Ball moving Out of Grid

Python Code :-

```
def find_ways (m,n,N,i,j):
 memo = {}
def dfs(x,y, steps_left):
if x<0  or  y<0  or  x>=m or y>=n:
return 1
if steps_left==0;
return 0
if (x,y,steps_left) in memo:
return   memo[(x,y,steps_left)]
moves= dfs(x-1,y,steps_left-1)+ dfs (x+1,y,steps
        _left-1)+ \dfs (x,y-1,steps_left-1)+
        dfs (x,y+1,steps_left-1) memo[(x,y,step
        _left)] =moves
 return moves
return dfs(i,j,N)

Print (find_Ways(2,2,2,0,0))
```

Output:-

```
6.
```

Maximum money Robbery without robbing two adjacent houses.

Python Code:-

```
def rob(nums):
if len(nums)==1:
return nums[0]
def rob_linear(houses):
Prev, Curr =0,0
for money in houses:
Prev, Curr = Curr, max(prev+money, Curr)
return Curr
return max(rob-linear(nums[::-1]), rob-linear
                        (nums[1::]))

nums= [2, 3, 2]"

Print(" robber of maximum money
without robbing two adjacent houses=",
rob(nums))
```

Output:-

```
3
```

Staircase climbing ~ in distinct w with 2 steps.

Python Code:-

```
def climb_stairs(n):
if n==1:
return 1
dp = [0] * (n+1)
dp [1], dp[2] = 1, 2
for i in range (3, n+1):
  dp[i] = dp [i-1] + dp[i-2]
return dp[n]

n = 4
Print (climb-stairs (n))
```

Output:-

```
5.
```

Number of unique paths for robot moving from top-left Corner to bottom-right corner

Python Code:-

```
def   unique_path (m,n):
dp= [[1] * n for _ in range(m)]:
for i in range (1,m):
 for j in range (1,n):
  dp[i][j] = dp[i-1][j] + dp[i][j-1]
return dp[-1][-1]

Print (unique_paths (7,3))
Print (unique_paths (3,2))
```

Output:-

```
28

3
```

Starting and ending positions of larg groups of 3 or more same cha

Python Code:-

```
def large-group(s):
result = [ ]
start = 0
for i in range (1,len(s)+1):
~~for~~ if i== len(s) or s[i]!= s[start]:
if i- start >= 3;
result. append ([start,i-1])
start = i
return result

Print (large-group ("abbxx xxzzy"))
Print (large-group ("abc"))
```

Output:-

```
[[3,6]]

[[ ]]
```

Starting and ending positions of large groups of 3 or more same cha

Python Code:-

```
def large-group(s):
result = [ ]
start = 0
for i in range (1, len(s)+1):
~~for~~ if i == len(s) or s[i] != s[start]:
if i - start >= 3;
result.append([start, i-1])
start = i
return result

Print (large-group ("abbxx xxzzy"))
Print (large-group ("abc"))
```

Output:-

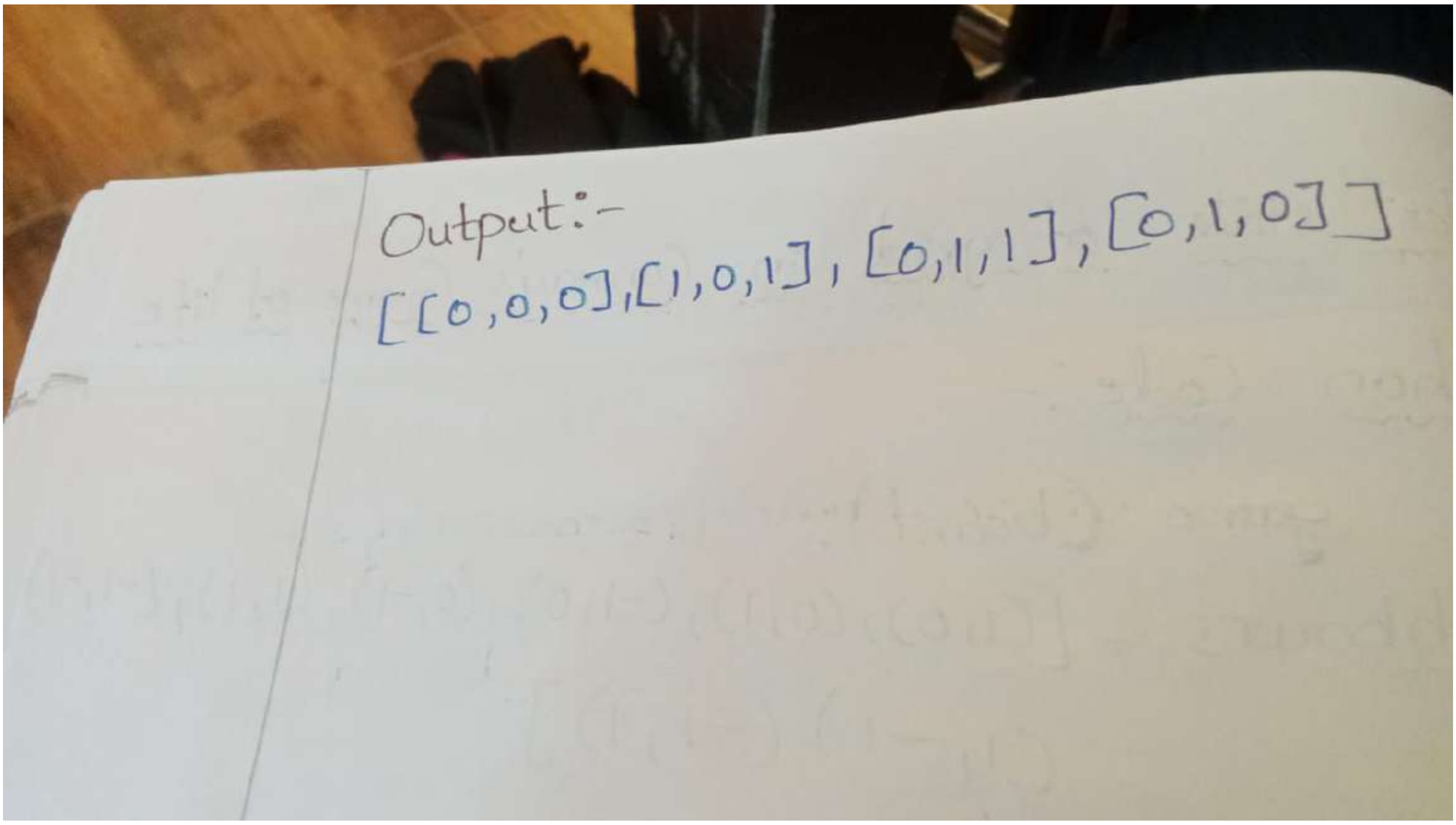
```
[[3, 6]]

[[  ]]
```

Next state of grid in Conway's Game of life

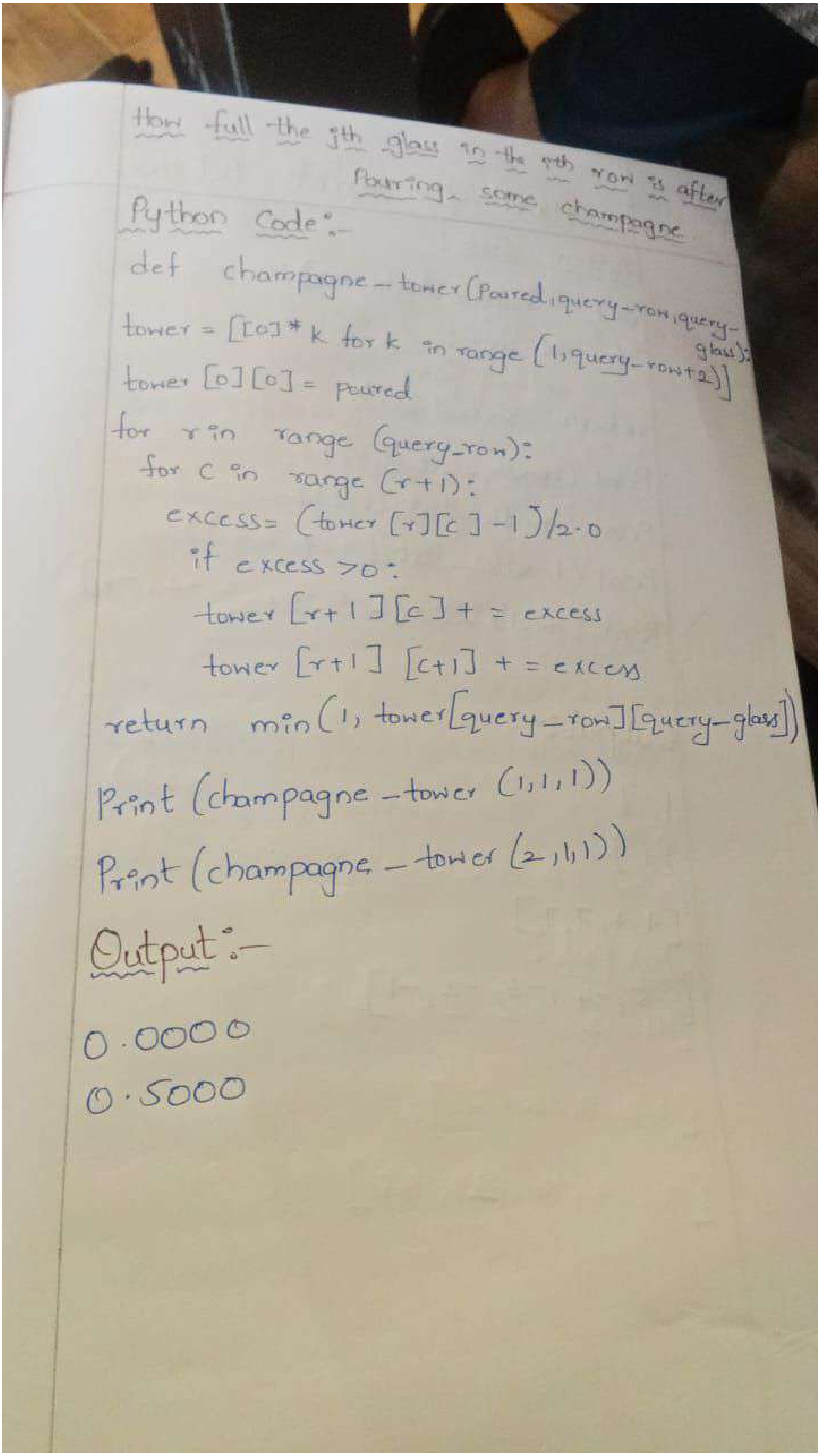
Python Code :-

```
def  game (board):
neighbours =[(1,0),(0,1),(-1,0),(0,-1),(1,1),(-1,-1)
              (1,-1) (-1,1)]
rows, cols = len(board), len (board [0])
for  row  in   range (rows):
 for  col  in   range (cols):
   live_neighbours = 0
 for neighbours  in  neighbours:
   r, c = row  + neighbour [0], col + neighbour[1]
if (0<=r < rows) and (0< = c<cols) and abs (board
[r] [c]==1 :
live_neighbours + = 1
if  board [row][col ]==1 and (live_neighbour<2 or
live_neighbours > 3):
board [row] [col] + =-1
if  board [row] [col ]==0  and  live neighbours=
                                              =3:
board [row] [col] =2
for   row  in  range (rows):
   for col  in   range (cols):
    board [row][col] =1 if board [row][col]>0
                                              else 0
board =[[0,1,0],[0,0,1],[1,1,1][0,0,0]]
Print (game (board))
```



Output:-

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



How full the jth glass in the ith row is after Pouring some champagne

Python Code:-

```
def champagne_tower(Poured,query_row,query_glass):
tower = [[0]*k for k in range(1,query_row+2)]
tower [0][0] = poured
for r in range (query_row):
  for c in range (r+1):
    excess= (tower [r][c] -1)/2.0
     if excess >0:
     tower [r+1][c] + = excess
      tower [r+1] [c+1] + = excess
return min(1, tower[query_row][query_glass])
Print (champagne_tower (1,1,1))
Print (champagne_tower (2,1,1))
```

Output:-

```
0.0000
0.5000
```