Programiz Python Online Compiler

main.py

Share Run

```python
def findPaths(m, n, N, i, j):
    MOD = 10**9 + 7
    dp = [[0] * n for _ in range(m)]
    dp[i][j] = 1
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    count = 0
    for _ in range(N):
        temp = [[0] * n for _ in range(m)]
        for r in range(m):
            for c in range(n):
                for dr, dc in directions:
                    nr, nc = r + dr, c + dc
                    if 0 <= nr < m and 0 <= nc < n:
                        temp[nr][nc] = (temp[nr][nc] + dp[r][c]) %
                            MOD
                    else:
                        count = (count + dp[r][c]) % MOD
        dp = temp

    return count

# Test Cases
```

Output

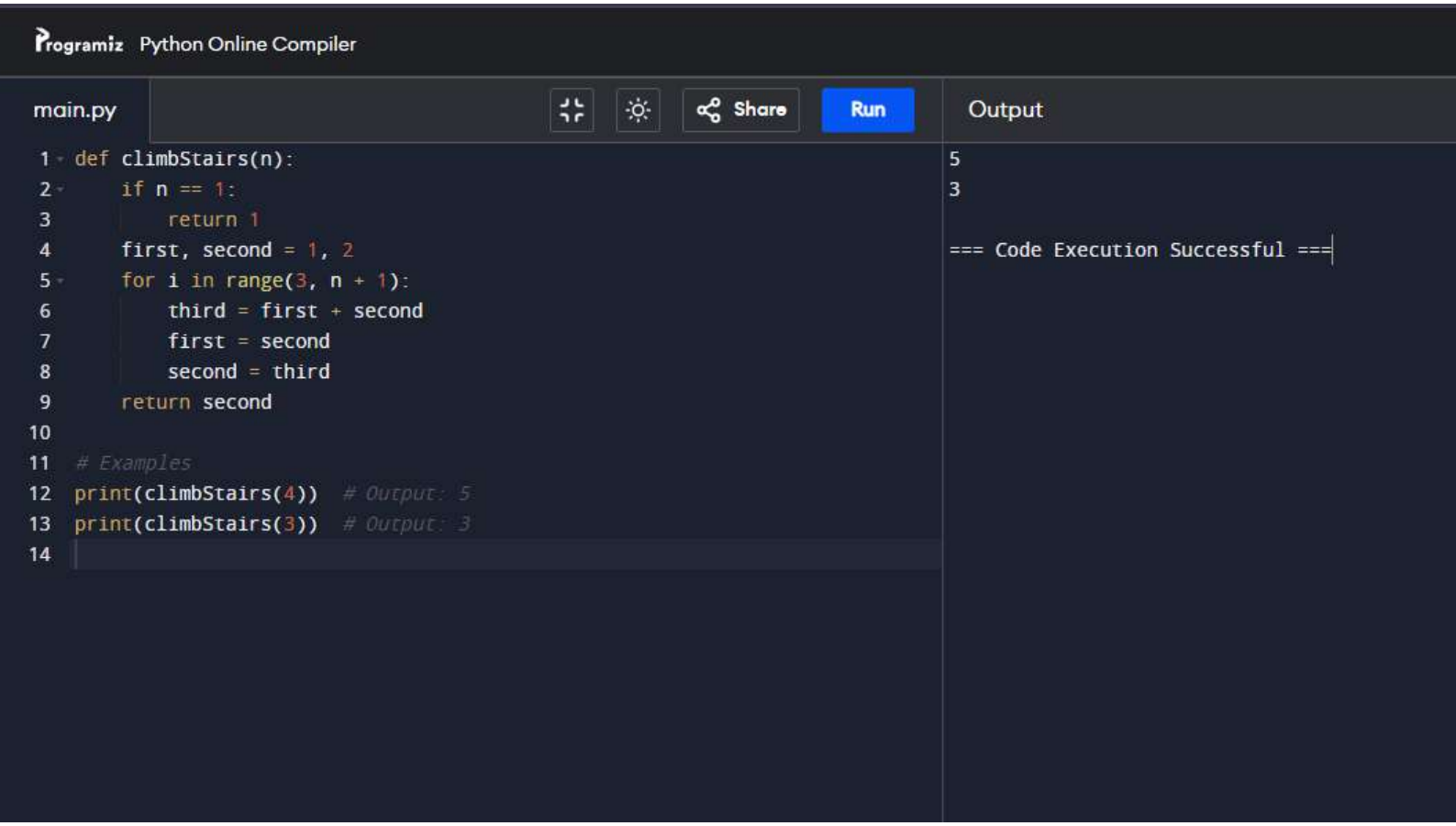
```
6
12

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

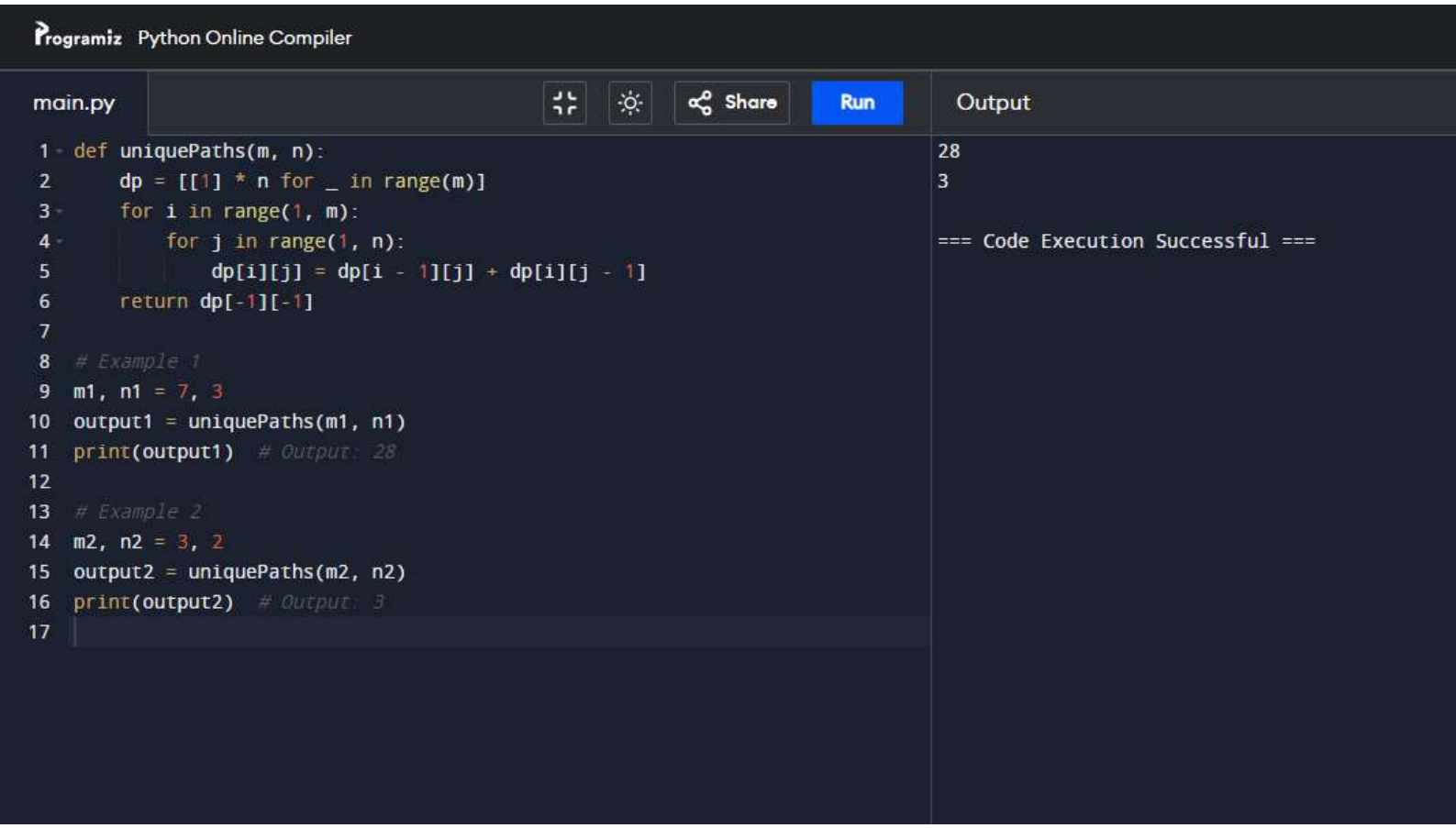
Programiz Python Online Compiler

main.py | Share | Run | Output

```python
def rob(nums):
    def rob_range(start, end):
        rob_next, rob_curr = 0, 0
        for i in range(start, end):
            rob_next, rob_curr = max(rob_curr + nums[i], rob_next),
                rob_next
        return rob_next

    if len(nums) == 1:
        return nums[0]
    return max(rob_range(0, len(nums) - 1), rob_range(1, len(nums)))

# Examples
nums1 = [2, 3, 2]
print("Output:", rob(nums1))  # Output: The maximum money you can rob
    without alerting the police is 3 (robbing house 1).

nums2 = [1, 2, 3, 1]
print("Output:", rob(nums2))  # Output: The maximum money you can rob
    without alerting the police is 4 (robbing house 1 and house 3).
```

```
Output: 3
Output: 4

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


Programiz Python Online Compiler
main.py
Share
Run
Output
def climbStairs(n):
if n == 1:
return 1
first, second = 1, 2
for i in range(3, n + 1):
third = first + second
first = second
second = third
return second
Examples
print(climbStairs(4)) # Output: 5
print(climbStairs(3)) # Output: 3
5
3
=== Code Execution Successful ===


Programiz Python Online Compiler
main.py
Share
Run
Output
def uniquePaths(m, n):
dp = [[1] * n for _ in range(m)]
for i in range(1, m):
for j in range(1, n):
dp[i][j] = dp[i - 1][j] + dp[i][j - 1]
return dp[-1][-1]
Example 1
m1, n1 = 7, 3
output1 = uniquePaths(m1, n1)
print(output1) # Output: 28
Example 2
m2, n2 = 3, 2
output2 = uniquePaths(m2, n2)
print(output2) # Output: 3
28
3
=== Code Execution Successful ===

Programiz Python Online Compiler

main.py

Share Run

```python
def largeGroupPositions(s):
    result = []
    start = 0
    for end in range(len(s)):
        if end == len(s) - 1 or s[end] != s[end + 1]:
            if end - start + 1 >= 3:
                result.append([start, end])
            start = end + 1
    return result

# Example 1
s1 = "abbxxxxzzy"
print(largeGroupPositions(s1))  # Output: [[3, 6]]

# Example 2
s2 = "abc"
print(largeGroupPositions(s2))  # Output: []
```

Output

```
[[3, 6]]
[]

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

Programiz Python Online Compiler

main.py

Share Run

```
def count_live_neighbors(board, i, j):
    count = 0
    for x in range(-1, 2):
        for y in range(-1, 2):
            if x == 0 and y == 0:
                continue
            if 0 <= i + x < len(board) and 0 <= j + y < len
                (board[0]):
                count += board[i + x][j + y] & 1
    return count

for i in range(len(board)):
    for j in range(len(board[0])):
        live_neighbors = count_live_neighbors(board, i, j)
        if board[i][j] == 1 and (live_neighbors < 2 or
            live_neighbors > 3):
            board[i][j] = 2
        if board[i][j] == 0 and live_neighbors == 3:
            board[i][j] = -1

for i in range(len(board)):
    for j in range(len(board[0])):

```

Output

```
[[0, 1, 0], [0, 0, 1], [1, 1, 1], [0, 0, 0]]
[[1, 1], [1, 0]]

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

Programiz Python Online Compiler

main.py

Share Run

```python
def champagneTower(poured, query_row, query_glass):
    glasses = [[0] * i for i in range(1, 102)]
    glasses[0][0] = poured
    for i in range(100):
        for j in range(i + 1):
            if glasses[i][j] > 1:
                excess = (glasses[i][j] - 1) / 2
                glasses[i + 1][j] += excess
                glasses[i + 1][j + 1] += excess
                glasses[i][j] = 1
    return min(1, glasses[query_row][query_glass])

```

Output

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