DAY-2

1)Given an m x 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.

Example:

· Input: m=2,n=2,N=2,i=0,j=0 · Output: 6

· Input: m=1,n=3,N=3,i=0,j=1 · Output: 12

CODE:

def findBallWays(m, n, N, i, j):

memo = {}

def countWays(steps, x, y):

if x < 0 or x >= m or y < 0 or y >= n:

return 1

if steps == 0:

return 0

if (steps, x, y) in memo:

return memo[(steps, x, y)]

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

ways = 0

for dx, dy in directions:

ways += countWays(steps - 1, x + dx, y + dy)

memo[(steps, x, y)] = ways

return ways

return countWays(N, i, j)

print(findBallWays(2, 2, 2, 0, 0))

OUTPUT:6

2)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?

Examples:

(i) Input: n=4 Output: 5

(ii) Input: n=3 Output: 3

CODE:

import math

def climbStairs\_combinations(n):

total\_ways = 0

for k in range(n // 2 + 1):

one\_steps = n - 2 \* k

total\_steps = one\_steps + k

# Calculate combinations (total\_steps choose k)

total\_ways += math.comb(total\_steps, k)

return total\_ways

n = 4

print(f"Input: n={n}, Output: {climbStairs\_combinations(n)}")

OUTPUT:5

3) You are a professional robber planning to rob houses 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 neighbor of the last one. Meanwhile, adjacent houses have

security systems connected, and it will automatically contact the police if two adjacent

houses were broken into on the same night.

Examples:

(i) Input : nums = [2, 3, 2]

Output : The maximum money you can rob without alerting the police is 3

CODE:

nums = [2, 3, 2]

if len(nums) == 1:

result = nums[0]

elif len(nums) == 2:

result = max(nums)

else:

rob1 = [0] \* len(nums)

rob1[0] = nums[0]

rob1[1] = max(nums[0], nums[1])

for i in range(2, len(nums) - 1):

rob1[i] = max(rob1[i - 1], nums[i] + rob1[i - 2])

max\_rob1 = rob1[len(nums) - 2]

rob2 = [0] \* len(nums)

rob2[1] = nums[1]

rob2[2] = max(nums[1], nums[2])

for i in range(3, len(nums)):

rob2[i] = max(rob2[i - 1], nums[i] + rob2[i - 2])

max\_rob2 = rob2[len(nums) - 1]

result = max(max\_rob1, max\_rob2)

print(f"Input: nums = {nums}")

print(f"Output: {result}")

OUTPUT: 3

4)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 paths are there?

Examples:

1. Input: m=7,n=3 Output: 28

CODE:

import math

m = 7

n = 3

unique\_paths = math.comb(m + n - 2, m - 1)

print(f"Input: m={m}, n={n}")

print(f"Output: {unique\_paths}")

OUTPUT: 28

5) 5. In a string S of lowercase letters, these letters form consecutive groups of the same

character. For example, a string like s = "abbxxxxzyy" has the groups "a", "bb", "xxxx",

"z", and "yy". A group is identified by an interval [start, end], where start and end denote

the start and end indices (inclusive) of the group. In the above example, "xxxx" has the

interval [3,6]. A group is considered large if it has 3 or more characters. Return the

intervals of every large group sorted in increasing order by start index.

Example 1:

Input: s = "abbxxxxzzy"

Output: [[3,6]]

CODE:

from collections import Counter

s = "abbxxxxzzy"

char\_count = Counter(s)

n = len(s)

result = []

count = 1

for i in range(1, n):

# If the current character is the same as the previous one, increment the count

if s[i] == s[i - 1]:

count += 1

else:

# If the count is 3 or more, add the interval to the result list

if count >= 3:

result.append([i - count, i - 1])

count = 1

if count >= 3:

result.append([n - count, n - 1])

print(f"Character Count: {char\_count}")

print(f"Input: s = \"{s}\"")

print(f"Output: {result}")

OUTPUT: [[3, 6]]

6) We stack glasses in a pyramid, where the first row has 1 glass, the second row has 2

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. any excess champagne will fall equally to the left

and right of those glasses, and so on. (A glass at the bottom row has its excess

champagne fall on the floor.) For example, after one cup of champagne is poured, the top

most glass is full. After two cups of champagne are poured, the two glasses on the

second row are half full. After three cups of champagne are poured, those two cups

become full - there are 3 full glasses total now. After four cups of champagne are

poured, the third row has the middle glass half full, and the two outside glasses are a

quarter full, as pictured below.

Example 1:

Input: poured = 1, query\_row = 1, query\_glass = 1

Output: 0.00000

CODE:

import numpy as np

poured = 1

query\_row = 1

query\_glass = 1

glasses = np.zeros((101, 101), dtype=float)

glasses[0][0] = poured

for r in range(100):

for c in range(r + 1):

if glasses[r][c] > 1.0:

overflow = (glasses[r][c] - 1) / 2.0

glasses[r][c] = 1.0

glasses[r + 1][c] += overflow

glasses[r + 1][c + 1] += overflow

result = min(1, glasses[query\_row][query\_glass])

print(f"Output: {result:.5f}")

OUTPUT:0

7) "The Game of Life, also known simply as Life, is a cellular automaton devised by the

British mathematician John Horton Conway in 1970." The board is made up of an m x n

grid of cells, where each cell has an initial state: live (represented by a 1) or dead

(represented by a 0). Each cell interacts with its eight neighbors (horizontal, vertical,

diagonal) using the following four rules

Any live cell with fewer than two live neighbors dies as if caused by under-

population.

1. Any live cell with two or three live neighbors lives on to the next generation.

2. Any live cell with more than three live neighbors dies, as if by over-

population.

3. Any dead cell with exactly three live neighbors becomes a live cell, as if by

reproduction.

The next state is created by applying the above rules simultaneously to every cell in

the current state, where births and deaths occur simultaneously. Given the current

state of the m x n grid board, return the next state.

CODE:

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

board = [

[0, 1, 0],

[0, 0, 1],

[1, 1, 1],

[0, 0, 0]

]

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

next\_state = [[board[r][c] for c in range(cols)] for r in range(rows)]

for r in range(rows):

for c in range(cols):

live\_neighbors = 0

for dr, dc in directions:

nr, nc = r + dr, c + dc

if 0 <= nr < rows and 0 <= nc < cols and board[nr][nc] == 1:

live\_neighbors += 1

if board[r][c] == 1

if live\_neighbors < 2 or live\_neighbors > 3:

next\_state[r][c] = 0

else:

# Rule 4: Dead cell becomes live if exactly 3 neighbors

if live\_neighbors == 3:

next\_state[r][c] = 1

for r in range(rows):

for c in range(cols):

board[r][c] = next\_state[r][c]

for row in board:

print(row)

OUTPUT:

[0, 0, 0]

[1, 0, 1]

[0, 1, 1]

[0, 1, 0]