

main.py



Share

Run

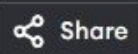
Output

```
1 def findWays(m, n, N, i, j, memo):
2     if i < 0 or i >= m or j < 0 or j >= n:
3         return 1
4     if N == 0:
5         return 0
6     if (i, j, N) in memo:
7         return memo[(i, j, N)]
8     ways = (findWays(m, n, N - 1, i + 1, j, memo) +
9             findWays(m, n, N - 1, i - 1, j, memo) +
10            findWays(m, n, N - 1, i, j + 1, memo) +
11            findWays(m, n, N - 1, i, j - 1, memo))
12     memo[(i, j, N)] = ways
13     return ways
14 def ballOutOfGrid(m, n, N, i, j):
15     memo = {}
16     return findWays(m, n, N, i, j, memo)
17 print(ballOutOfGrid(2, 2, 2, 0, 0))
```

6

=== Code Execution Successful ===

main.py



Share

Run

Output

```
1 def rob_linear(houses):
2     prev = 0
3     curr = 0
4     for amount in houses:
5         new_curr = max(curr, prev + amount)
6         prev = curr
7         curr = new_curr
8     return curr
9 def rob_circular(nums):
10     if len(nums) == 1:
11         return nums[0]
12     case1 = rob_linear(nums[:-1])
13     case2 = rob_linear(nums[1:])
14     return max(case1, case2)
15 print(rob_circular([2, 3, 2]))
```

3

=== Code Execution Successful ===

main.py



Share

Run

Output

```
1 def stairs(n):
2     if n <= 0:
3         return 0
4     elif n == 1:
5         return 1
6     else:
7         return stairs(n - 1) + stairs(n - 2)
8 sum=1
9 for i in range(5):
10     sum+=stairs(i)
11 print(sum)
```

8

=== Code Execution Successful ===

main.py	   Share 	Output
<pre>1 def fact(a): 2 if a==1: 3 return a 4 else: 5 return a*fact(a-1) 6 def paths(m,n): 7 return fact((m+n)-2)//(fact(m-1)*fact(n-1)) 8 print(paths(7,3))</pre>		<pre>28 === Code Execution Successful ===</pre>

main.py



Share

Run

Output

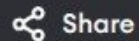
```
1 def large_group_positions(s):
2     result = []
3     n = len(s)
4     i = 0
5     while i < n:
6         start = i
7         while i < n and s[i] == s[start]:
8             i += 1
9         if i - start >= 3:
10            result.append([start, i - 1])
11    return result
12 print(large_group_positions("abbxxxxzzy"))
13 print(large_group_positions("abc"))
```

[[3, 6]]

[]

=== Code Execution Successful ===

main.py



Run

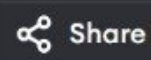
Output

```
1 def game_of_life(board):
2     if not board:
3         return
4     m, n = len(board), len(board[0])
5     directions = [(-1, -1), (-1, 0), (-1, 1),
6                  (0, -1), (0, 1),
7                  (1, -1), (1, 0), (1, 1)]
8     for i in range(m):
9         for j in range(n):
10            live_neighbors = 0
11            for di, dj in directions:
12                ni, nj = i + di, j + dj
13                if 0 <= ni < m and 0 <= nj < n:
14                    if board[ni][nj] == 1 or board[ni][nj] == 3:
15                        live_neighbors += 1
16            if board[i][j] == 1 and (live_neighbors < 2 or live_neighbors > 3):
17                board[i][j] = 3
18            elif board[i][j] == 0 and live_neighbors == 3:
19                board[i][j] = 2
20     for i in range(m):
21         for j in range(n):
22             if board[i][j] == 2:
23                 board[i][j] = 1
24             elif board[i][j] == 3:
25                 board[i][j] = 0
26     board = [[0, 1, 0], [0, 0, 1], [1, 1, 1], [0, 0, 0]]
27     game_of_life(board)
```

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

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

main.py



Share

Run

Output

```
1 def champagneTower(poured: int, query_row: int, query_glass: int) -> float:
2     glasses = [[0] * (i + 1) for i in range(101)]
3     glasses[0][0] = poured
4     for i in range(100):
5         for j in range(i + 1):
6             if glasses[i][j] > 1:
7                 overflow = glasses[i][j] - 1
8                 glasses[i][j] = 1
9                 glasses[i + 1][j] += overflow / 2
10                glasses[i + 1][j + 1] += overflow / 2
11     return min(1, glasses[query_row][query_glass])
12 print(champagneTower(2, 1, 1))
```

0.5

=== Code Execution Successful ===