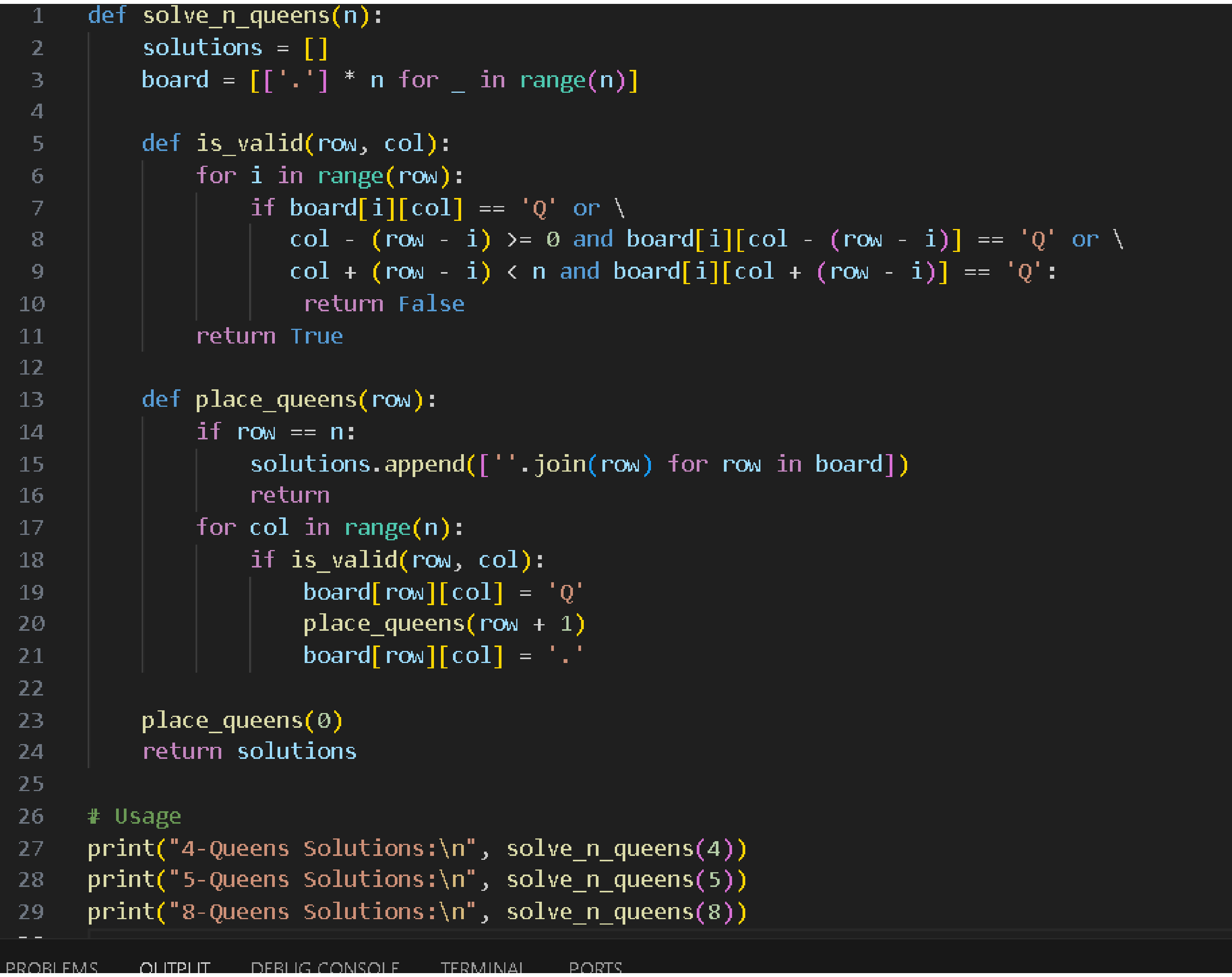
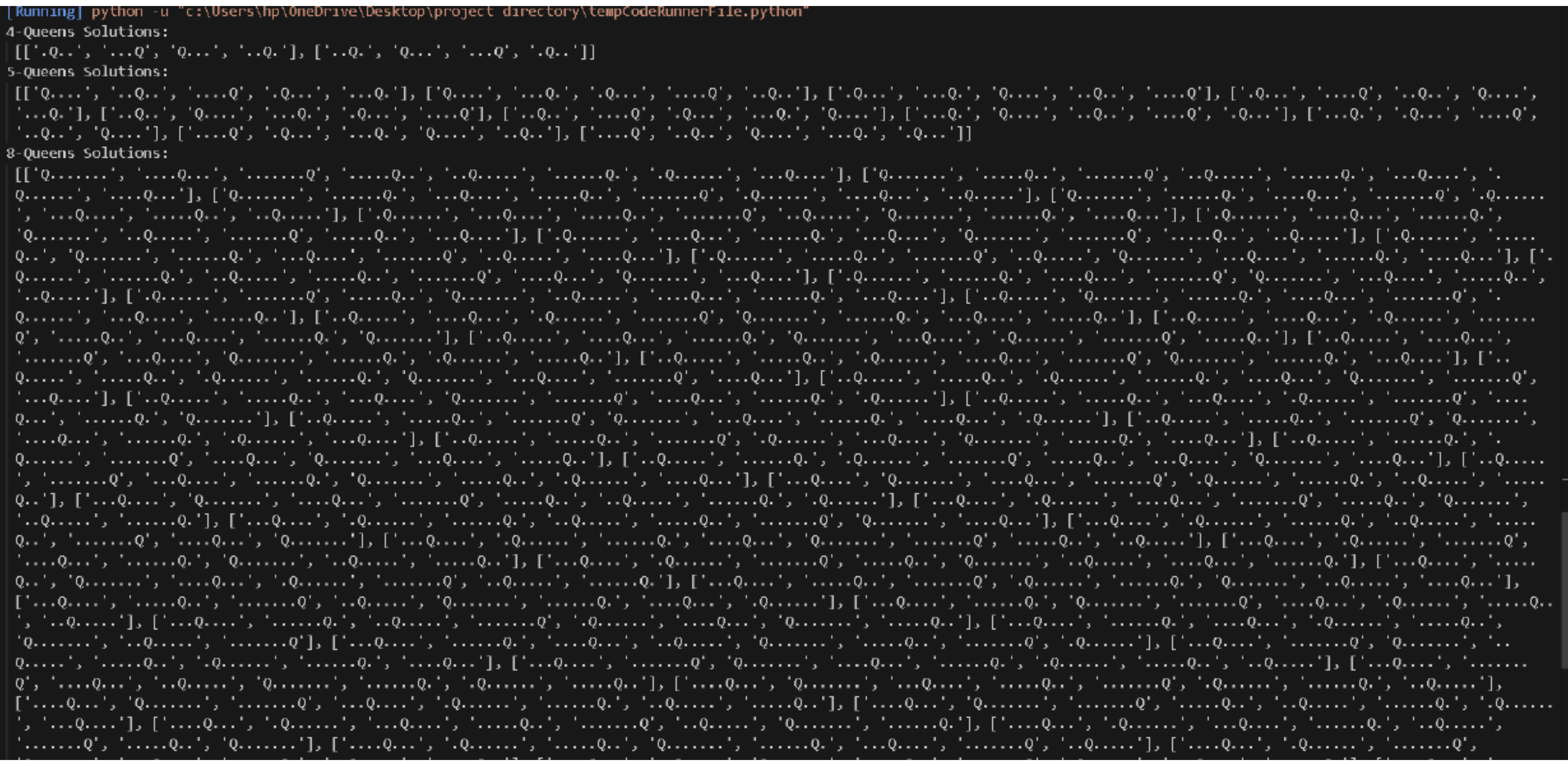
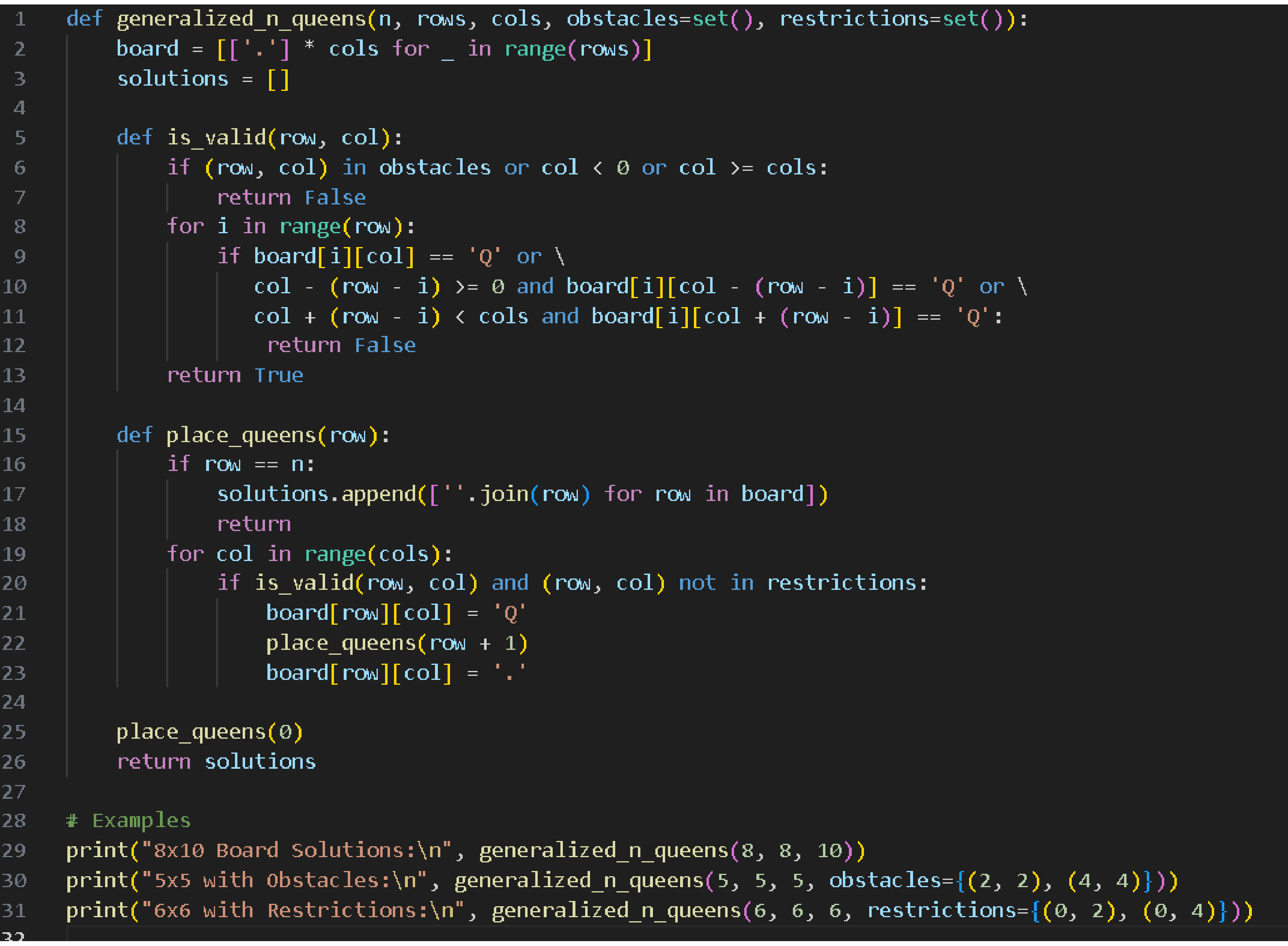


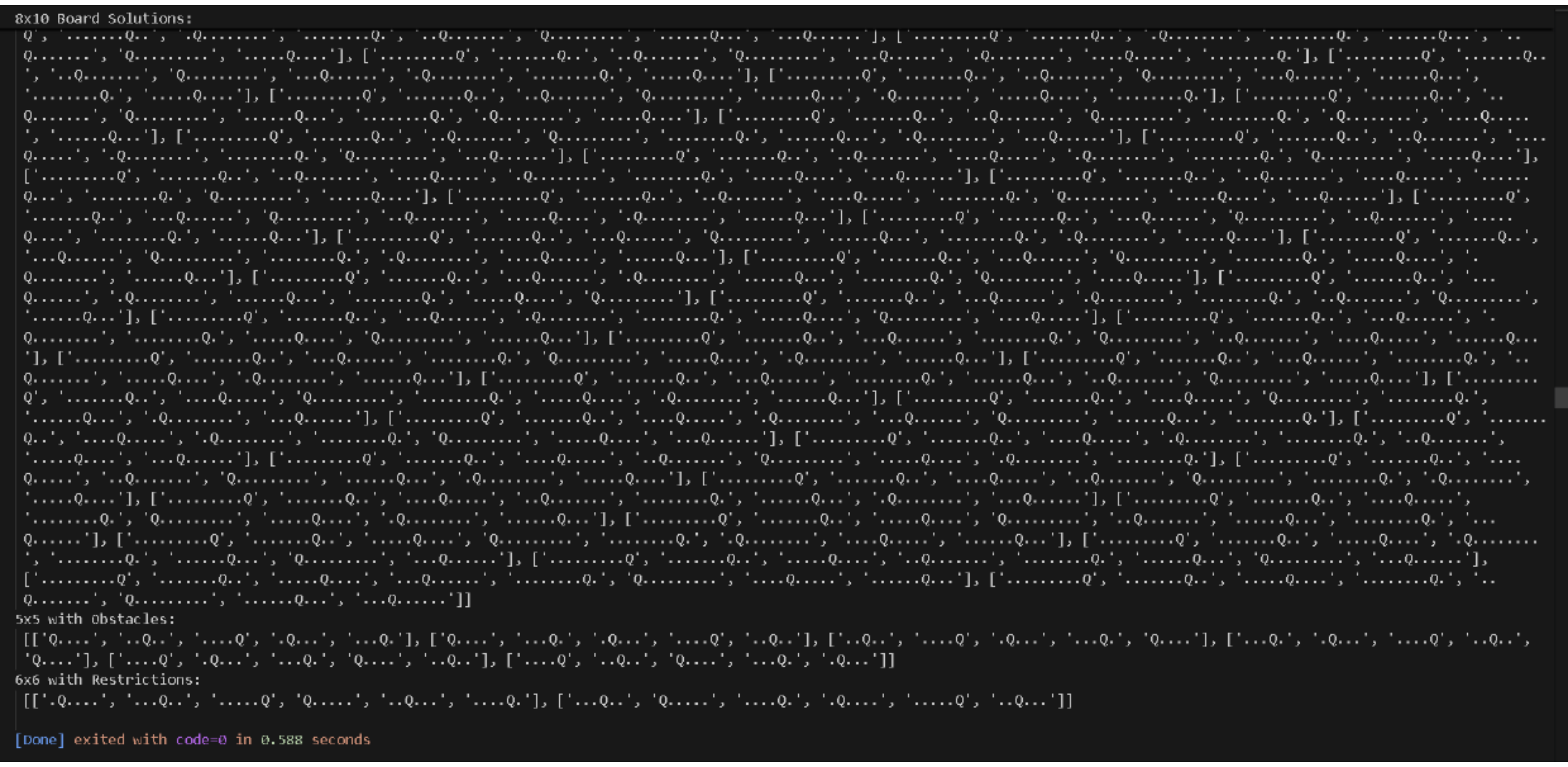
```python
def solve_n_queens(n):
    solutions = []
    board = [['.'] * n for _ in range(n)]

    def is_valid(row, col):
        for i in range(row):
            if board[i][col] == 'Q' or \
               col - (row - i) >= 0 and board[i][col - (row - i)] == 'Q' or \
               col + (row - i) < n and board[i][col + (row - i)] == 'Q':
                return False
        return True

    def place_queens(row):
        if row == n:
            solutions.append([''.join(row) for row in board])
            return
        for col in range(n):
            if is_valid(row, col):
                board[row][col] = 'Q'
                place_queens(row + 1)
                board[row][col] = '.'

    place_queens(0)
    return solutions

# Usage
print("4-Queens Solutions:\n", solve_n_queens(4))
print("5-Queens Solutions:\n", solve_n_queens(5))
print("8-Queens Solutions:\n", solve_n_queens(8))
```



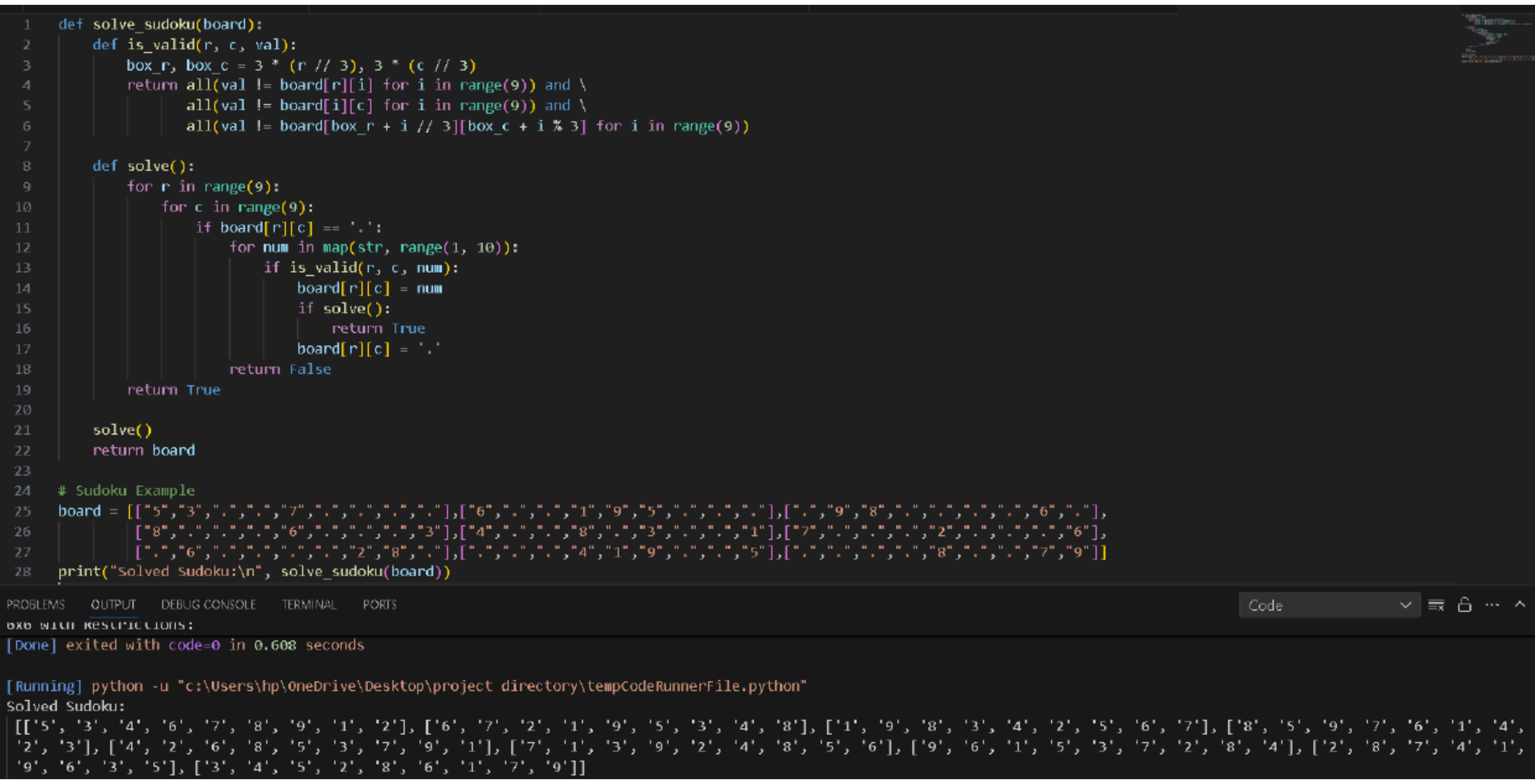
```
[Running] python -u "c:\Users\hp\OneDrive\Desktop\project directory\tempCodeRunnerFile.python"
4-Queens Solutions:
[['.Q..', '...Q', 'Q...', '..Q.'], ['..Q.', 'Q...', '...Q', '.Q..']]
5-Queens Solutions:
[['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['.Q...', '...Q.', 'Q....', '..Q..', '....Q'], ['.Q...', '....Q', '..Q..', 'Q....',
'...Q.'], ['..Q..', 'Q....', '...Q.', '.Q...', '....Q'], ['..Q..', '....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', 'Q....', '..Q..', '....Q', '.Q...'], ['...Q.', '.Q...', '....Q',
'..Q..', 'Q....'], ['....Q', '.Q...', '...Q.', 'Q....', '..Q..'], ['....Q', '..Q..', 'Q....', '...Q.', '.Q...']]
8-Queens Solutions:
[['Q.......', '....Q...', '.......Q', '.....Q..', '..Q.....', '......Q.', '.Q......', '...Q....'], ['Q.......', '.....Q..', '.......Q', '..Q.....', '......Q.', '...Q....', '.
Q......', '....Q...'], ['Q.......', '......Q.', '...Q....', '.....Q..', '.......Q', '.Q......', '....Q...', '..Q.....'], ['Q.......', '......Q.', '....Q...', '.......Q', '.Q......
', '...Q....', '.....Q..', '..Q.....'], ['.Q......', '...Q....', '.....Q..', '.......Q', '..Q.....', 'Q.......', '......Q.', '....Q...'], ['.Q......', '....Q...', '......Q.',
'Q.......', '..Q.....', '.......Q', '.....Q..', '...Q....'], ['.Q......', '....Q...', '......Q.', '...Q....', 'Q.......', '.......Q', '.....Q..', '..Q.....'], ['.Q......', '.....
Q..', 'Q.......', '......Q.', '...Q....', '.......Q', '..Q.....', '....Q...'], ['.Q......', '.....Q..', '.......Q', '..Q.....', 'Q.......', '...Q....', '......Q.', '....Q...'], ['.
Q......', '......Q.', '..Q.....', '.....Q..', '.......Q', '....Q...', 'Q.......', '...Q....'], ['.Q......', '......Q.', '....Q...', '.......Q', 'Q.......', '...Q....', '.....Q..',
'..Q.....'], ['.Q......', '.......Q', '.....Q..', 'Q.......', '..Q.....', '....Q...', '......Q.', '...Q....'], ['..Q.....', 'Q.......', '......Q.', '....Q...', '.......Q', '.
Q......', '...Q....', '.....Q..'], ['..Q.....', '....Q...', '.Q......', '.......Q', 'Q.......', '......Q.', '...Q....', '.....Q..'], ['..Q.....', '....Q...', '.Q......', '.......
Q', '.....Q..', '...Q....', '......Q.', 'Q.......'], ['..Q.....', '....Q...', '......Q.', 'Q.......', '...Q....', '.Q......', '.......Q', '.....Q..'], ['..Q.....', '....Q...',
'.......Q', '...Q....', 'Q.......', '......Q.', '.Q......', '.....Q..'], ['..Q.....', '.....Q..', '.Q......', '....Q...', '.......Q', 'Q.......', '......Q.', '...Q....'], ['..
Q.....', '.....Q..', '.Q......', '......Q.', 'Q.......', '...Q....', '.......Q', '....Q...'], ['..Q.....', '.....Q..', '.Q......', '......Q.', '....Q...', 'Q.......', '.......Q',
'...Q....'], ['..Q.....', '.....Q..', '...Q....', 'Q.......', '.......Q', '....Q...', '......Q.', '.Q......'], ['..Q.....', '.....Q..', '...Q....', '.Q......', '.......Q', '....
Q...', '......Q.', 'Q.......'], ['..Q.....', '.....Q..', '.......Q', 'Q.......', '...Q....', '......Q.', '....Q...', '.Q......'], ['..Q.....', '.....Q..', '.......Q', 'Q.......',
'....Q...', '......Q.', '.Q......', '...Q....'], ['..Q.....', '.....Q..', '.......Q', '.Q......', '...Q....', 'Q.......', '......Q.', '....Q...'], ['..Q.....', '......Q.', '.
Q......', '.......Q', '....Q...', 'Q.......', '...Q....', '.....Q..'], ['..Q.....', '......Q.', '.Q......', '.......Q', '.....Q..', '...Q....', 'Q.......', '....Q...'], ['..Q.....
', '.......Q', '...Q....', '......Q.', 'Q.......', '.....Q..', '.Q......', '....Q...'], ['...Q....', 'Q.......', '....Q...', '.......Q', '.Q......', '......Q.', '..Q.....', '.....
Q..'], ['...Q....', 'Q.......', '....Q...', '.......Q', '.....Q..', '..Q.....', '......Q.', '.Q......'], ['...Q....', '.Q......', '....Q...', '.......Q', '.....Q..', 'Q.......',
'..Q.....', '......Q.'], ['...Q....', '.Q......', '......Q.', '..Q.....', '.....Q..', '.......Q', 'Q.......', '....Q...'], ['...Q....', '.Q......', '......Q.', '..Q.....', '.....
Q..', '.......Q', '....Q...', 'Q.......'], ['...Q....', '.Q......', '......Q.', '....Q...', 'Q.......', '.......Q', '.....Q..', '..Q.....'], ['...Q....', '.Q......', '.......Q',
'....Q...', '......Q.', 'Q.......', '..Q.....', '.....Q..'], ['...Q....', '.Q......', '.......Q', '.....Q..', 'Q.......', '..Q.....', '....Q...', '......Q.'], ['...Q....', '.....
Q..', 'Q.......', '....Q...', '.Q......', '.......Q', '..Q.....', '......Q.'], ['...Q....', '.....Q..', '.......Q', '.Q......', '......Q.', 'Q.......', '..Q.....', '....Q...'],
['...Q....', '.....Q..', '.......Q', '..Q.....', 'Q.......', '......Q.', '....Q...', '.Q......'], ['...Q....', '......Q.', 'Q.......', '.......Q', '....Q...', '.Q......', '.....Q..
', '..Q.....'], ['...Q....', '......Q.', '..Q.....', '.......Q', '.Q......', '....Q...', 'Q.......', '.....Q..'], ['...Q....', '......Q.', '....Q...', '.Q......', '.....Q..',
'Q.......', '..Q.....', '.......Q'], ['...Q....', '......Q.', '....Q...', '..Q.....', 'Q.......', '.....Q..', '.......Q', '.Q......'], ['...Q....', '.......Q', 'Q.......', '..
Q.....', '.....Q..', '.Q......', '......Q.', '....Q...'], ['...Q....', '.......Q', 'Q.......', '....Q...', '......Q.', '.Q......', '.....Q..', '..Q.....'], ['...Q....', '.......
Q', '....Q...', '..Q.....', 'Q.......', '......Q.', '.Q......', '.....Q..'], ['....Q...', 'Q.......', '...Q....', '.....Q..', '.......Q', '.Q......', '......Q.', '..Q.....'],
['....Q...', 'Q.......', '.......Q', '...Q....', '.Q......', '......Q.', '..Q.....', '.....Q..'], ['....Q...', 'Q.......', '.......Q', '.....Q..', '..Q.....', '......Q.', '.Q......
', '...Q....'], ['....Q...', '.Q......', '...Q....', '.....Q..', '.......Q', '..Q.....', 'Q.......', '......Q.'], ['....Q...', '.Q......', '...Q....', '......Q.', '..Q.....',
'.......Q', '.....Q..', 'Q.......'], ['....Q...', '.Q......', '.....Q..', 'Q.......', '......Q.', '...Q....', '.......Q', '..Q.....'], ['....Q...', '.Q......', '.......Q',
```



```python
def generalized_n_queens(n, rows, cols, obstacles=set(), restrictions=set()):
    board = [['.'] * cols for _ in range(rows)]
    solutions = []

    def is_valid(row, col):
        if (row, col) in obstacles or col < 0 or col >= cols:
            return False
        for i in range(row):
            if board[i][col] == 'Q' or \
               col - (row - i) >= 0 and board[i][col - (row - i)] == 'Q' or \
               col + (row - i) < cols and board[i][col + (row - i)] == 'Q':
                return False
        return True

    def place_queens(row):
        if row == n:
            solutions.append([''.join(row) for row in board])
            return
        for col in range(cols):
            if is_valid(row, col) and (row, col) not in restrictions:
                board[row][col] = 'Q'
                place_queens(row + 1)
                board[row][col] = '.'

    place_queens(0)
    return solutions

# Examples
print("8x10 Board Solutions:\n", generalized_n_queens(8, 8, 10))
print("5x5 with Obstacles:\n", generalized_n_queens(5, 5, 5, obstacles={(2, 2), (4, 4)}))
print("6x6 with Restrictions:\n", generalized_n_queens(6, 6, 6, restrictions={(0, 2), (0, 4)}))
```

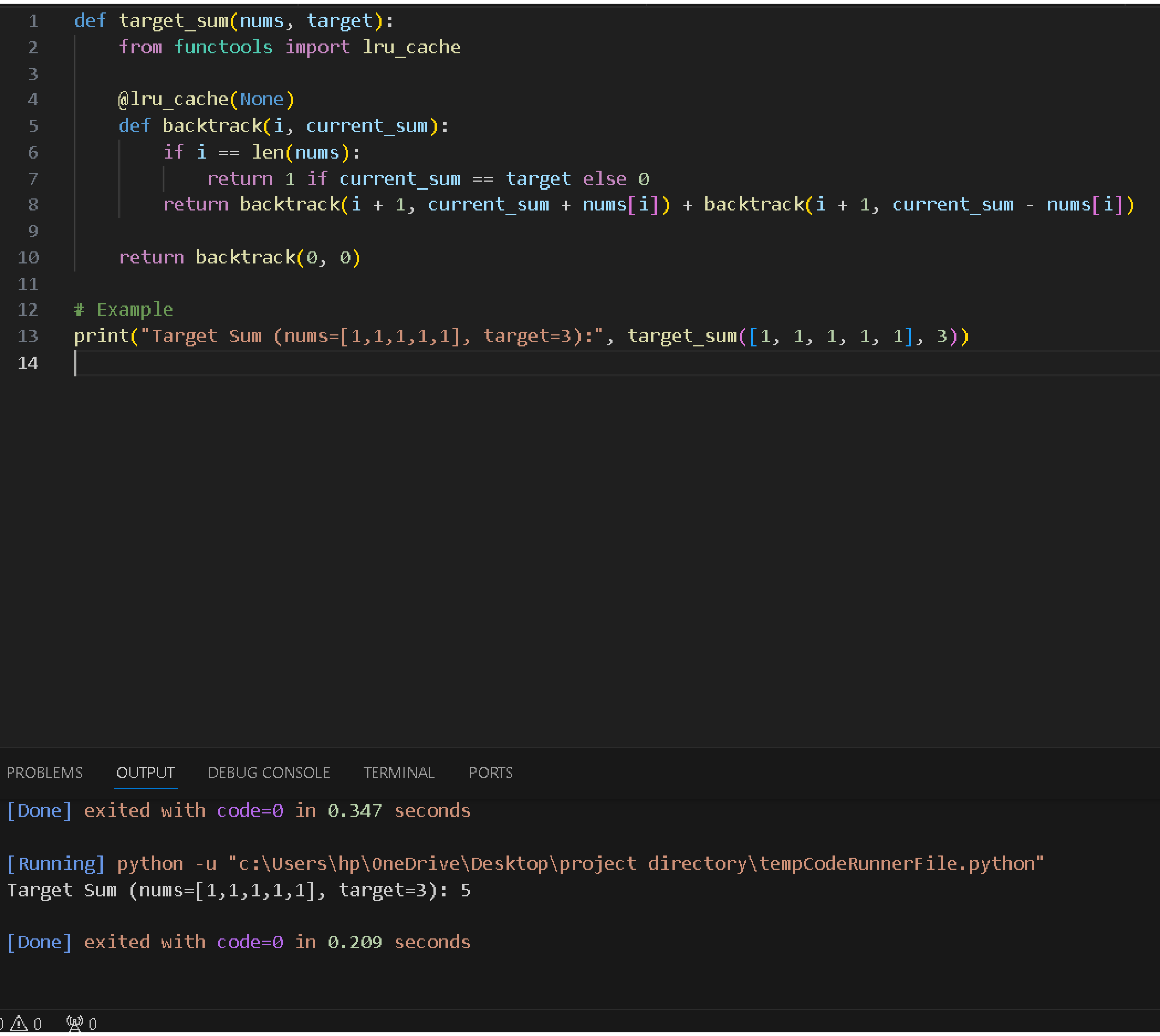


```
8x10 Board Solutions:
 Q', '.......Q..', '.Q........', '........Q.', '..Q.......', 'Q.........', '......Q...', '...Q......'], ['.........Q', '.......Q..', '.Q........', '........Q.', '......Q...', '..
 Q.......', 'Q.........', '.....Q....'], ['.........Q', '.......Q..', '..Q.......', 'Q.........', '...Q......', '.Q........', '....Q.....', '........Q.'], ['.........Q', '.......Q..
 ', '..Q.......', 'Q.........', '...Q......', '.Q........', '........Q.', '.....Q....'], ['.........Q', '.......Q..', '..Q.......', 'Q.........', '...Q......', '......Q...',
 '........Q.', '.....Q....'], ['.........Q', '.......Q..', '..Q.......', 'Q.........', '......Q...', '.Q........', '.....Q....', '........Q.'], ['.........Q', '.......Q..', '..
 Q.......', 'Q.........', '......Q...', '........Q.', '.Q........', '.....Q....'], ['.........Q', '.......Q..', '..Q.......', 'Q.........', '........Q.', '.Q........', '....Q.....
 ', '......Q...'], ['.........Q', '.......Q..', '..Q.......', 'Q.........', '........Q.', '......Q...', '.Q........', '...Q......'], ['.........Q', '.......Q..', '..Q.......', '....
 Q.....', '.Q........', '........Q.', 'Q.........', '...Q......'], ['.........Q', '.......Q..', '..Q.......', '....Q.....', '.Q........', '........Q.', 'Q.........', '.....Q....'],
 ['.........Q', '.......Q..', '..Q.......', '....Q.....', '.Q........', '........Q.', '.....Q....', '...Q......'], ['.........Q', '.......Q..', '..Q.......', '....Q.....', '......
 Q...', '........Q.', 'Q.........', '.....Q....'], ['.........Q', '.......Q..', '..Q.......', '....Q.....', '........Q.', 'Q.........', '.....Q....', '...Q......'], ['.........Q',
 '.......Q..', '...Q......', 'Q.........', '..Q.......', '.....Q....', '.Q........', '......Q...'], ['.........Q', '.......Q..', '...Q......', 'Q.........', '..Q.......', '.....
 Q....', '........Q.', '......Q...'], ['.........Q', '.......Q..', '...Q......', 'Q.........', '......Q...', '........Q.', '.Q........', '.....Q....'], ['.........Q', '.......Q..',
 '...Q......', 'Q.........', '........Q.', '.Q........', '....Q.....', '......Q...'], ['.........Q', '.......Q..', '...Q......', 'Q.........', '........Q.', '.....Q....', '.
 Q........', '......Q...'], ['.........Q', '.......Q..', '...Q......', '.Q........', '......Q...', '........Q.', 'Q.........', '....Q.....'], ['.........Q', '.......Q..', '...
 Q......', '.Q........', '......Q...', '........Q.', '.....Q....', 'Q.........'], ['.........Q', '.......Q..', '...Q......', '.Q........', '........Q.', '..Q.......', 'Q.........',
 '......Q...'], ['.........Q', '.......Q..', '...Q......', '.Q........', '........Q.', '.....Q....', 'Q.........', '....Q.....'], ['.........Q', '.......Q..', '...Q......', '.
 Q........', '........Q.', '.....Q....', 'Q.........', '......Q...'], ['.........Q', '.......Q..', '...Q......', '........Q.', 'Q.........', '..Q.......', '....Q.....', '......Q...
 '], ['.........Q', '.......Q..', '...Q......', '........Q.', 'Q.........', '.....Q....', '.Q........', '......Q...'], ['.........Q', '.......Q..', '...Q......', '........Q.', '..
 Q.......', '.....Q....', '.Q........', '......Q...'], ['.........Q', '.......Q..', '...Q......', '........Q.', '......Q...', '..Q.......', 'Q.........', '.....Q....'], ['.........
 Q', '.......Q..', '....Q.....', 'Q.........', '........Q.', '.....Q....', '.Q........', '......Q...'], ['.........Q', '.......Q..', '....Q.....', 'Q.........', '........Q.',
 '......Q...', '.Q........', '...Q......'], ['.........Q', '.......Q..', '....Q.....', '.Q........', '...Q......', 'Q.........', '......Q...', '........Q.'], ['.........Q', '.......
 Q..', '....Q.....', '.Q........', '........Q.', 'Q.........', '.....Q....', '...Q......'], ['.........Q', '.......Q..', '....Q.....', '.Q........', '........Q.', '..Q.......',
 '.....Q....', '...Q......'], ['.........Q', '.......Q..', '....Q.....', '..Q.......', 'Q.........', '.....Q....', '.Q........', '........Q.'], ['.........Q', '.......Q..', '....
 Q.....', '..Q.......', 'Q.........', '......Q...', '.Q........', '.....Q....'], ['.........Q', '.......Q..', '....Q.....', '..Q.......', 'Q.........', '........Q.', '.Q........',
 '.....Q....'], ['.........Q', '.......Q..', '....Q.....', '..Q.......', '........Q.', '......Q...', '.Q........', '...Q......'], ['.........Q', '.......Q..', '....Q.....',
 '........Q.', 'Q.........', '.....Q....', '.Q........', '......Q...'], ['.........Q', '.......Q..', '.....Q....', 'Q.........', '..Q.......', '......Q...', '........Q.', '...
 Q......'], ['.........Q', '.......Q..', '.....Q....', 'Q.........', '........Q.', '.Q........', '....Q.....', '......Q...'], ['.........Q', '.......Q..', '.....Q....', '.Q........
 ', '........Q.', '......Q...', 'Q.........', '...Q......'], ['.........Q', '.......Q..', '.....Q....', '..Q.......', '........Q.', '......Q...', 'Q.........', '...Q......'],
 ['.........Q', '.......Q..', '.....Q....', '...Q......', '........Q.', 'Q.........', '....Q.....', '......Q...'], ['.........Q', '.......Q..', '.....Q....', '........Q.', '..
 Q.......', 'Q.........', '......Q...', '...Q......']]
5x5 with Obstacles:
 [['Q....', '..Q..', '....Q', '.Q...', '...Q.'], ['Q....', '...Q.', '.Q...', '....Q', '..Q..'], ['..Q..', '....Q', '.Q...', '...Q.', 'Q....'], ['...Q.', '.Q...', '....Q', '..Q..',
 'Q....'], ['....Q', '.Q...', '...Q.', 'Q....', '..Q..'], ['....Q', '..Q..', 'Q....', '...Q.', '.Q...']]
6x6 with Restrictions:
 [['.Q....', '...Q..', '.....Q', 'Q.....', '..Q...', '....Q.'], ['...Q..', 'Q.....', '....Q.', '.Q....', '.....Q', '..Q...']]

[Done] exited with code=0 in 0.588 seconds
```
]



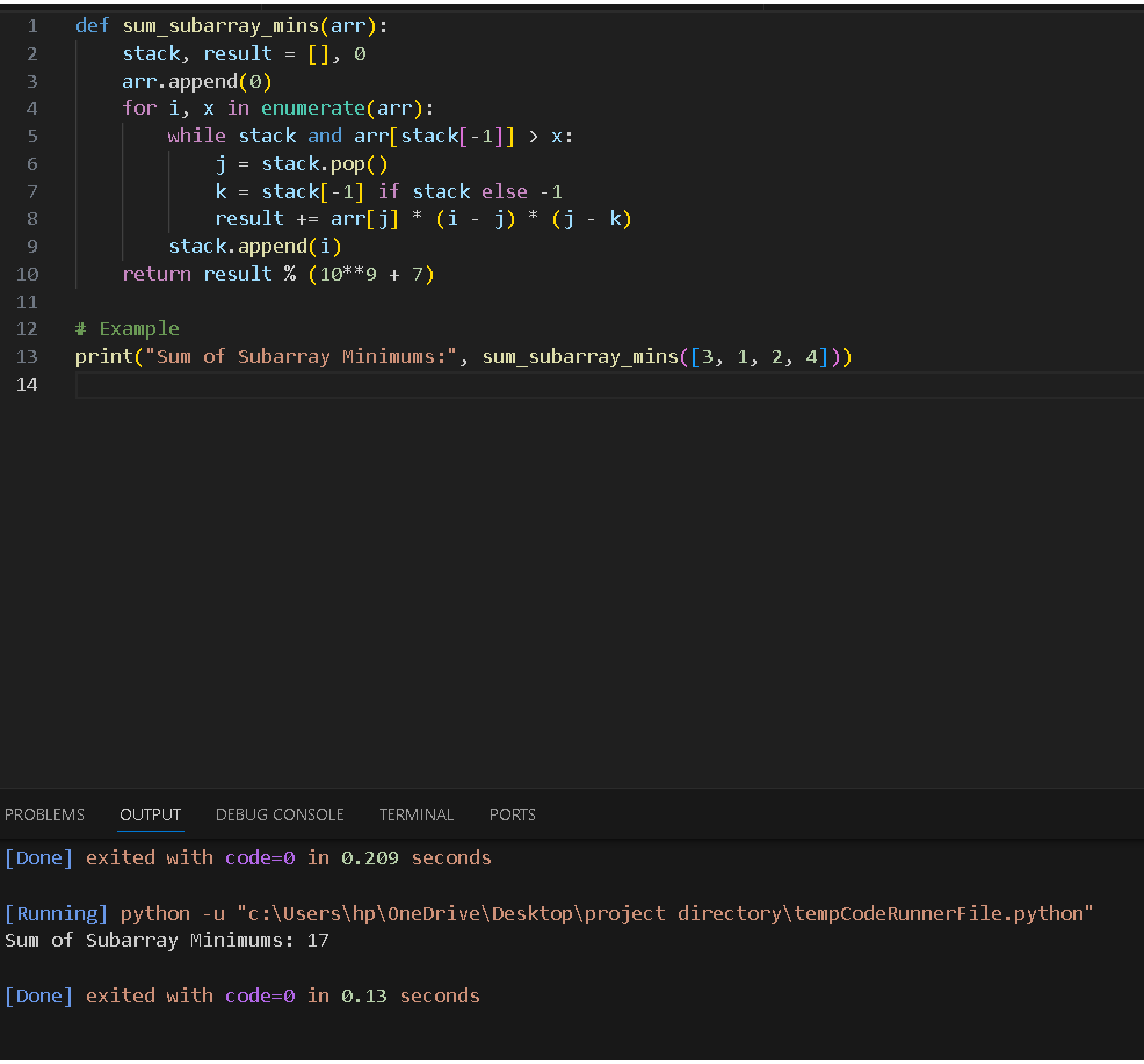
```python
def solve_sudoku(board):
    def is_valid(r, c, val):
        box_r, box_c = 3 * (r // 3), 3 * (c // 3)
        return all(val != board[r][i] for i in range(9)) and \
               all(val != board[i][c] for i in range(9)) and \
               all(val != board[box_r + i // 3][box_c + i % 3] for i in range(9))

    def solve():
        for r in range(9):
            for c in range(9):
                if board[r][c] == '.':
                    for num in map(str, range(1, 10)):
                        if is_valid(r, c, num):
                            board[r][c] = num
                            if solve():
                                return True
                            board[r][c] = '.'
                    return False
        return True

    solve()
    return board

# Sudoku Example
board = [["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"]]
print("Solved Sudoku:\n", solve_sudoku(board))
```

```
6x6 with Restrictions:
[Done] exited with code=0 in 0.608 seconds

[Running] python -u "c:\Users\hp\OneDrive\Desktop\project directory\tempCodeRunnerFile.python"
Solved Sudoku:
 [['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']]
```



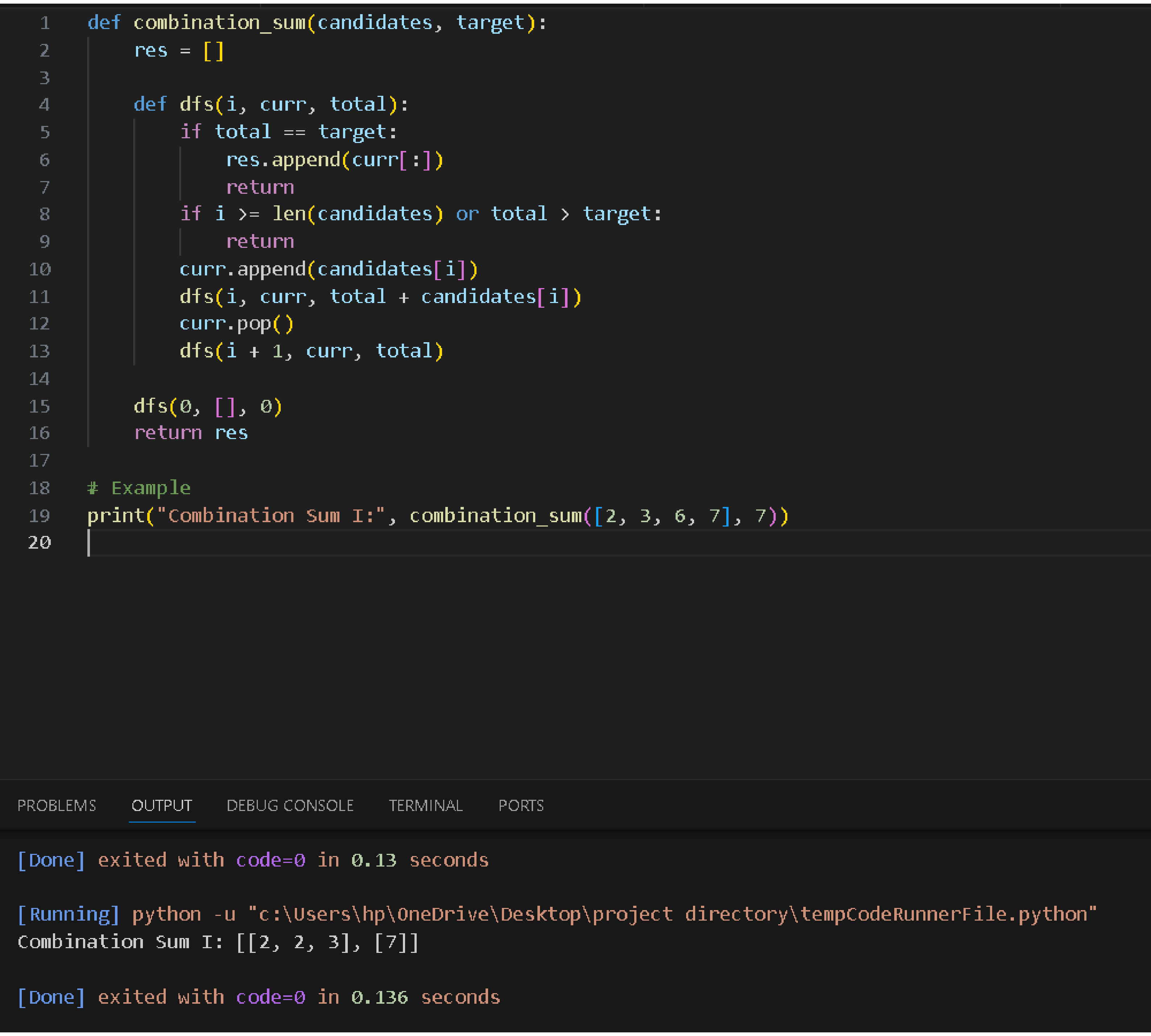
```python
def target_sum(nums, target):
    from functools import lru_cache

    @lru_cache(None)
    def backtrack(i, current_sum):
        if i == len(nums):
            return 1 if current_sum == target else 0
        return backtrack(i + 1, current_sum + nums[i]) + backtrack(i + 1, current_sum - nums[i])

    return backtrack(0, 0)

# Example
print("Target Sum (nums=[1,1,1,1,1], target=3):", target_sum([1, 1, 1, 1, 1], 3))

```

```
[Done] exited with code=0 in 0.347 seconds

[Running] python -u "c:\Users\hp\OneDrive\Desktop\project directory\tempCodeRunnerFile.python"
Target Sum (nums=[1,1,1,1,1], target=3): 5

[Done] exited with code=0 in 0.209 seconds
```



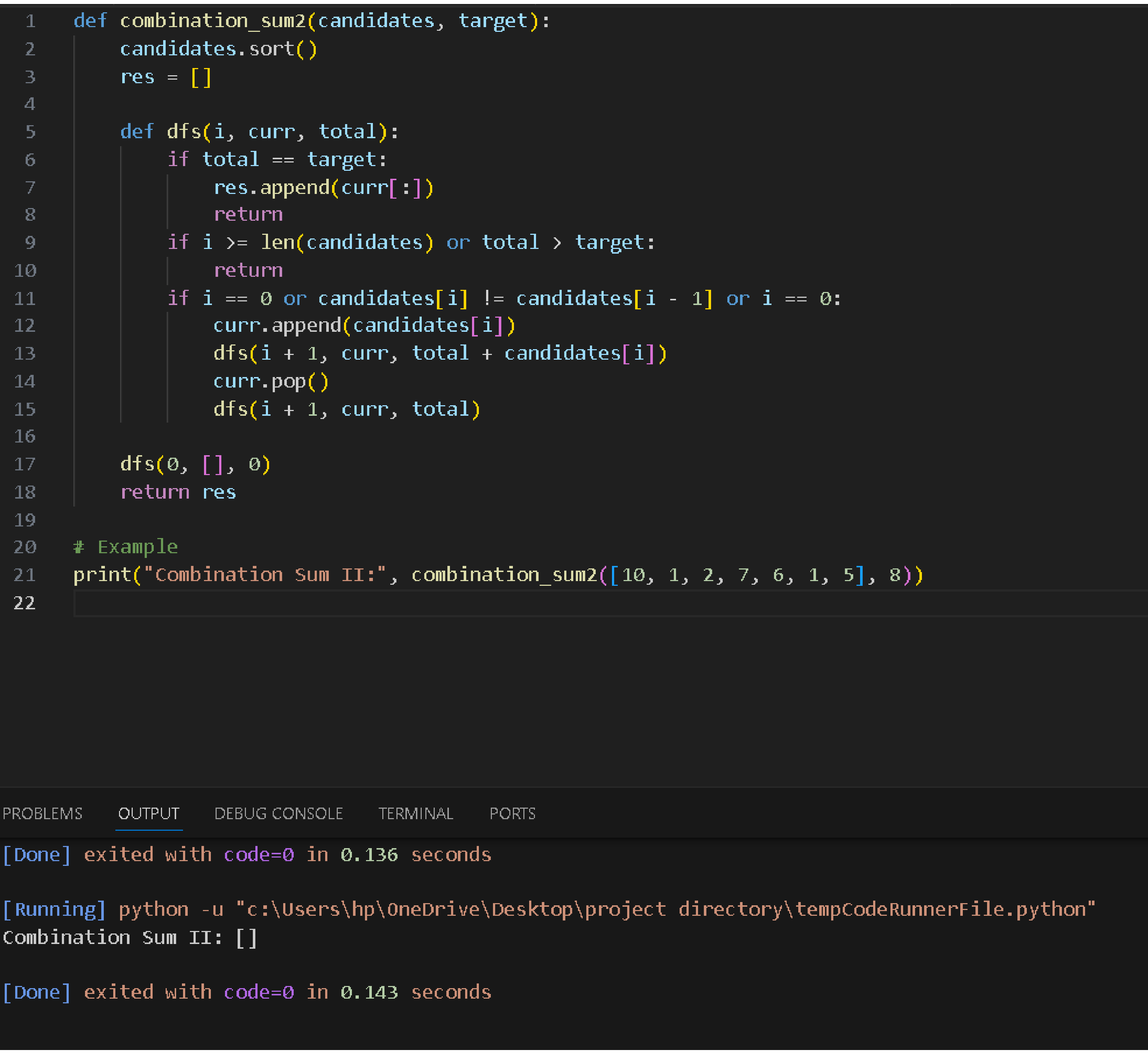
```python
def sum_subarray_mins(arr):
    stack, result = [], 0
    arr.append(0)
    for i, x in enumerate(arr):
        while stack and arr[stack[-1]] > x:
            j = stack.pop()
            k = stack[-1] if stack else -1
            result += arr[j] * (i - j) * (j - k)
        stack.append(i)
    return result % (10**9 + 7)

# Example
print("Sum of Subarray Minimums:", sum_subarray_mins([3, 1, 2, 4]))
```

```
[Done] exited with code=0 in 0.209 seconds

[Running] python -u "c:\Users\hp\OneDrive\Desktop\project directory\tempCodeRunnerFile.python"
Sum of Subarray Minimums: 17

[Done] exited with code=0 in 0.13 seconds
```



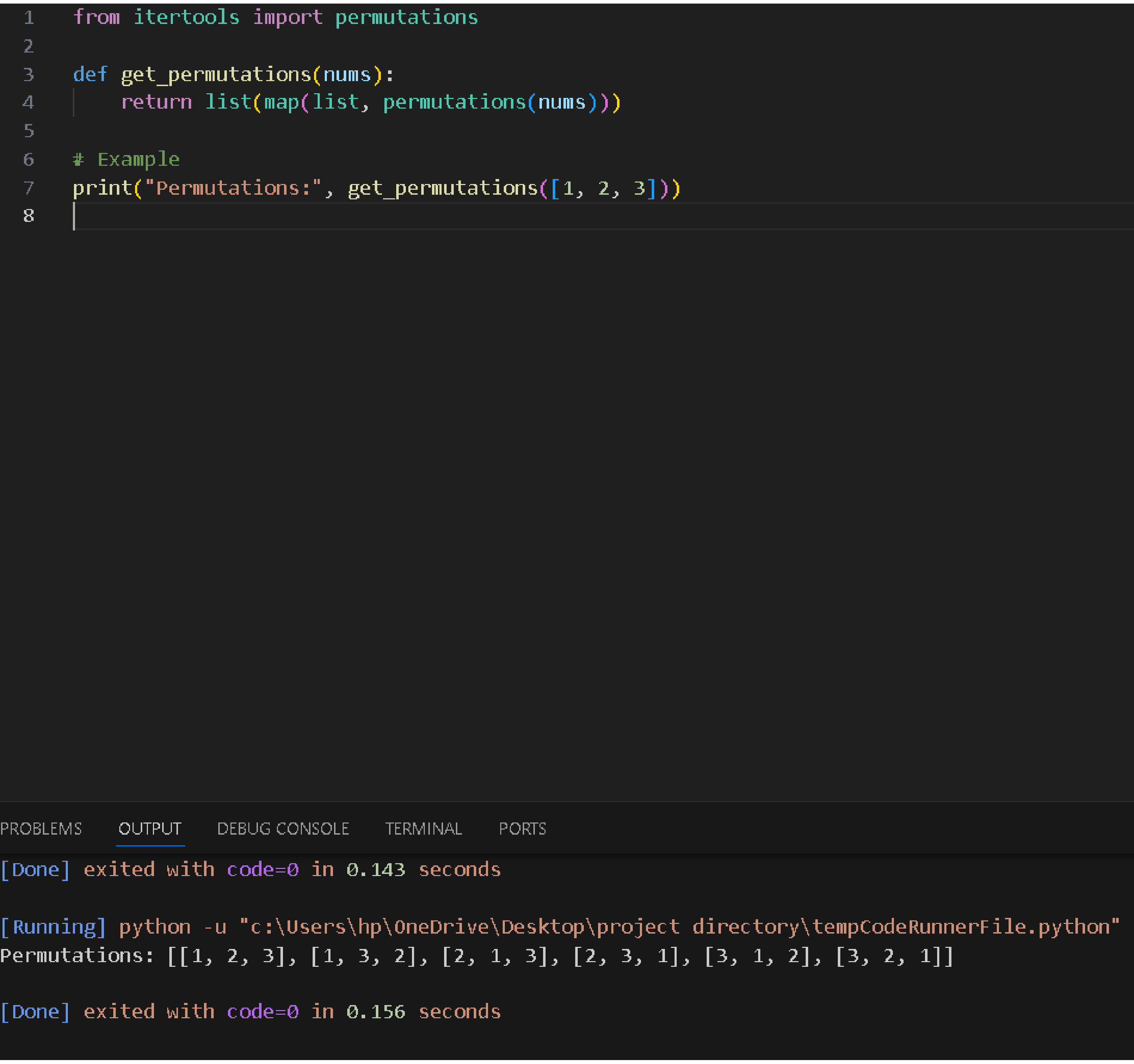
```python
def combination_sum(candidates, target):
    res = []

    def dfs(i, curr, total):
        if total == target:
            res.append(curr[:])
            return
        if i >= len(candidates) or total > target:
            return
        curr.append(candidates[i])
        dfs(i, curr, total + candidates[i])
        curr.pop()
        dfs(i + 1, curr, total)

    dfs(0, [], 0)
    return res

# Example
print("Combination Sum I:", combination_sum([2, 3, 6, 7], 7))
```

```
[Done] exited with code=0 in 0.13 seconds

[Running] python -u "c:\Users\hp\OneDrive\Desktop\project directory\tempCodeRunnerFile.python"
Combination Sum I: [[2, 2, 3], [7]]

[Done] exited with code=0 in 0.136 seconds
```



```python
def combination_sum2(candidates, target):
    candidates.sort()
    res = []

    def dfs(i, curr, total):
        if total == target:
            res.append(curr[:])
            return
        if i >= len(candidates) or total > target:
            return
        if i == 0 or candidates[i] != candidates[i - 1] or i == 0:
            curr.append(candidates[i])
            dfs(i + 1, curr, total + candidates[i])
            curr.pop()
            dfs(i + 1, curr, total)

    dfs(0, [], 0)
    return res

# Example
print("Combination Sum II:", combination_sum2([10, 1, 2, 7, 6, 1, 5], 8))
```

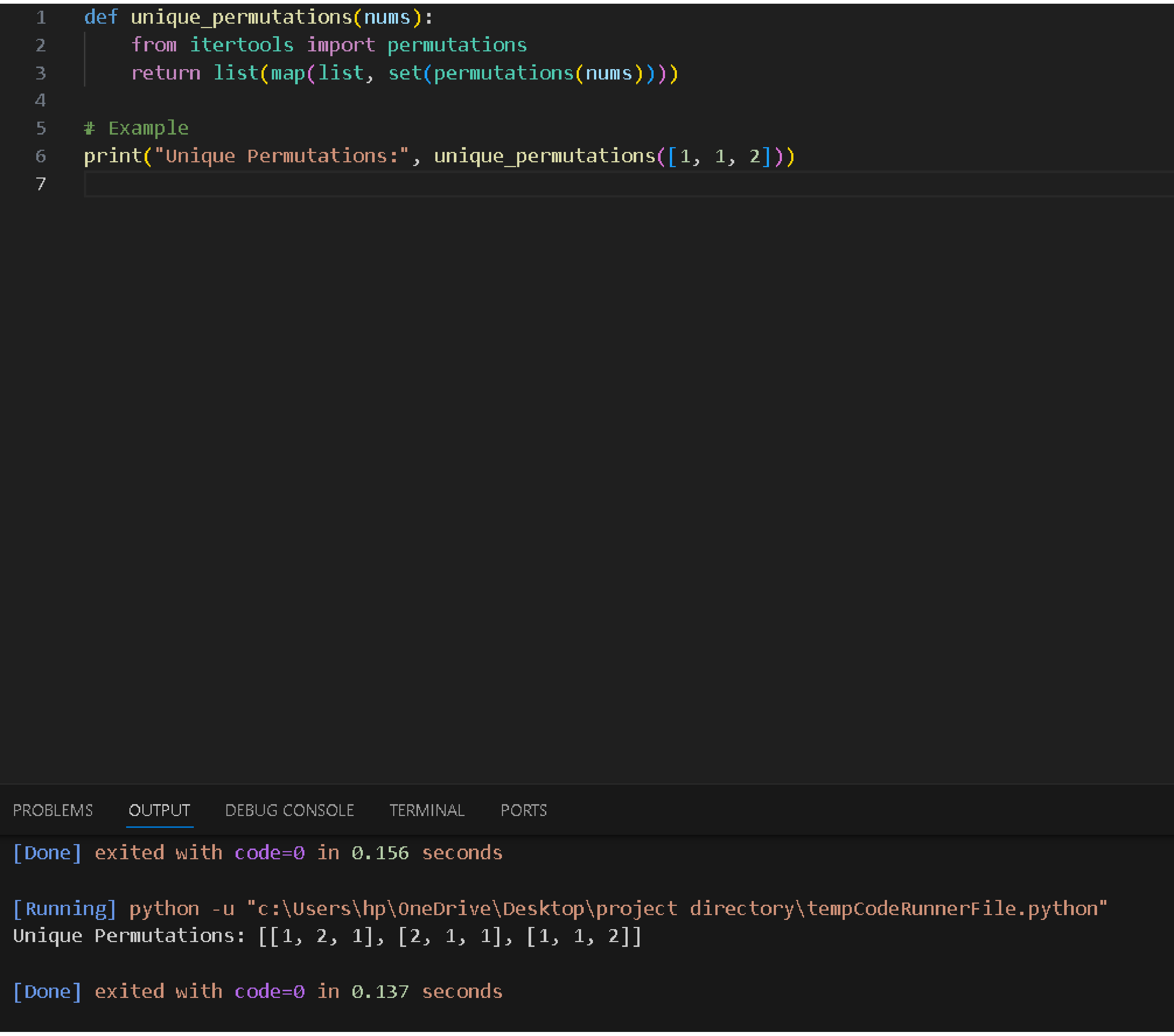
```
[Done] exited with code=0 in 0.136 seconds

[Running] python -u "c:\Users\hp\OneDrive\Desktop\project directory\tempCodeRunnerFile.python"
Combination Sum II: []

[Done] exited with code=0 in 0.143 seconds
```



```python
from itertools import permutations

def get_permutations(nums):
    return list(map(list, permutations(nums)))

# Example
print("Permutations:", get_permutations([1, 2, 3]))

```

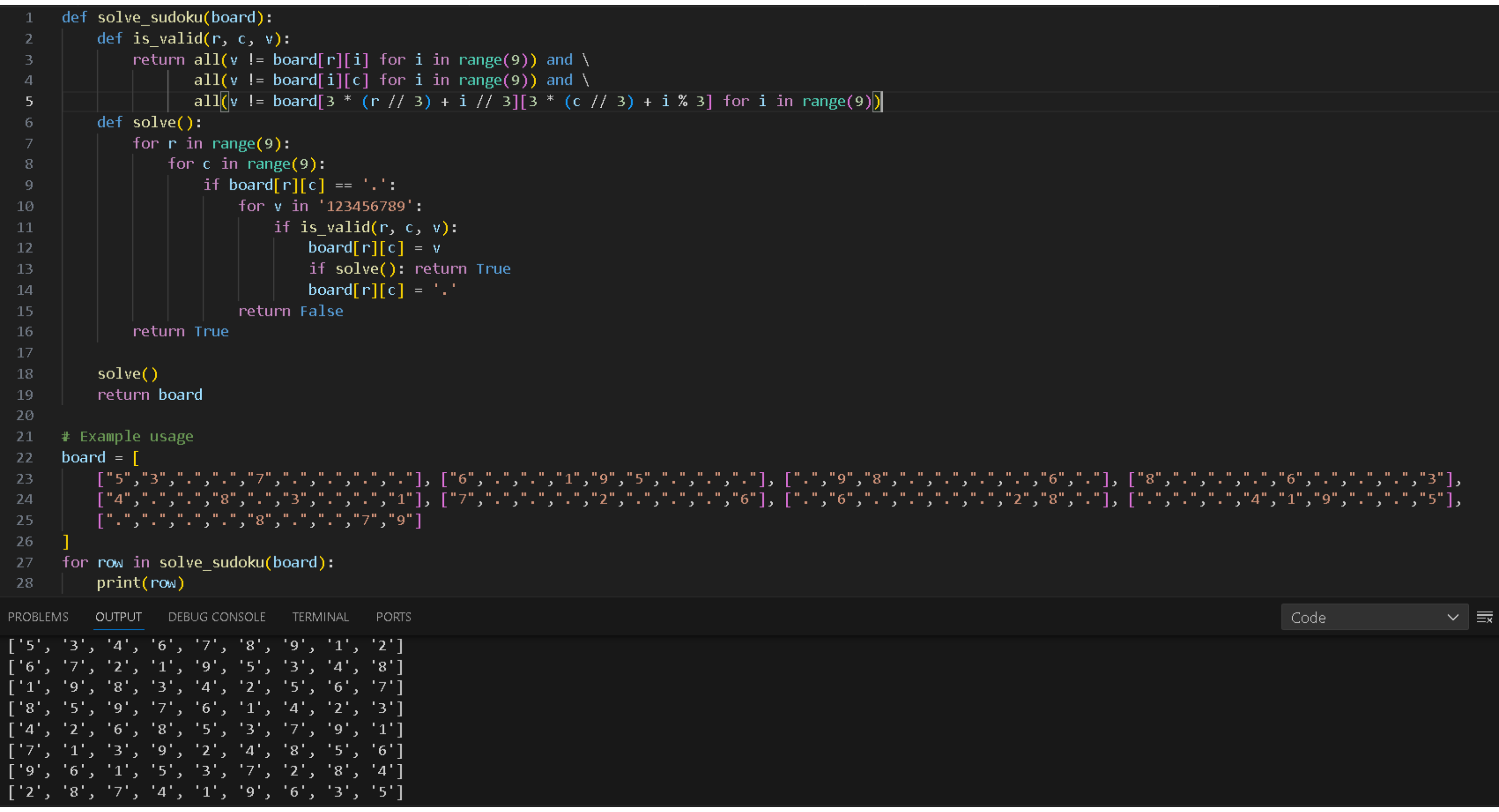
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Done] exited with code=0 in 0.143 seconds

[Running] python -u "c:\Users\hp\OneDrive\Desktop\project directory\tempCodeRunnerFile.python"
Permutations: [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]

[Done] exited with code=0 in 0.156 seconds
```



```python
def unique_permutations(nums):
    from itertools import permutations
    return list(map(list, set(permutations(nums))))

# Example
print("Unique Permutations:", unique_permutations([1, 1, 2]))

```

```
[Done] exited with code=0 in 0.156 seconds

[Running] python -u "c:\Users\hp\OneDrive\Desktop\project directory\tempCodeRunnerFile.python"
Unique Permutations: [[1, 2, 1], [2, 1, 1], [1, 1, 2]]

[Done] exited with code=0 in 0.137 seconds
```



```python
def solve_sudoku(board):
    def is_valid(r, c, v):
        return all(v != board[r][i] for i in range(9)) and \
               all(v != board[i][c] for i in range(9)) and \
               all(v != board[3 * (r // 3) + i // 3][3 * (c // 3) + i % 3] for i in range(9))
    def solve():
        for r in range(9):
            for c in range(9):
                if board[r][c] == '.':
                    for v in '123456789':
                        if is_valid(r, c, v):
                            board[r][c] = v
                            if solve(): return True
                            board[r][c] = '.'
                    return False
        return True

    solve()
    return board

# Example usage
board = [
    ["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"]
]
for row in solve_sudoku(board):
    print(row)
```

```
['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']
```