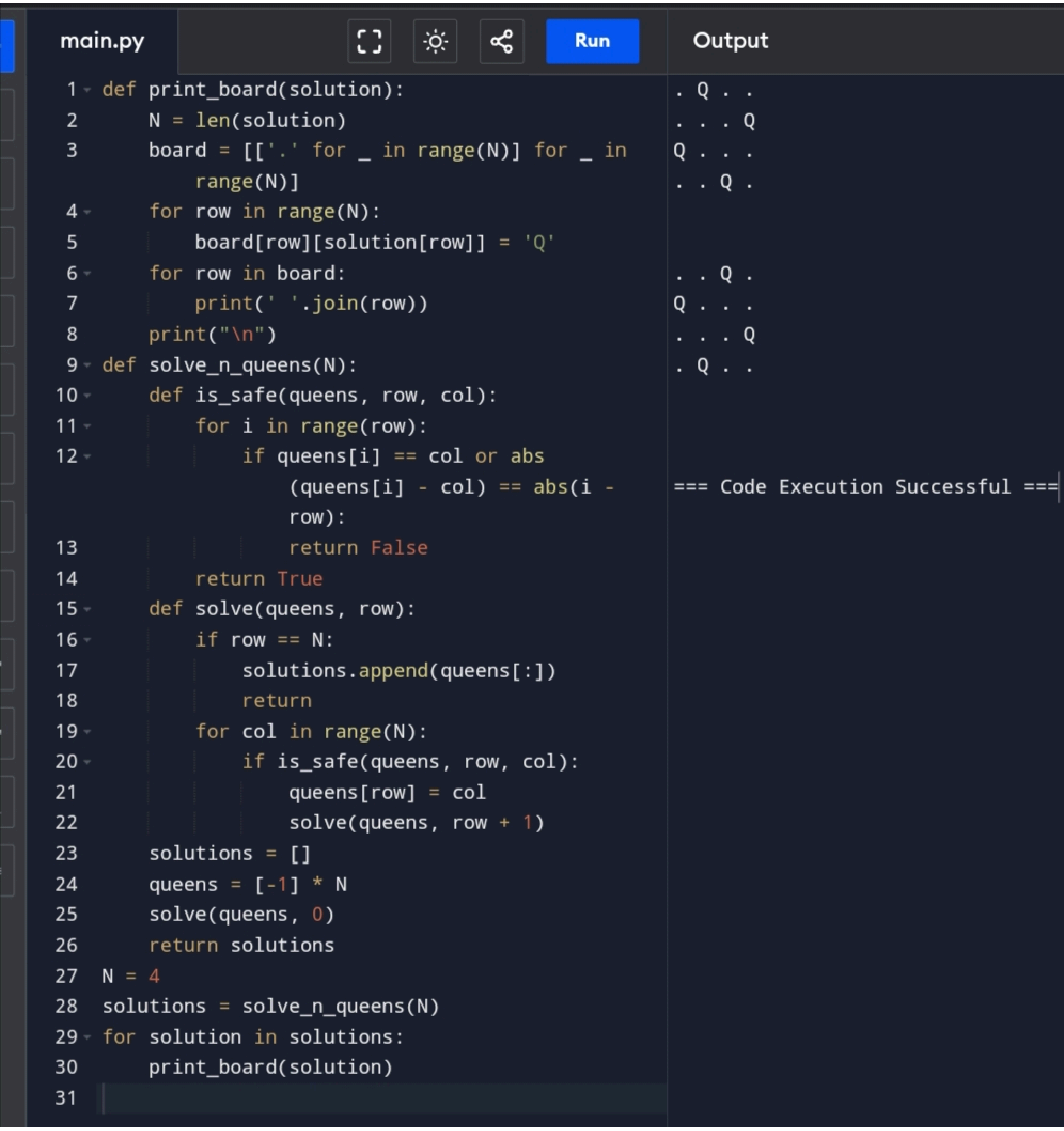
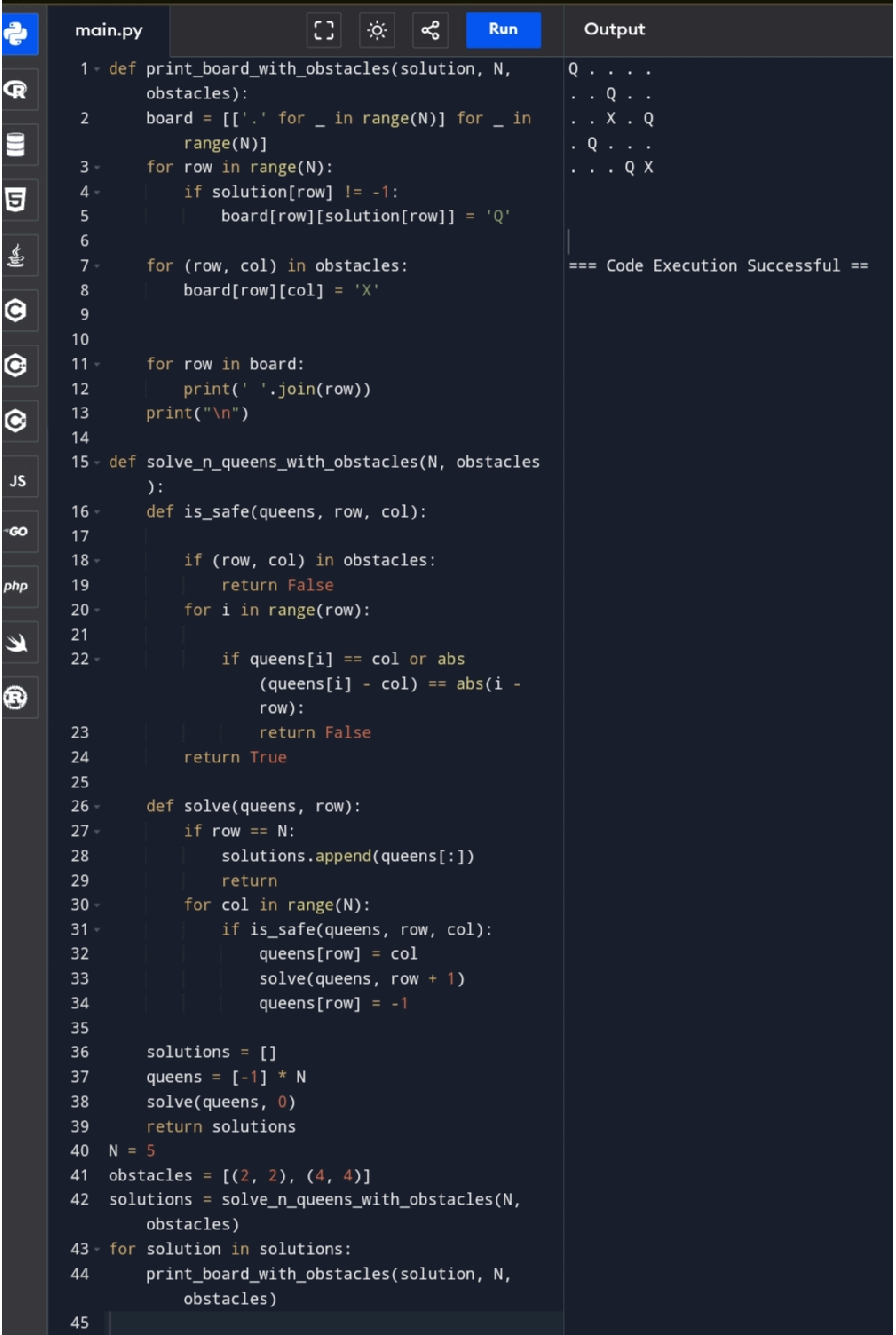


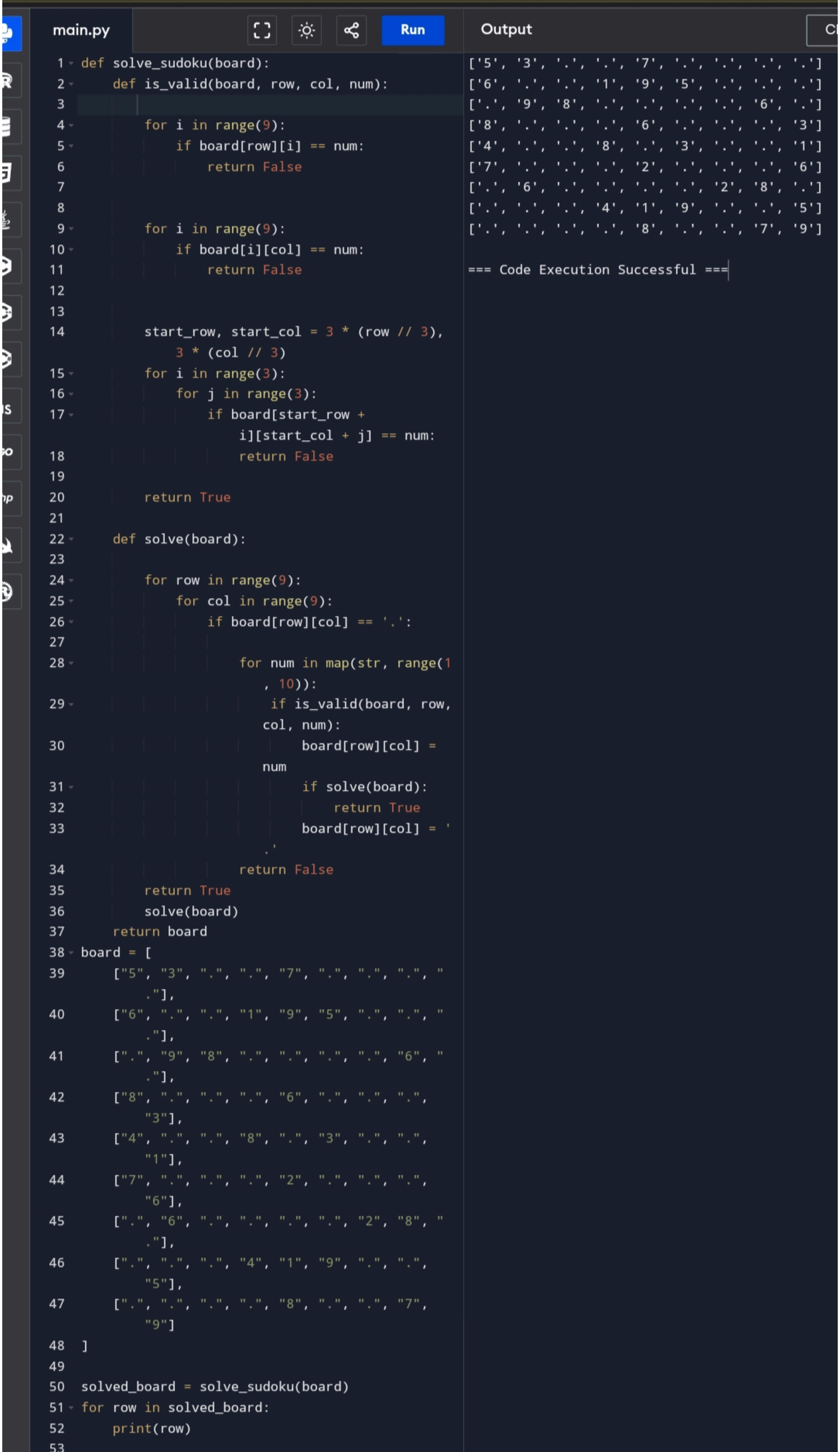
```python
def print_board(solution):
    N = len(solution)
    board = [['.' for _ in range(N)] for _ in range(N)]
    for row in range(N):
        board[row][solution[row]] = 'Q'
    for row in board:
        print(' '.join(row))
    print("\n")
def solve_n_queens(N):
    def is_safe(queens, row, col):
        for i in range(row):
            if queens[i] == col or abs(queens[i] - col) == abs(i - row):
                return False
        return True
    def solve(queens, row):
        if row == N:
            solutions.append(queens[:])
            return
        for col in range(N):
            if is_safe(queens, row, col):
                queens[row] = col
                solve(queens, row + 1)
    solutions = []
    queens = [-1] * N
    solve(queens, 0)
    return solutions
N = 4
solutions = solve_n_queens(N)
for solution in solutions:
    print_board(solution)
```

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



. . Q .
Q . . .
. . . Q
. Q . .




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



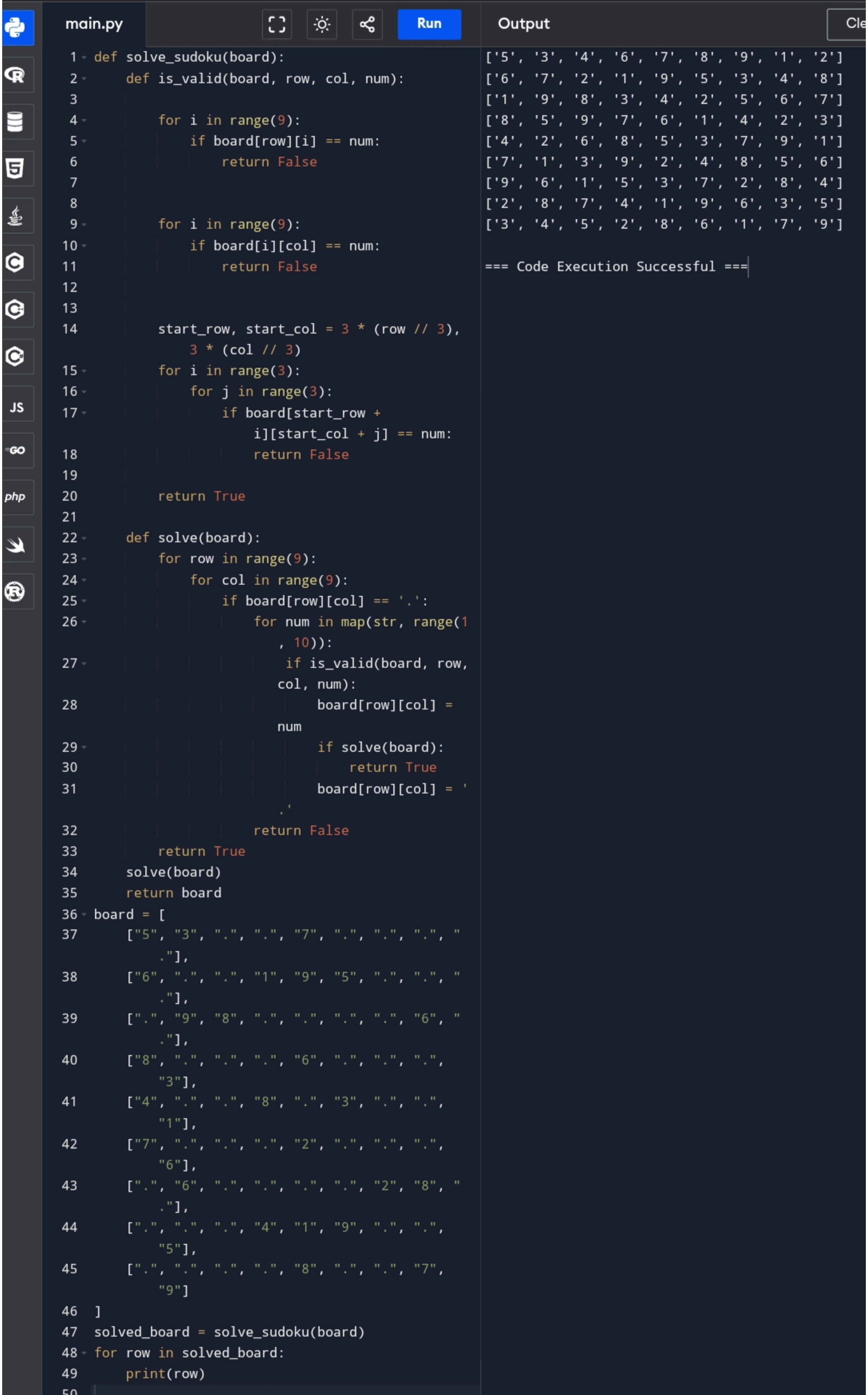
```python
def print_board_with_obstacles(solution, N, obstacles):
    board = [['.' for _ in range(N)] for _ in range(N)]
    for row in range(N):
        if solution[row] != -1:
            board[row][solution[row]] = 'Q'

    for (row, col) in obstacles:
        board[row][col] = 'X'


    for row in board:
        print(' '.join(row))
    print("\n")

def solve_n_queens_with_obstacles(N, obstacles):
    def is_safe(queens, row, col):

        if (row, col) in obstacles:
            return False
        for i in range(row):

            if queens[i] == col or abs(queens[i] - col) == abs(i - row):
                return False
        return True

    def solve(queens, row):
        if row == N:
            solutions.append(queens[:])
            return
        for col in range(N):
            if is_safe(queens, row, col):
                queens[row] = col
                solve(queens, row + 1)
                queens[row] = -1

    solutions = []
    queens = [-1] * N
    solve(queens, 0)
    return solutions
N = 5
obstacles = [(2, 2), (4, 4)]
solutions = solve_n_queens_with_obstacles(N, obstacles)
for solution in solutions:
    print_board_with_obstacles(solution, N, obstacles)
```

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



=== Code Execution Successful ==
```



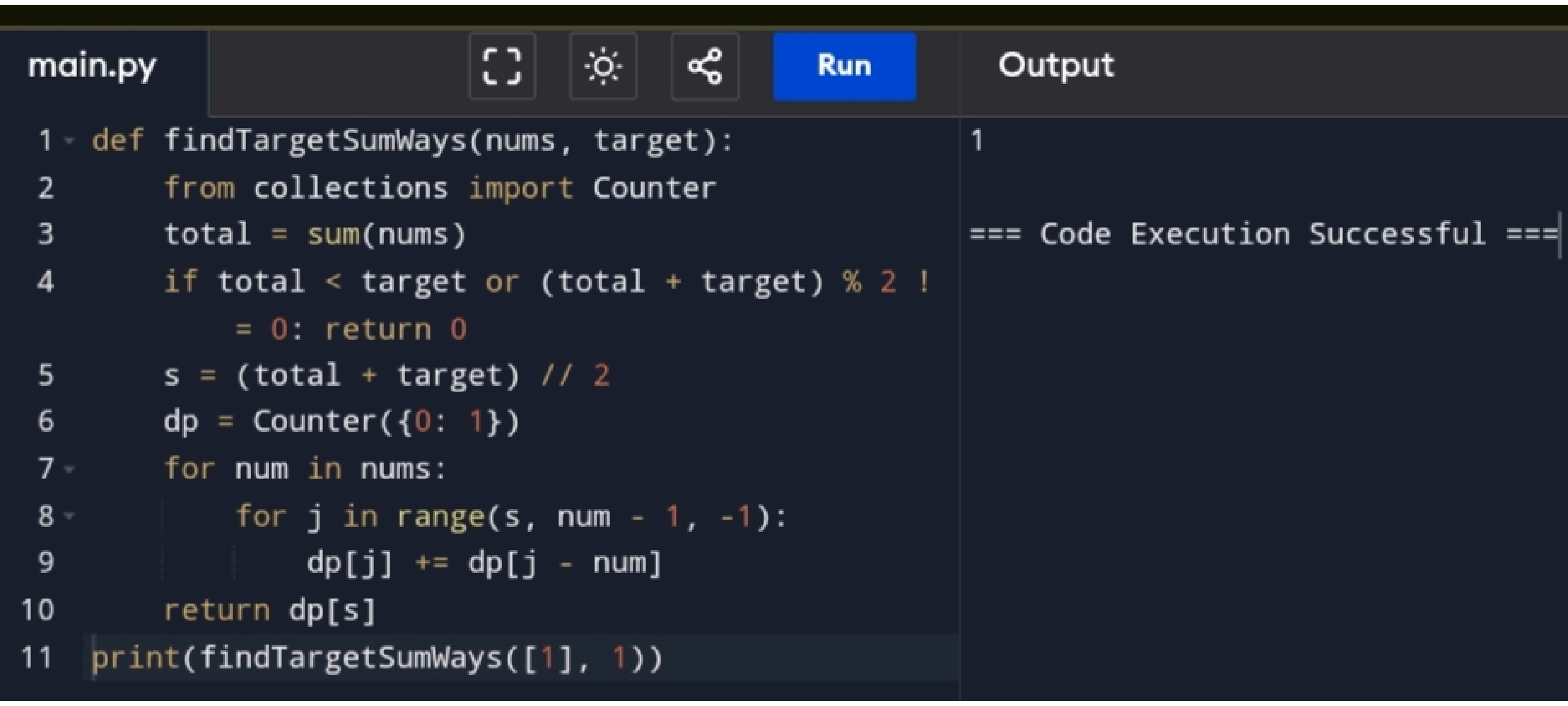
```python
def solve_sudoku(board):
    def is_valid(board, row, col, num):

        for i in range(9):
            if board[row][i] == num:
                return False


        for i in range(9):
            if board[i][col] == num:
                return False


        start_row, start_col = 3 * (row // 3), 3 * (col // 3)
        for i in range(3):
            for j in range(3):
                if board[start_row + i][start_col + j] == num:
                    return False

        return True

    def solve(board):

        for row in range(9):
            for col in range(9):
                if board[row][col] == '.':

                    for num in map(str, range(1, 10)):
                        if is_valid(board, row, col, num):
                            board[row][col] = num
                            if solve(board):
                                return True
                            board[row][col] = '.'
                    return False
        return True
        solve(board)
    return board
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"]
]

solved_board = solve_sudoku(board)
for row in solved_board:
    print(row)
```

```
['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 ===
```



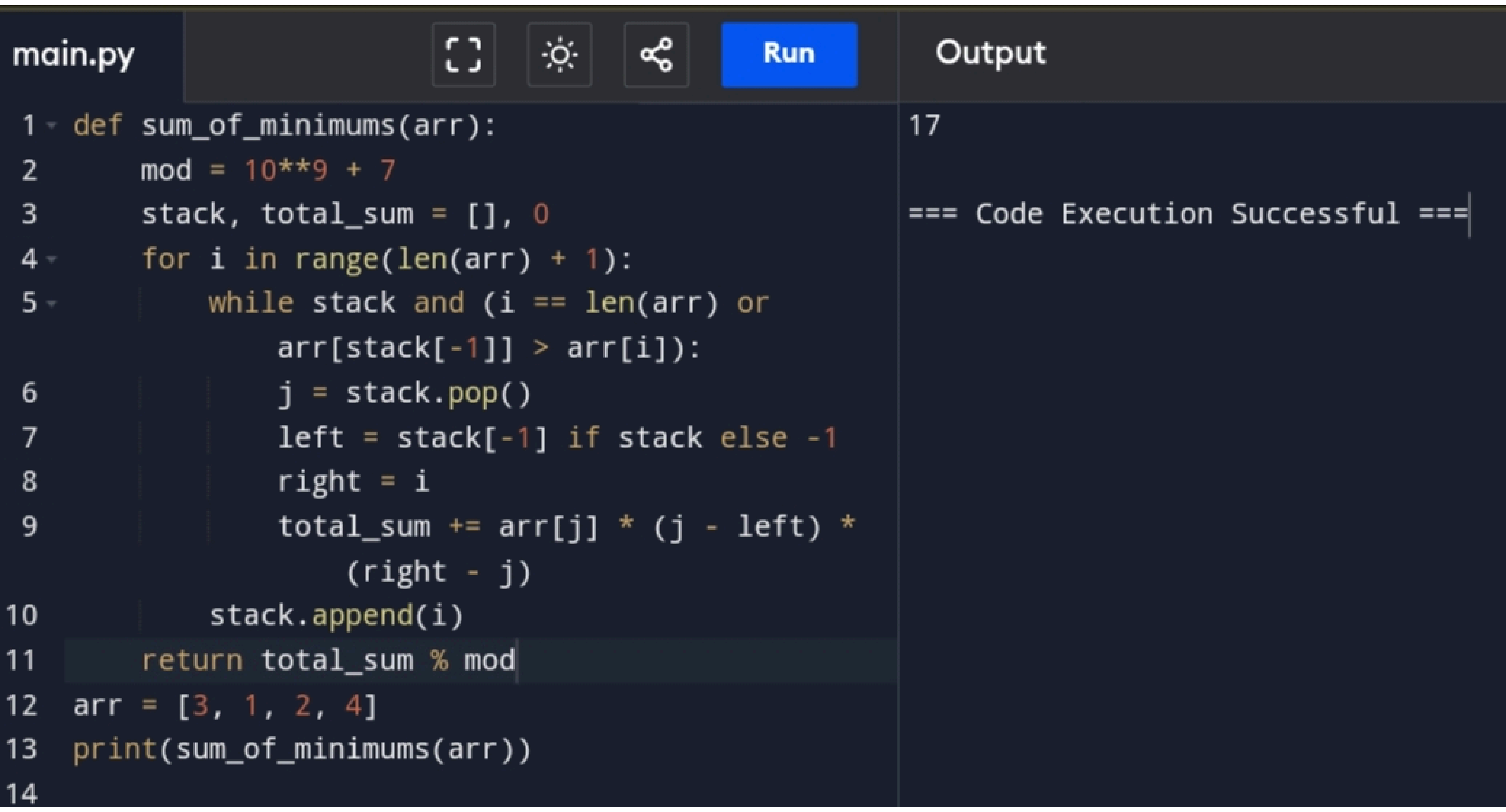
```python
def solve_sudoku(board):
    def is_valid(board, row, col, num):

        for i in range(9):
            if board[row][i] == num:
                return False


        for i in range(9):
            if board[i][col] == num:
                return False


        start_row, start_col = 3 * (row // 3), 3 * (col // 3)
        for i in range(3):
            for j in range(3):
                if board[start_row + i][start_col + j] == num:
                    return False

        return True

    def solve(board):
        for row in range(9):
            for col in range(9):
                if board[row][col] == '.':
                    for num in map(str, range(1, 10)):
                        if is_valid(board, row, col, num):
                            board[row][col] = num
                            if solve(board):
                                return True
                            board[row][col] = '.'
                    return False
        return True
    solve(board)
    return board
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"]
]
solved_board = solve_sudoku(board)
for row in solved_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']
['3', '4', '5', '2', '8', '6', '1', '7', '9']

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



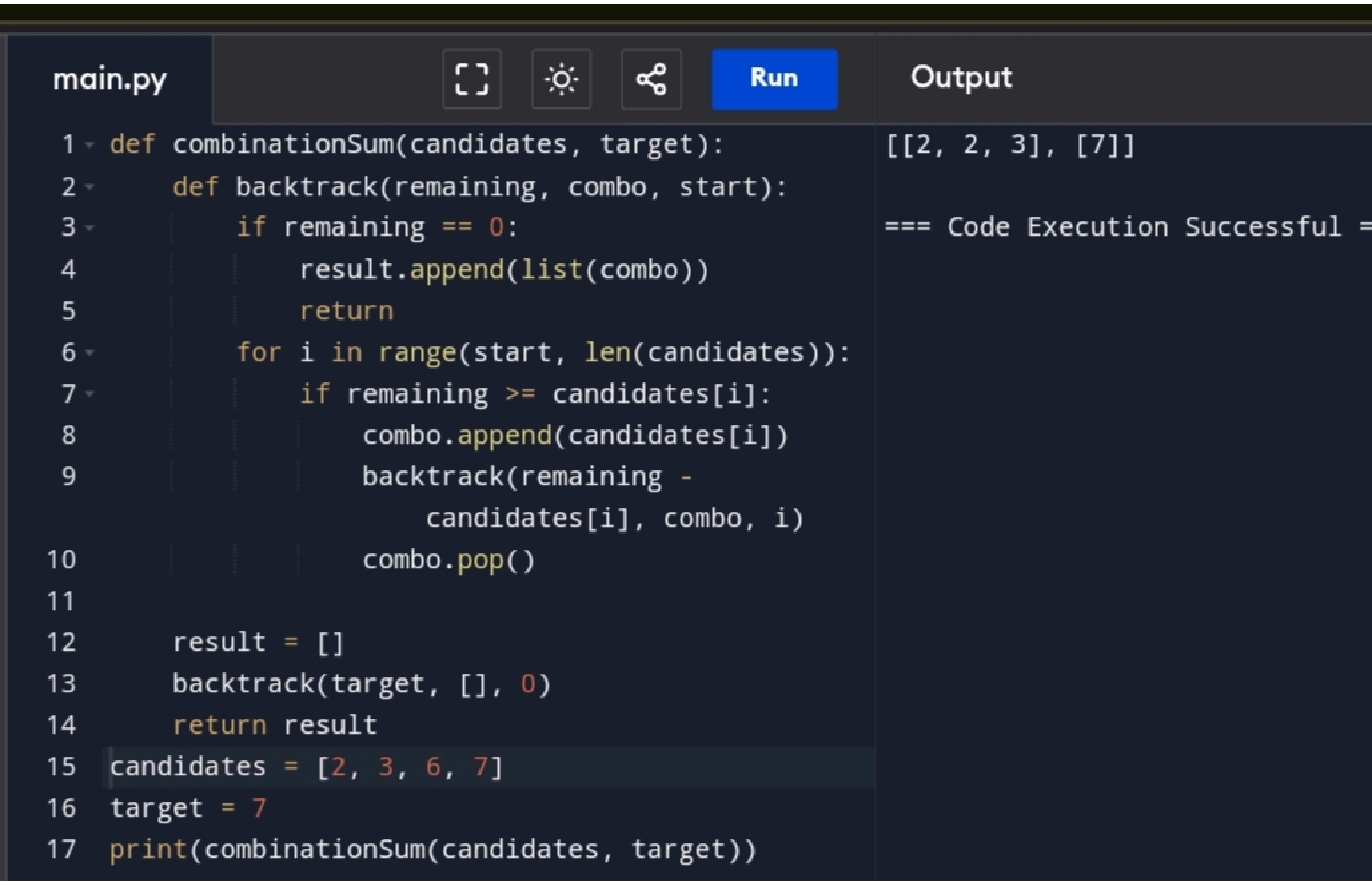
```python
def findTargetSumWays(nums, target):
    from collections import Counter
    total = sum(nums)
    if total < target or (total + target) % 2 != 0: return 0
    s = (total + target) // 2
    dp = Counter({0: 1})
    for num in nums:
        for j in range(s, num - 1, -1):
            dp[j] +=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

```
1


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



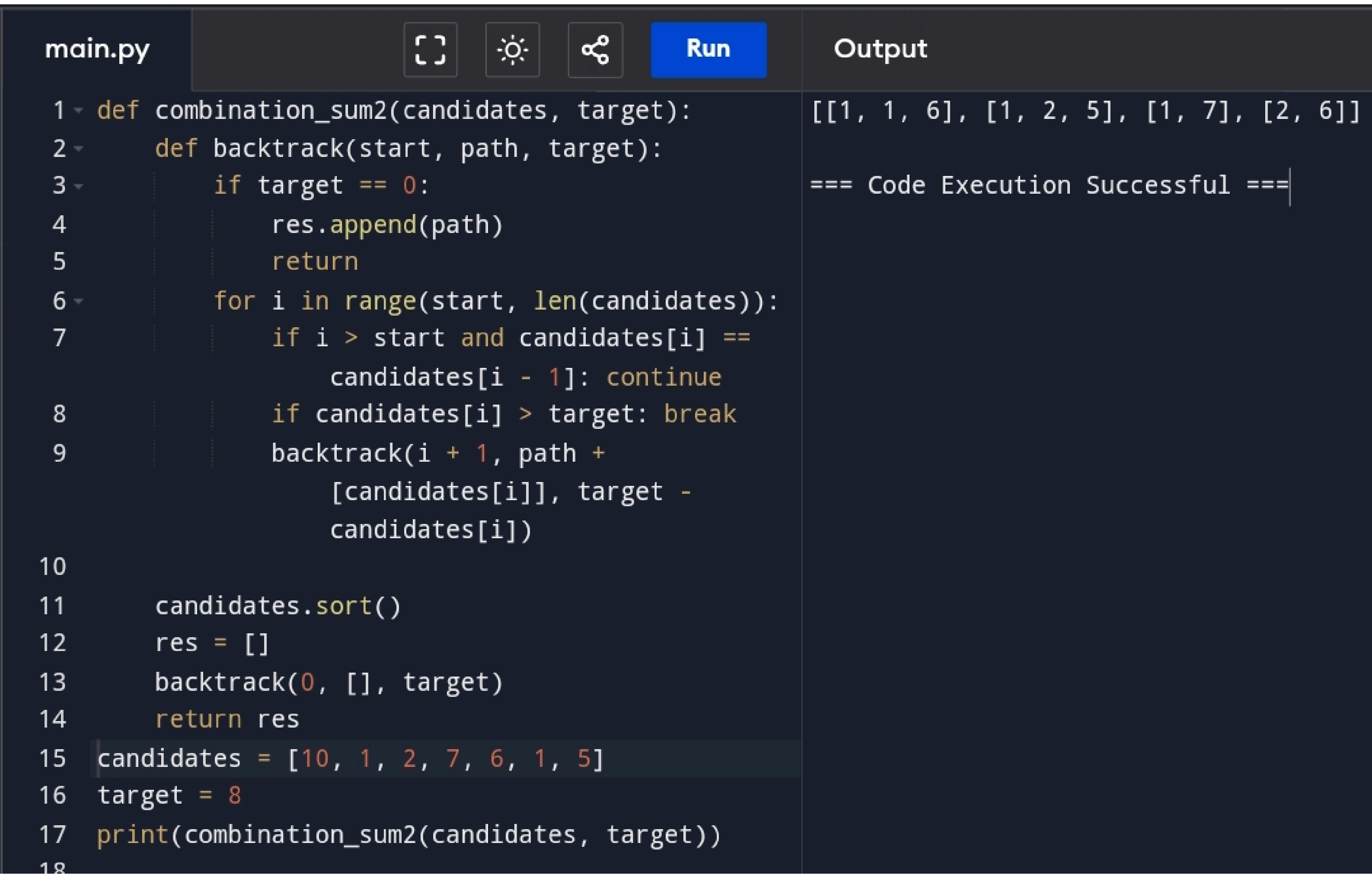
```python
def sum_of_minimums(arr):
    mod = 10**9 + 7
    stack, total_sum = [], 0
    for i in range(len(arr) + 1):
        while stack and (i == len(arr) or
            arr[stack[-1]] > arr[i]):
            j = stack.pop()
            left = stack[-1] if stack else -1
            right = i
            total_sum += arr[j] * (j - left) *
                (right - j)
        stack.append(i)
    return total_sum % mod
arr = [3, 1, 2, 4]
print(sum_of_minimums(arr))
```

```
17

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



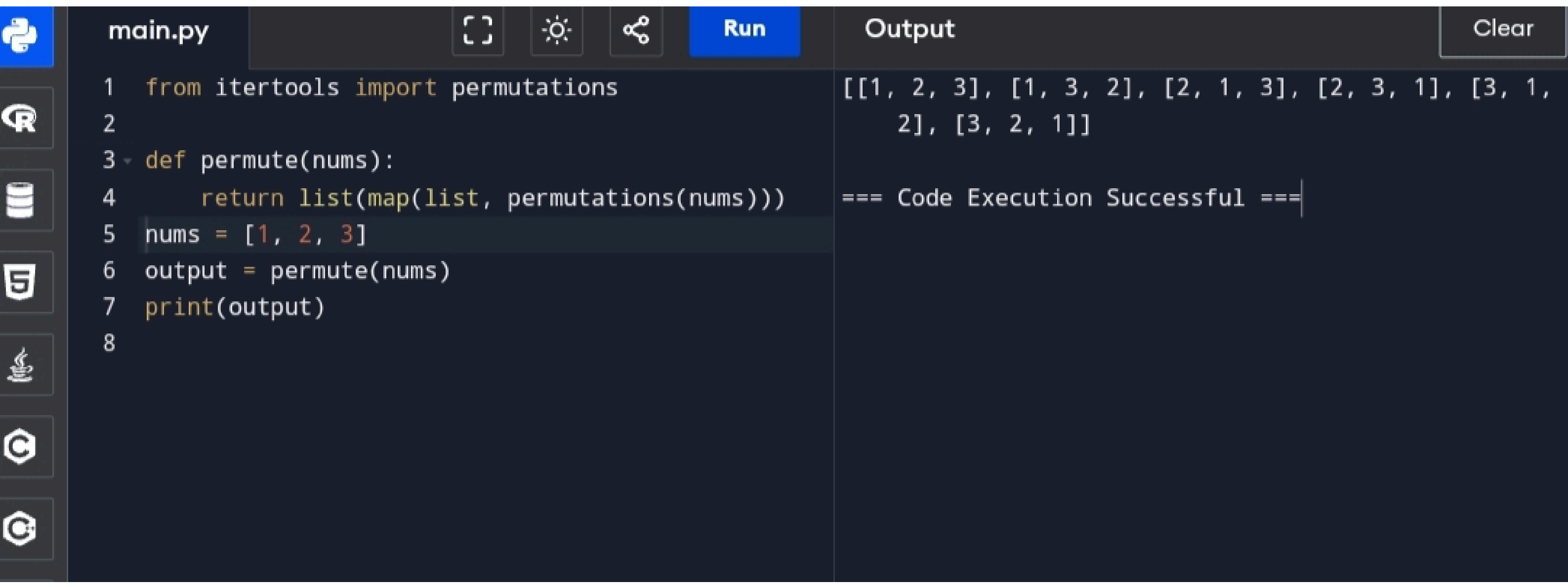
```python
def combinationSum(candidates, target):
    def backtrack(remaining, combo, start):
        if remaining == 0:
            result.append(list(combo))
            return
        for i in range(start, len(candidates)):
            if remaining >= candidates[i]:
                combo.append(candidates[i])
                backtrack(remaining -
                    candidates[i], combo, i)
                combo.pop()

    result = []
    backtrack(target, [], 0)
    return result
candidates = [2, 3, 6, 7]
target = 7
print(combinationSum(candidates, target))
```

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

=== Code Execution Successful =
```



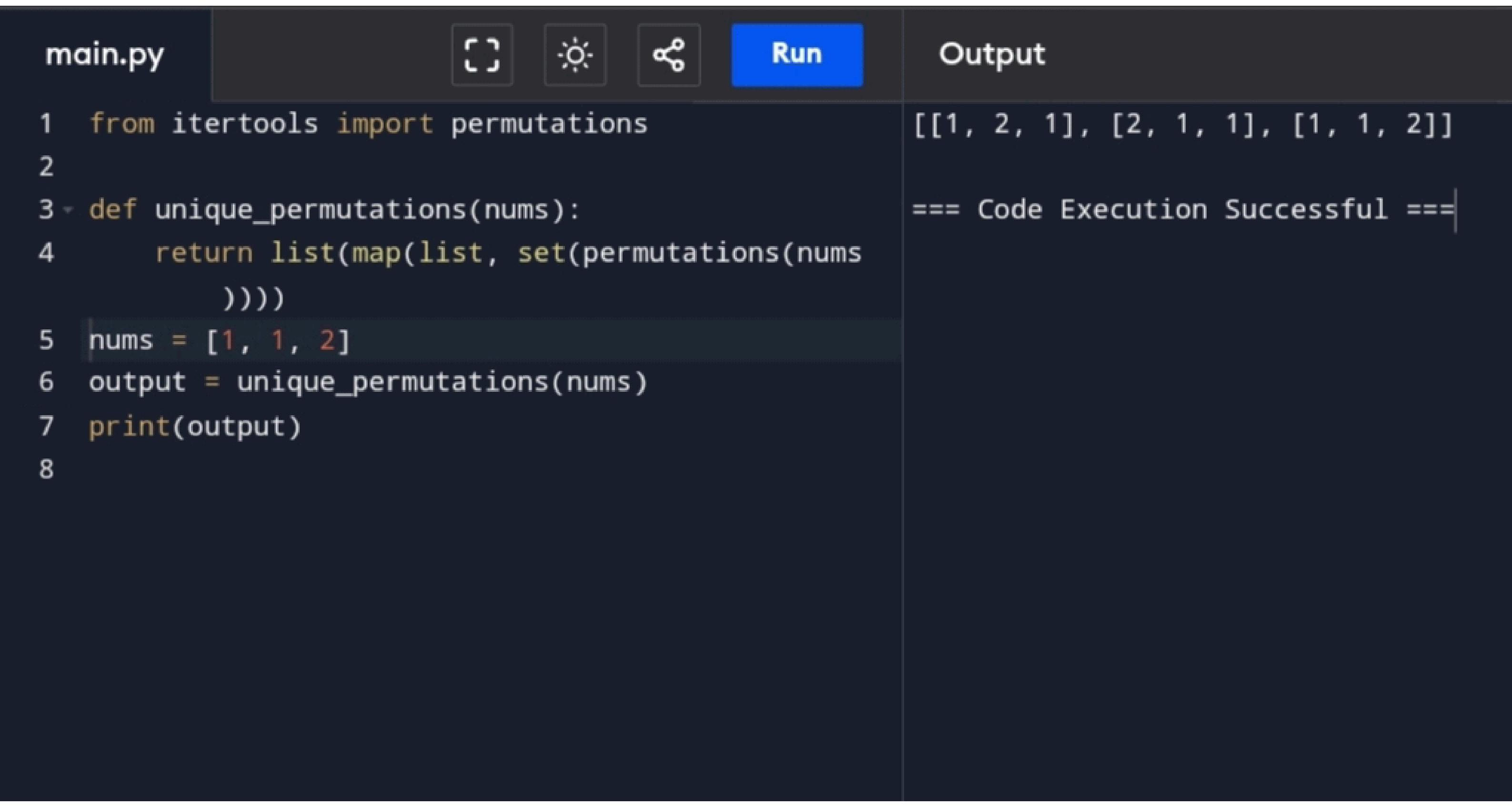
```python
def combination_sum2(candidates, target):
    def backtrack(start, path, target):
        if target == 0:
            res.append(path)
            return
        for i in range(start, len(candidates)):
            if i > start and candidates[i] ==
                candidates[i - 1]: continue
            if candidates[i] > target: break
            backtrack(i + 1, path +
                [candidates[i]], target -
                candidates[i])

    candidates.sort()
    res = []
    backtrack(0, [], target)
    return res
candidates = [10, 1, 2, 7, 6, 1, 5]
target = 8
print(combination_sum2(candidates, target))
```

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

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



```python
from itertools import permutations

def permute(nums):
    return list(map(list, permutations(nums)))
nums = [1, 2, 3]
output = permute(nums)
print(output)
```

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

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



```python
from itertools import permutations

def unique_permutations(nums):
    return list(map(list, set(permutations(nums
        ))))
nums = [1, 1, 2]
output = unique_permutations(nums)
print(output)
```

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

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