DAY-2

Given an m×n grid and a ball at a starting cell, find the number of ways to move the ball out of the grid boundary in exactly N steps.

Input:- m=2, n=2, i=0, j=0

output:- 6

Python code:-

```
def findways (m,n,N, startRow, startCol):
    exit_count=[0]
def moveBall (steps, row, col):
    if row<0 or row>=m or col<0 or col>=n:
        exit_count[0]+=1
        return
    if steps==0:
        return
moveball (steps-1, row+1, col)
moveball (steps-1, row-1, col)
moveball (steps-1, row, col+1)
moveball (steps-1 row, col-1)
moveball (steps-1 m, startRow, startCol)
return exit_count[0]
print findways (2,2,0,0,0)
```

Result:-

ouput = 6

You are a professional robot planning to rob house along a street each house has a certain amount of money stashed. All houses at this place are arranged in a circle. that means the first house is the neighbour of the last one. Meanwhile, adjacent houses have security system connected and it will automatically contact the police if 2 adjacent houses were broken into on the same night

I/p nums :- [2, 3, 2]

O/p :- 3

Python code :- def rob_linear(nums):

prev1, prev2 = 0, 0

for num in nums:

current = max(prev1, prev2 + num)

prev2 = prev1

prev1 = current

return prev1

def rob(nums):

if len(nums) == 1:

return nums[0]

return max(rob_linear(nums[1:]), rob_linear(nums[:-1]))

print(rob([2, 3, 2]))

Result :-

output = 3

You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top

I/p :- n=4

o/p :- 5

Python code

```
def climbstairs(n):
    if n == 1:
        return 1
    if n == 2:
        return 2
    a, b = 1, 1
    for i in range(2, n+1):
        a, b = b, a+b
    return b
print(climbstairs(4))
```

Result :-

output :- 5

A robot is located at the top-left corner of a m×n grid. the robot can only move either down or right at any point in time. the robot is trying to reach the bottom-right corner of the grid. How many possible unique path are there?

I/P:- m=7, n=3

O/P:- 28

Python Code:-

```
import math
def unique_path(m, n):
    return math.comb(m+n-2, m-1)
print(unique_paths(7, 3))
```

Result:-

output: 28

In a string S of lowercase letters, these letters form consecutive groups of the same characters. For a string like S = "abbxxxxzyy" has the groups "a", "bb", "xxxx", "z", "yy". A group is identified by an interval [start, end] where start and end denote [3, 6] Return the intervals of every large group started in increasing order by start index

I/P :- S = "abbxxxxzyy"

O/P :- [[3, 6]]

Python code

```
def largeGroupPositions(s):
    res = []
    n = len(s)
    i = 0
    while i < n:
        j = i
        while j < n and s[j] == s[i]:
            j += 1
        if j - i >= 3:
            res.append([i, j-1])
        i = j
    return res
```

S = "abbxxxxyzyy"

output : largeGroupPosition(s)

print(output)

Result :-

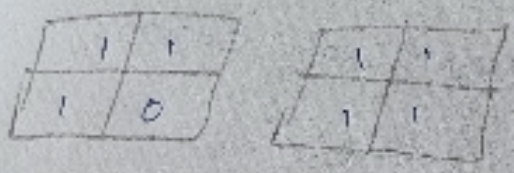
output [[3, 6]]

The game of life, also known simply as life, is a cellular automaton devised by the british Mathematician John Horton in 1970. The board is made up of an mxn grid of cells, where each cell has an initial state live (represented by a 1) or dead (represented by 0).

1) Any live cell with two or three live neighbours lives to the next generation

2) Any live cells with more than three live neighbours dies, as if by over-population

3) Any dead cell with exactly three live neighbours a live cells, as if by reproduction



IIP:- [[1,1],[1,0]]

OIP:- [[1,1],[1,1]]

python code:- def game of life(board):

rows, cols = len(board), len(board[0])

def count_live_neighbour(r,c):

direction = [(-1,-1),(-1,0),(-1,1),(0,-1)
(0,1),(1,-1)(1,0),(1,1)]

count = 0

for dr, dc in directions.

nr, nc = r+dr, c+dc

if 0<=nr<len(board) and 0<=nc<len(board[0])

and abs(board[nr][nc])==1:

```
                count += 1
        return count
    for i in range(len(board)):
        for c in range(len(board[0])):
            live_neighbour = count_live_neighbour(r,c)
    if board[i][c] == 1 and (live_neighbour < 2 or
                             live_neighbour > 3):
        board[i][c] = -1
    if board[r][c] == 0 and live_neighbour == 3:
        board[r][c] = 2
    for r in range(len(board)):
    for c in range(len(board[0])):
        if board[r][c] == 2:
            board[r][c] = 0
    return board
board = [[1,1], [1,0]]
print(game_of_life(board))
```

Result:-

output

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

We stack glasses in a pyramid where the first row has one glass, the second row has two glasses and so on until the 100th row. Each glass holds one cup of champagne. Then, some champagne is poured into the first glass at the top. When the topmost glass is full, any excess liquid poured will fall equally to the glass immediately to the left and right of it after pouring some non negative integer cups of champagne. Return how full the j^{th} glass in the i^{th} row is (both i and j are 0-indexed.

I/p:- poured = 2, query_row = 1, query_glass = 1

O/p:- 0.5

python code:

```
poured = 2
query_row = 1
query_glass = 1
current_row = [0] * (query_row + 1)
current_row[0] = poured
for row in range(query_row):
    next_row = [0] * (row + 2)
    for glass in range(row + 1):
        if current_row[glass] > 1:
            overflow = (current_row[glass] - 1) / 2.0
            next_row[glass] += overflow
            next_row[glass + 1] += overflow
    current_row = next_row
result = min(1, current_row[query_glass])
print(result)
```

Result:-

output = 0.5