main.py

Share Run

Output

```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 ===
```

main.py

Share Run

```python
def rob_linear(houses):
    prev = 0
    curr = 0
    for amount in houses:
        new_curr = max(curr, prev + amount)
        prev = curr
        curr = new_curr
    return curr
def rob_circular(nums):
    if len(nums) == 1:
        return nums[0]
    case1 = rob_linear(nums[:-1])
    case2 = rob_linear(nums[1:])
    return max(case1, case2)
print(rob_circular([2, 3, 2]))
```

Output

```
3

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

main.py

Share Run

```python
def stairs(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return stairs(n - 1) + stairs(n - 2)
sum=1
for i in range(5):
    sum+=stairs(i)
print(sum)
```

Output

```
8

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

main.py

Share Run Output

```
def fact(a):
    if a==1:
        return a
    else:
        return a*fact(a-1)
def paths(m,n):
    return fact((m+n)-2)//(fact(m-1)*fact(n-1))
print(paths(7,3))
```

```
28

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

main.py

Share Run

```python
def large_group_positions(s):
    result = []
    n = len(s)
    i = 0
    while i < n:
        start = i
        while i < n and s[i] == s[start]:
            i += 1
        if i - start >= 3:
            result.append([start, i - 1])
    return result
print(large_group_positions("abbxxxxzzy"))
print(large_group_positions("abc"))
```

Output

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

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

main.py

Share Run

```python
def game_of_life(board):
    if not board:
        return
    m, n = len(board), len(board[0])
    directions = [(-1, -1), (-1, 0), (-1, 1),
                  (0, -1),           (0, 1),
                  (1, -1), (1, 0), (1, 1)]
    for i in range(m):
        for j in range(n):
            live_neighbors = 0
            for di, dj in directions:
                ni, nj = i + di, j + dj
                if 0 <= ni < m and 0 <= nj < n:
                    if board[ni][nj] == 1 or board[ni][nj] == 3:
                        live_neighbors += 1
            if board[i][j] == 1 and (live_neighbors < 2 or live_neighbors > 3):
                board[i][j] = 3
            elif board[i][j] == 0 and live_neighbors == 3:
                board[i][j] = 2
    for i in range(m):
        for j in range(n):
            if board[i][j] == 2:
                board[i][j] = 1
            elif board[i][j] == 3:
                board[i][j] = 0
board = [[0, 1, 0], [0, 0, 1], [1, 1, 1], [0, 0, 0]]
game_of_life(board)
```

Output

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

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

main.py

Share Run

```python
def champagneTower(poured: int, query_row: int, query_glass: int) -> float:
    glasses = [[0] * (i + 1) for i in range(101)]
    glasses[0][0] = poured
    for i in range(100):
        for j in range(i + 1):
            if glasses[i][j] > 1:
                overflow = glasses[i][j] - 1
                glasses[i][j] = 1
                glasses[i + 1][j] += overflow / 2
                glasses[i + 1][j + 1] += overflow / 2
    return min(1, glasses[query_row][query_glass])
print(champagneTower(2, 1, 1))
```

Output

```
0.5

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