TOPIC 1 : INTRODUCTION

1.Given an array of strings words, return the first palindromic string in the array. If there is no such string, return an empty string "". A string is palindromic if it reads the same forward and backward.

Example 1:

Input: words = ["abc","car","ada","racecar","cool"]

Output: "ada"

Explanation: The first string that is palindromic is "ada".

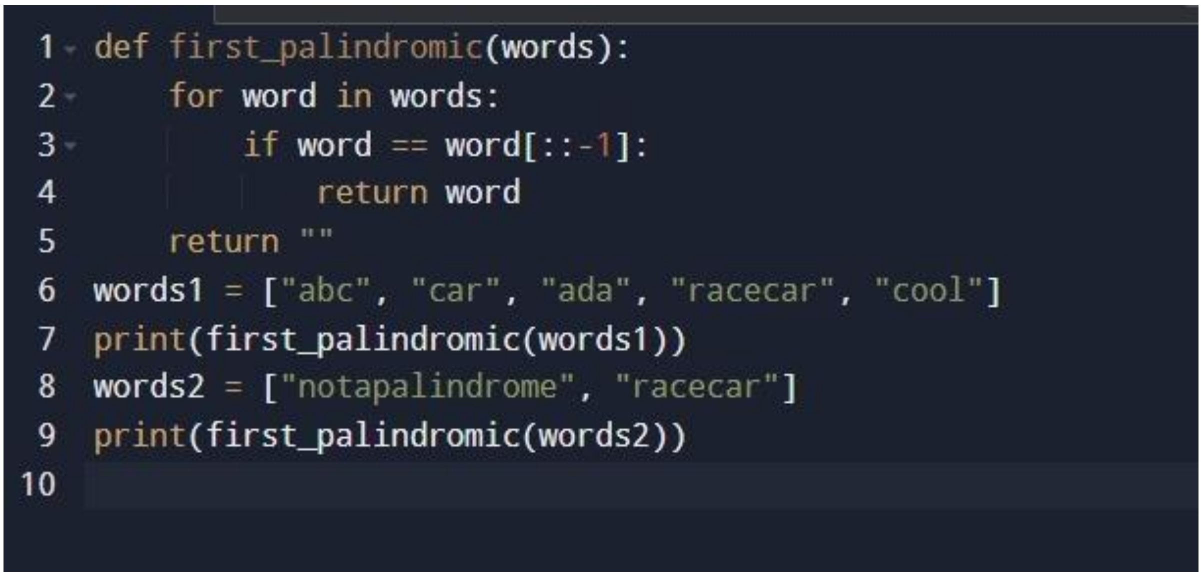
Note that "racecar" is also palindromic, but it is not the first.

Example 2:

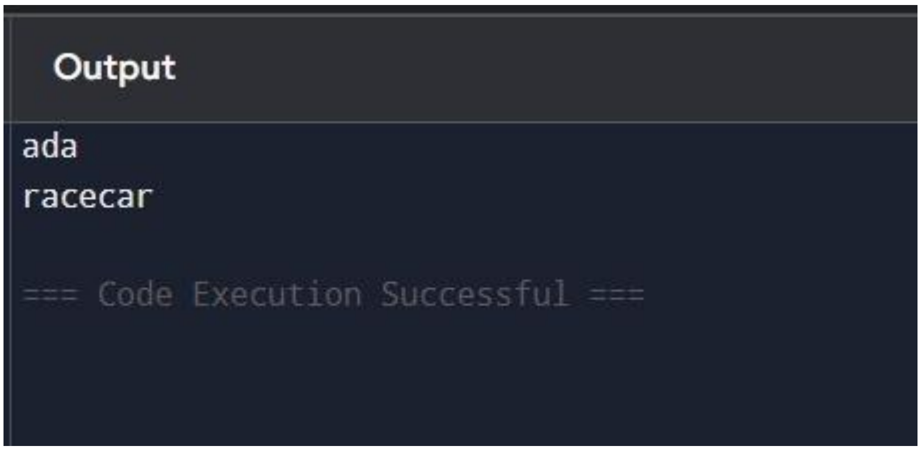
Input: words = ["notapalindrome","racecar"]

Output: "racecar"

Explanation: The first and only string that is palindromic is "racecar".



```python
def first_palindromic(words):
    for word in words:
        if word == word[::-1]:
            return word
    return ""
words1 = ["abc", "car", "ada", "racecar", "cool"]
print(first_palindromic(words1))
words2 = ["notapalindrome", "racecar"]
print(first_palindromic(words2))
```



```
ada
racecar

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

2.You are given two integer arrays nums1 and nums2 of sizes n and m, respectively. Calculate the following values: answer1 : the number of indices i such that nums1[i] exists in nums2. answer2 : the number of indices i such that nums2[i] exists in nums1 Return [answer1,answer2].

Example 1:

Input: nums1 = [2,3,2], nums2 = [1,2]

Output: [2,1]

Explanation:

Example 2:

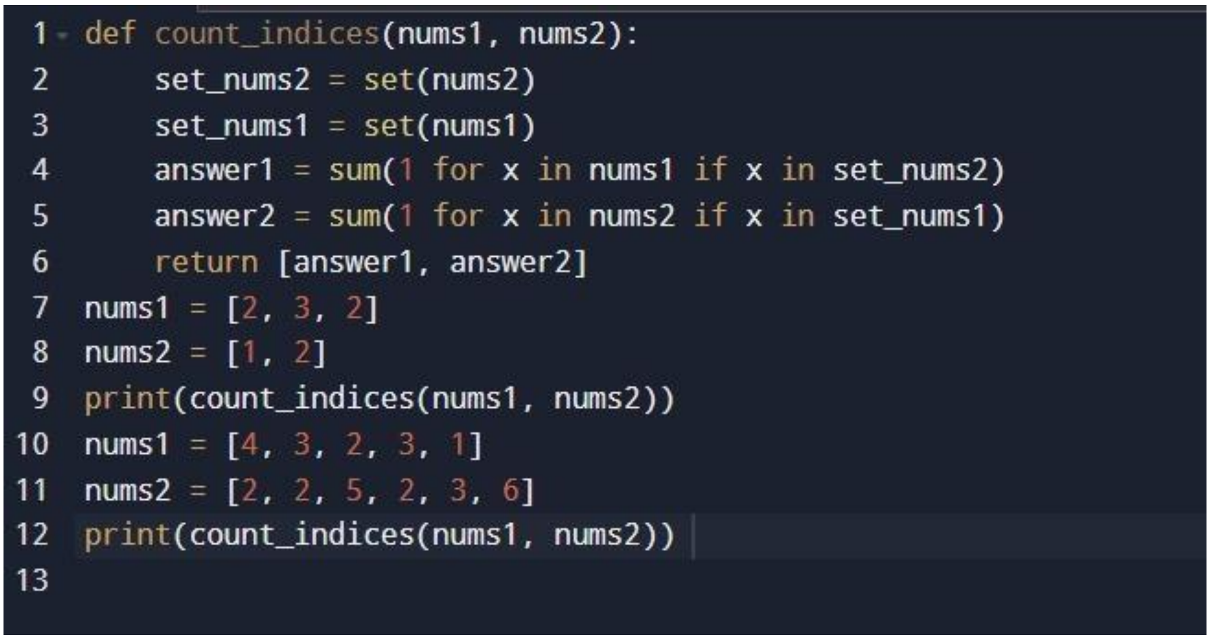
Input: nums1 = [4,3,2,3,1], nums2 = [2,2,5,2,3,6]

Output: [3,4]

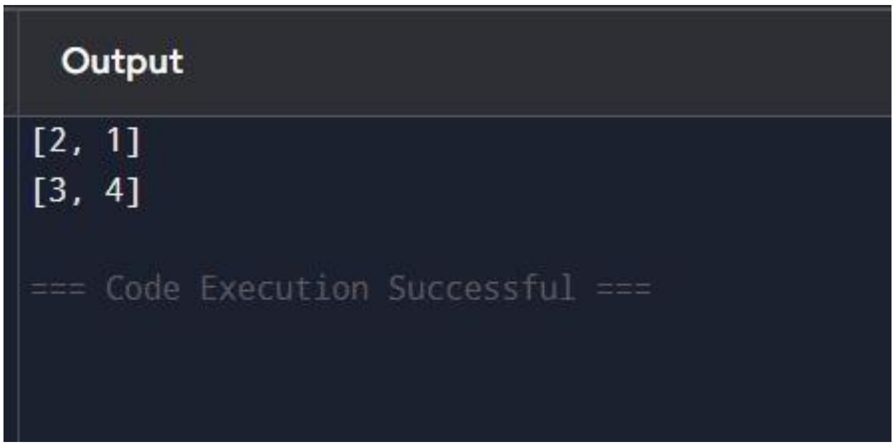
Explanation:

The elements at indices 1, 2, and 3 in nums1 exist in nums2 as well. So answer1 is 3.

The elements at indices 0, 1, 3, and 4 in nums2 exist in nums1. So answer2 is 4.



```python
def count_indices(nums1, nums2):
    set_nums2 = set(nums2)
    set_nums1 = set(nums1)
    answer1 = sum(1 for x in nums1 if x in set_nums2)
    answer2 = sum(1 for x in nums2 if x in set_nums1)
    return [answer1, answer2]
nums1 = [2, 3, 2]
nums2 = [1, 2]
print(count_indices(nums1, nums2))
nums1 = [4, 3, 2, 3, 1]
nums2 = [2, 2, 5, 2, 3, 6]
print(count_indices(nums1, nums2))
```



Output

```
[2, 1]
[3, 4]

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

3.You are given a 0-indexed integer array nums. The distinct count of a subarray of nums is defined as: Let nums[i..j] be a subarray of nums consisting of all the indices from i to j such that 0 <= i <= j < nums.length. Then the number of distinct values in nums[i..j] is called the distinct count of nums[i..j]. Return the sum of the squares of distinct counts of all subarrays of nums. A subarray is a contiguous non-empty sequence of elements within an array.

Example 1:

Input: nums = [1,2,1]

Output: 15

Explanation: Six possible subarrays are:

[1]: 1 distinct value

[2]: 1 distinct value

[1]: 1 distinct value

[1,2]: 2 distinct values

[2,1]: 2 distinct values

[1,2,1]: 2 distinct values

The sum of the squares of the distinct counts in all subarrays is equal to 12 + 12 + 12 + 22 + 22 + 22 = 15.

Example 2:

Input: nums = [1,1]

Output: 3

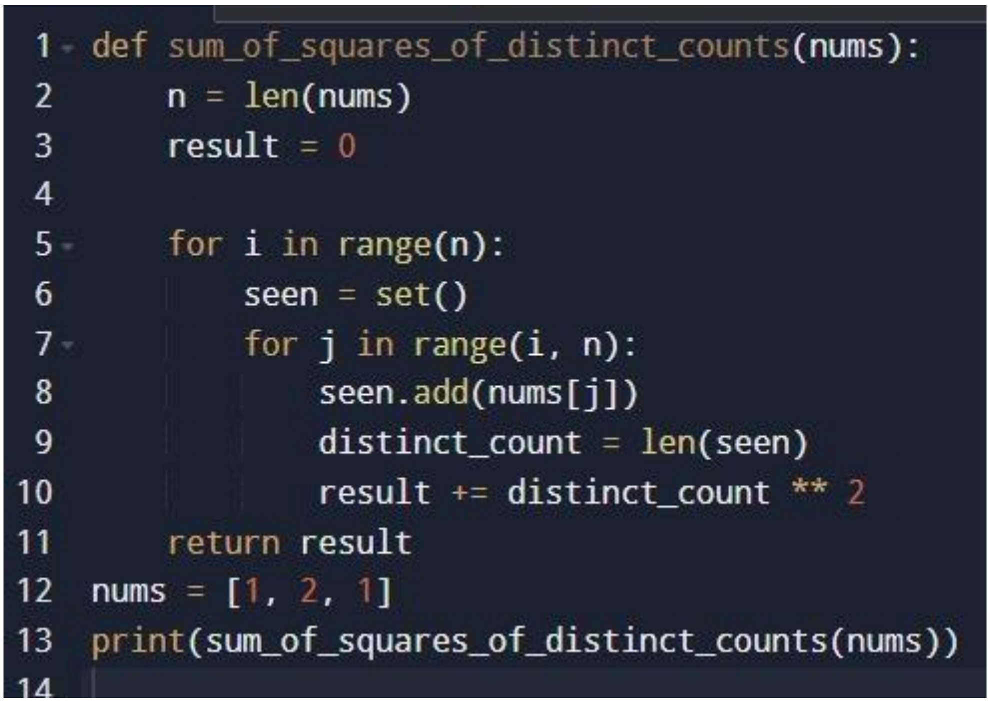
Explanation: Three possible subarrays are:

[1]: 1 distinct value

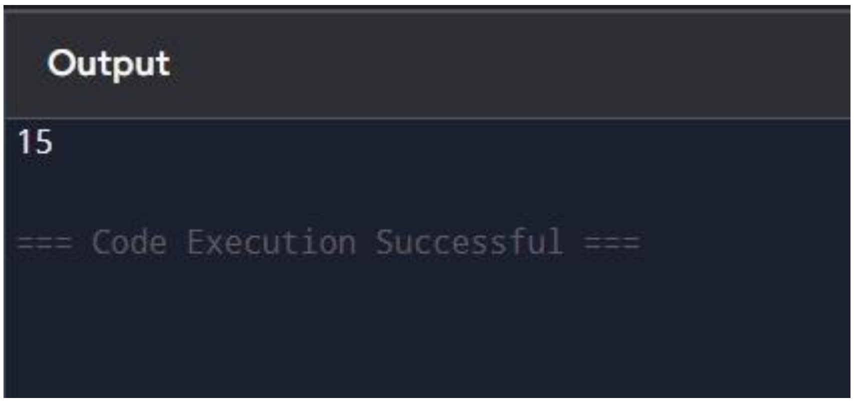
[1]: 1 distinct value

[1,1]: 1 distinct value

The sum of the squares of the distinct counts in all subarrays is equal to 12 + 12 + 12 = 3.



```python
def sum_of_squares_of_distinct_counts(nums):
    n = len(nums)
    result = 0

    for i in range(n):
        seen = set()
        for j in range(i, n):
            seen.add(nums[j])
            distinct_count = len(seen)
            result += distinct_count ** 2
    return result
nums = [1, 2, 1]
print(sum_of_squares_of_distinct_counts(nums))
```



Output

```
15

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

4. Given a 0-indexed integer array nums of length n and an integer k, return the number of pairs (i, j) where 0 <= i < j < n, such that nums[i] == nums[j] and (i * j) is divisible by k.

Example 1:

Input: nums = [3,1,2,2,2,1,3], k = 2

Output: 4

Explanation:

There are 4 pairs that meet all the requirements:

- nums[0] == nums[6], and 0 * 6 == 0, which is divisible by 2.

- nums[2] == nums[3], and 2 * 3 == 6, which is divisible by 2.

- nums[2] == nums[4], and 2 * 4 == 8, which is divisible by 2.

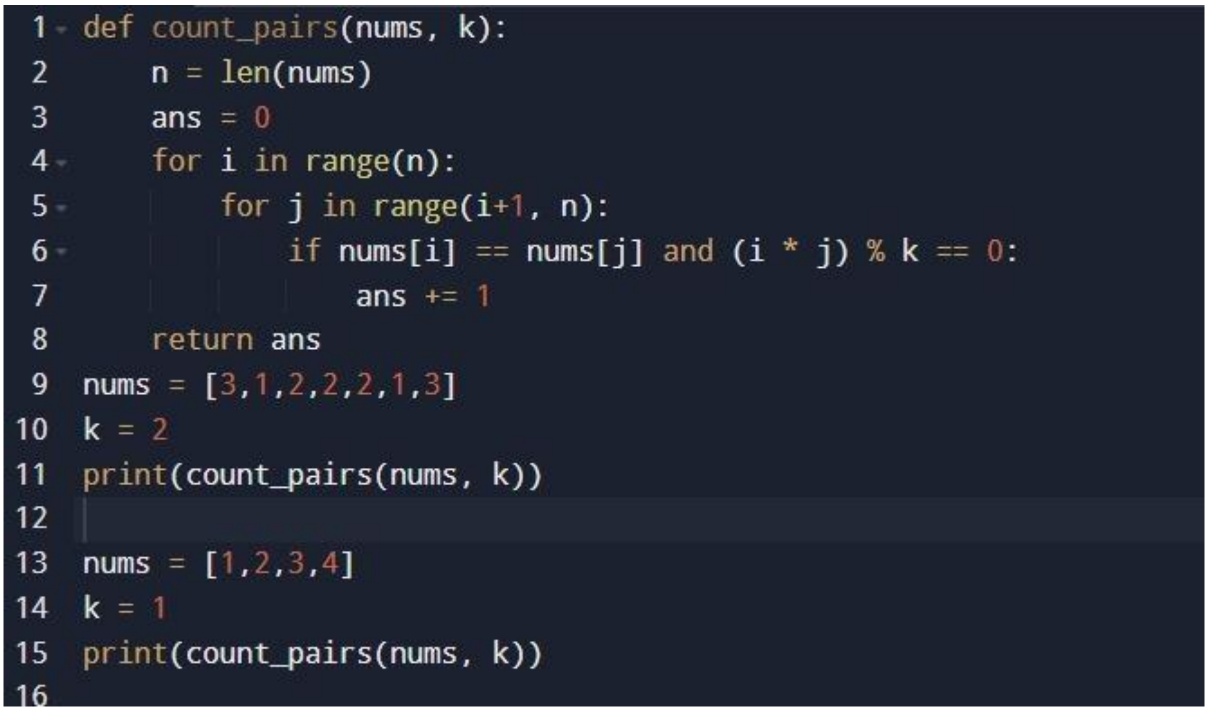
- nums[3] == nums[4], and 3 * 4 == 12, which is divisible by 2.

Example 2:

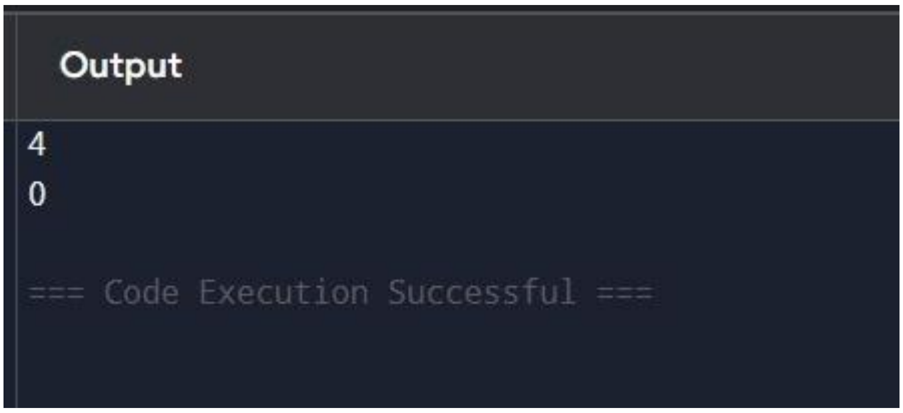
Input: nums = [1,2,3,4], k = 1

Output: 0

Explanation: Since no value in nums is repeated, there are no pairs (i,j) that meet all the requirements.



```python
def count_pairs(nums, k):
    n = len(nums)
    ans = 0
    for i in range(n):
        for j in range(i+1, n):
            if nums[i] == nums[j] and (i * j) % k == 0:
                ans += 1
    return ans
nums = [3,1,2,2,2,1,3]
k = 2
print(count_pairs(nums, k))

nums = [1,2,3,4]
k = 1
print(count_pairs(nums, k))

```



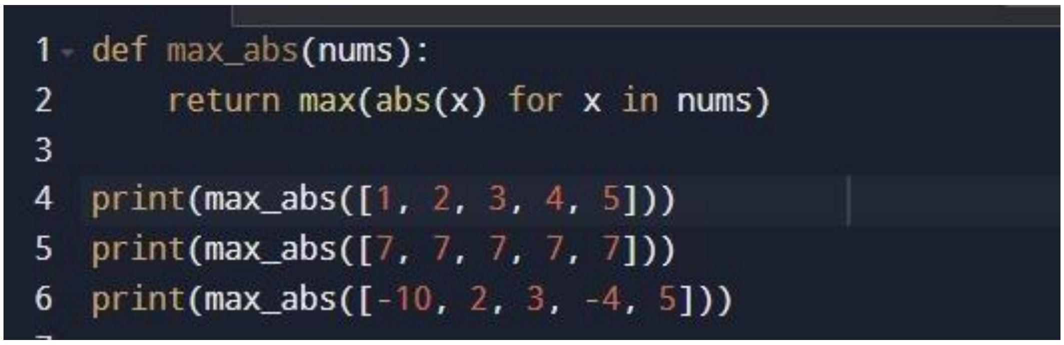
```
Output
4
0

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

5. Write a program FOR THE BELOW TEST CASES with least time complexity Test Cases: -

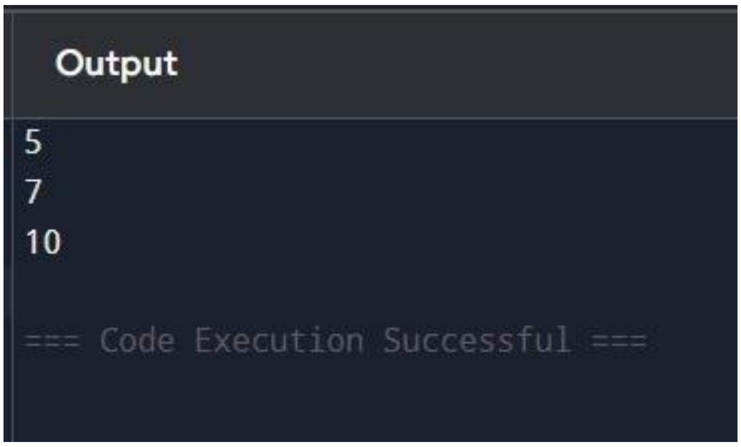
Input: {1, 2, 3, 4, 5} Expected Output: 5

Input: {7, 7, 7, 7, 7} Expected Output: 7

Input: {-10, 2, 3, -4, 5} Expected Output: 5



```python
def max_abs(nums):
    return max(abs(x) for x in nums)

print(max_abs([1, 2, 3, 4, 5]))
print(max_abs([7, 7, 7, 7, 7]))
print(max_abs([-10, 2, 3, -4, 5]))
```



```
Output
5
7
10

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

6. You have an algorithm that process a list of numbers. It firsts sorts the list using an efficient sorting algorithm and then finds the maximum element in sorted list. Write the code for the same.

Test Cases

1. Empty List

1. Input: []

2. Expected Output: None or an appropriate message indicating that the list is empty.

2. Single Element List

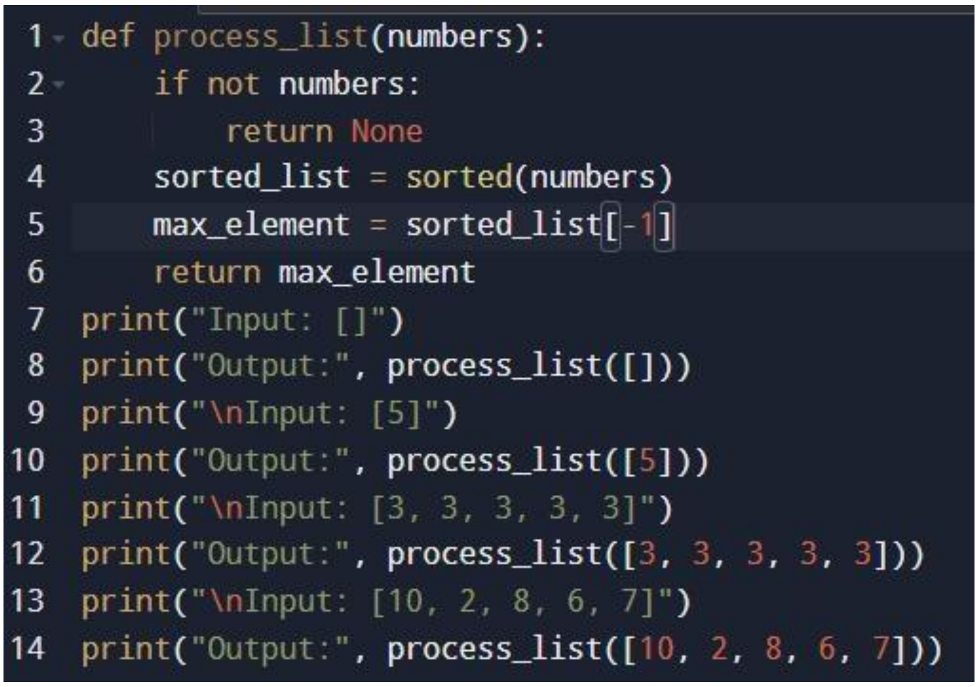
1. Input: [5]

2. Expected Output: 5

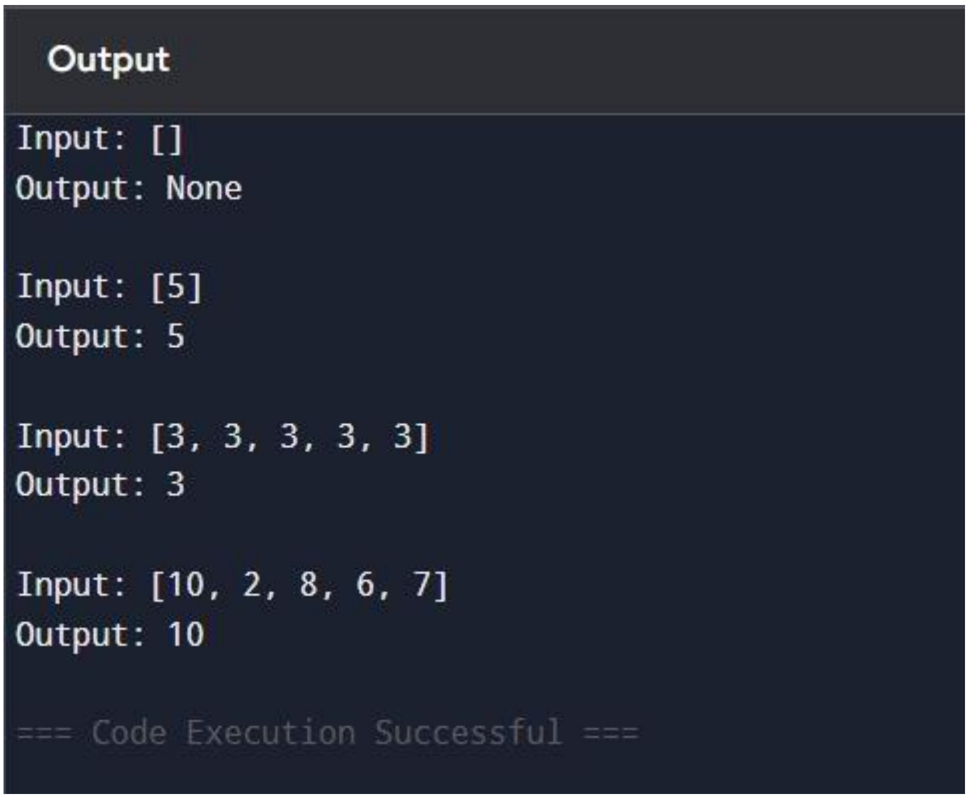
3. All Elements are the Same

1. Input: [3, 3, 3, 3, 3]

2. Expected Output: 3



```python
def process_list(numbers):
    if not numbers:
        return None
    sorted_list = sorted(numbers)
    max_element = sorted_list[-1]
    return max_element
print("Input: []")
print("Output:", process_list([]))
print("\nInput: [5]")
print("Output:", process_list([5]))
print("\nInput: [3, 3, 3, 3, 3]")
print("Output:", process_list([3, 3, 3, 3, 3]))
print("\nInput: [10, 2, 8, 6, 7]")
print("Output:", process_list([10, 2, 8, 6, 7]))
```



```
Input: []
Output: None

Input: [5]
Output: 5

Input: [3, 3, 3, 3, 3]
Output: 3

Input: [10, 2, 8, 6, 7]
Output: 10

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

7. Write a program that takes an input list of n numbers and creates a new list containing only the unique elements from the original list. What is the space complexity of the algorithm?

Test Cases

Some Duplicate Elements

Input: [3, 7, 3, 5, 2, 5, 9, 2]

Expected Output: [3, 7, 5, 2, 9] (Order may vary based on the algorithm used)

Negative and Positive Numbers

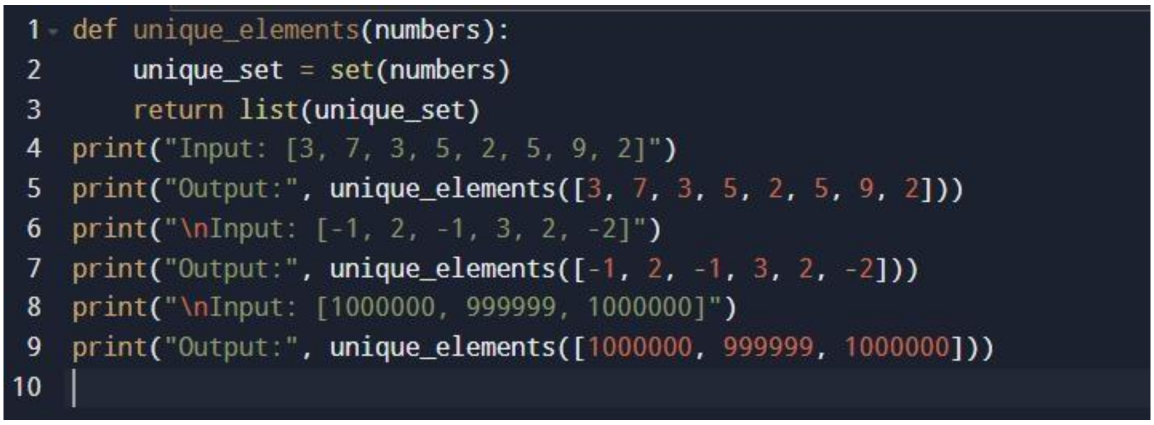
Input: [-1, 2, -1, 3, 2, -2]

Expected Output: [-1, 2, 3, -2] (Order may vary)

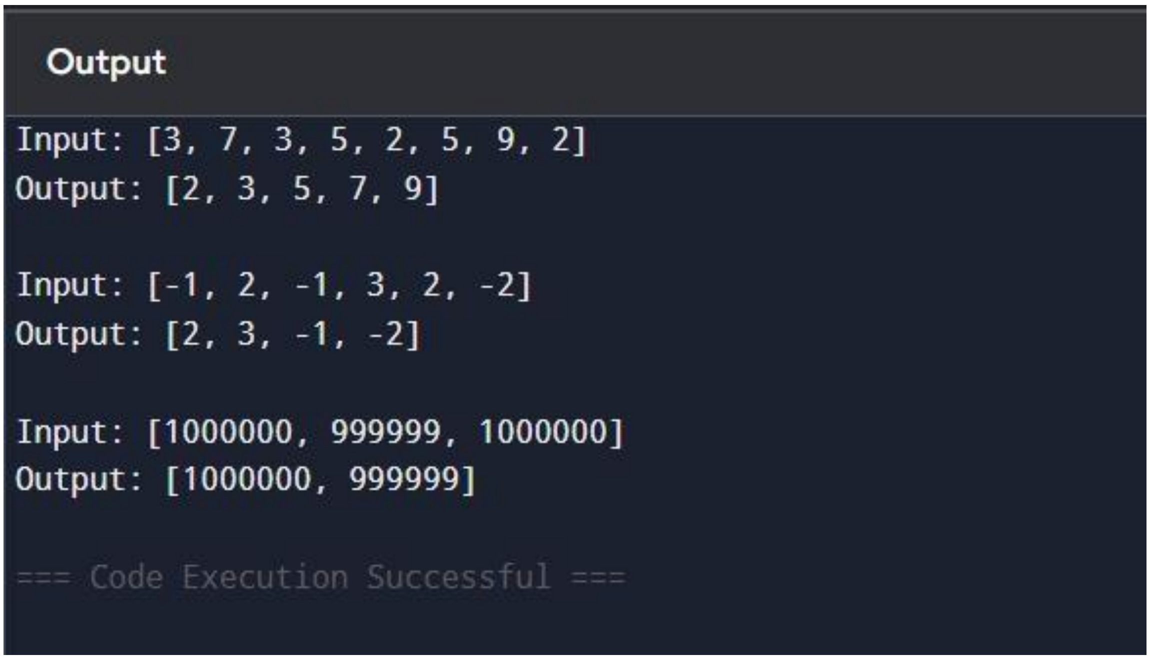
List with Large Numbers

Input: [1000000, 999999, 1000000]

Expected Output: [1000000, 999999]

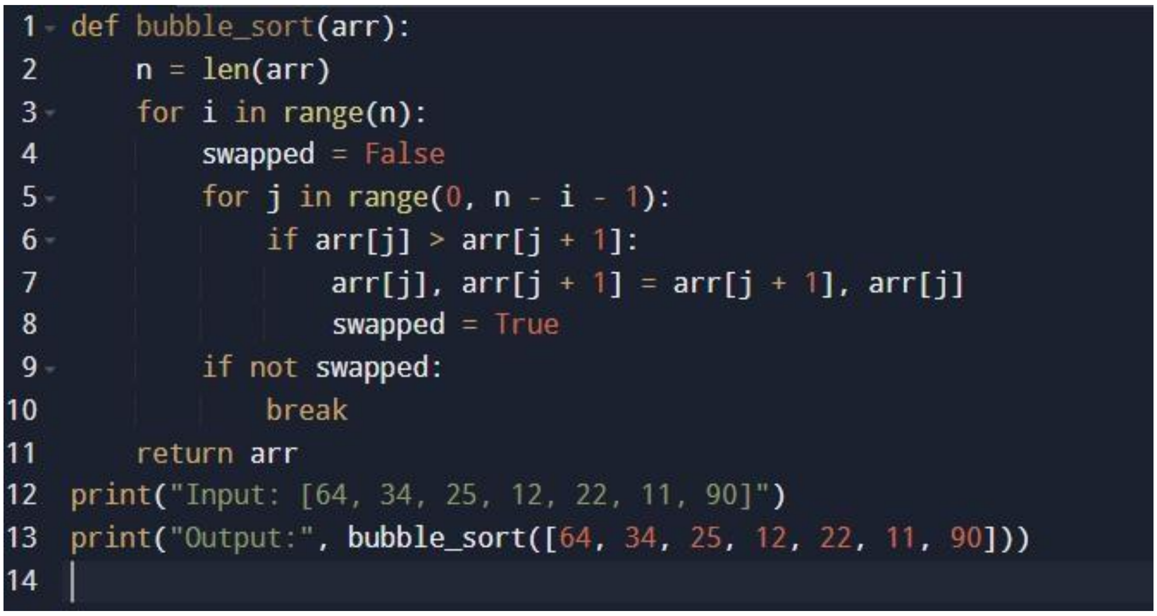


```python
def unique_elements(numbers):
    unique_set = set(numbers)
    return list(unique_set)
print("Input: [3, 7, 3, 5, 2, 5, 9, 2]")
print("Output:", unique_elements([3, 7, 3, 5, 2, 5, 9, 2]))
print("\nInput: [-1, 2, -1, 3, 2, -2]")
print("Output:", unique_elements([-1, 2, -1, 3, 2, -2]))
print("\nInput: [1000000, 999999, 1000000]")
print("Output:", unique_elements([1000000, 999999, 1000000]))
```

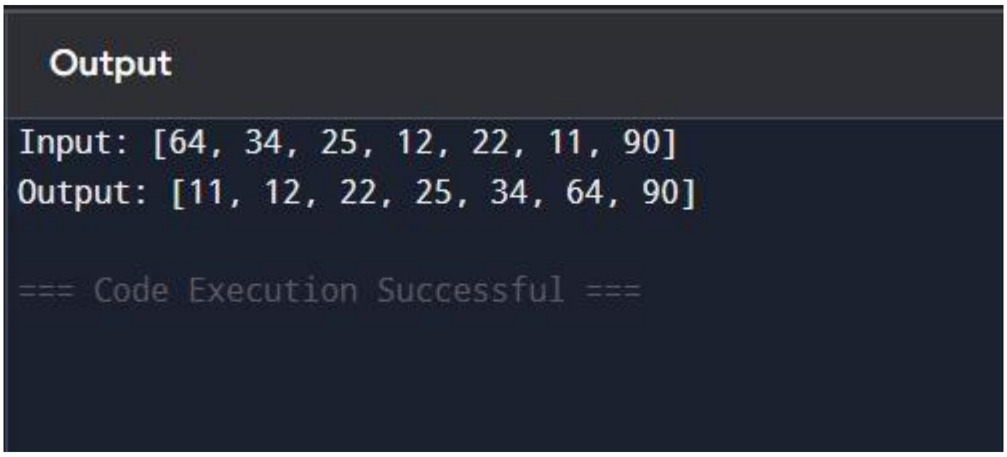


```
Input: [3, 7, 3, 5, 2, 5, 9, 2]
Output: [2, 3, 5, 7, 9]

Input: [-1, 2, -1, 3, 2, -2]
Output: [2, 3, -1, -2]

Input: [1000000, 999999, 1000000]
Output: [1000000, 999999]

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

8. Sort an array of integers using the bubble sort technique. Analyze its time complexity using Big-O notation. Write the code



```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
print("Input: [64, 34, 25, 12, 22, 11, 90]")
print("Output:", bubble_sort([64, 34, 25, 12, 22, 11, 90]))
```



Output

```
Input: [64, 34, 25, 12, 22, 11, 90]
Output: [11, 12, 22, 25, 34, 64, 90]

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

9. Checks if a given number x exists in a sorted array arr using binary search. Analyze its time complexity using Big-O notation.

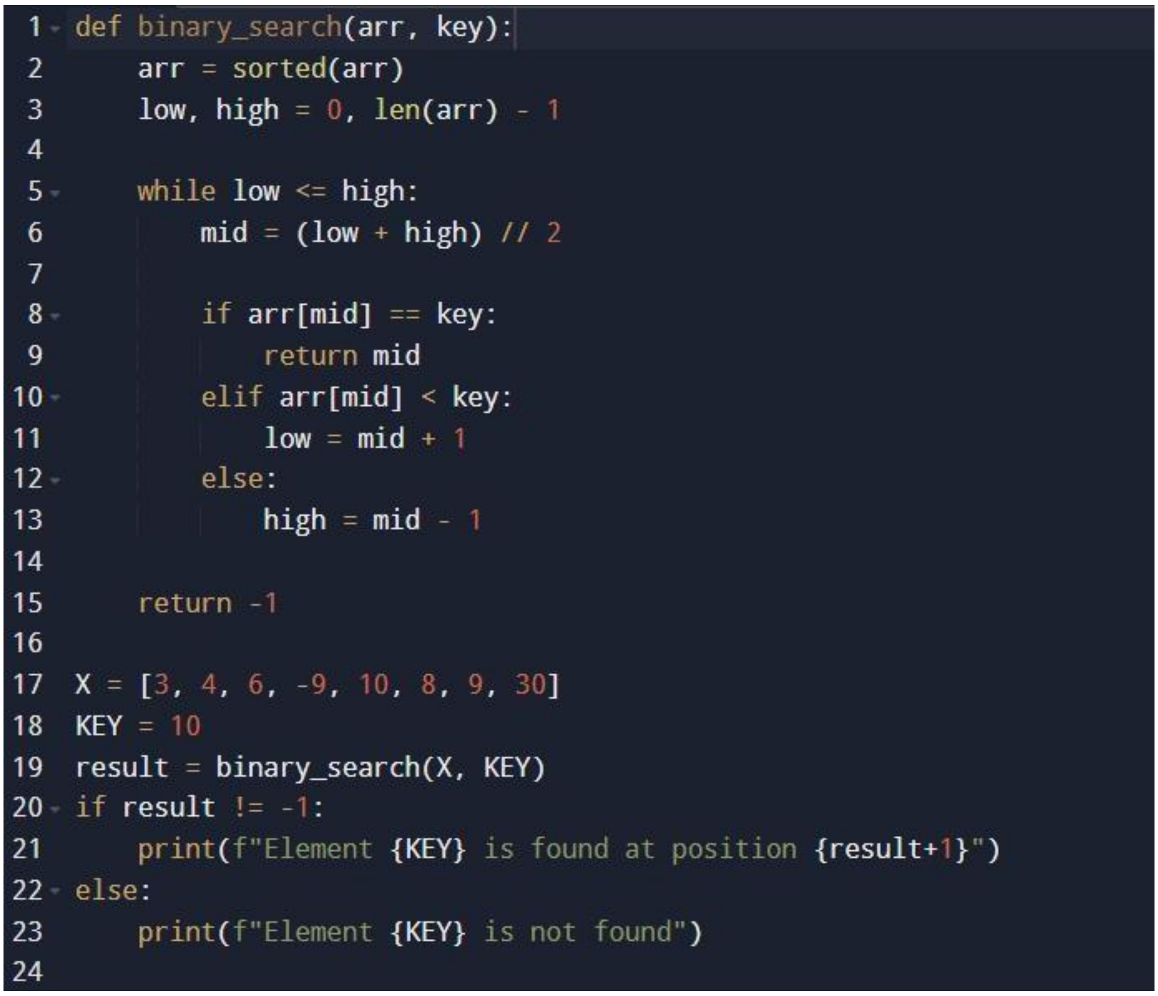
Test Case:

Example X={ 3,4,6,-9,10,8,9,30} KEY=10

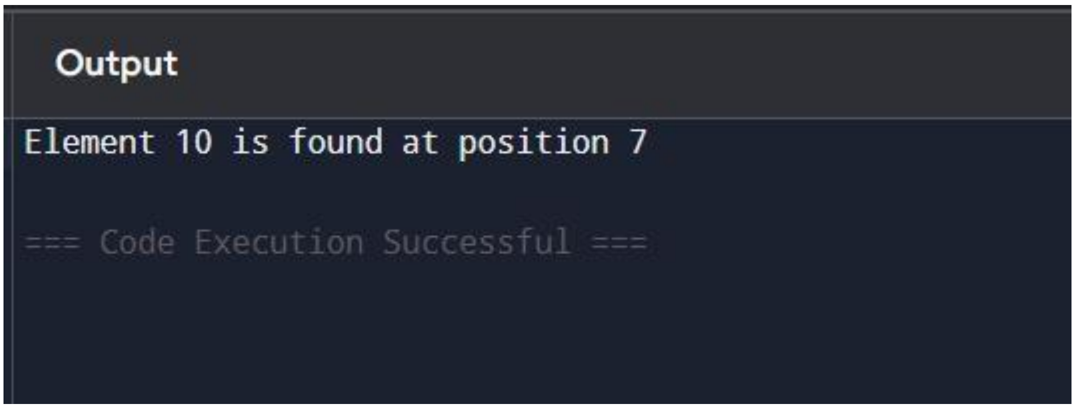
Output: Element 10 is found at position 5

Example X={ 3,4,6,-9,10,8,9,30} KEY=100

Output : Element 100 is not found



```python
def binary_search(arr, key):
    arr = sorted(arr)
    low, high = 0, len(arr) - 1

    while low <= high:
        mid = (low + high) // 2

        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1

    return -1

X = [3, 4, 6, -9, 10, 8, 9, 30]
KEY = 10
result = binary_search(X, KEY)
if result != -1:
    print(f"Element {KEY} is found at position {result+1}")
else:
    print(f"Element {KEY} is not found")

```

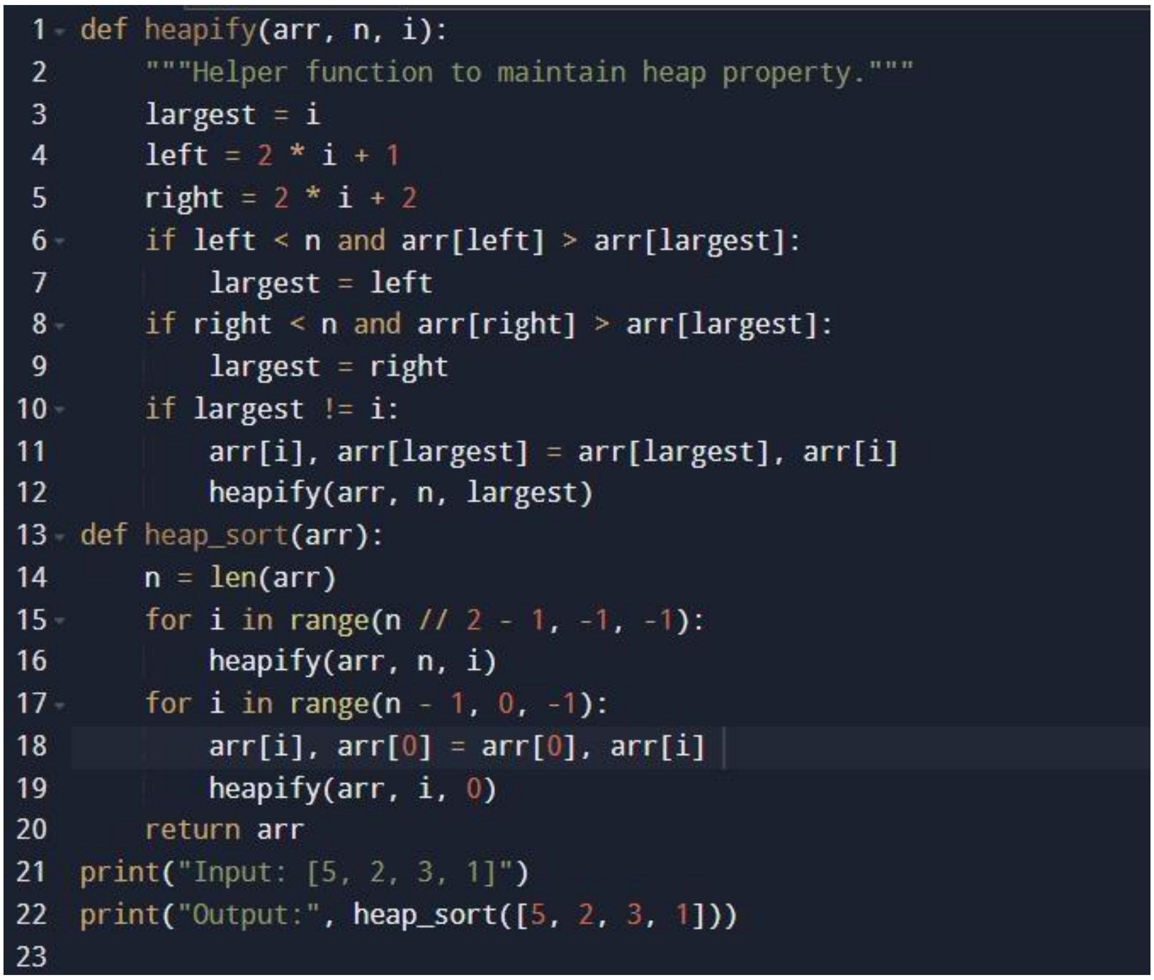


Output

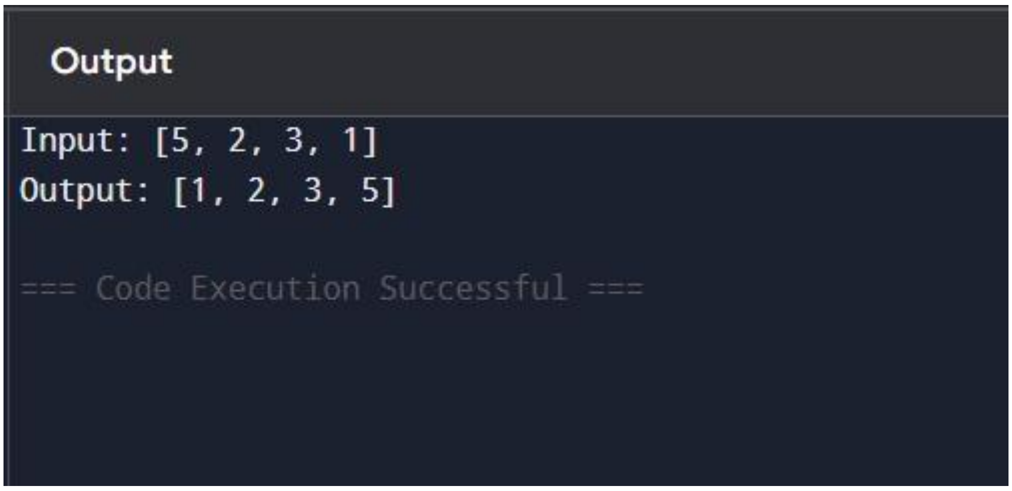
```
Element 10 is found at position 7


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

10. Given an array of integers nums, sort the array in ascending order and return it. You must solve the problem without using any built-in

functions in O(nlog(n)) time complexity and with the smallest space complexity possible.



```python
def heapify(arr, n, i):
    """Helper function to maintain heap property."""
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2
    if left < n and arr[left] > arr[largest]:
        largest = left
    if right < n and arr[right] > arr[largest]:
        largest = right
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)
def heap_sort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
    return arr
print("Input: [5, 2, 3, 1]")
print("Output:", heap_sort([5, 2, 3, 1]))

```



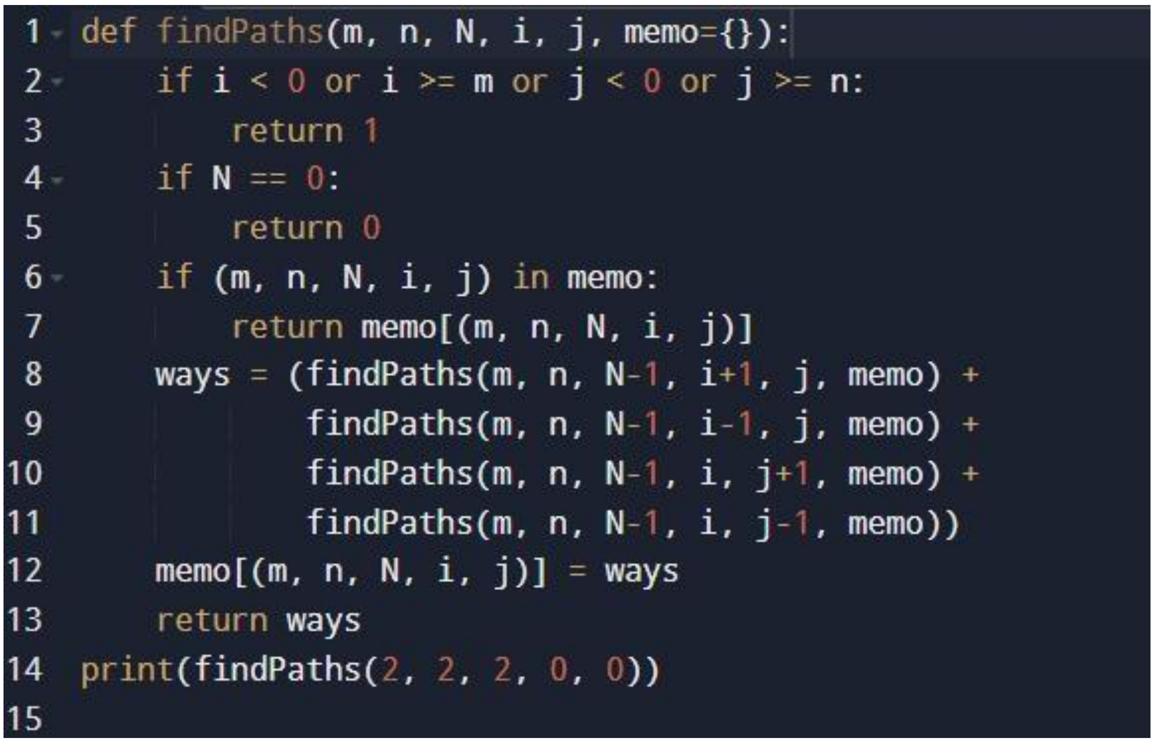
Output

```
Input: [5, 2, 3, 1]
Output: [1, 2, 3, 5]

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

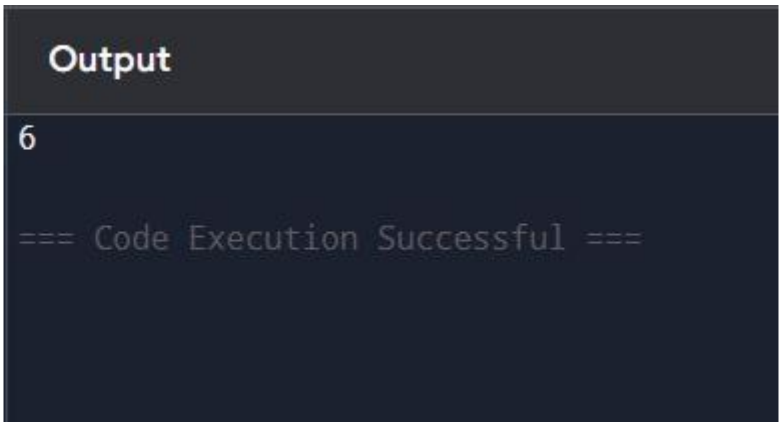
11. Given an m x n grid and a ball at a starting cell, find the number of ways to move the ball out of the grid boundary in exactly N steps.

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



```python
def findPaths(m, n, N, i, j, memo={}):
    if i < 0 or i >= m or j < 0 or j >= n:
        return 1
    if N == 0:
        return 0
    if (m, n, N, i, j) in memo:
        return memo[(m, n, N, i, j)]
    ways = (findPaths(m, n, N-1, i+1, j, memo) +
            findPaths(m, n, N-1, i-1, j, memo) +
            findPaths(m, n, N-1, i, j+1, memo) +
            findPaths(m, n, N-1, i, j-1, memo))
    memo[(m, n, N, i, j)] = ways
    return ways
print(findPaths(2, 2, 2, 0, 0))
```



Output

```
6

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

12. 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.

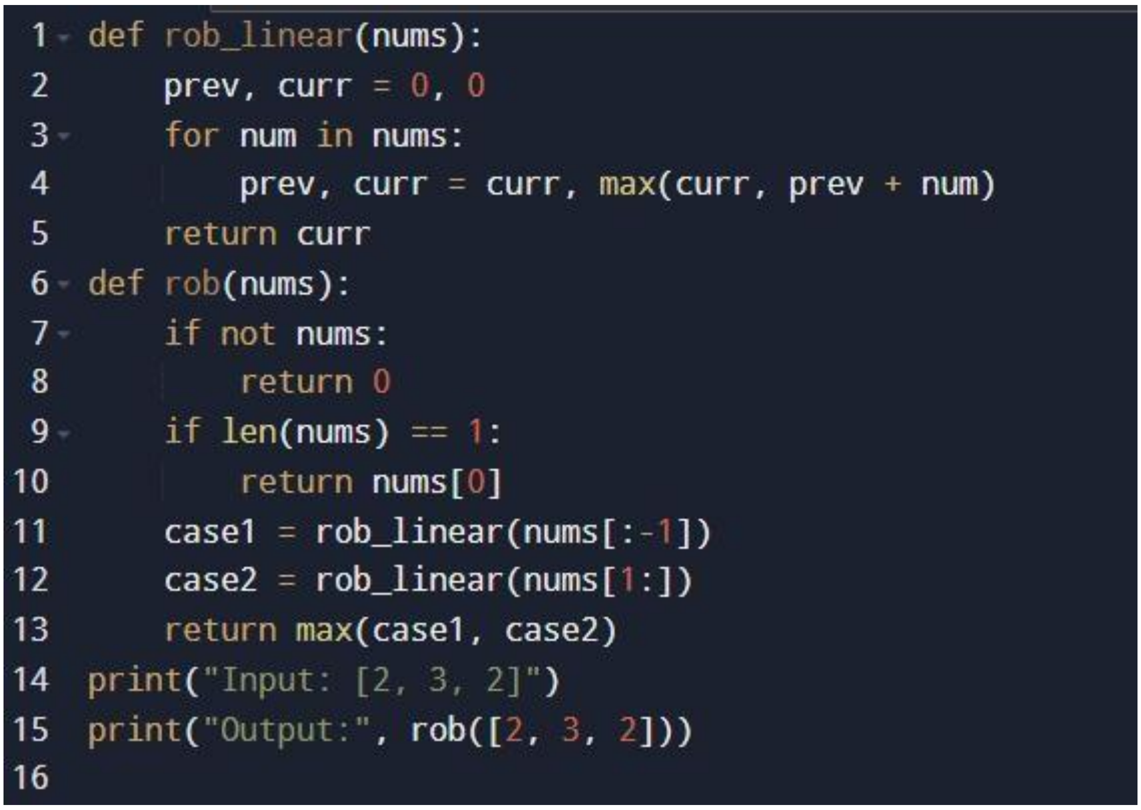
Examples:

Input : nums = [2, 3, 2]

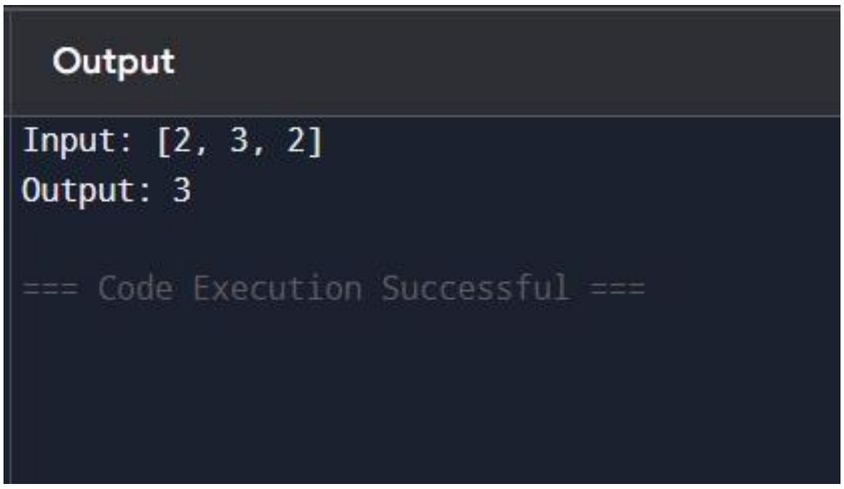
Output : The maximum money you can rob without alerting the police is 3(robbing house 1).

(ii) Input : nums = [1, 2, 3, 1]

Output : The maximum money you can rob without alerting the police is 4 (robbing house 1 and house 3).



```python
def rob_linear(nums):
    prev, curr = 0, 0
    for num in nums:
        prev, curr = curr, max(curr, prev + num)
    return curr
def rob(nums):
    if not nums:
        return 0
    if len(nums) == 1:
        return nums[0]
    case1 = rob_linear(nums[:-1])
    case2 = rob_linear(nums[1:])
    return max(case1, case2)
print("Input: [2, 3, 2]")
print("Output:", rob([2, 3, 2]))
```



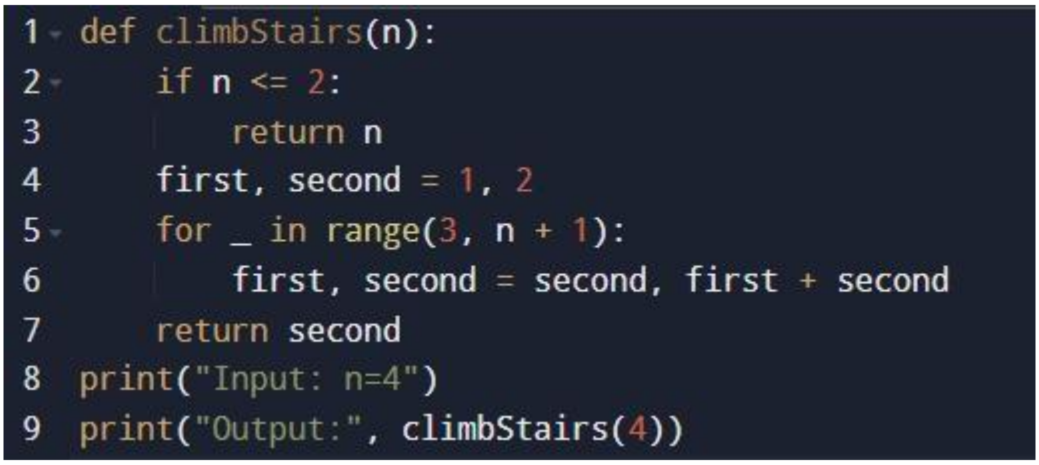
```
Input: [2, 3, 2]
Output: 3

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

13. 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?

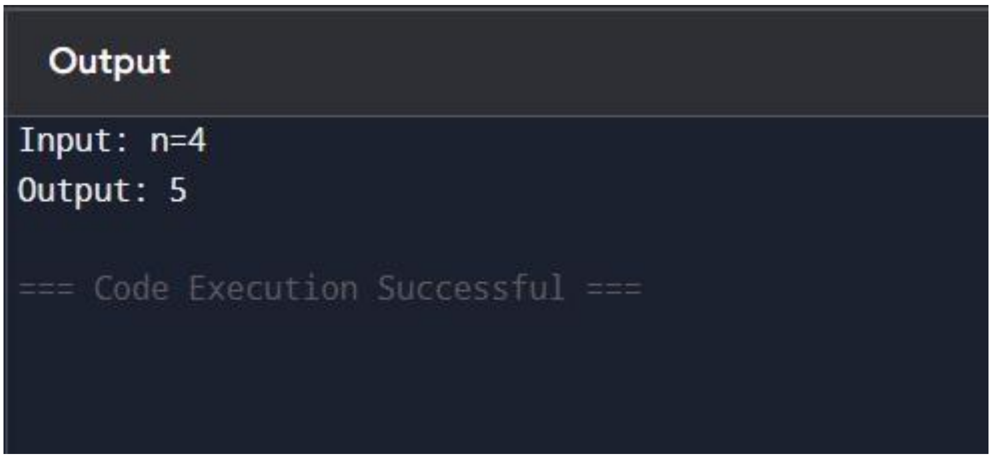
Examples:

Input: n=4   Output: 5

Input: n=3 Output: 3



```python
def climbStairs(n):
    if n <= 2:
        return n
    first, second = 1, 2
    for _ in range(3, n + 1):
        first, second = second, first + second
    return second
print("Input: n=4")
print("Output:", climbStairs(4))
```


Output

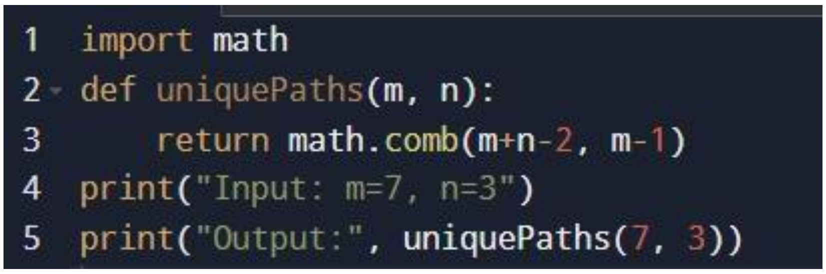
```
Input: n=4
Output: 5

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

14. 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?

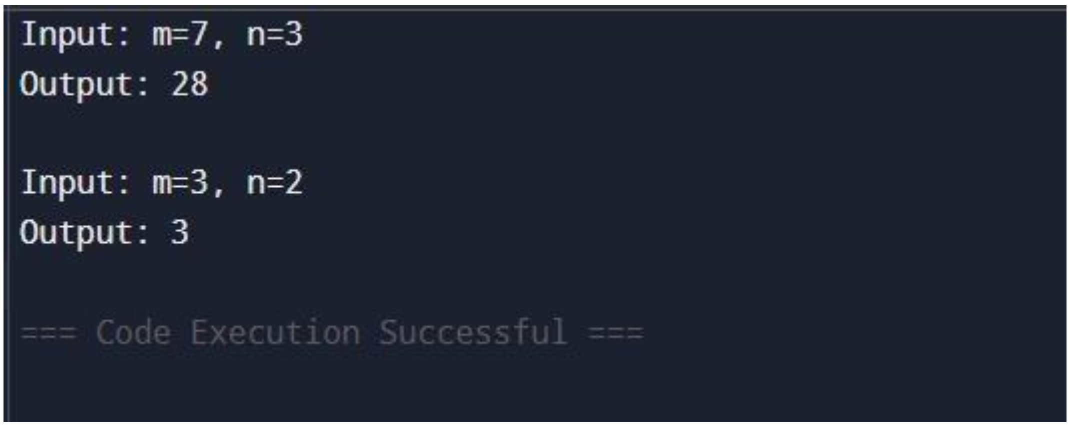
Examples:

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

Input: m=3,n=2   Output: 3



```python
import math
def uniquePaths(m, n):
    return math.comb(m+n-2, m-1)
print("Input: m=7, n=3")
print("Output:", uniquePaths(7, 3))
```



```
Input: m=7, n=3
Output: 28

Input: m=3, n=2
Output: 3

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

15. 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]]

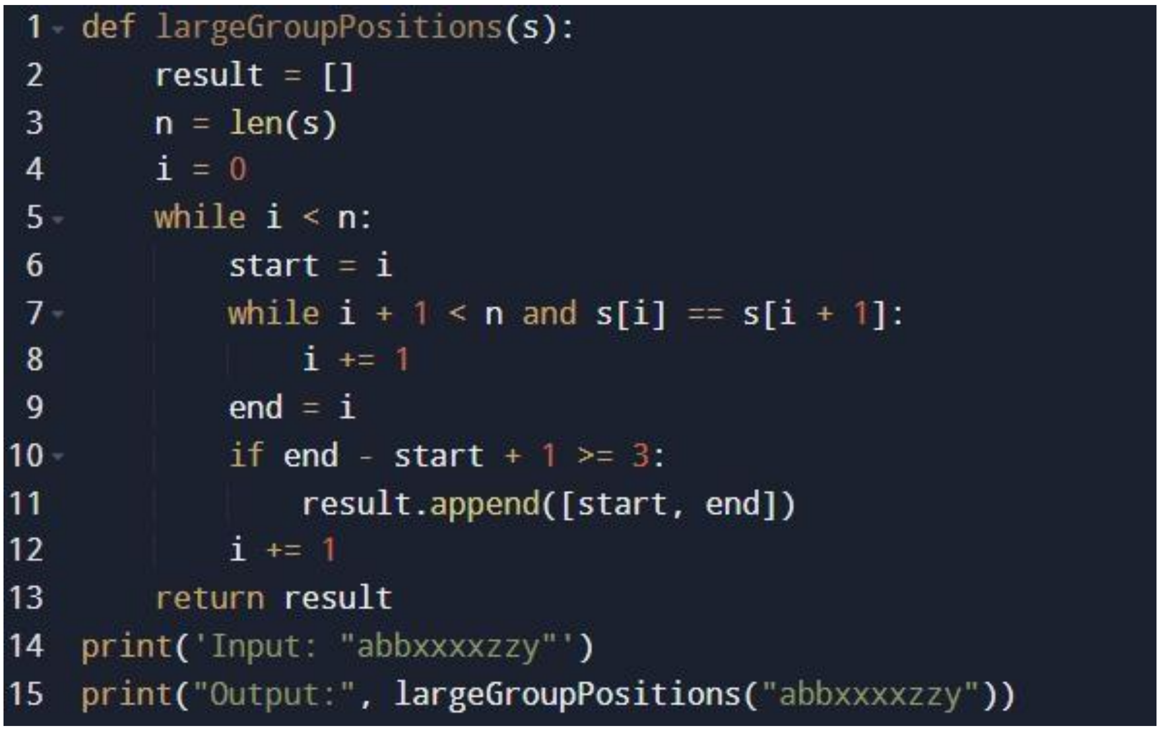
Explanation: "xxxx" is the only large group with start index 3 and end index 6.

Example 2:

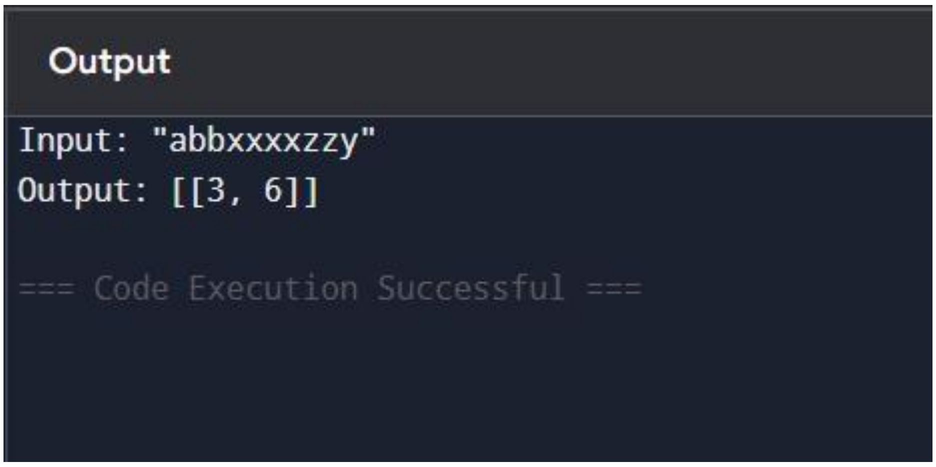
Input: s = "abc"

Output: []

Explanation: We have groups "a", "b", and "c", none of which are large groups.



```python
def largeGroupPositions(s):
    result = []
    n = len(s)
    i = 0
    while i < n:
        start = i
        while i + 1 < n and s[i] == s[i + 1]:
            i += 1
        end = i
        if end - start + 1 >= 3:
            result.append([start, end])
        i += 1
    return result
print('Input: "abbxxxxzzy"')
print("Output:", largeGroupPositions("abbxxxxzzy"))
```



```
Input: "abbxxxxzzy"
Output: [[3, 6]]

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

15. "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.

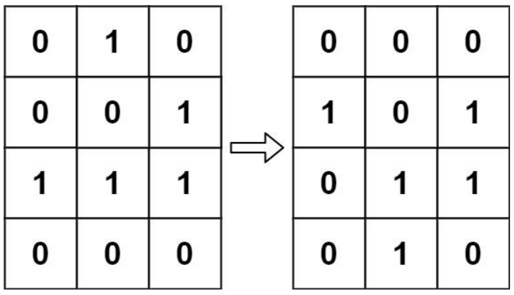
Any live cell with two or three live neighbors lives on to the next generation.

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

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.

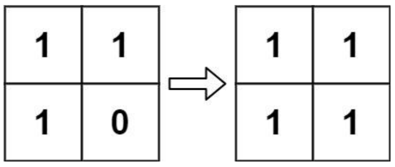
Example 1:



Input: board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]

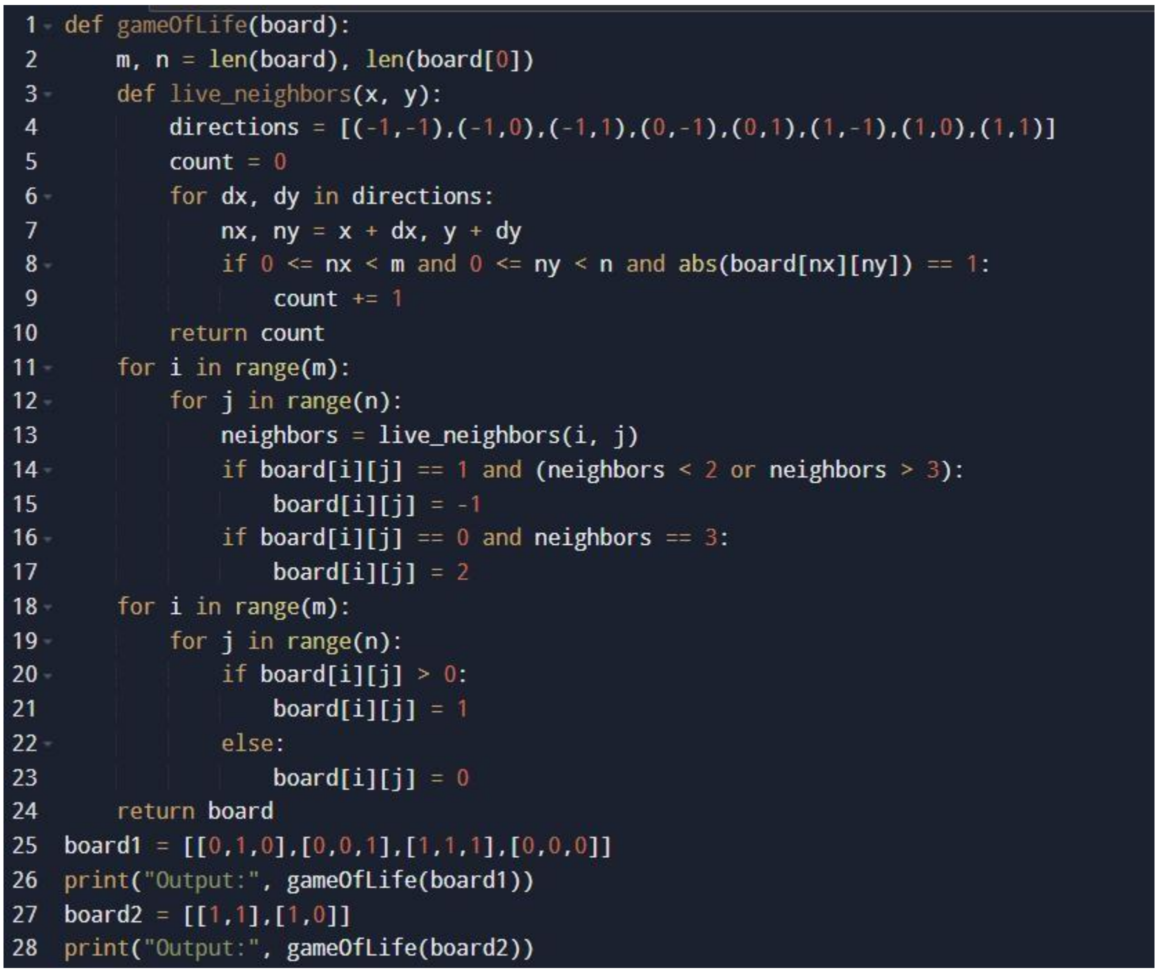
Output: [[0,0,0],[1,0,1],[0,1,1],[0,1,0]]

Example 2:

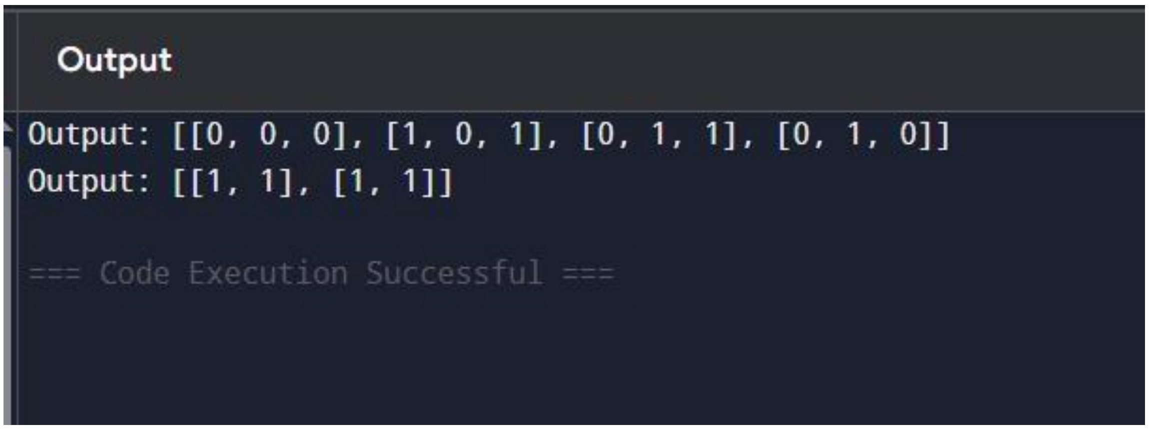


Input: board = [[1,1],[1,0]]

Output: [[1,1],[1,1]]



```python
def gameOfLife(board):
    m, n = len(board), len(board[0])
    def live_neighbors(x, y):
        directions = [(-1,-1),(-1,0),(-1,1),(0,-1),(0,1),(1,-1),(1,0),(1,1)]
        count = 0
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < m and 0 <= ny < n and abs(board[nx][ny]) == 1:
                count += 1
        return count
    for i in range(m):
        for j in range(n):
            neighbors = live_neighbors(i, j)
            if board[i][j] == 1 and (neighbors < 2 or neighbors > 3):
                board[i][j] = -1
            if board[i][j] == 0 and neighbors == 3:
                board[i][j] = 2
    for i in range(m):
        for j in range(n):
            if board[i][j] > 0:
                board[i][j] = 1
            else:
                board[i][j] = 0
    return board
board1 = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]
print("Output:", gameOfLife(board1))
board2 = [[1,1],[1,0]]
print("Output:", gameOfLife(board2))
```

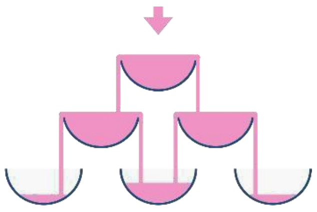


Output

```
Output: [[0, 0, 0], [1, 0, 1], [0, 1, 1], [0, 1, 0]]
Output: [[1, 1], [1, 1]]

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

16. 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. When the topmost glass is full, any excess liquid poured will fall equally to the glass immediately to the left and right of it. When those glasses become full, 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.



Now after pouring some non-negative integer cups of champagne, return how full the jth glass in the ith row is (both i and j are 0-indexed.)

Example 1:

Input: poured = 1, query_row = 1, query_glass = 1

Output: 0.00000

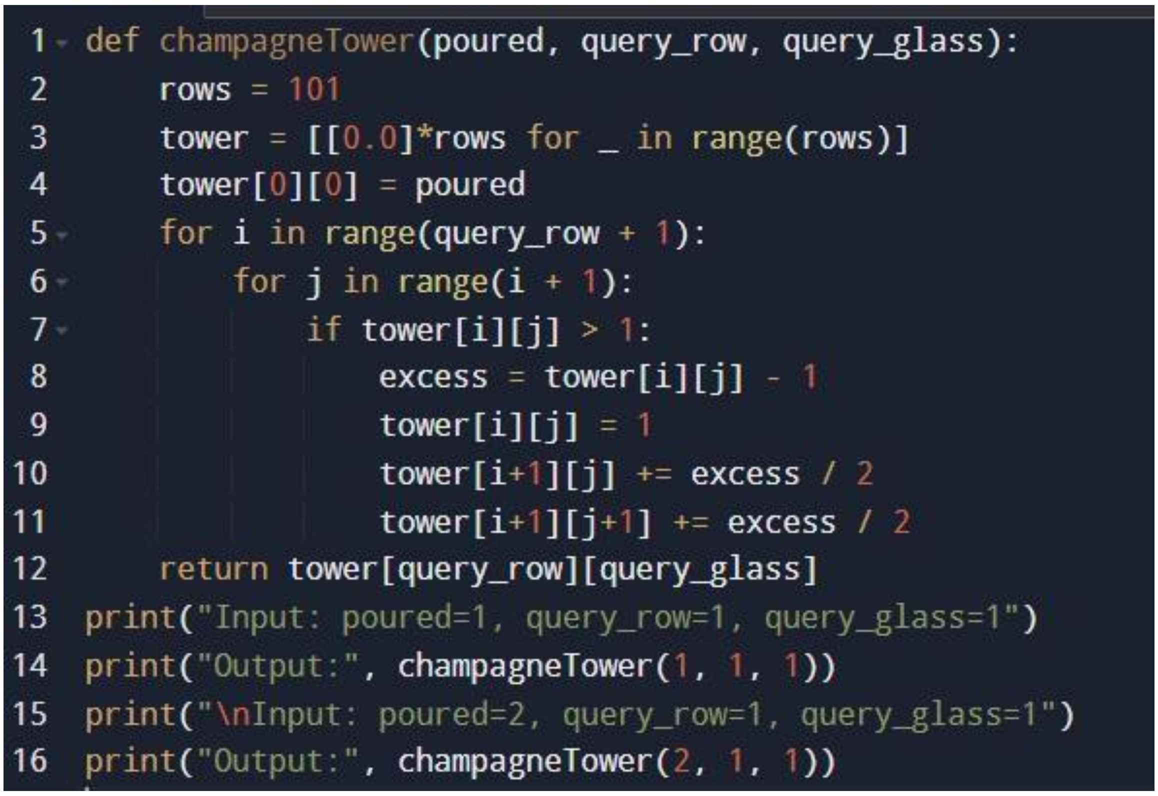
Explanation: We poured 1 cup of champange to the top glass of the tower (which is indexed as (0, 0)). There will be no excess liquid so all the glasses under the top glass will remain empty.

Example 2:

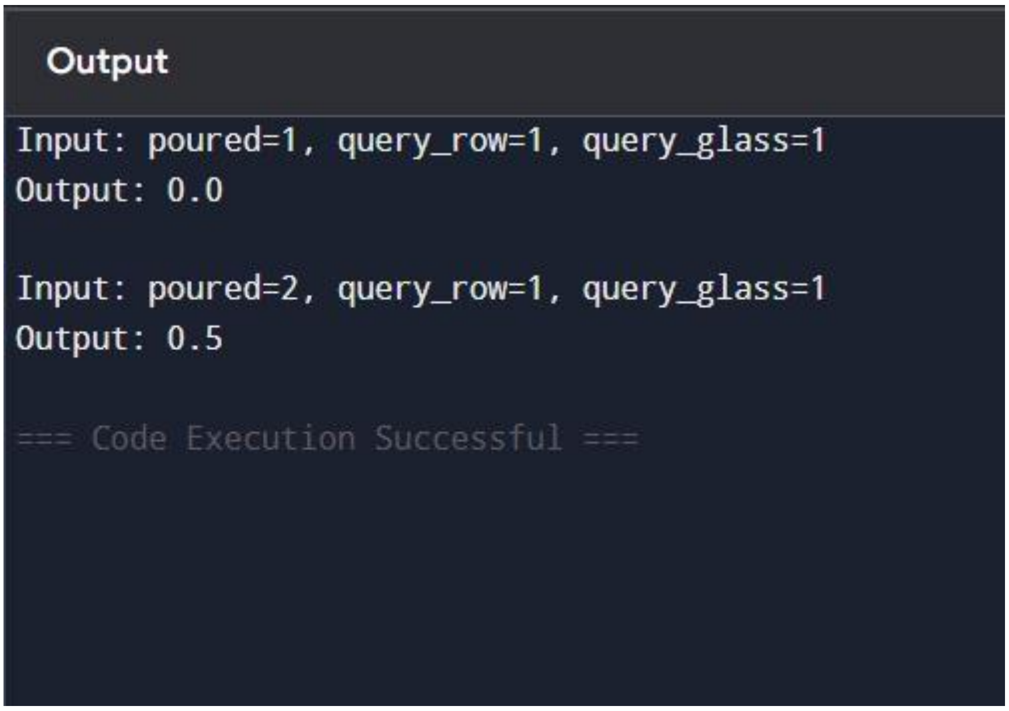
Input: poured = 2, query_row = 1, query_glass = 1

Output: 0.50000

Explanation: We poured 2 cups of champange to the top glass of the tower (which is indexed as (0, 0)). There is one cup of excess liquid. The glass indexed as (1, 0) and the glass indexed as (1, 1) will share the excess liquid equally, and each will get half cup of champange.



```python
def champagneTower(poured, query_row, query_glass):
    rows = 101
    tower = [[0.0]*rows for _ in range(rows)]
    tower[0][0] = poured
    for i in range(query_row + 1):
        for j in range(i + 1):
            if tower[i][j] > 1:
                excess = tower[i][j] - 1
                tower[i][j] = 1
                tower[i+1][j] += excess / 2
                tower[i+1][j+1] += excess / 2
    return tower[query_row][query_glass]
print("Input: poured=1, query_row=1, query_glass=1")
print("Output:", champagneTower(1, 1, 1))
print("\nInput: poured=2, query_row=1, query_glass=1")
print("Output:", champagneTower(2, 1, 1))
```



Output

```
Input: poured=1, query_row=1, query_glass=1
Output: 0.0

Input: poured=2, query_row=1, query_glass=1
Output: 0.5


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