6                 return False
7
8
9         for i in range(9):
10            if board[i][col] == num:
11                return False
12
13
14            start_row, start_col = 3 * (row // 3),
15                3 * (col // 3)
16            for i in range(3):
17                for j in range(3):
18                    if board[start_row +
19                        i][start_col + j] == num:
20                        return False
21
22            return True
23
24    def solve(board):
25        for row in range(9):
26            for col in range(9):
27                if board[row][col] == '.':
28                    for num in map(str, range(1
29                        , 10)):
30                        if is_valid(board, row,
31                            col, num):
32                            board[row][col] =
33                                num
34                            if solve(board):
35                                return True
36                            board[row][col] = '
37                                '
38                        return False
39            return True
40    solve(board)
41    return board
42
43 board = [
44     ["5", "3", ".", ".", "7", ".", ".", ".", "."],
45     ["6", ".", ".", "1", "9", "5", ".", ".", "."],
46     [".", "9", "8", ".", ".", ".", ".", "6", "."],
47     ["8", ".", ".", ".", "6", ".", ".", ".", "3"],
48     ["4", ".", ".", "8", ".", "3", ".", ".", "1"],
49     ["7", ".", ".", ".", "2", ".", ".", ".", "6"],
50     [".", "6", ".", ".", ".", ".", "2", "8", "."],
51     [".", ".", ".", "4", "1", "9", ".", ".", "5"],
52     [".", ".", ".", ".", "8", ".", ".", "7", "9"]
53 ]
54
55 solved_board = solve_sudoku(board)
56 for row in solved_board:
57     print(row)
58
```

```
['5', '3', '4', '6', '7', '8', '9', '1', '2']
['6', '7', '2', '1', '9', '5', '3', '4', '8']
['1', '9', '8', '3', '4', '2', '5', '6', '7']
['8', '5', '9', '7', '6', '1', '4', '2', '3']
['4', '2', '6', '8', '5', '3', '7', '9', '1']
['7', '1', '3', '9', '2', '4', '8', '5', '6']
['9', '6', '1', '5', '3', '7', '2', '8', '4']
['2', '8', '7', '4', '1', '9', '6', '3', '5']
['3', '4', '5', '2', '8', '6', '1', '7', '9']

=== Code Execution Successful ===
```

main.py



Run

Output

```
1 def findTargetSumWays(nums, target):
2     from collections import Counter
3     total = sum(nums)
4     if total < target or (total + target) % 2 !=
        = 0: return 0
5     s = (total + target) // 2
6     dp = Counter({0: 1})
7     for num in nums:
8         for j in range(s, num - 1, -1):
9             dp[j] += dp[j - num]
10    return dp[s]
11 print(findTargetSumWays([1], 1))
```

1

=== Code Execution Successful ===

main.py	<div><div></div><div></div><div></div><div>Run</div></div>	Output
	<pre>1 def sum_of_minimums(arr): 2     mod = 10**9 + 7 3     stack, total_sum = [], 0 4     for i in range(len(arr) + 1): 5         while stack and (i == len(arr) or 6             arr[stack[-1]] &gt; arr[i]): 7             j = stack.pop() 8             left = stack[-1] if stack else -1 9             right = i 10            total_sum += arr[j] * (j - left) * 11                (right - j) 12            stack.append(i) 13     return total_sum % mod 14 arr = [3, 1, 2, 4] 15 print(sum_of_minimums(arr))</pre>	<pre>17 === Code Execution Successful ===</pre>

main.py



Run

Output

```
1 def combinationSum(candidates, target):
2     def backtrack(remaining, combo, start):
3         if remaining == 0:
4             result.append(list(combo))
5             return
6         for i in range(start, len(candidates)):
7             if remaining >= candidates[i]:
8                 combo.append(candidates[i])
9                 backtrack(remaining -
                           candidates[i], combo, i)
10                combo.pop()
11
12     result = []
13     backtrack(target, [], 0)
14     return result
15 candidates = [2, 3, 6, 7]
16 target = 7
17 print(combinationSum(candidates, target))
```

[[2, 2, 3], [7]]

=== Code Execution Successful ===

main.py



Run








Output

```
1 def combination_sum2(candidates, target):
2     def backtrack(start, path, target):
3         if target == 0:
4             res.append(path)
5             return
6         for i in range(start, len(candidates)):
7             if i > start and candidates[i] ==
                candidates[i - 1]: continue
8             if candidates[i] > target: break
9             backtrack(i + 1, path +
                [candidates[i]], target -
                candidates[i])
10
11     candidates.sort()
12     res = []
13     backtrack(0, [], target)
14     return res
15 candidates = [10, 1, 2, 7, 6, 1, 5]
16 target = 8
17 print(combination_sum2(candidates, target))
18
```




[[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]

=== Code Execution Successful ===





main.py



Run

```
1 from itertools import permutations
2
3 def permute(nums):
4     return list(map(list, permutations(nums)))
5 nums = [1, 2, 3]
6 output = permute(nums)
7 print(output)
8
```

Output

Clear

[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]

=== Code Execution Successful ===

main.py



Run

Output

```
1 from itertools import permutations
2
3 def unique_permutations(nums):
4     return list(map(list, set(permutations(nums
5                             ))))
6
7 nums = [1, 1, 2]
8 output = unique_permutations(nums)
9 print(output)
```

```
[[1, 2, 1], [2, 1, 1], [1, 1, 2]]
```

```
=== Code Execution Successful ===
```