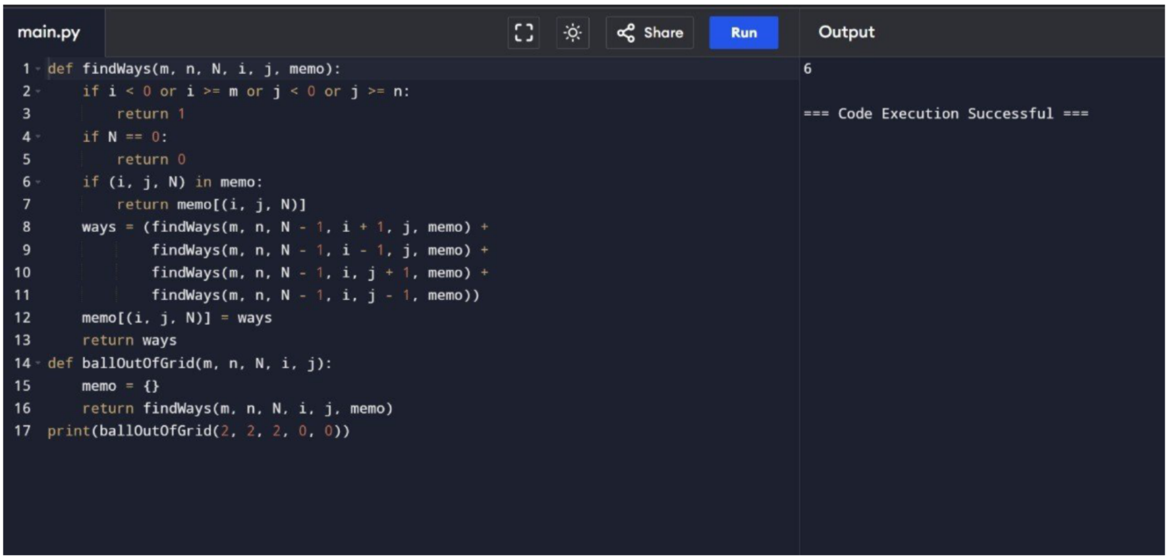
Day 2:

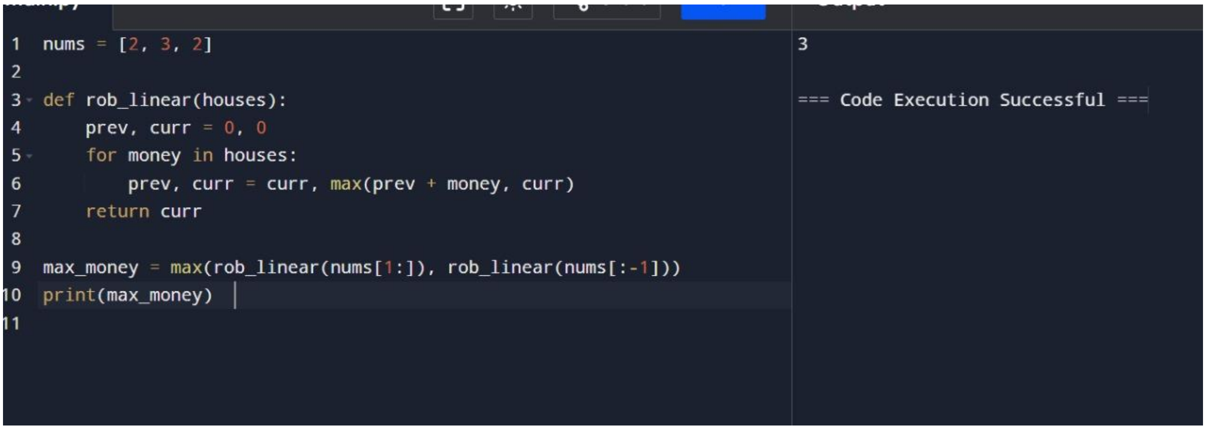
1



```python
def findWays(m, n, N, i, j, memo):
    if i < 0 or i >= m or j < 0 or j >= n:
        return 1
    if N == 0:
        return 0
    if (i, j, N) in memo:
        return memo[(i, j, N)]
    ways = (findWays(m, n, N - 1, i + 1, j, memo) +
            findWays(m, n, N - 1, i - 1, j, memo) +
            findWays(m, n, N - 1, i, j + 1, memo) +
            findWays(m, n, N - 1, i, j - 1, memo))
    memo[(i, j, N)] = ways
    return ways
def ballOutOfGrid(m, n, N, i, j):
    memo = {}
    return findWays(m, n, N, i, j, memo)
print(ballOutOfGrid(2, 2, 2, 0, 0))
```

```
6

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

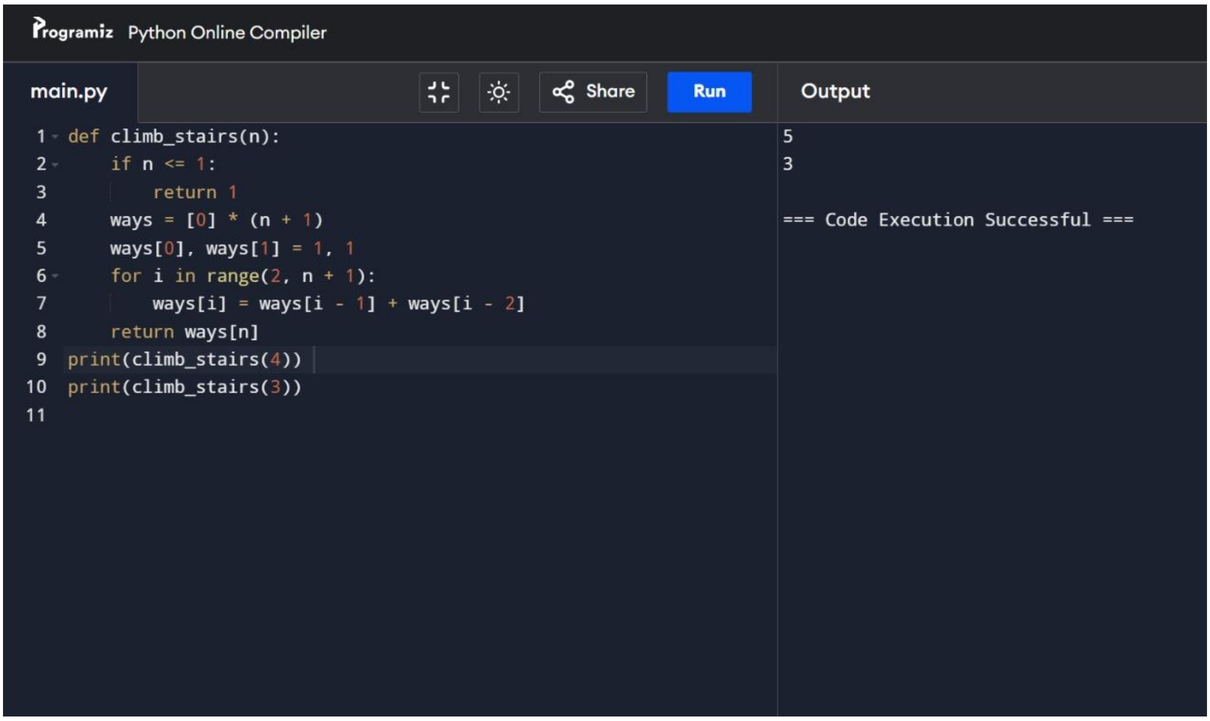
2



```python
nums = [2, 3, 2]

def rob_linear(houses):
    prev, curr = 0, 0
    for money in houses:
        prev, curr = curr, max(prev + money, curr)
    return curr

max_money = max(rob_linear(nums[1:]), rob_linear(nums[:-1]))
print(max_money)
```

```
3

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

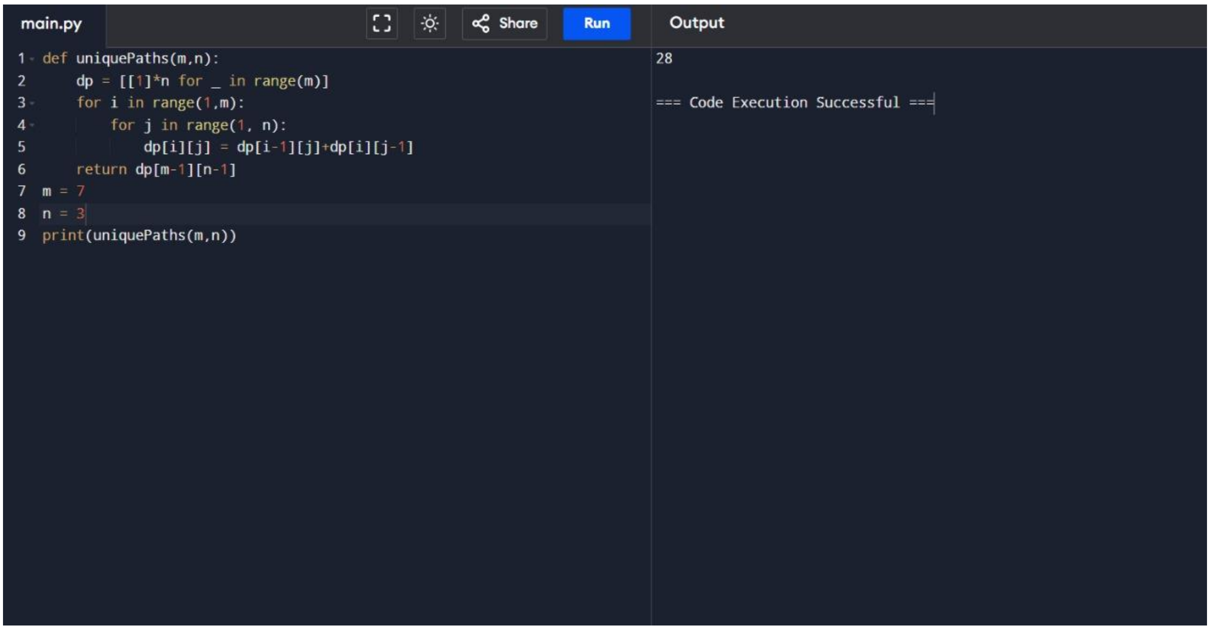
3



```python
def climb_stairs(n):
    if n <= 1:
        return 1
    ways = [0] * (n + 1)
    ways[0], ways[1] = 1, 1
    for i in range(2, n + 1):
        ways[i] = ways[i - 1] + ways[i - 2]
    return ways[n]
print(climb_stairs(4))
print(climb_stairs(3))
```

```
5
3

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

4

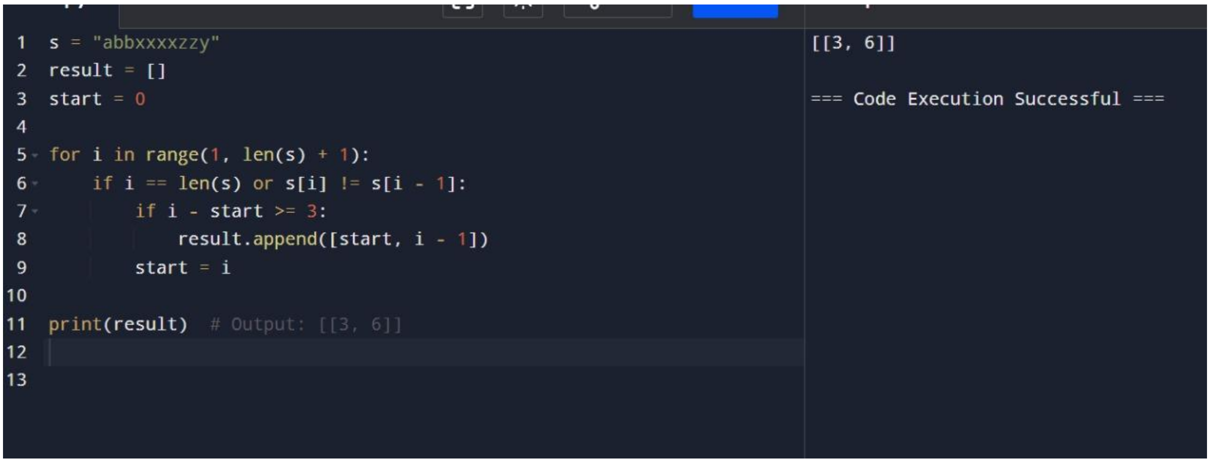


```python
def uniquePaths(m,n):
    dp = [[1]*n for _ in range(m)]
    for i in range(1,m):
        for j in range(1, n):
            dp[i][j] = dp[i-1][j]+dp[i][j-1]
    return dp[m-1][n-1]
m = 7
n = 3
print(uniquePaths(m,n))
```

```
28

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

5

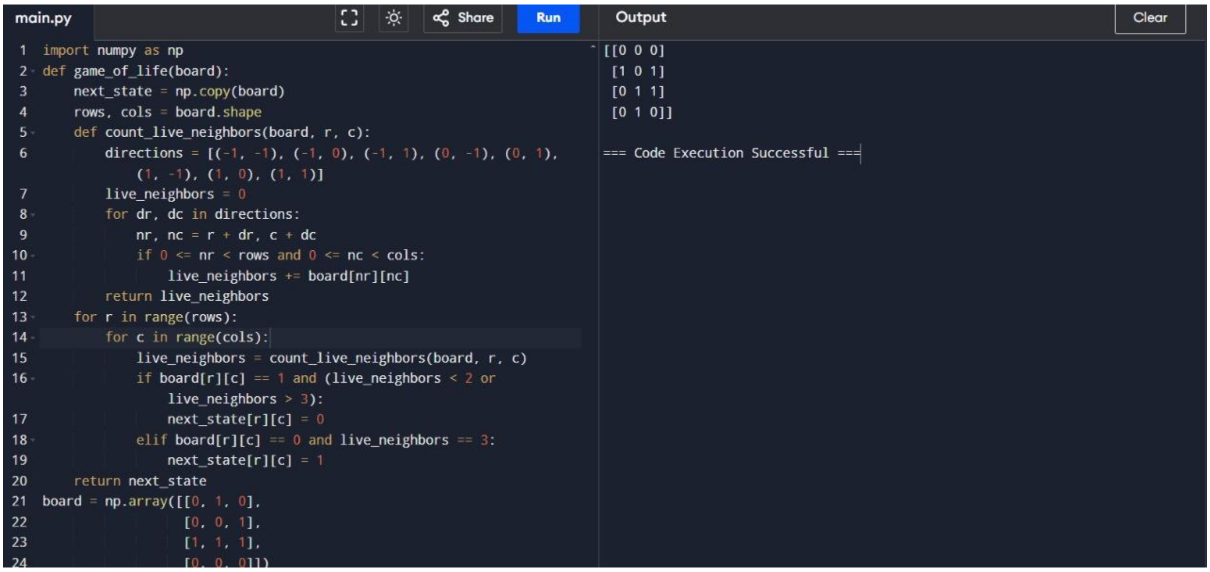
5



```python
s = "abbxxxxzzy"
result = []
start = 0

for i in range(1, len(s) + 1):
    if i == len(s) or s[i] != s[i - 1]:
        if i - start >= 3:
            result.append([start, i - 1])
        start = i

print(result)  # Output: [[3, 6]]
```

```
[[3, 6]]

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

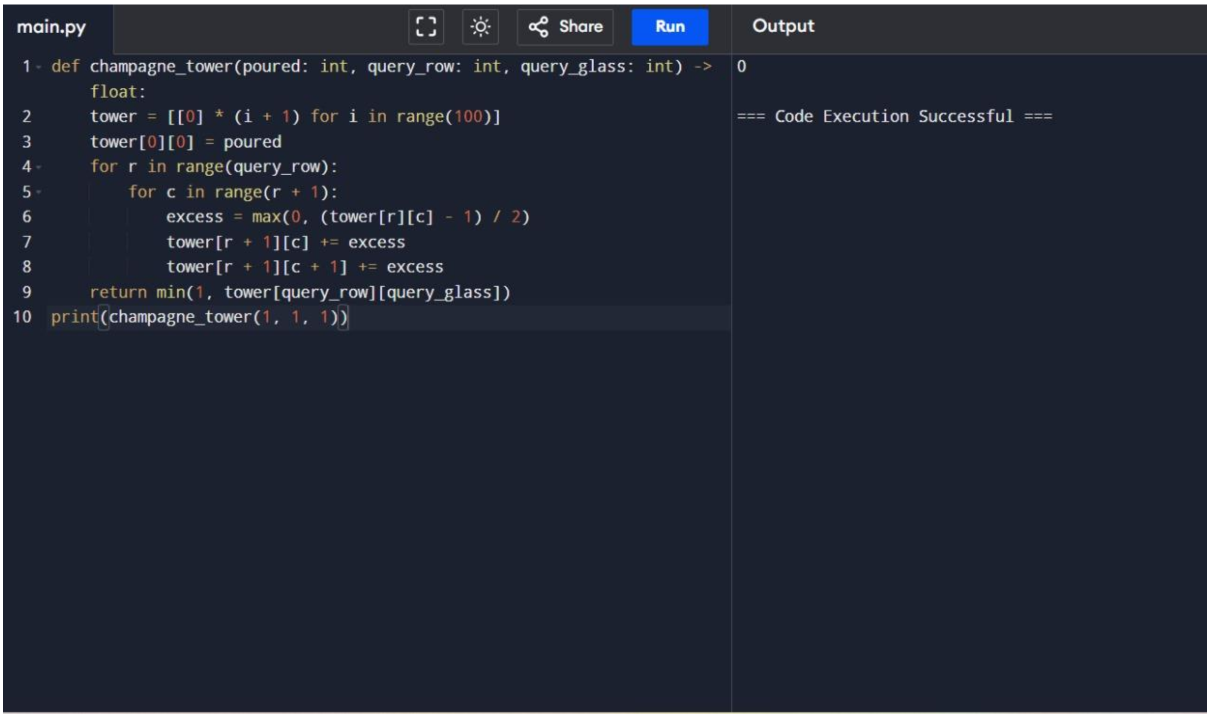
6



```python
import numpy as np
def game_of_life(board):
    next_state = np.copy(board)
    rows, cols = board.shape
    def count_live_neighbors(board, r, c):
        directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1),
            (1, -1), (1, 0), (1, 1)]
        live_neighbors = 0
        for dr, dc in directions:
            nr, nc = r + dr, c + dc
            if 0 <= nr < rows and 0 <= nc < cols:
                live_neighbors += board[nr][nc]
        return live_neighbors
    for r in range(rows):
        for c in range(cols):
            live_neighbors = count_live_neighbors(board, r, c)
            if board[r][c] == 1 and (live_neighbors < 2 or
                live_neighbors > 3):
                next_state[r][c] = 0
            elif board[r][c] == 0 and live_neighbors == 3:
                next_state[r][c] = 1
    return next_state
board = np.array([[0, 1, 0],
                  [0, 0, 1],
                  [1, 1, 1],
                  [0, 0, 0]])
```

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

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

7



```python
def champagne_tower(poured: int, query_row: int, query_glass: int) -> float:
    tower = [[0] * (i + 1) for i in range(100)]
    tower[0][0] = poured
    for r in range(query_row):
        for c in range(r + 1):
            excess = max(0, (tower[r][c] - 1) / 2)
            tower[r + 1][c] += excess
            tower[r + 1][c + 1] += excess
    return min(1, tower[query_row][query_glass])
print(champagne_tower(1, 1, 1))
```

```
0

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