1.

```python
from functools import lru_cache

m, n, N, i, j = 2, 2, 2, 0, 0

@lru_cache(None)
def dfs(x, y, steps):
    if x < 0 or x >= m or y < 0 or y >= n:
        return 1
    if steps == 0:
        return 0
    return (dfs(x+1, y, steps-1) + dfs(x-1, y, steps-1) +
            dfs(x, y+1, steps-1) + dfs(x, y-1, steps-1))

print(dfs(i, j, N))
```

```
6

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

2.

```python
def rob_linear(nums):
    prev, curr = 0, 0
    for num in nums:
        prev, curr = curr, max(curr, prev + num)
    return curr

def rob(nums):
    if len(nums) <= 1:
        return nums[0] if nums else 0
    return max(rob_linear(nums[:-1]), rob_linear(nums[1:]))

# Example usage:
nums = [2, 3, 2]
print(rob(nums))
```

```
3

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

3.

```python
def climbStairs(n):
    a, b = 1, 1
    for _ in range(n-1):
        a, b = b, a + b
    return b

print(climbStairs(4))
print(climbStairs(3))
```

```
5
3

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

4.

```python
import math
m, n = 3, 2
print(math.comb(m + n - 2, m - 1))
```

```
3

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

5.

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

```
[[3, 6]]

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

6.

```python
def gameOfLife(board):
    if not board:
        return
    m, n = len(board), len(board[0])
    def count_live_neighbors(i, j):
        count = 0
        for x in range(max(i-1, 0), min(i+2, m)):
            for y in range(max(j-1, 0), min(j+2, n)):
                if (x != i or y != j) and board[x][y] & 1:
                    count += 1
        return count
    for i in range(m):
        for j in range(n):
            live_neighbors = count_live_neighbors(i, j)
            if board[i][j] & 1:
                if live_neighbors < 2 or live_neighbors > 3:
                    board[i][j] = 1
                else:
                    board[i][j] = 3
            else:
                if live_neighbors == 3:
                    board[i][j] = 2
    for i in range(m):
        for j in range(n):
            board[i][j] >>= 1
board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]
gameOfLife(board)
print(board)
```

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

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

7.

```python
def champagneTower(poured, query_row, query_glass):
    dp = [[0] * (i + 1) for i in range(101)]
    dp[0][0] = poured

    # Process each row of the pyramid.
    for i in range(100):
        for j in range(i + 1):
            if dp[i][j] > 1:
                excess = (dp[i][j] - 1) / 2
                dp[i+1][j] += excess
                dp[i+1][j+1] += excess
    return min(1, dp[query_row][query_glass])
poured = 4
query_row = 2
query_glass = 1
print(champagneTower(poured, query_row, query_glass))
```

```
0.5

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