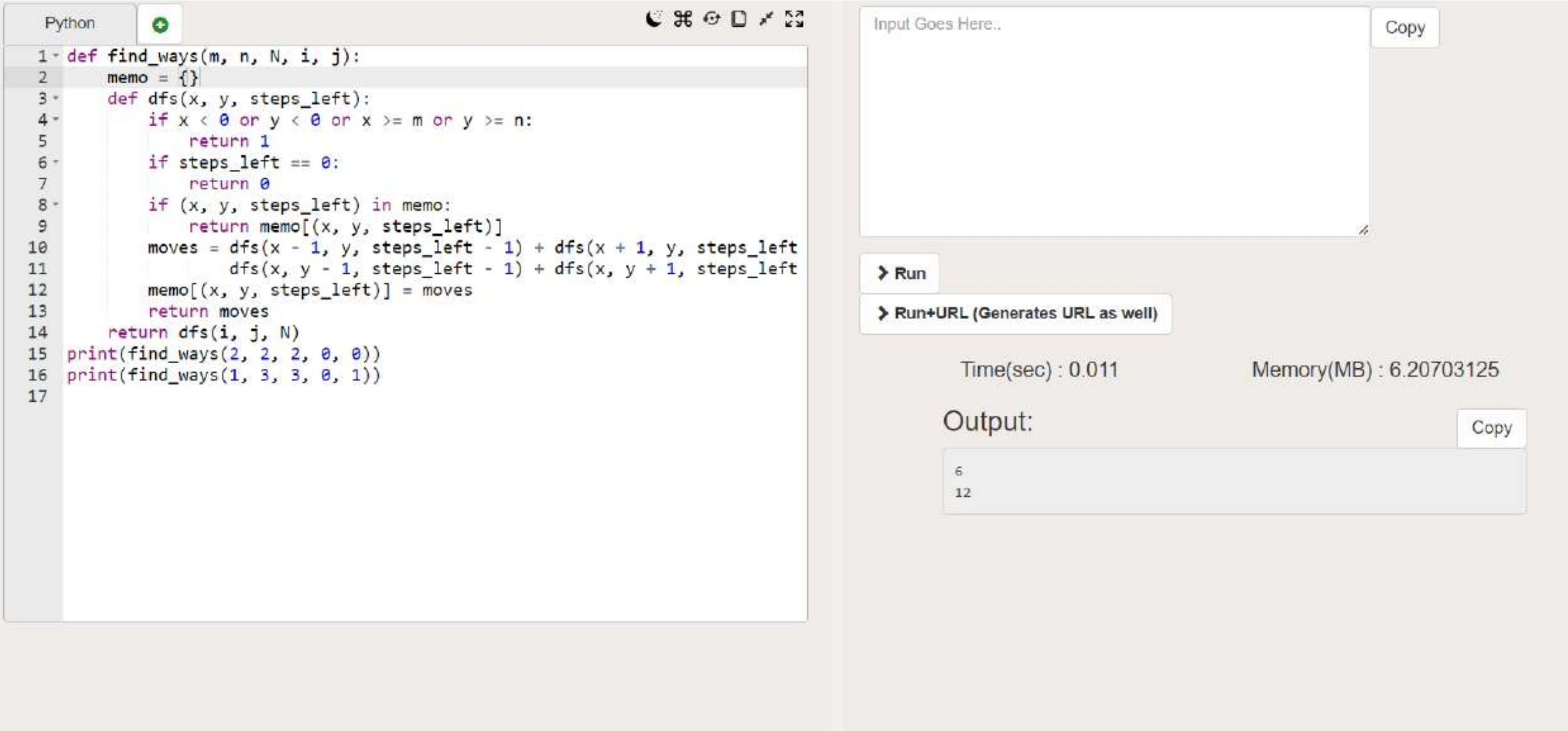


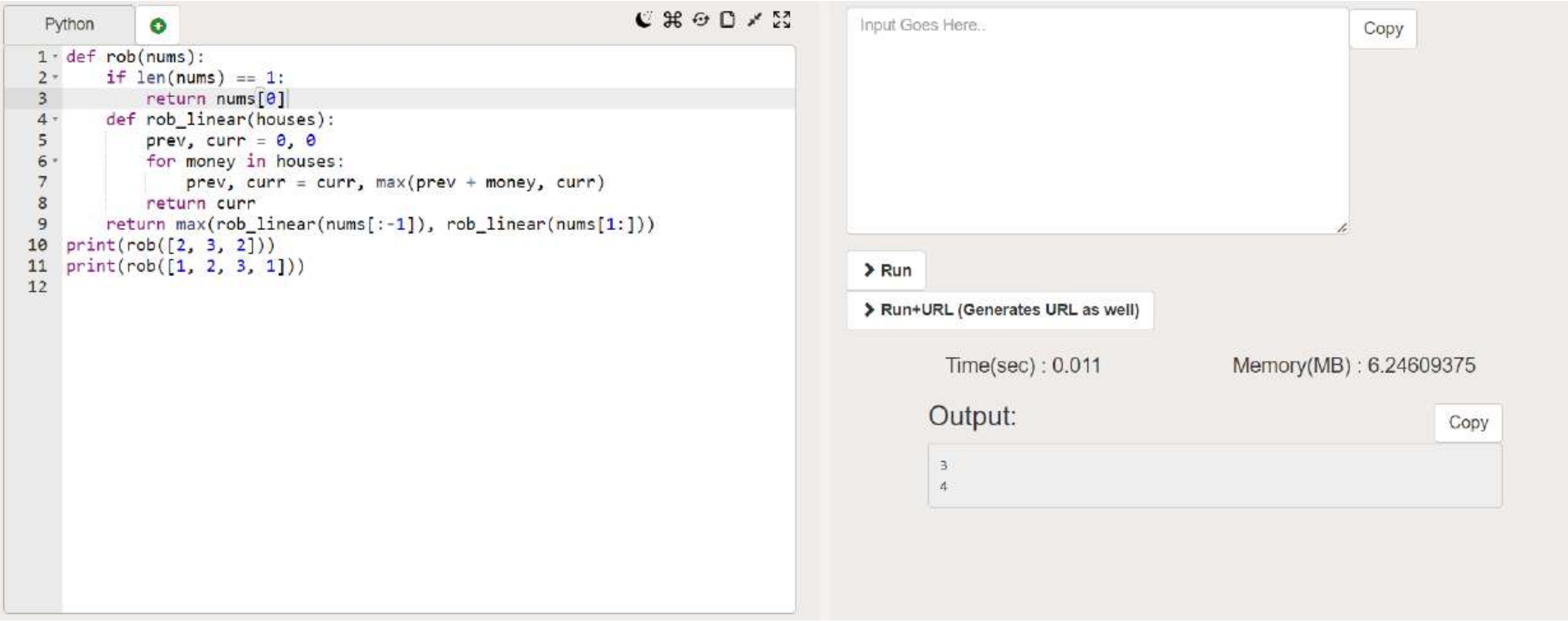
```python
def find_ways(m, n, N, i, j):
    memo = {}
    def dfs(x, y, steps_left):
        if x < 0 or y < 0 or x >= m or y >= n:
            return 1
        if steps_left == 0:
            return 0
        if (x, y, steps_left) in memo:
            return memo[(x, y, steps_left)]
        moves = dfs(x - 1, y, steps_left - 1) + dfs(x + 1, y, steps_left
                dfs(x, y - 1, steps_left - 1) + dfs(x, y + 1, steps_left
        memo[(x, y, steps_left)] = moves
        return moves
    return dfs(i, j, N)
print(find_ways(2, 2, 2, 0, 0))
print(find_ways(1, 3, 3, 0, 1))
```

Time(sec) : 0.011

Memory(MB) : 6.20703125

Output:

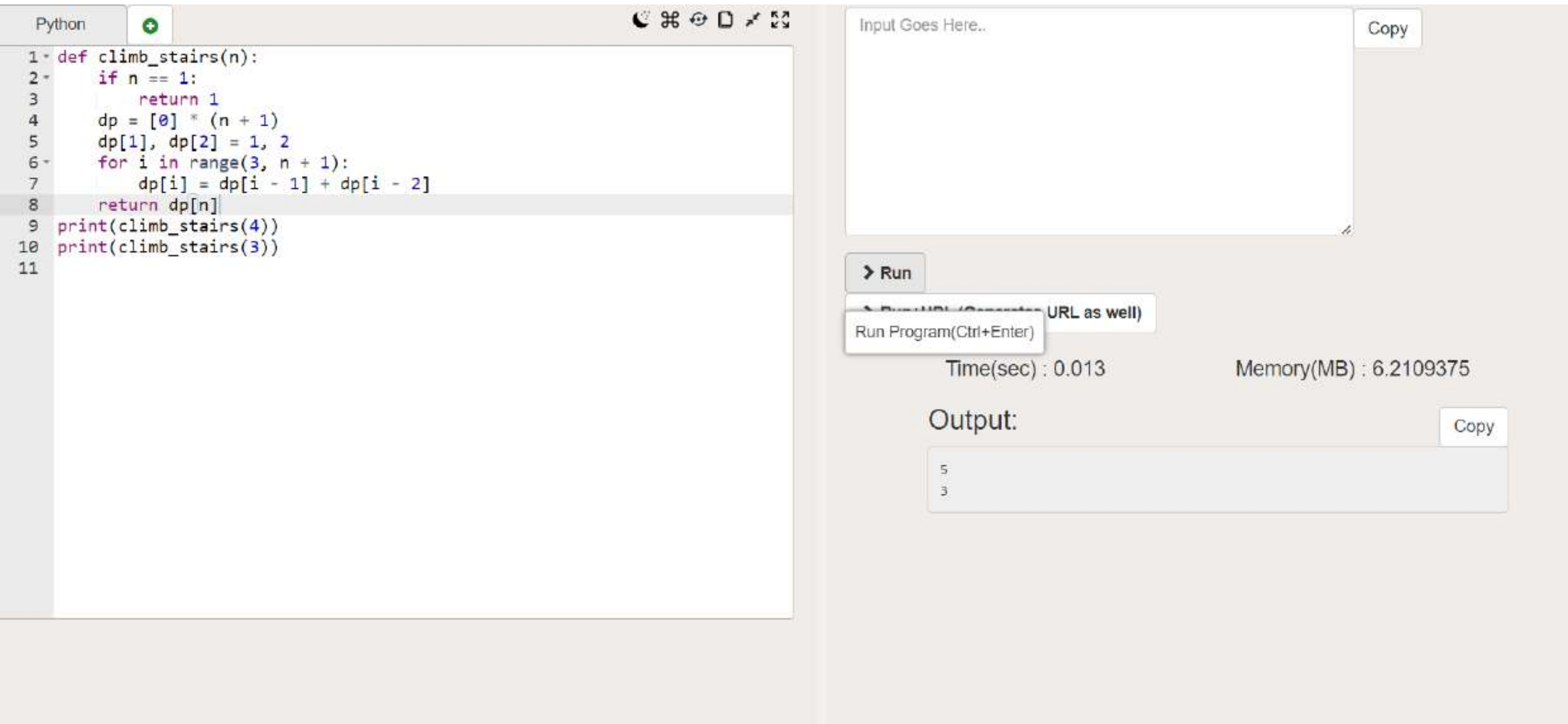
```
6
12
```



```
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:]))
print(rob([2, 3, 2]))
print(rob([1, 2, 3, 1]))
```

Time(sec) : 0.011 Memory(MB) : 6.24609375

```
3
4
```

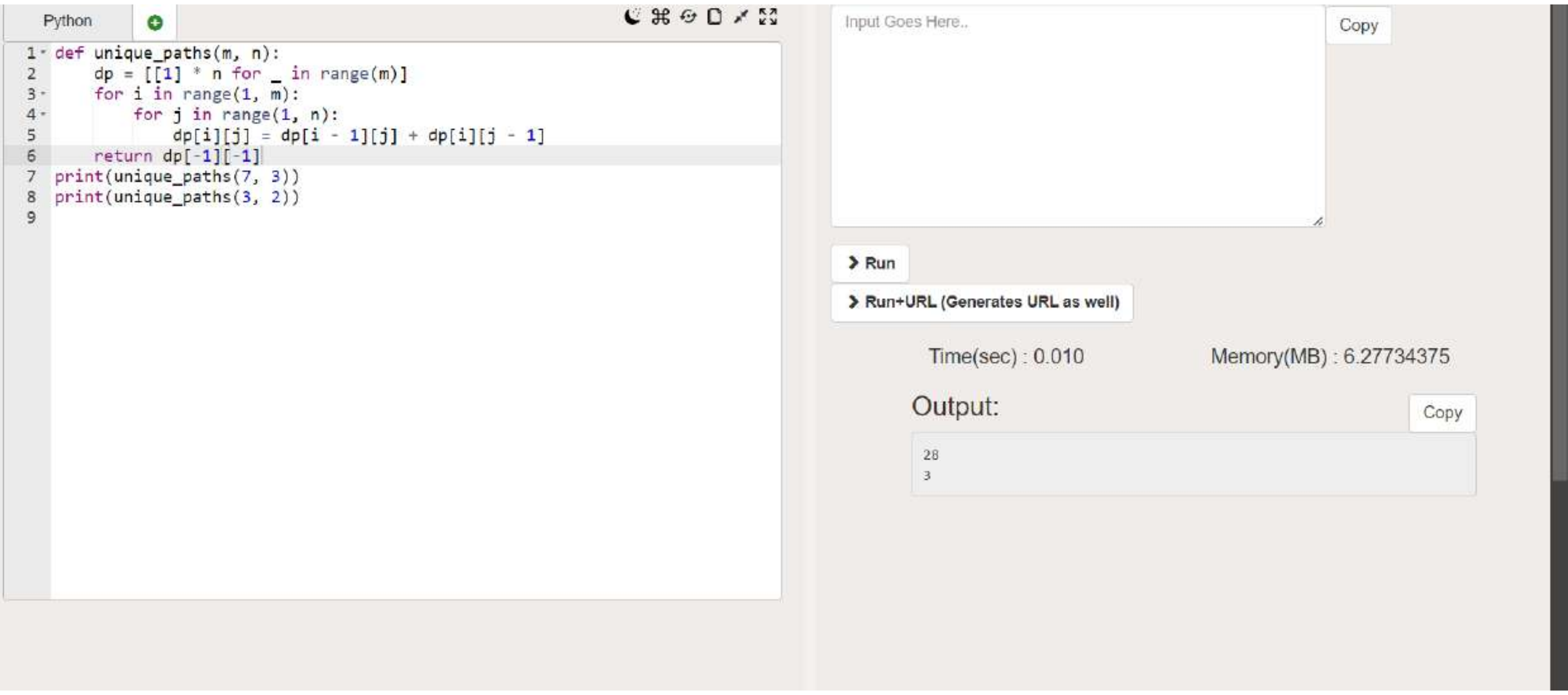


```python
def climb_stairs(n):
    if n == 1:
        return 1
    dp = [0] * (n + 1)
    dp[1], dp[2] = 1, 2
    for i in range(3, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]
    return dp[n]
print(climb_stairs(4))
print(climb_stairs(3))
```

Time(sec) : 0.013   Memory(MB) : 6.2109375

Output:

```
5
3
```



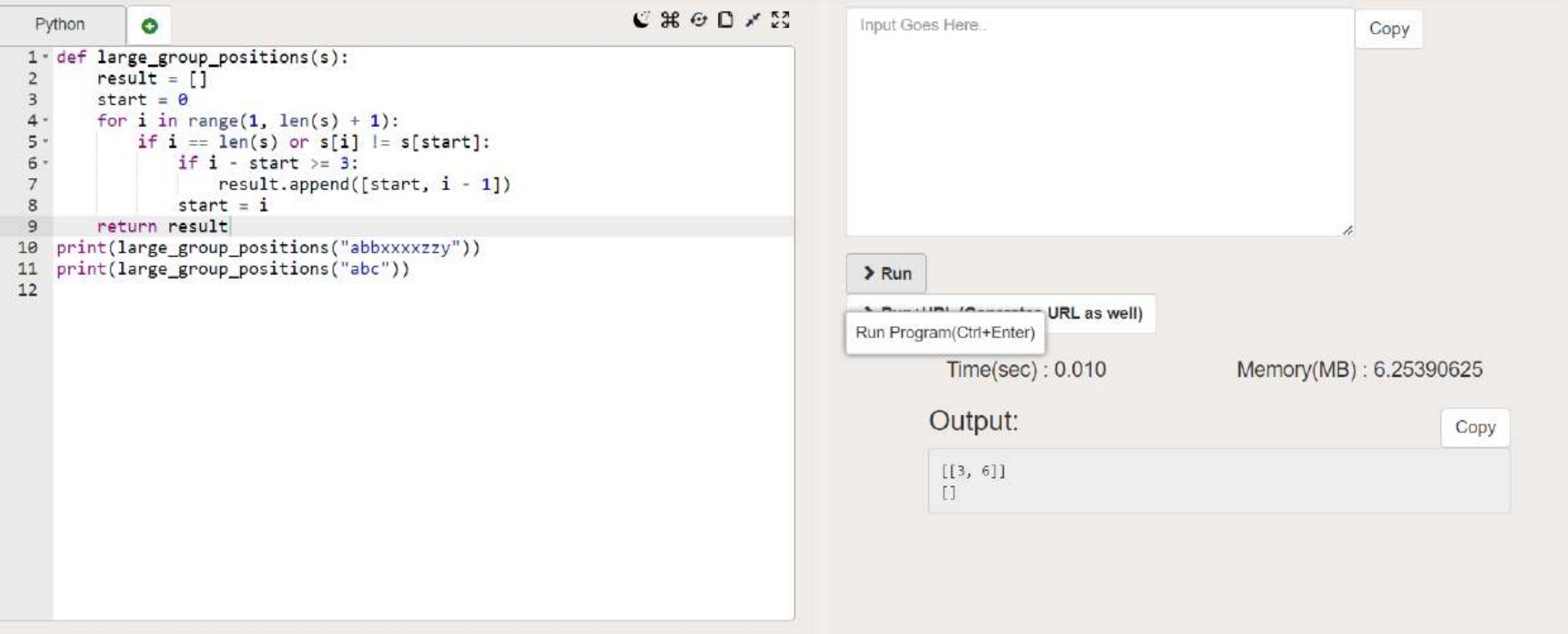
```python
def unique_paths(m, n):
    dp = [[1] * n for _ in range(m)]
    for i in range(1, m):
        for j in range(1, n):
            dp[i][j] = dp[i - 1][j] + dp[i][j - 1]
    return dp[-1][-1]
print(unique_paths(7, 3))
print(unique_paths(3, 2))
```

Time(sec) : 0.010

Memory(MB) : 6.27734375

Output:

```
28
3
```



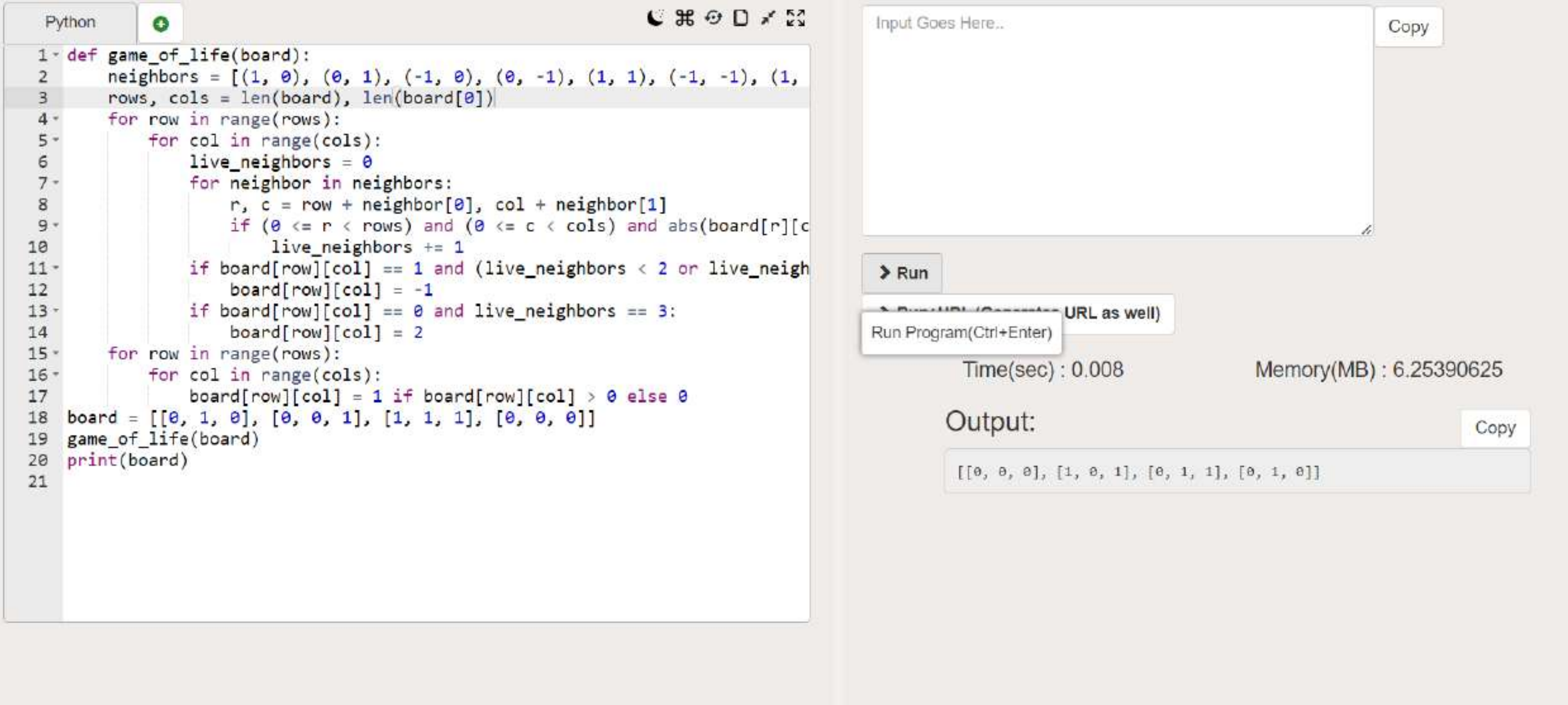
```python
def large_group_positions(s):
    result = []
    start = 0
    for i in range(1, len(s) + 1):
        if i == len(s) or s[i] != s[start]:
            if i - start >= 3:
                result.append([start, i - 1])
            start = i
    return result
print(large_group_positions("abbxxxxzzy"))
print(large_group_positions("abc"))
```

Time(sec) : 0.010

Memory(MB) : 6.25390625

Output:

```
[[3, 6]]
[]
```

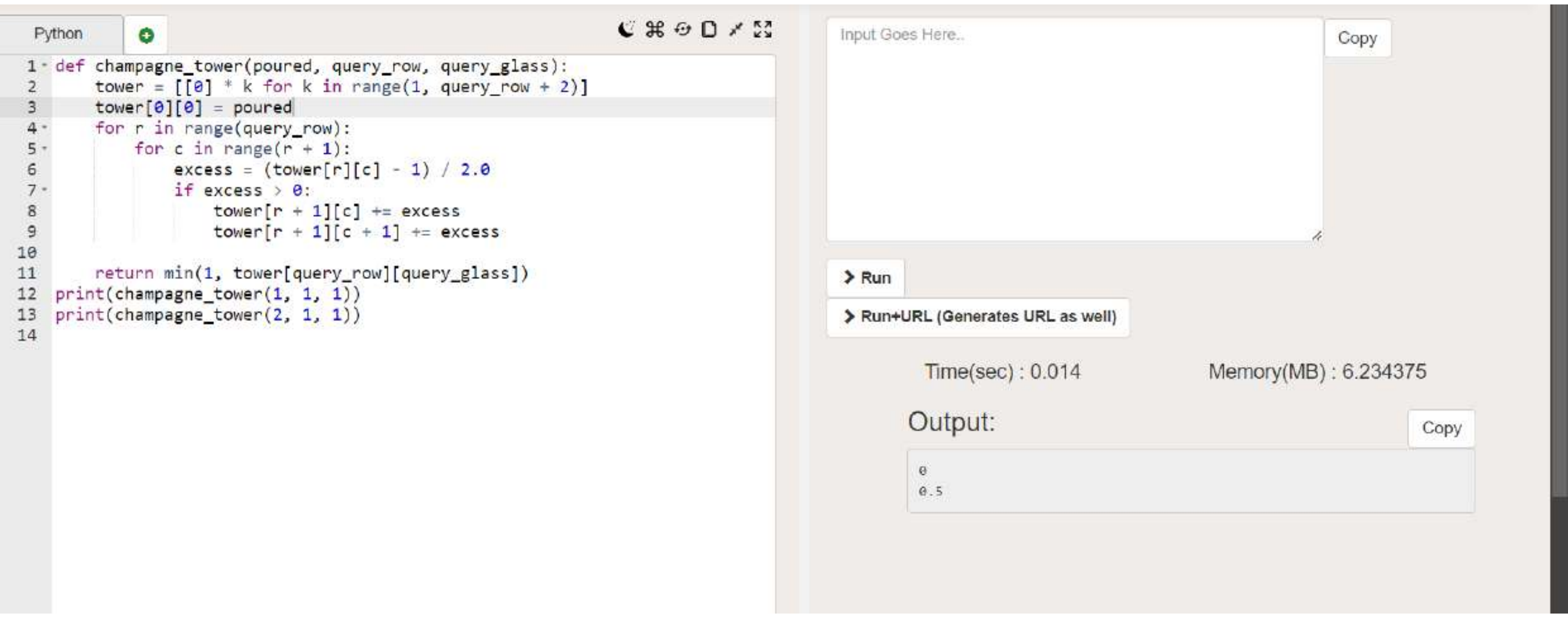


```python
def game_of_life(board):
    neighbors = [(1, 0), (0, 1), (-1, 0), (0, -1), (1, 1), (-1, -1), (1,
    rows, cols = len(board), len(board[0])
    for row in range(rows):
        for col in range(cols):
            live_neighbors = 0
            for neighbor in neighbors:
                r, c = row + neighbor[0], col + neighbor[1]
                if (0 <= r < rows) and (0 <= c < cols) and abs(board[r][c
                    live_neighbors += 1
            if board[row][col] == 1 and (live_neighbors < 2 or live_neigh
                board[row][col] = -1
            if board[row][col] == 0 and live_neighbors == 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]]
game_of_life(board)
print(board)
```

Time(sec) : 0.008 Memory(MB) : 6.25390625

Output:

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



```python
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))
```

Time(sec) : 0.014   Memory(MB) : 6.234375

Output:

```
0
0.5
```