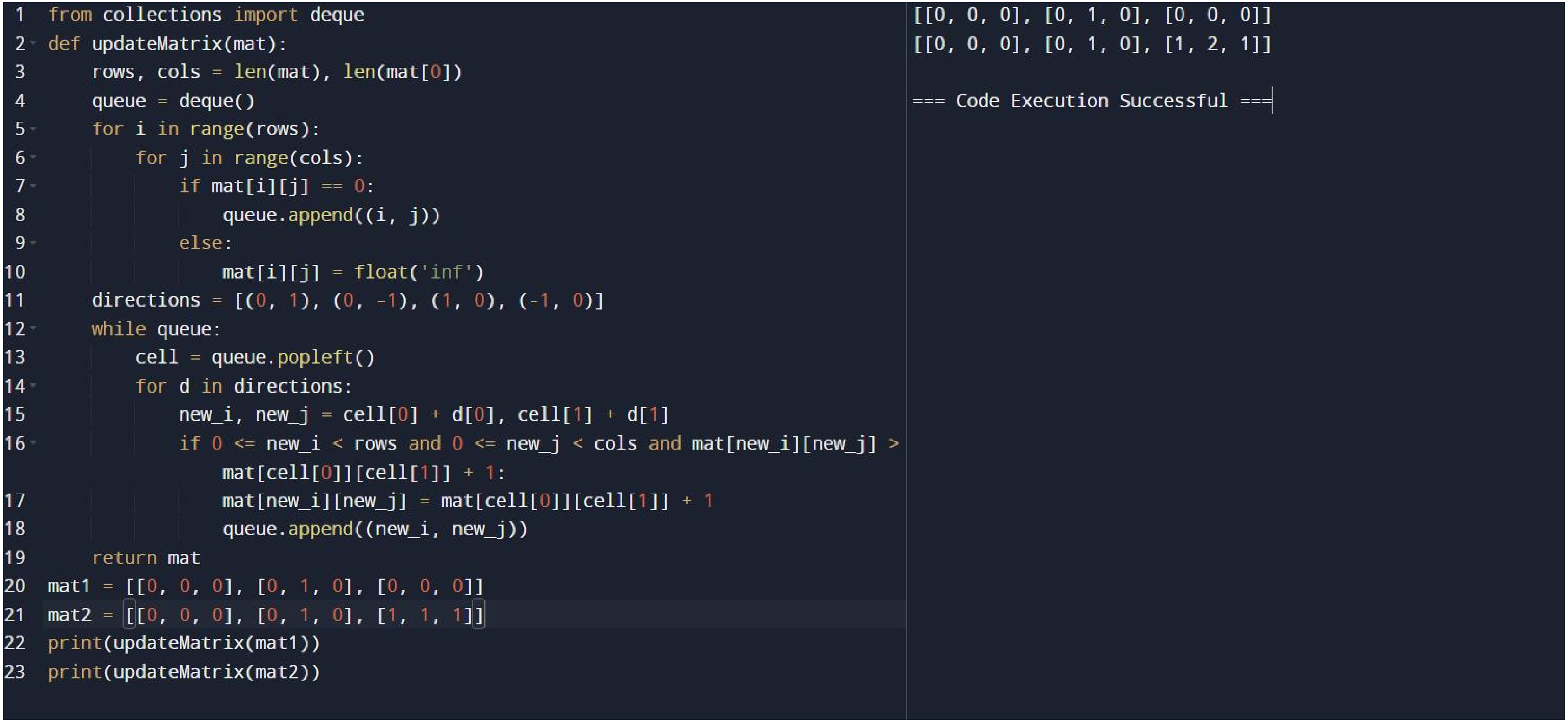
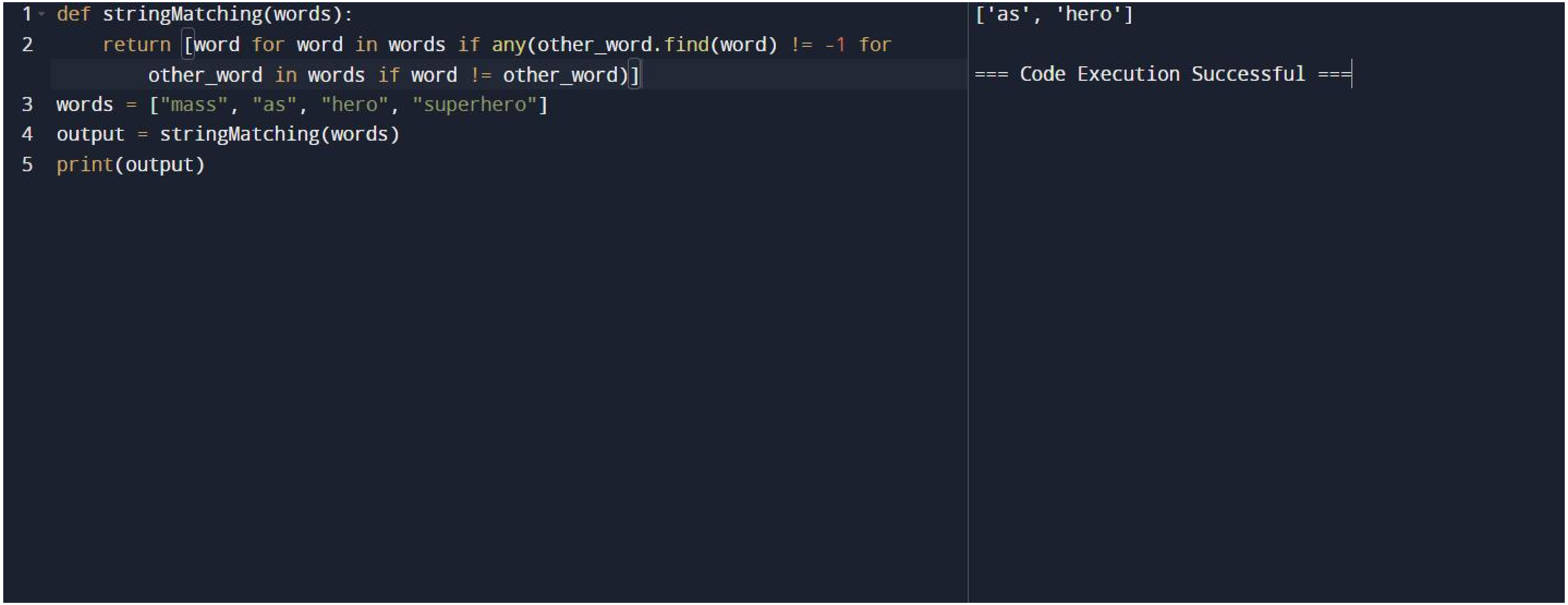


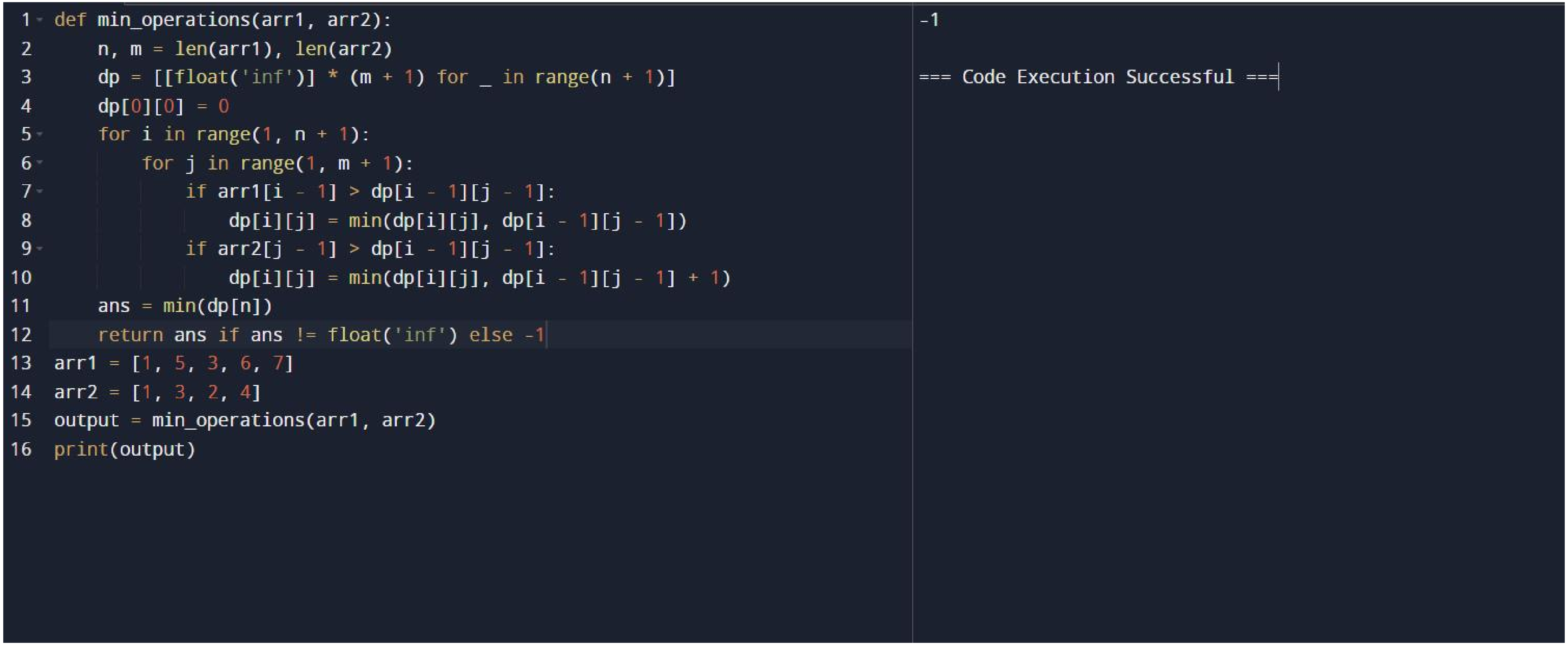
```python
from collections import deque
def updateMatrix(mat):
    rows, cols = len(mat), len(mat[0])
    queue = deque()
    for i in range(rows):
        for j in range(cols):
            if mat[i][j] == 0:
                queue.append((i, j))
            else:
                mat[i][j] = float('inf')
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
    while queue:
        cell = queue.popleft()
        for d in directions:
            new_i, new_j = cell[0] + d[0], cell[1] + d[1]
            if 0 <= new_i < rows and 0 <= new_j < cols and mat[new_i][new_j] > mat[cell[0]][cell[1]] + 1:
                mat[new_i][new_j] = mat[cell[0]][cell[1]] + 1
                queue.append((new_i, new_j))
    return mat
mat1 = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
mat2 = [[0, 0, 0], [0, 1, 0], [1, 1, 1]]
print(updateMatrix(mat1))
print(updateMatrix(mat2))
```

```
[[0, 0, 0], [0, 1, 0], [0, 0, 0]]
[[0, 0, 0], [0, 1, 0], [1, 2, 1]]

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



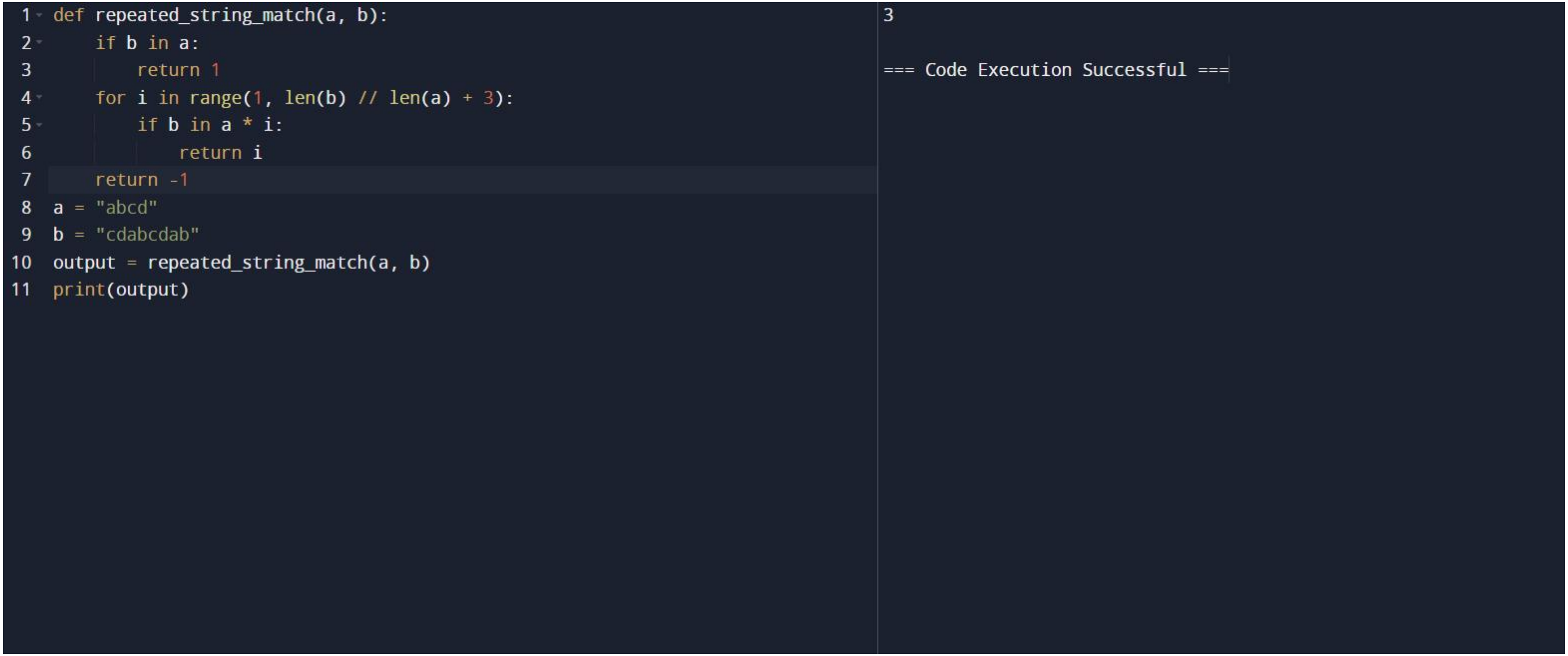
```python
def stringMatching(words):
    return [word for word in words if any(other_word.find(word) != -1 for
        other_word in words if word != other_word)]
words = ["mass", "as", "hero", "superhero"]
output = stringMatching(words)
print(output)
```

```
['as', 'hero']

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



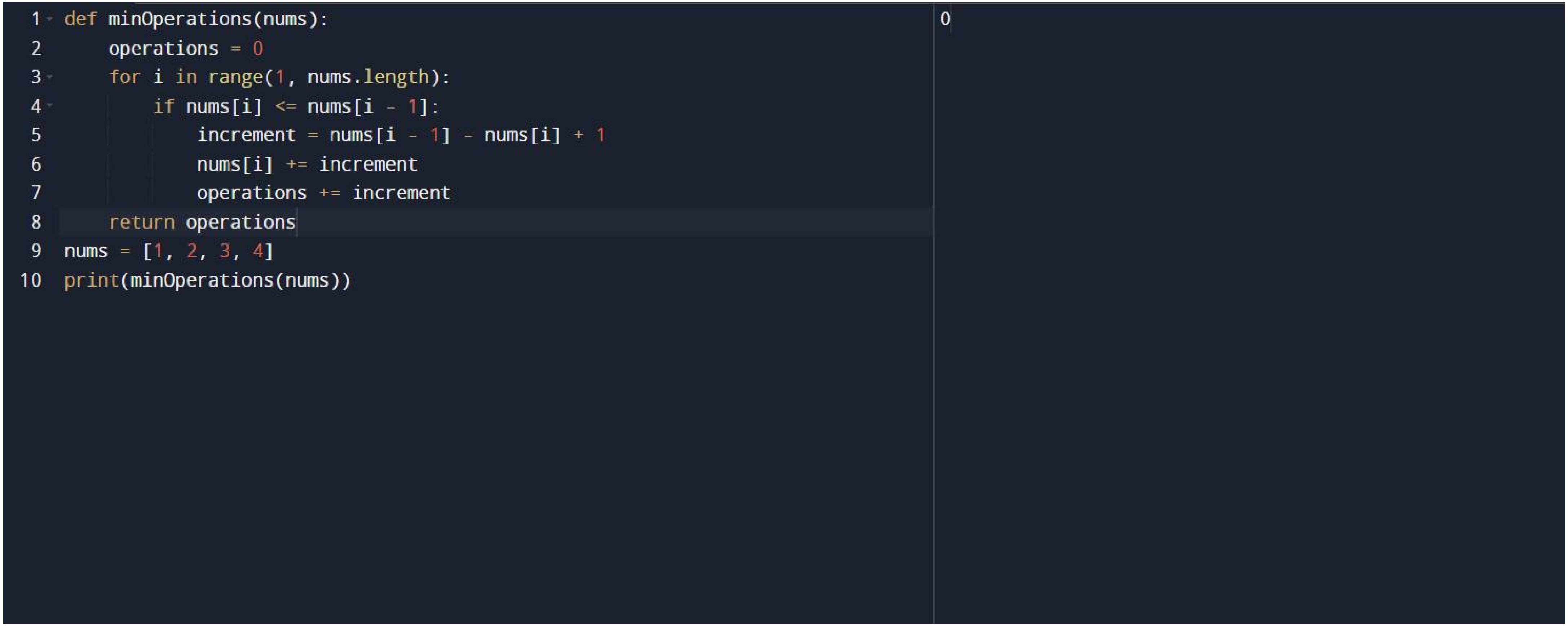
```python
def min_operations(arr1, arr2):
    n, m = len(arr1), len(arr2)
    dp = [[float('inf')] * (m + 1) for _ in range(n + 1)]
    dp[0][0] = 0
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if arr1[i - 1] > dp[i - 1][j - 1]:
                dp[i][j] = min(dp[i][j], dp[i - 1][j - 1])
            if arr2[j - 1] > dp[i - 1][j - 1]:
                dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + 1)
    ans = min(dp[n])
    return ans if ans != float('inf') else -1
arr1 = [1, 5, 3, 6, 7]
arr2 = [1, 3, 2, 4]
output = min_operations(arr1, arr2)
print(output)
```

```
-1

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



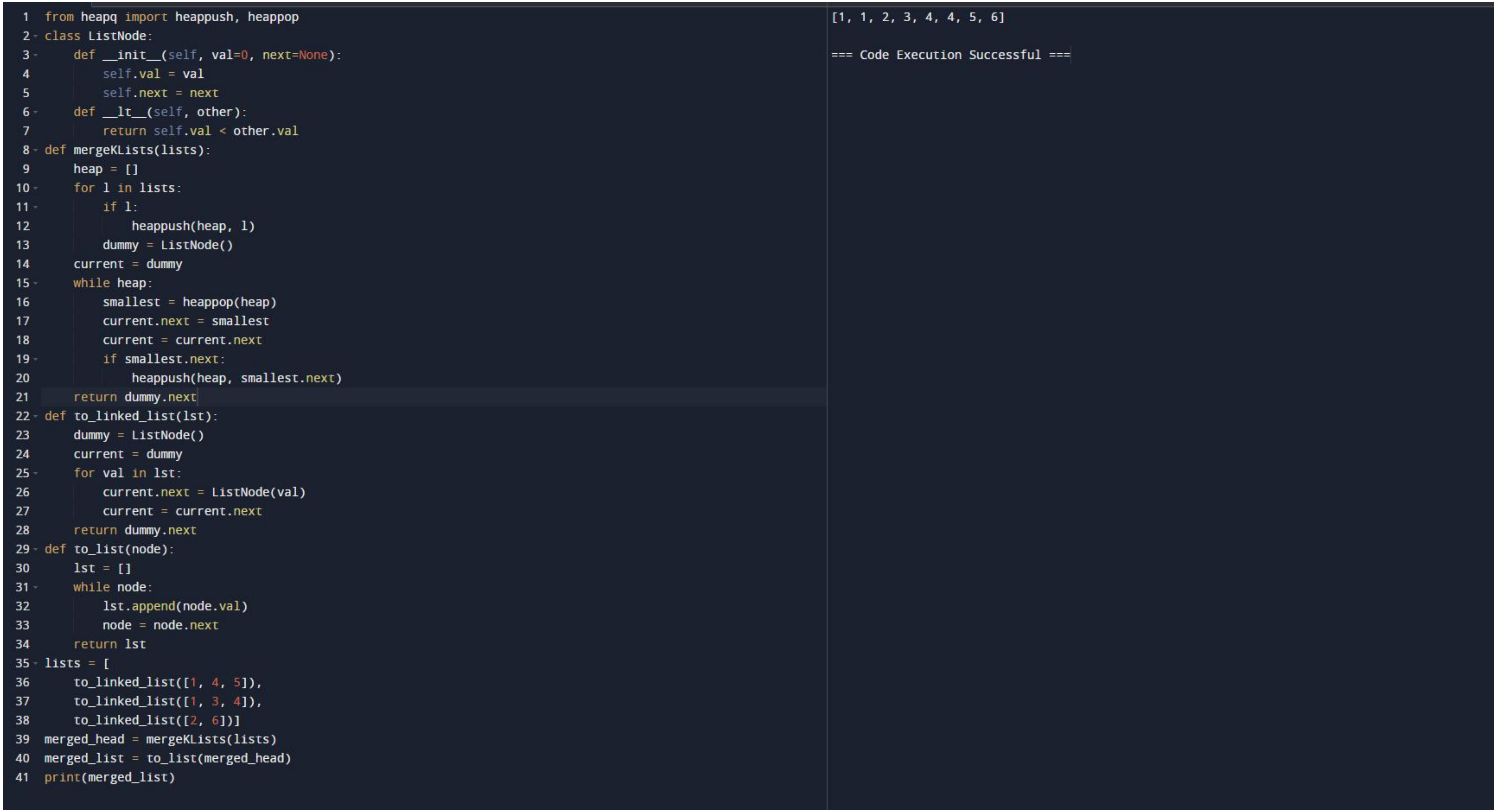
```python
def repeated_string_match(a, b):
    if b in a:
        return 1
    for i in range(1, len(b) // len(a) + 3):
        if b in a * i:
            return i
    return -1
a = "abcd"
b = "cdabcdab"
output = repeated_string_match(a, b)
print(output)
```

```
3

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



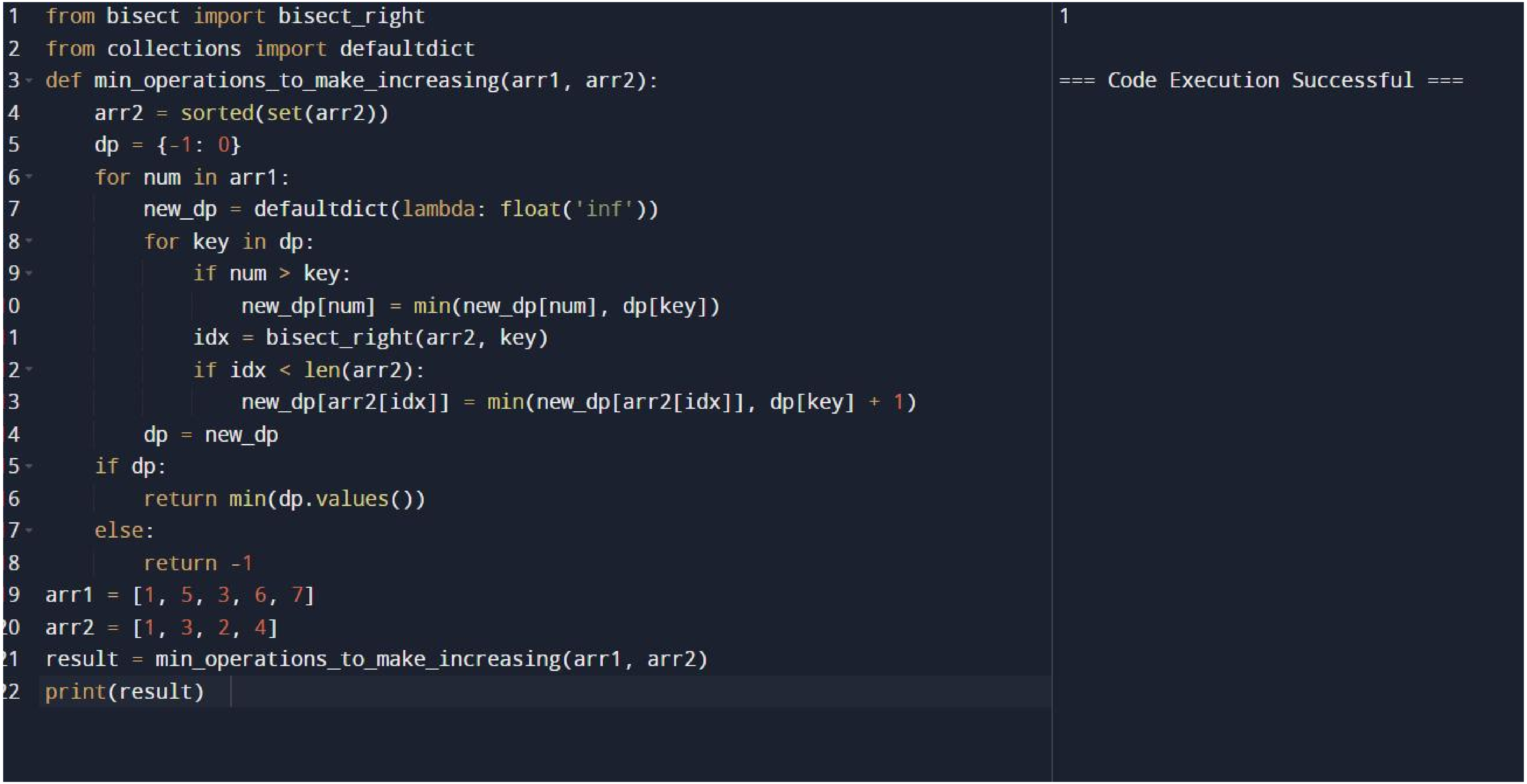
```python
def minOperations(nums):
    operations = 0
    for i in range(1, nums.length):
        if nums[i] <= nums[i - 1]:
            increment = nums[i - 1] - nums[i] + 1
            nums[i] += increment
            operations += increment
    return operations
nums = [1, 2, 3, 4]
print(minOperations(nums))
```

```
0
```



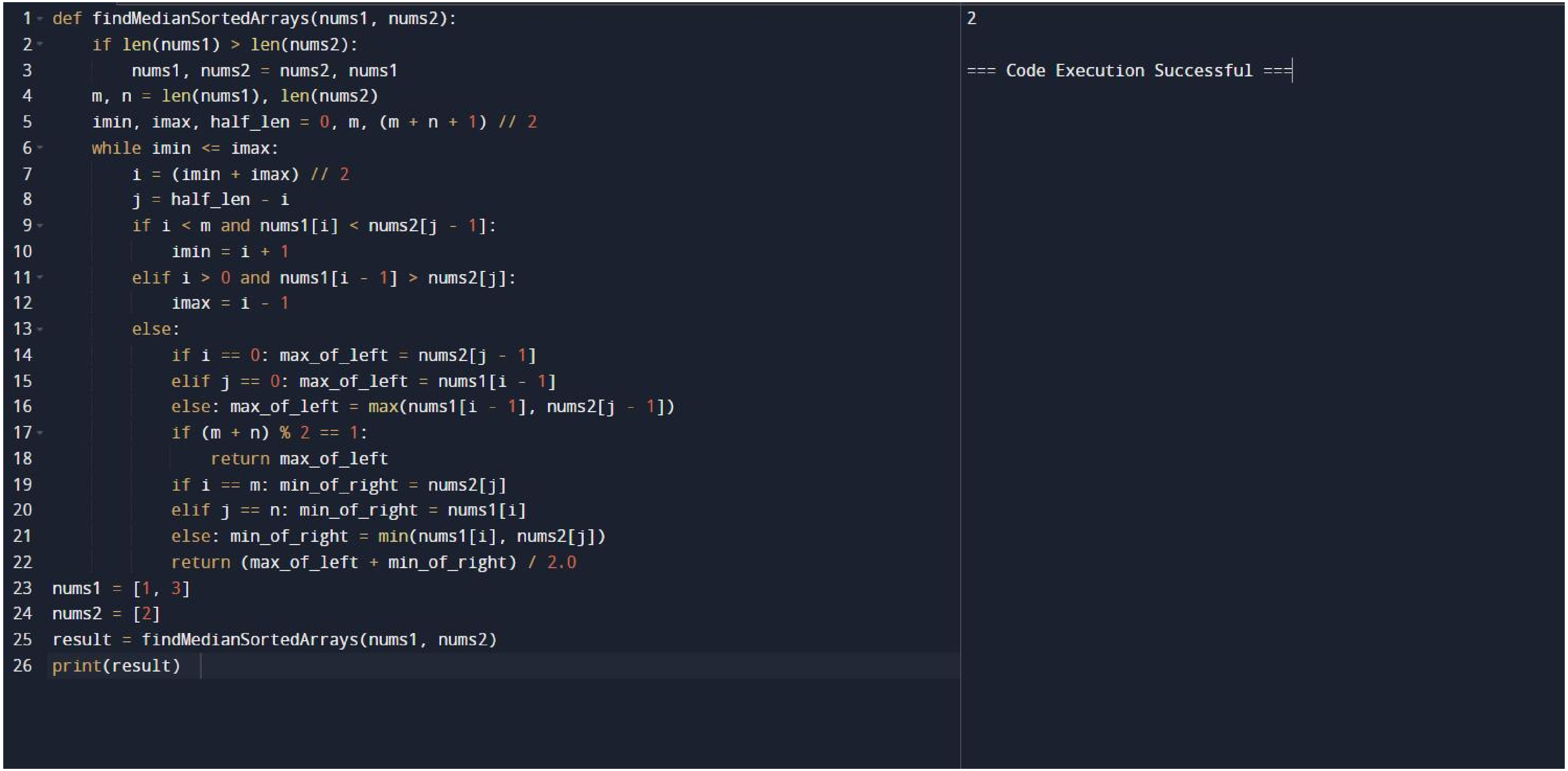
```python
from heapq import heappush, heappop
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
    def __lt__(self, other):
        return self.val < other.val
def mergeKLists(lists):
    heap = []
    for l in lists:
        if l:
            heappush(heap, l)
        dummy = ListNode()
    current = dummy
    while heap:
        smallest = heappop(heap)
        current.next = smallest
        current = current.next
        if smallest.next:
            heappush(heap, smallest.next)
    return dummy.next
def to_linked_list(lst):
    dummy = ListNode()
    current = dummy
    for val in lst:
        current.next = ListNode(val)
        current = current.next
    return dummy.next
def to_list(node):
    lst = []
    while node:
        lst.append(node.val)
        node = node.next
    return lst
lists = [
    to_linked_list([1, 4, 5]),
    to_linked_list([1, 3, 4]),
    to_linked_list([2, 6])]
merged_head = mergeKLists(lists)
merged_list = to_list(merged_head)
print(merged_list)
```

```
[1, 1, 2, 3, 4, 4, 5, 6]

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



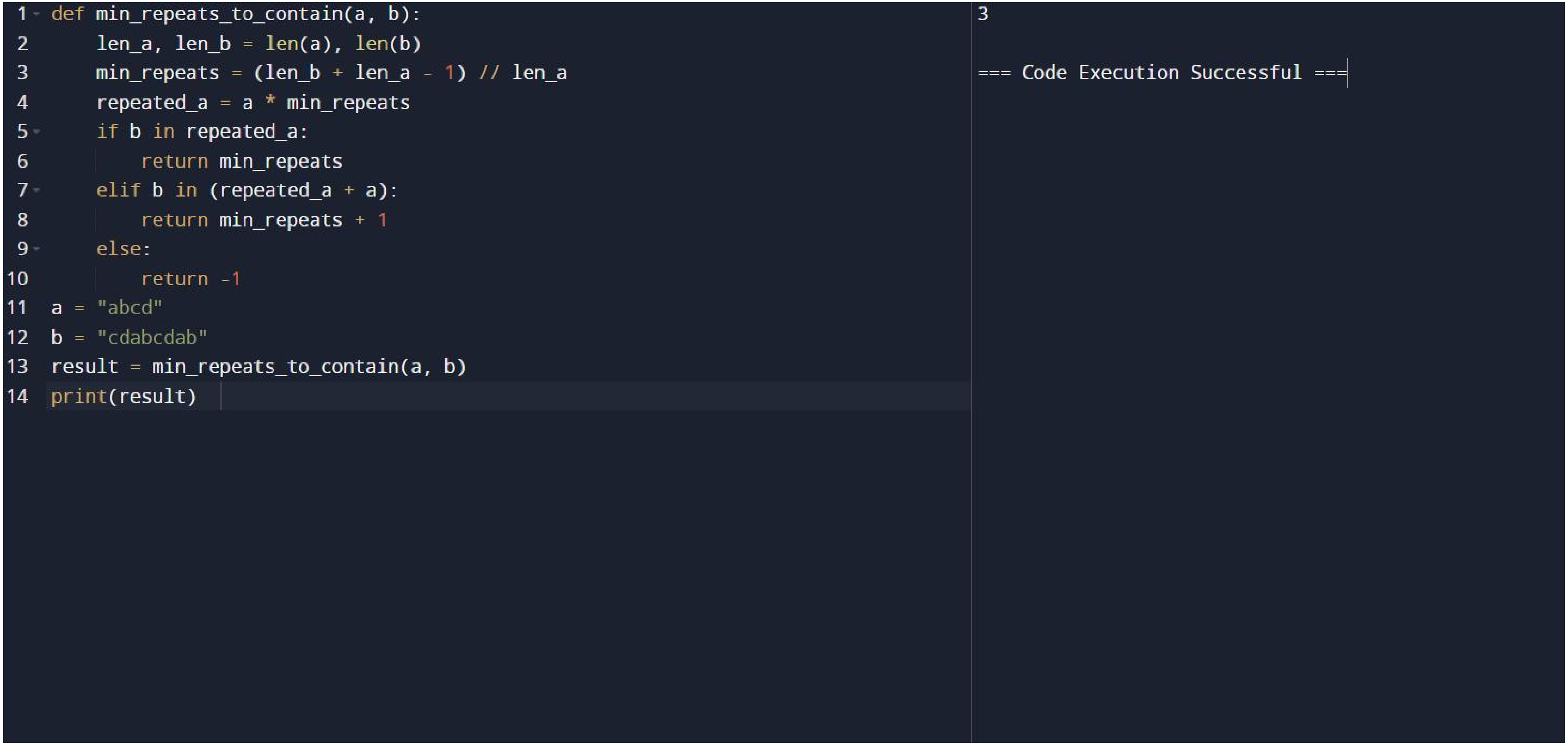
```python
from bisect import bisect_right
from collections import defaultdict
def min_operations_to_make_increasing(arr1, arr2):
    arr2 = sorted(set(arr2))
    dp = {-1: 0}
    for num in arr1:
        new_dp = defaultdict(lambda: float('inf'))
        for key in dp:
            if num > key:
                new_dp[num] = min(new_dp[num], dp[key])
            idx = bisect_right(arr2, key)
            if idx < len(arr2):
                new_dp[arr2[idx]] = min(new_dp[arr2[idx]], dp[key] + 1)
        dp = new_dp
    if dp:
        return min(dp.values())
    else:
        return -1
arr1 = [1, 5, 3, 6, 7]
arr2 = [1, 3, 2, 4]
result = min_operations_to_make_increasing(arr1, arr2)
print(result)
```

```
1

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



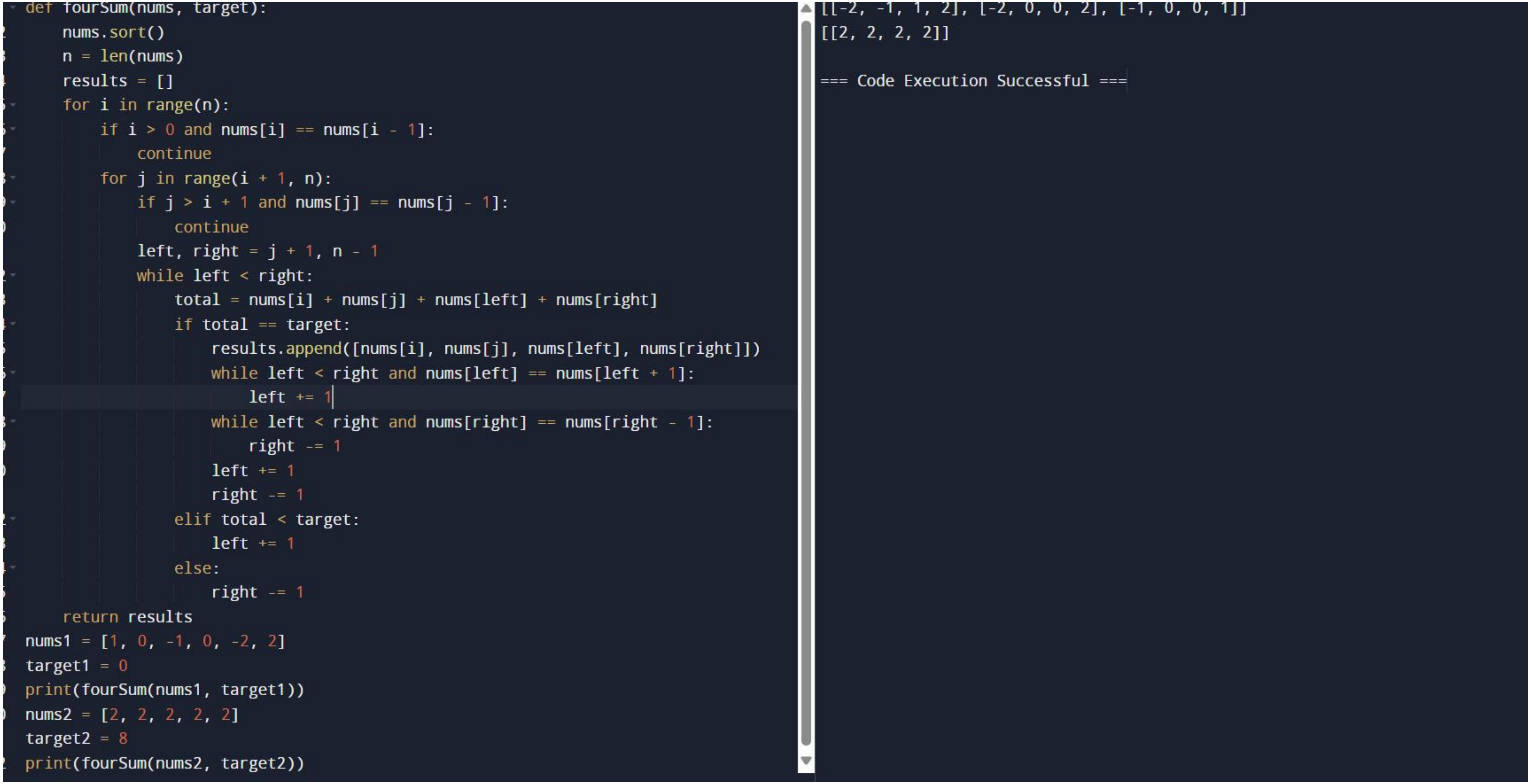
```python
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    m, n = len(nums1), len(nums2)
    imin, imax, half_len = 0, m, (m + n + 1) // 2
    while imin <= imax:
        i = (imin + imax) // 2
        j = half_len - i
        if i < m and nums1[i] < nums2[j - 1]:
            imin = i + 1
        elif i > 0 and nums1[i - 1] > nums2[j]:
            imax = i - 1
        else:
            if i == 0: max_of_left = nums2[j - 1]
            elif j == 0: max_of_left = nums1[i - 1]
            else: max_of_left = max(nums1[i - 1], nums2[j - 1])
            if (m + n) % 2 == 1:
                return max_of_left
            if i == m: min_of_right = nums2[j]
            elif j == n: min_of_right = nums1[i]
            else: min_of_right = min(nums1[i], nums2[j])
            return (max_of_left + min_of_right) / 2.0
nums1 = [1, 3]
nums2 = [2]
result = findMedianSortedArrays(nums1, nums2)
print(result)
```

```
2

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



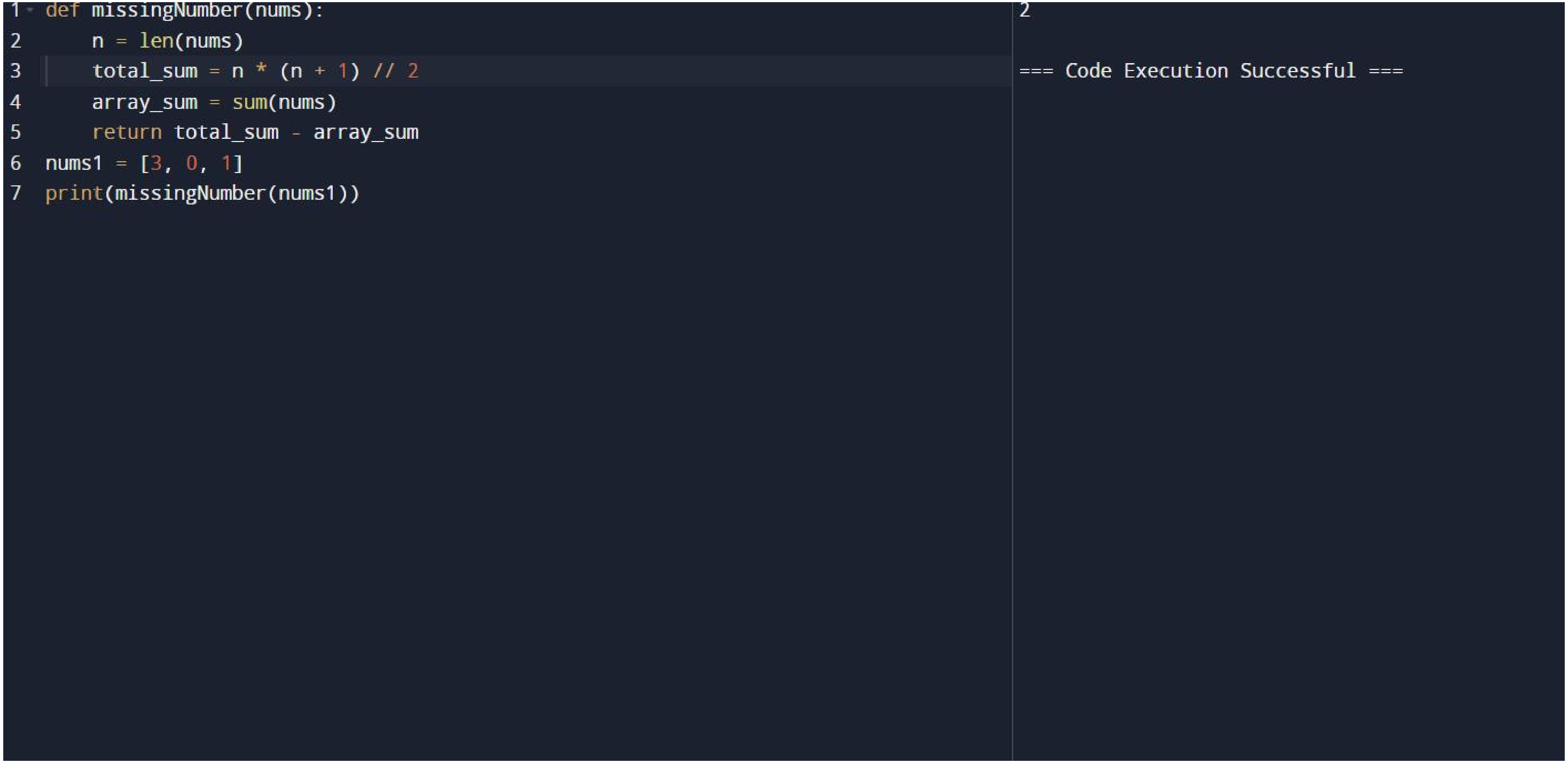
```python
def min_repeats_to_contain(a, b):
    len_a, len_b = len(a), len(b)
    min_repeats = (len_b + len_a - 1) // len_a
    repeated_a = a * min_repeats
    if b in repeated_a:
        return min_repeats
    elif b in (repeated_a + a):
        return min_repeats + 1
    else:
        return -1
a = "abcd"
b = "cdabcdab"
result = min_repeats_to_contain(a, b)
print(result)
```

```
3

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



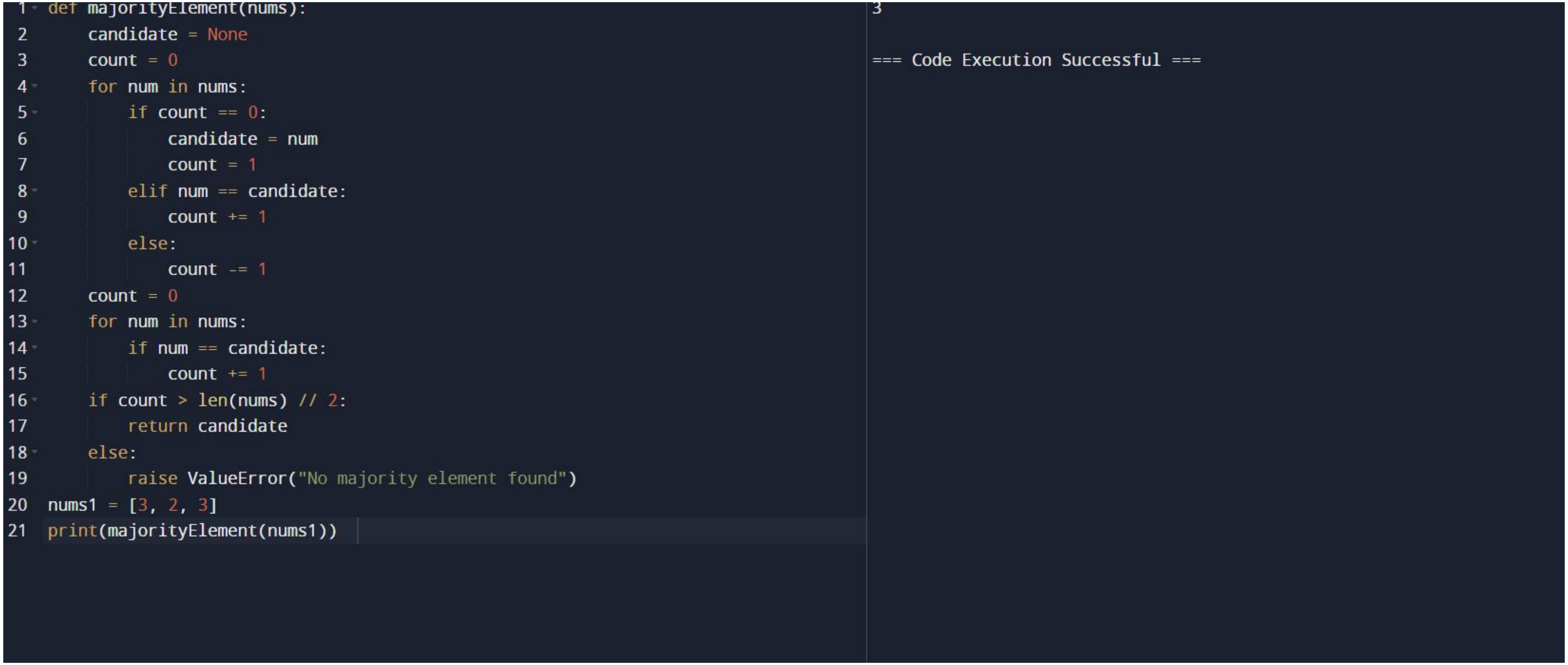
```python
def fourSum(nums, target):
    nums.sort()
    n = len(nums)
    results = []
    for i in range(n):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        for j in range(i + 1, n):
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue
            left, right = j + 1, n - 1
            while left < right:
                total = nums[i] + nums[j] + nums[left] + nums[right]
                if total == target:
                    results.append([nums[i], nums[j], nums[left], nums[right]])
                    while left < right and nums[left] == nums[left + 1]:
                        left += 1
                    while left < right and nums[right] == nums[right - 1]:
                        right -= 1
                    left += 1
                    right -= 1
                elif total < target:
                    left += 1
                else:
                    right -= 1
    return results
nums1 = [1, 0, -1, 0, -2, 2]
target1 = 0
print(fourSum(nums1, target1))
nums2 = [2, 2, 2, 2, 2]
target2 = 8
print(fourSum(nums2, target2))
```

```
[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
[[2, 2, 2, 2]]

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



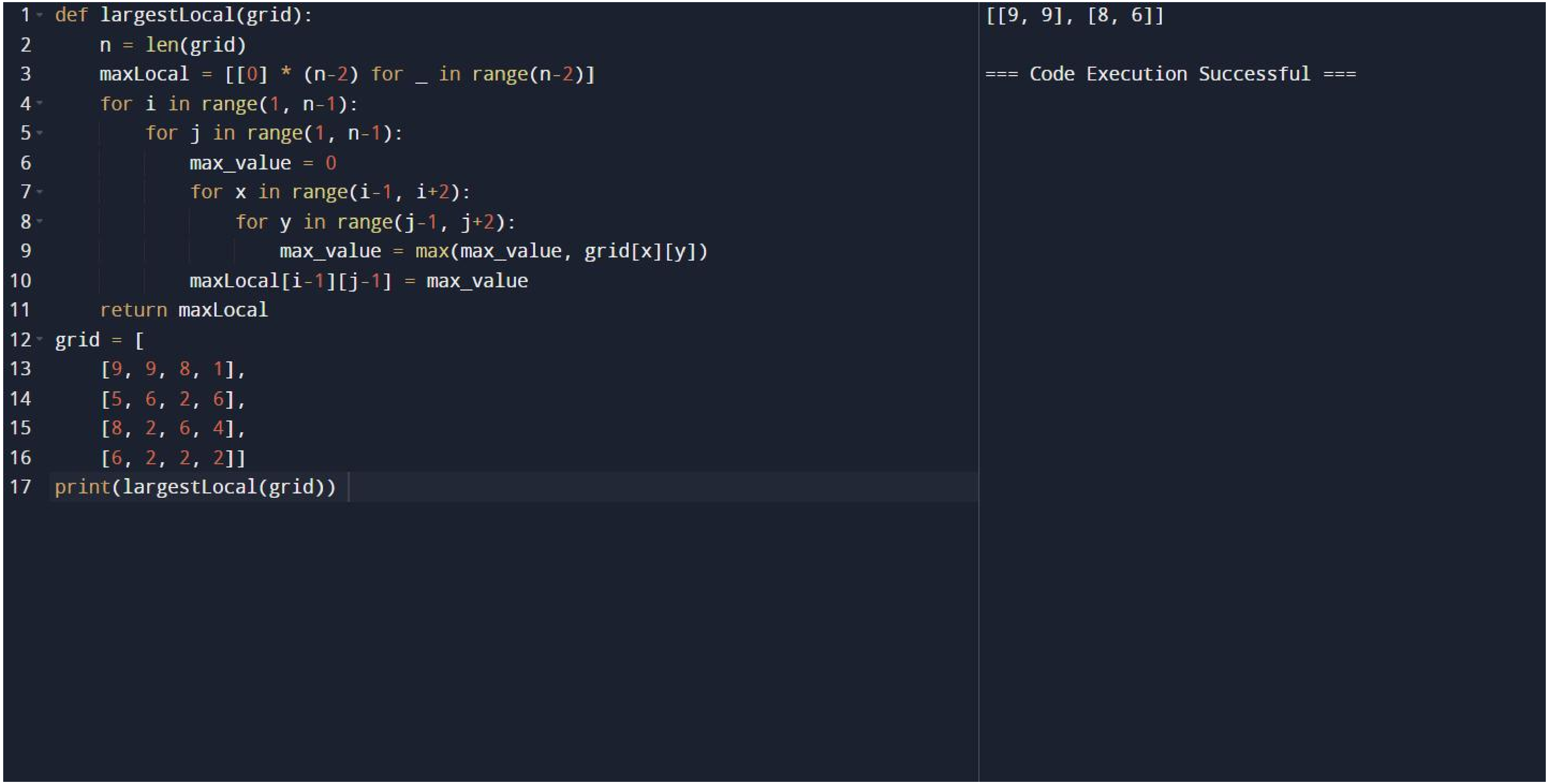
```python
def missingNumber(nums):
    n = len(nums)
    total_sum = n * (n + 1) // 2
    array_sum = sum(nums)
    return total_sum - array_sum
nums1 = [3, 0, 1]
print(missingNumber(nums1))
```

```
2


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



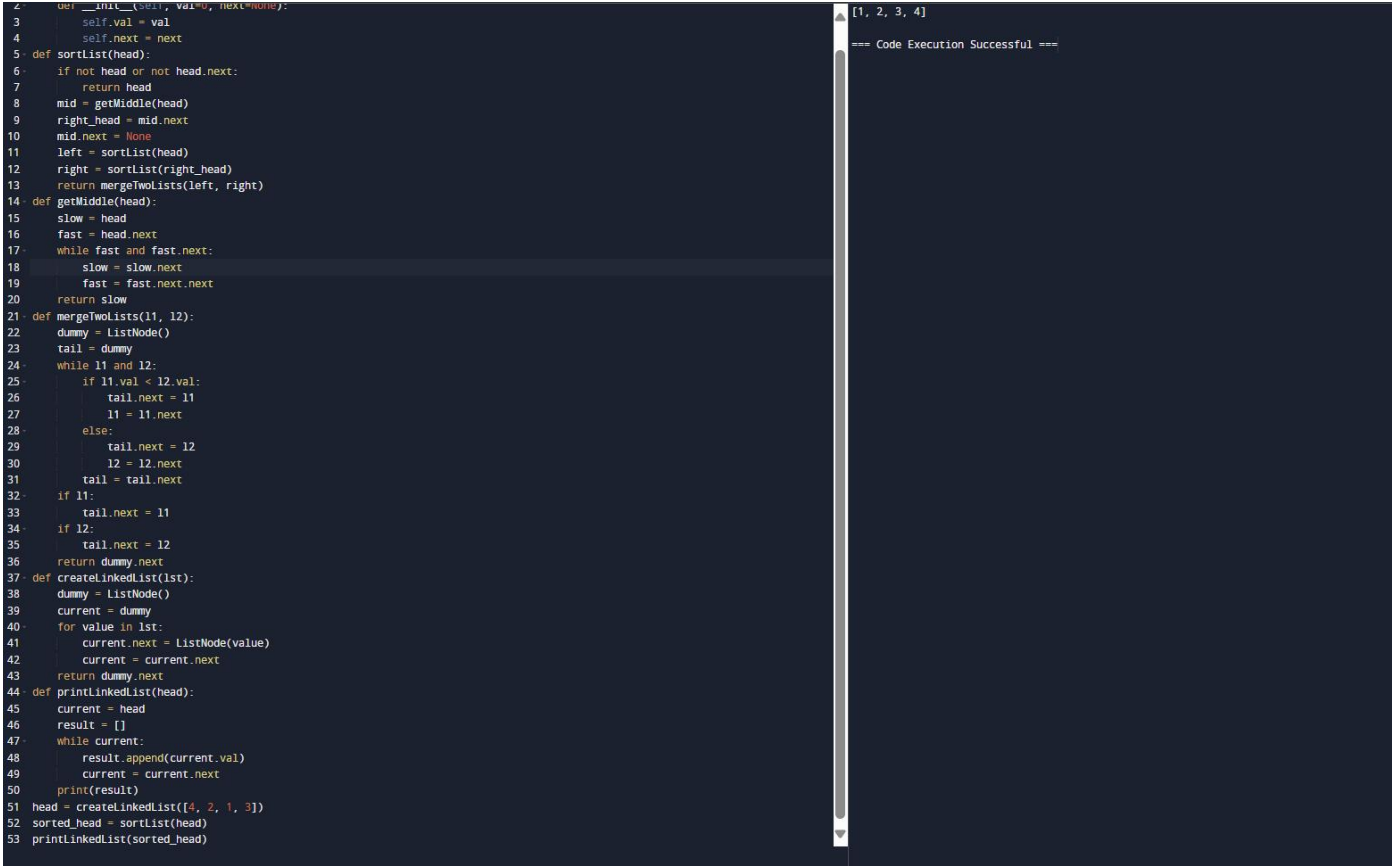
```python
def majorityElement(nums):
    candidate = None
    count = 0
    for num in nums:
        if count == 0:
            candidate = num
            count = 1
        elif num == candidate:
            count += 1
        else:
            count -= 1
    count = 0
    for num in nums:
        if num == candidate:
            count += 1
    if count > len(nums) // 2:
        return candidate
    else:
        raise ValueError("No majority element found")
nums1 = [3, 2, 3]
print(majorityElement(nums1))
```

```
3

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



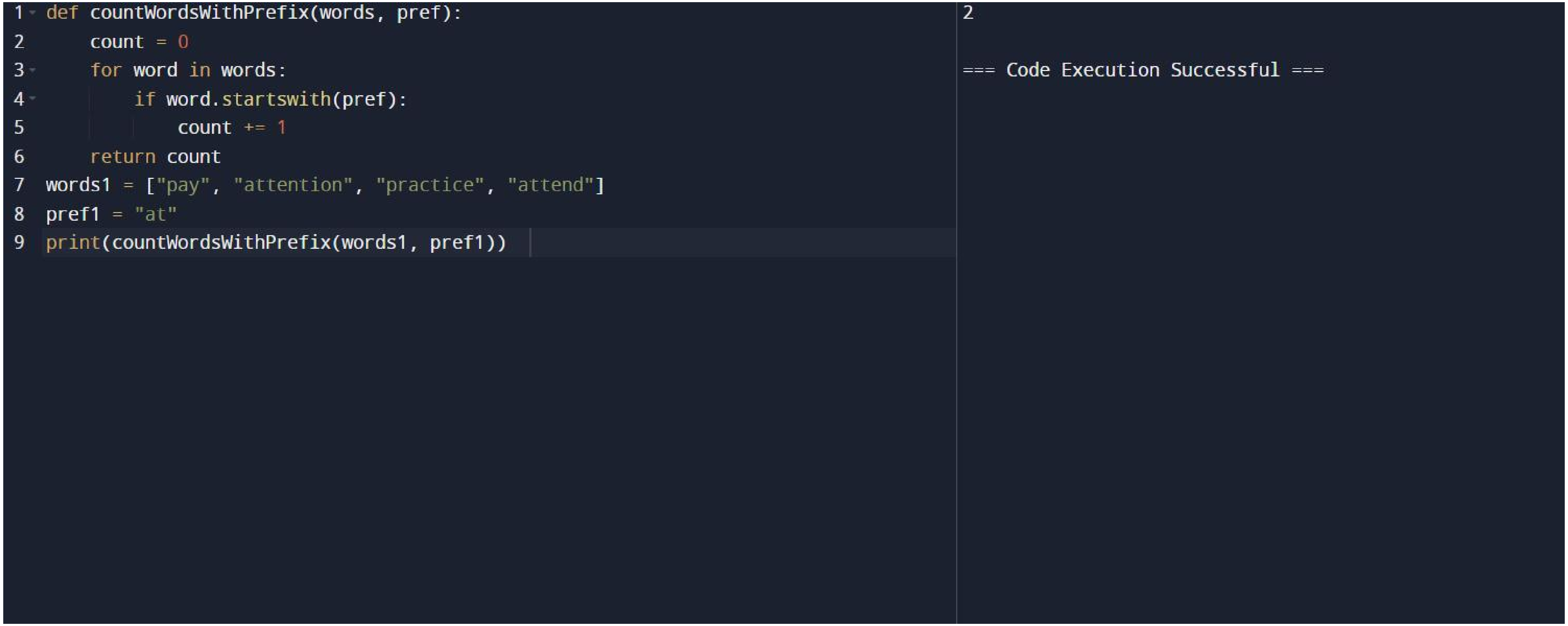
```python
def largestLocal(grid):
    n = len(grid)
    maxLocal = [[0] * (n-2) for _ in range(n-2)]
    for i in range(1, n-1):
        for j in range(1, n-1):
            max_value = 0
            for x in range(i-1, i+2):
                for y in range(j-1, j+2):
                    max_value = max(max_value, grid[x][y])
            maxLocal[i-1][j-1] = max_value
    return maxLocal
grid = [
    [9, 9, 8, 1],
    [5, 6, 2, 6],
    [8, 2, 6, 4],
    [6, 2, 2, 2]]
print(largestLocal(grid))
```

```
[[9, 9], [8, 6]]

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



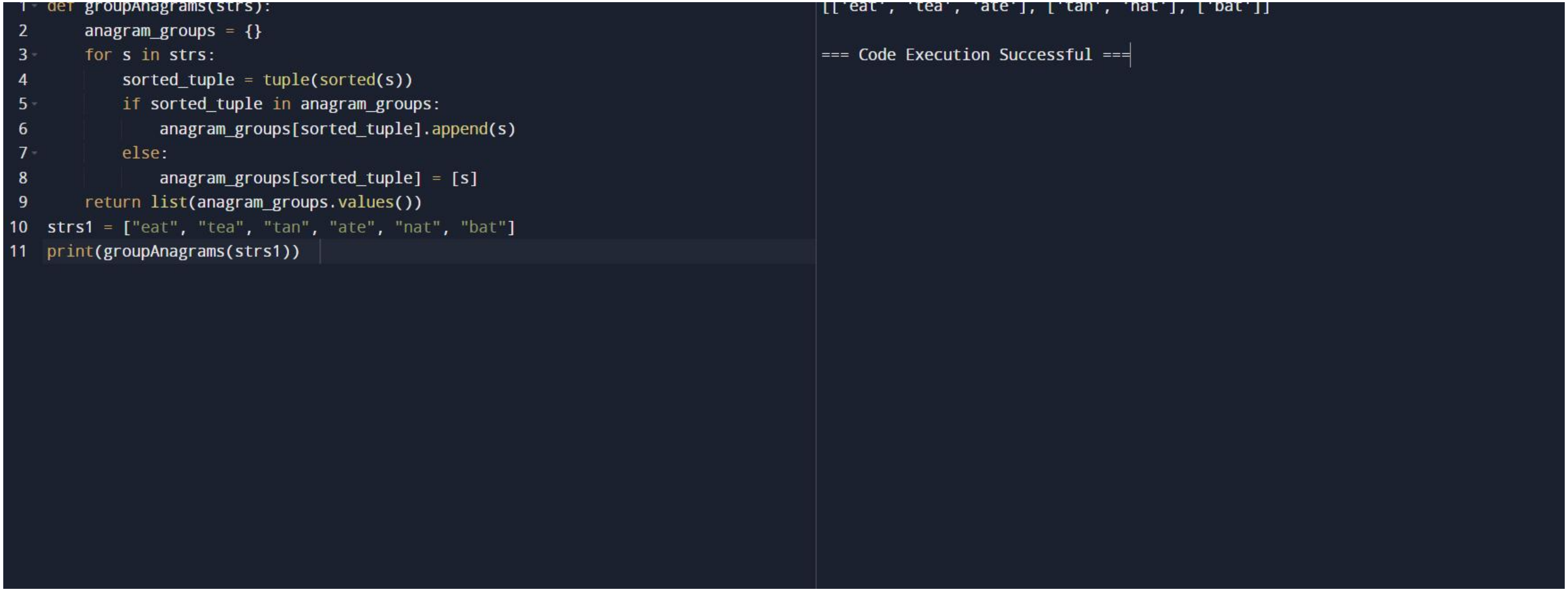
```
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def sortList(head):
    if not head or not head.next:
        return head
    mid = getMiddle(head)
    right_head = mid.next
    mid.next = None
    left = sortList(head)
    right = sortList(right_head)
    return mergeTwoLists(left, right)
def getMiddle(head):
    slow = head
    fast = head.next
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
    return slow
def mergeTwoLists(l1, l2):
    dummy = ListNode()
    tail = dummy
    while l1 and l2:
        if l1.val < l2.val:
            tail.next = l1
            l1 = l1.next
        else:
            tail.next = l2
            l2 = l2.next
        tail = tail.next
    if l1:
        tail.next = l1
    if l2:
        tail.next = l2
    return dummy.next
def createLinkedList(lst):
    dummy = ListNode()
    current = dummy
    for value in lst:
        current.next = ListNode(value)
        current = current.next
    return dummy.next
def printLinkedList(head):
    current = head
    result = []
    while current:
        result.append(current.val)
        current = current.next
    print(result)
head = createLinkedList([4, 2, 1, 3])
sorted_head = sortList(head)
printLinkedList(sorted_head)
```

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

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



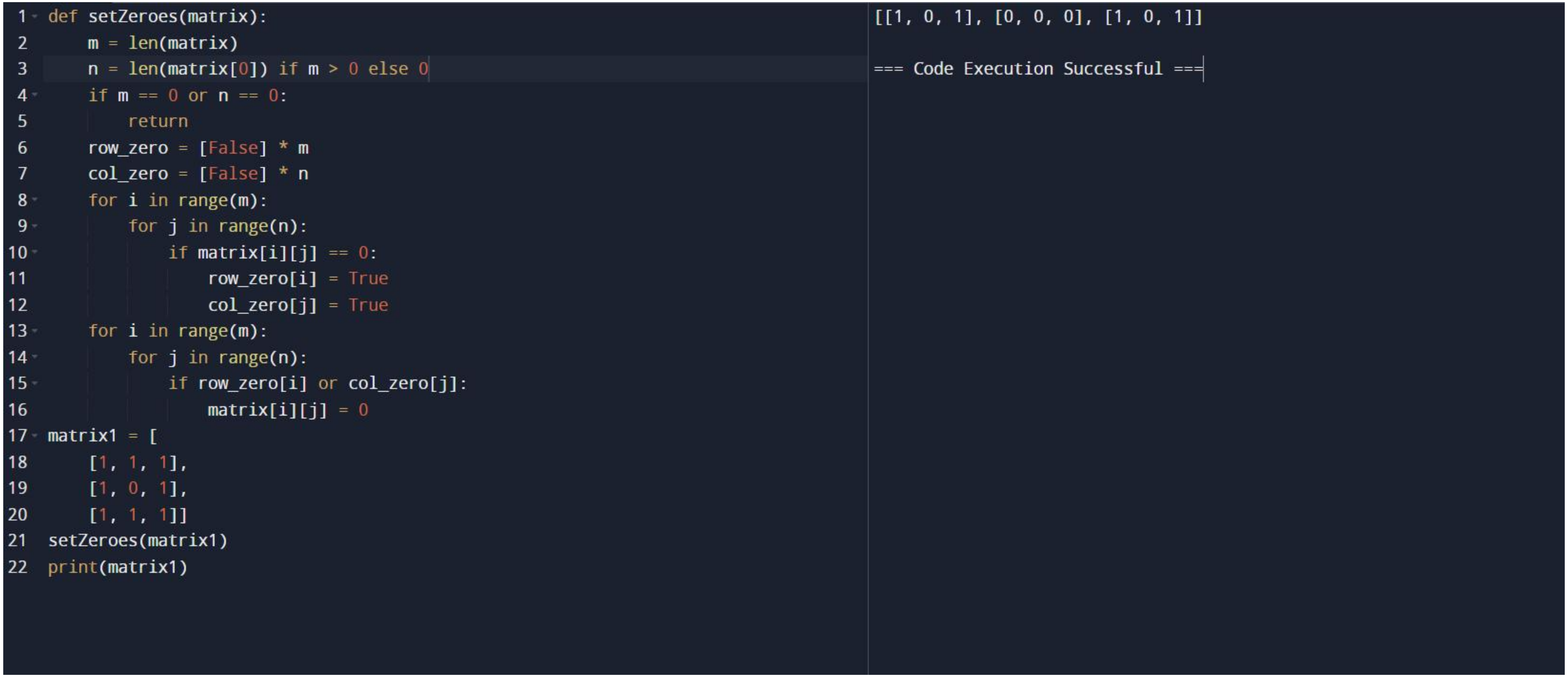
```python
def countWordsWithPrefix(words, pref):
    count = 0
    for word in words:
        if word.startswith(pref):
            count += 1
    return count
words1 = ["pay", "attention", "practice", "attend"]
pref1 = "at"
print(countWordsWithPrefix(words1, pref1))
```

```
2


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



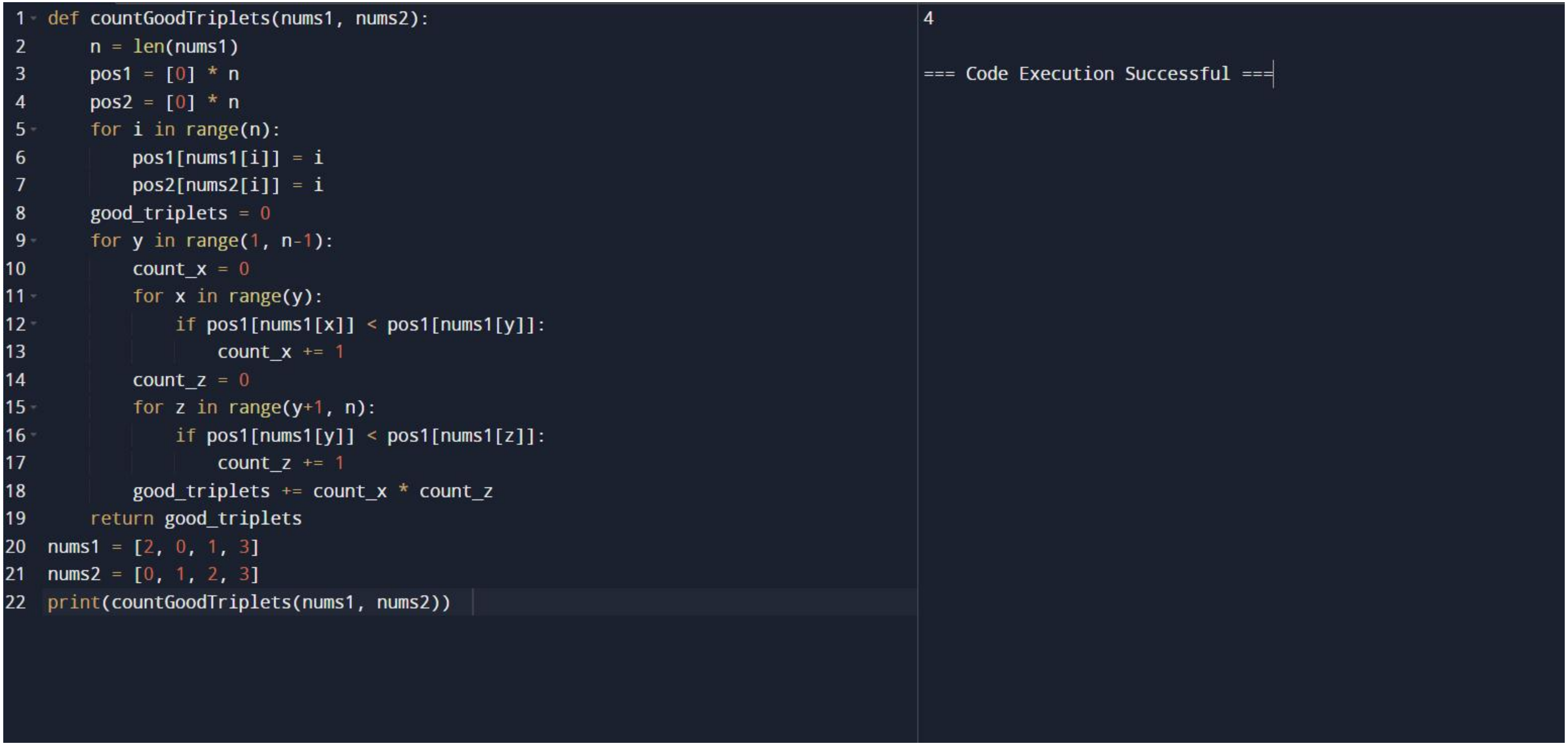
```python
def groupAnagrams(strs):
    anagram_groups = {}
    for s in strs:
        sorted_tuple = tuple(sorted(s))
        if sorted_tuple in anagram_groups:
            anagram_groups[sorted_tuple].append(s)
        else:
            anagram_groups[sorted_tuple] = [s]
    return list(anagram_groups.values())
strs1 = ["eat", "tea", "tan", "ate", "nat", "bat"]
print(groupAnagrams(strs1))
```

```
[['eat', 'tea', 'ate'], ['tan', 'nat'], ['bat']]

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



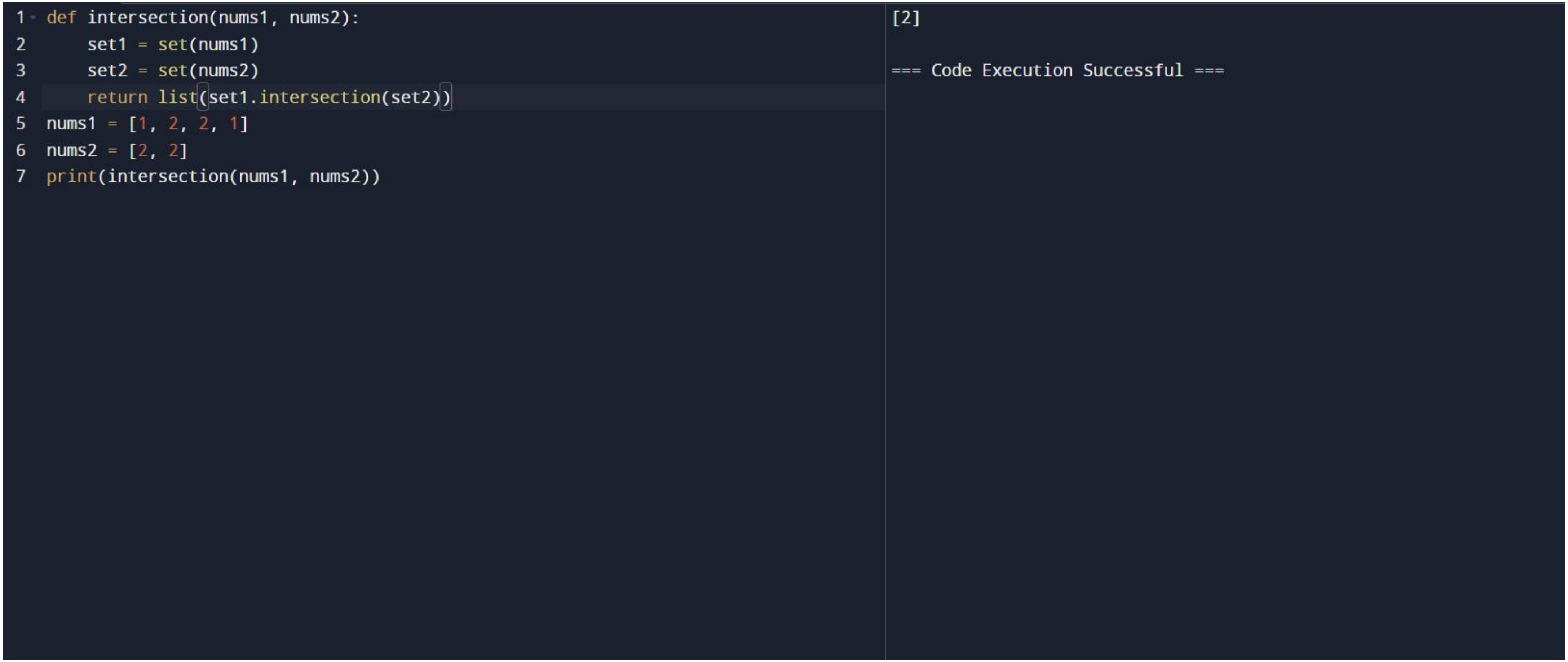
```python
def setZeroes(matrix):
    m = len(matrix)
    n = len(matrix[0]) if m > 0 else 0
    if m == 0 or n == 0:
        return
    row_zero = [False] * m
    col_zero = [False] * n
    for i in range(m):
        for j in range(n):
            if matrix[i][j] == 0:
                row_zero[i] = True
                col_zero[j] = True
    for i in range(m):
        for j in range(n):
            if row_zero[i] or col_zero[j]:
                matrix[i][j] = 0
matrix1 = [
    [1, 1, 1],
    [1, 0, 1],
    [1, 1, 1]]
setZeroes(matrix1)
print(matrix1)
```

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

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



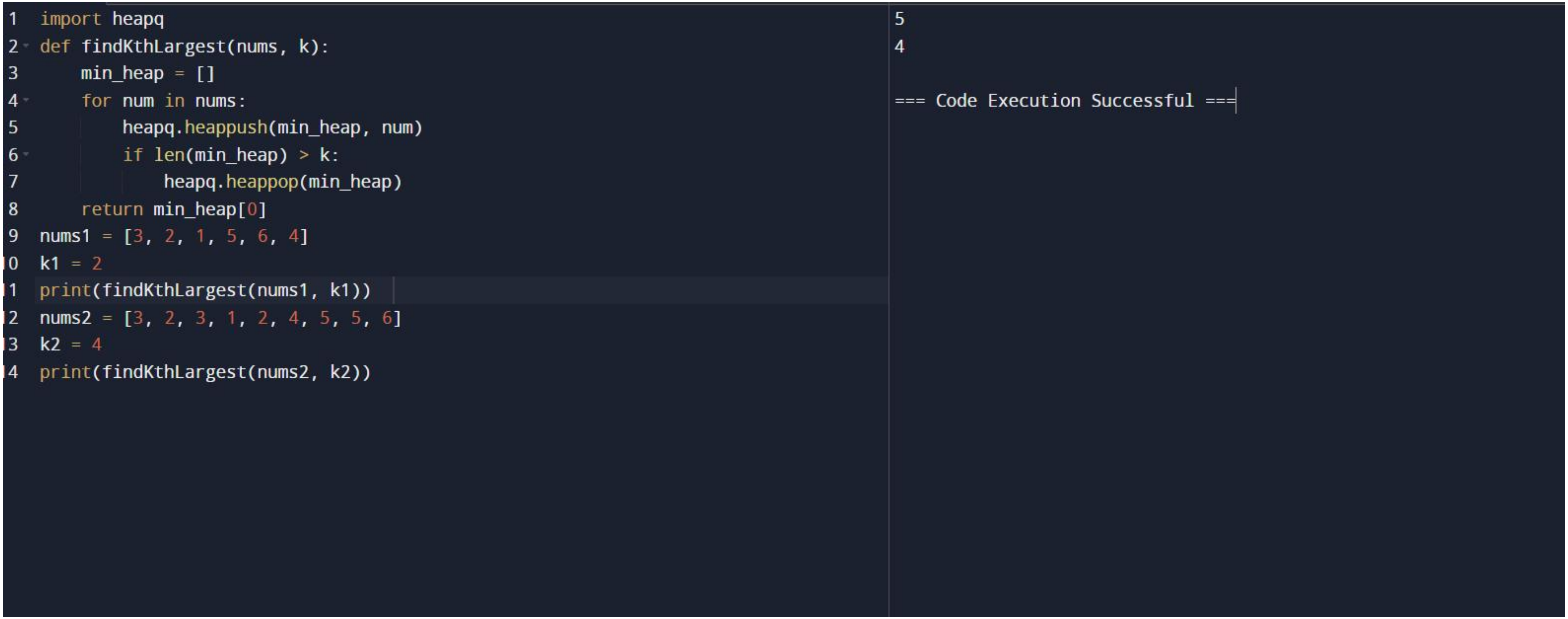
```python
def countGoodTriplets(nums1, nums2):
    n = len(nums1)
    pos1 = [0] * n
    pos2 = [0] * n
    for i in range(n):
        pos1[nums1[i]] = i
        pos2[nums2[i]] = i
    good_triplets = 0
    for y in range(1, n-1):
        count_x = 0
        for x in range(y):
            if pos1[nums1[x]] < pos1[nums1[y]]:
                count_x += 1
        count_z = 0
        for z in range(y+1, n):
            if pos1[nums1[y]] < pos1[nums1[z]]:
                count_z += 1
        good_triplets += count_x * count_z
    return good_triplets
nums1 = [2, 0, 1, 3]
nums2 = [0, 1, 2, 3]
print(countGoodTriplets(nums1, nums2))
```

```
4

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



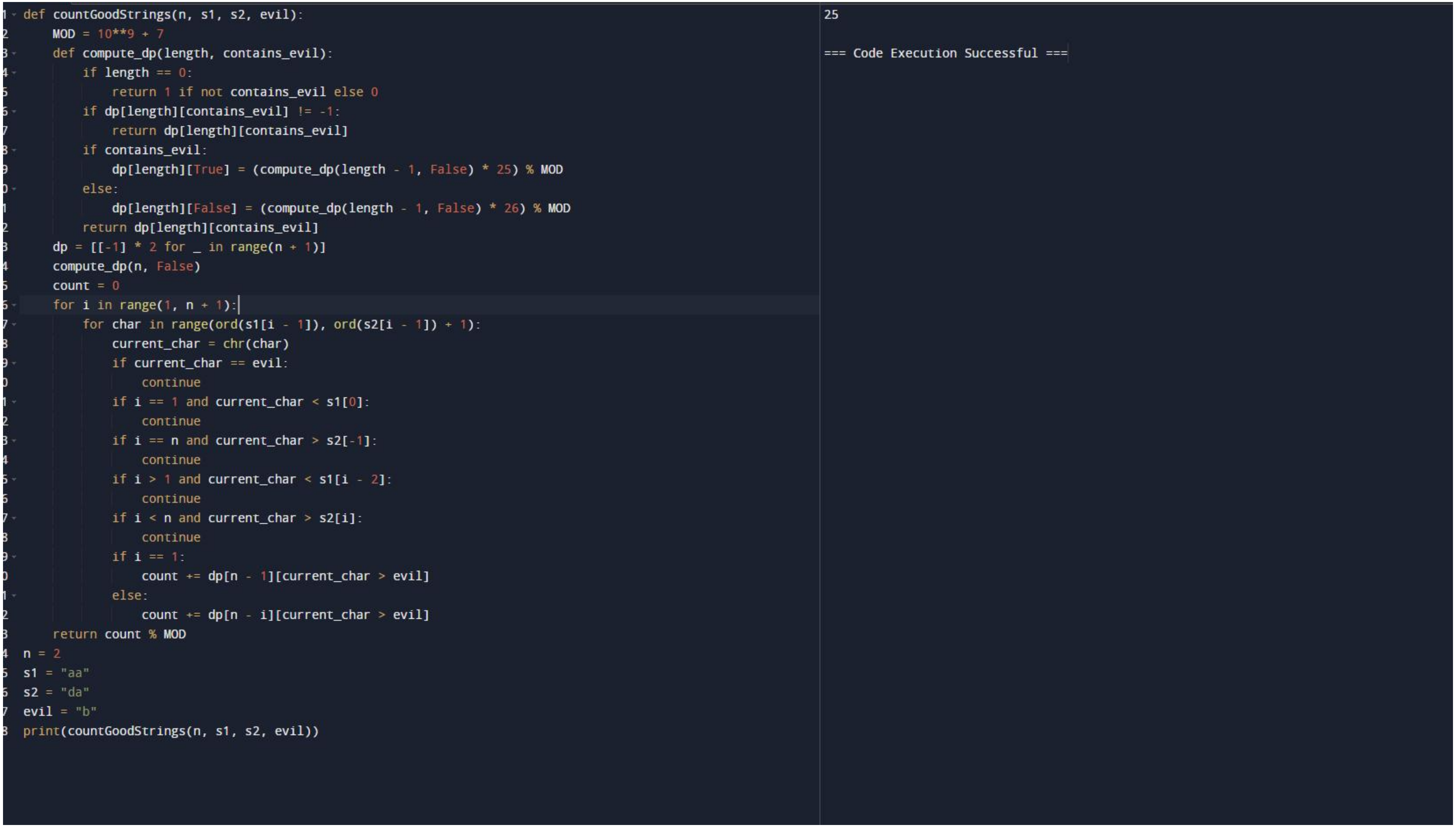
```python
def intersection(nums1, nums2):
    set1 = set(nums1)
    set2 = set(nums2)
    return list(set1.intersection(set2))
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
print(intersection(nums1, nums2))
```

```
[2]

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



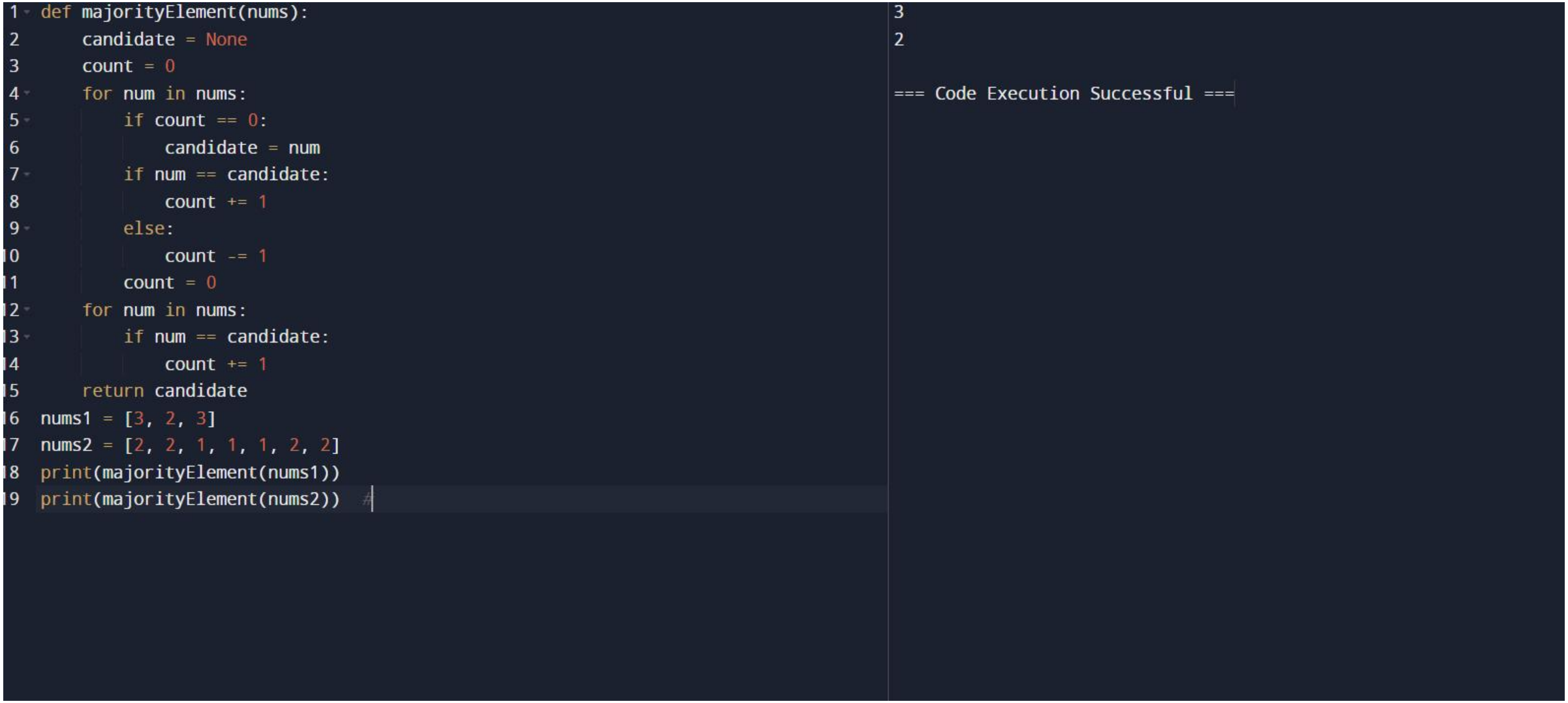
```python
import heapq
def findKthLargest(nums, k):
    min_heap = []
    for num in nums:
        heapq.heappush(min_heap, num)
        if len(min_heap) > k:
            heapq.heappop(min_heap)
    return min_heap[0]
nums1 = [3, 2, 1, 5, 6, 4]
k1 = 2
print(findKthLargest(nums1, k1))
nums2 = [3, 2, 3, 1, 2, 4, 5, 5, 6]
k2 = 4
print(findKthLargest(nums2, k2))
```

```
5
4

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



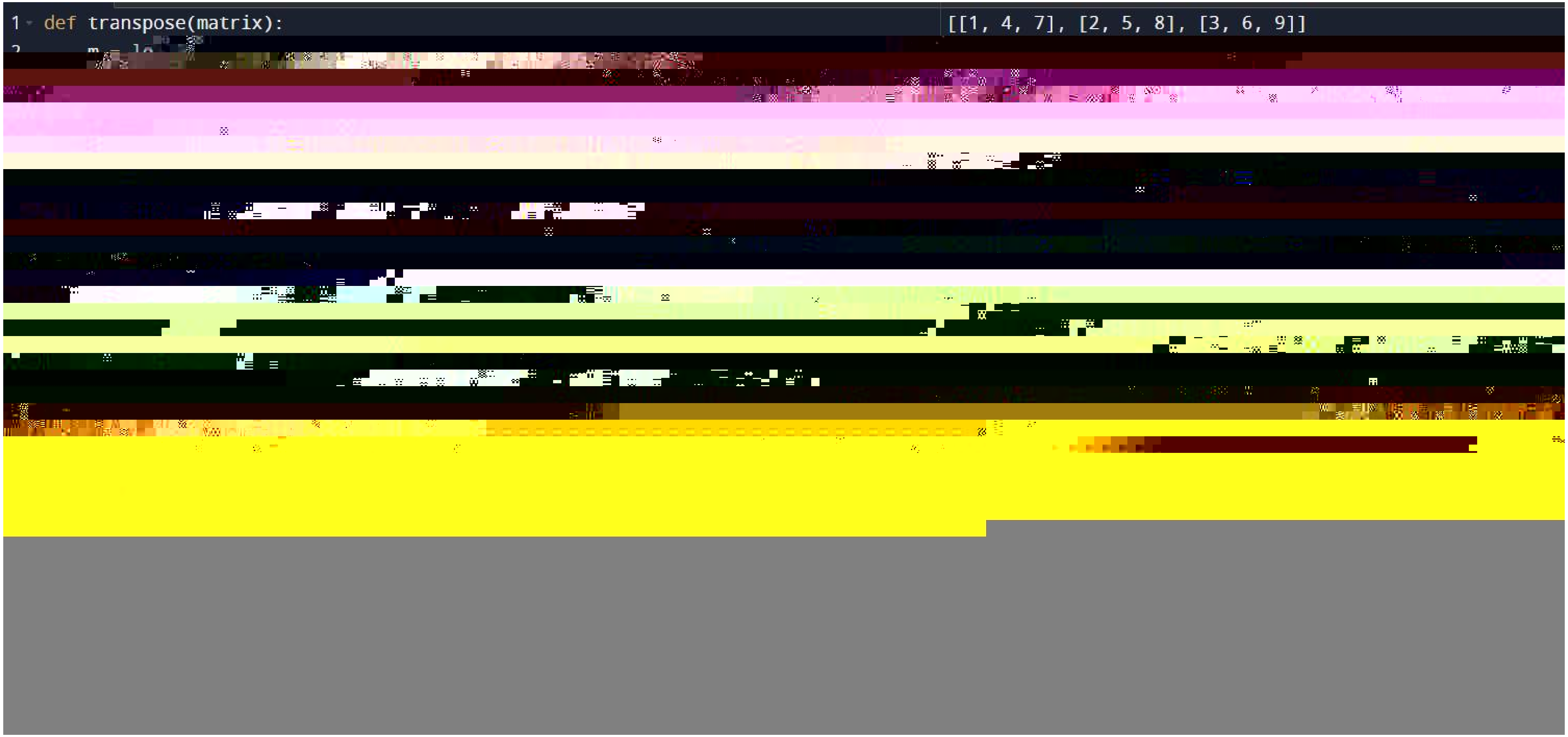
```python
def countGoodStrings(n, s1, s2, evil):
    MOD = 10**9 + 7
    def compute_dp(length, contains_evil):
        if length == 0:
            return 1 if not contains_evil else 0
        if dp[length][contains_evil] != -1:
            return dp[length][contains_evil]
        if contains_evil:
            dp[length][True] = (compute_dp(length - 1, False) * 25) % MOD
        else:
            dp[length][False] = (compute_dp(length - 1, False) * 26) % MOD
        return dp[length][contains_evil]
    dp = [[-1] * 2 for _ in range(n + 1)]
    compute_dp(n, False)
    count = 0
    for i in range(1, n + 1):
        for char in range(ord(s1[i - 1]), ord(s2[i - 1]) + 1):
            current_char = chr(char)
            if current_char == evil:
                continue
            if i == 1 and current_char < s1[0]:
                continue
            if i == n and current_char > s2[-1]:
                continue
            if i > 1 and current_char < s1[i - 2]:
                continue
            if i < n and current_char > s2[i]:
                continue
            if i == 1:
                count += dp[n - 1][current_char > evil]
            else:
                count += dp[n - i][current_char > evil]
    return count % MOD
n = 2
s1 = "aa"
s2 = "da"
evil = "b"
print(countGoodStrings(n, s1, s2, evil))
```

```
25

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



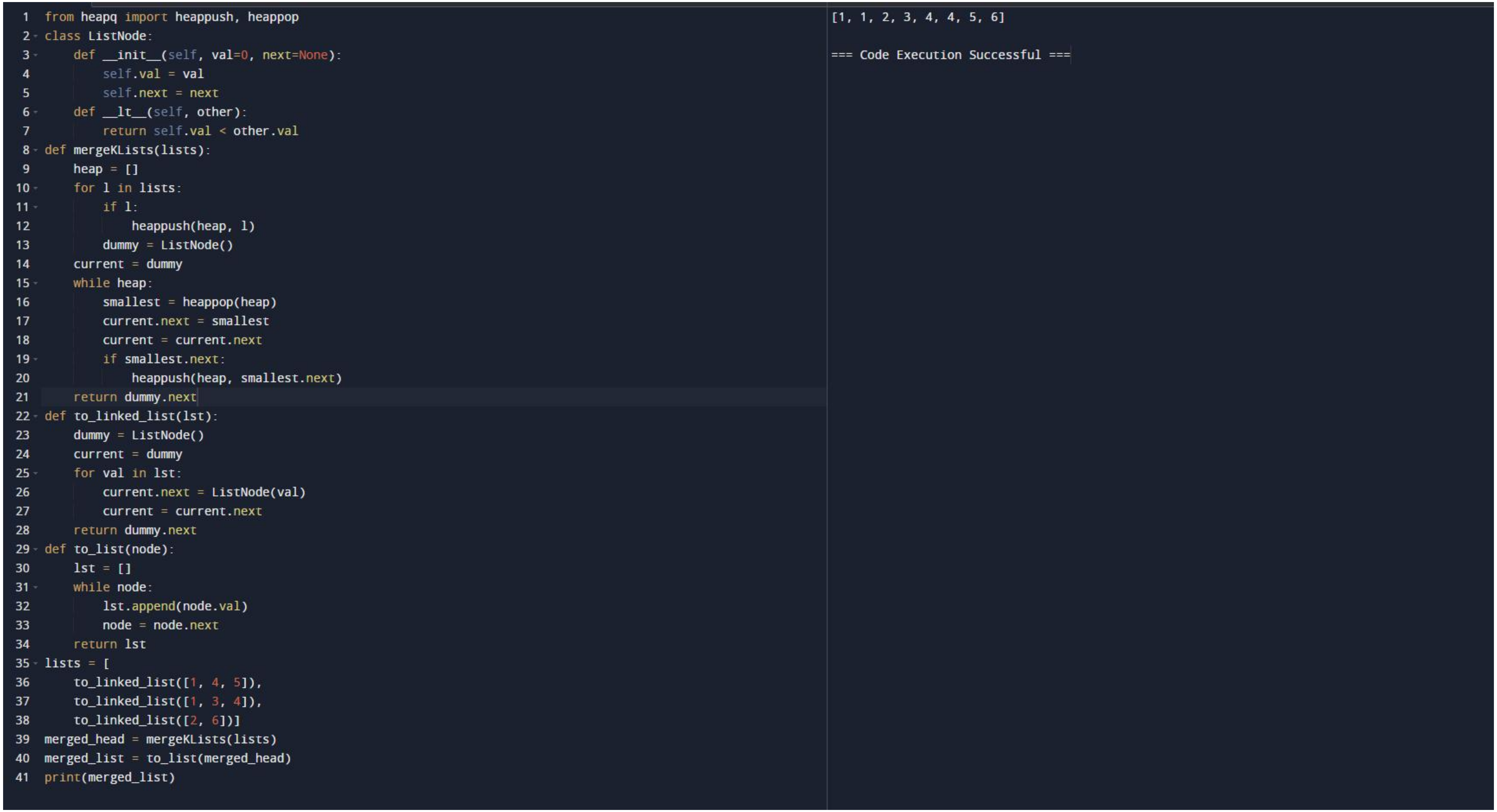
```python
def majorityElement(nums):
    candidate = None
    count = 0
    for num in nums:
        if count == 0:
            candidate = num
        if num == candidate:
            count += 1
        else:
            count -= 1
        count = 0
    for num in nums:
        if num == candidate:
            count += 1
    return candidate
nums1 = [3, 2, 3]
nums2 = [2, 2, 1, 1, 1, 2, 2]
print(majorityElement(nums1))
print(majorityElement(nums2))  #
```

```
3
2

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



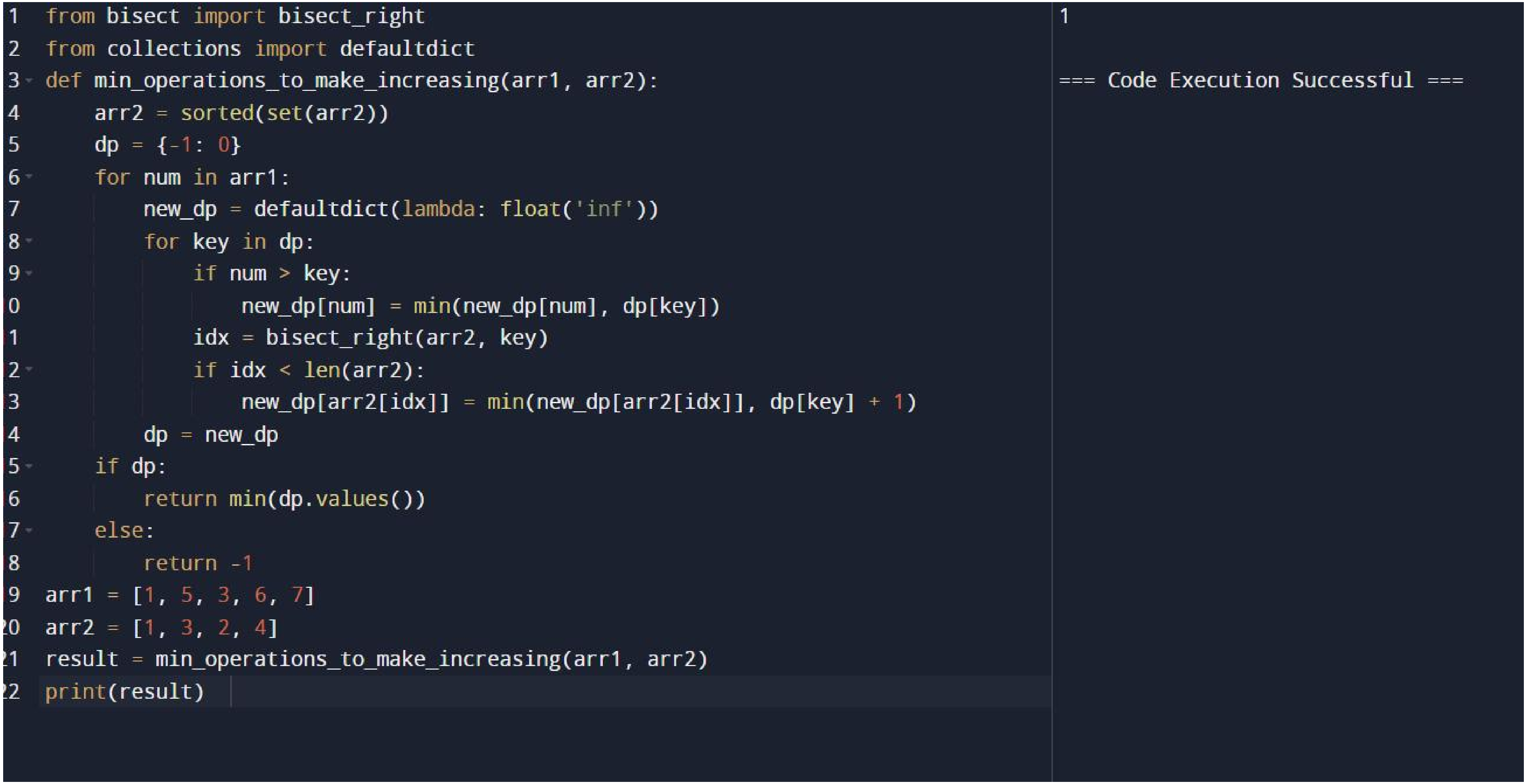
```
def transpose(matrix):
```

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```



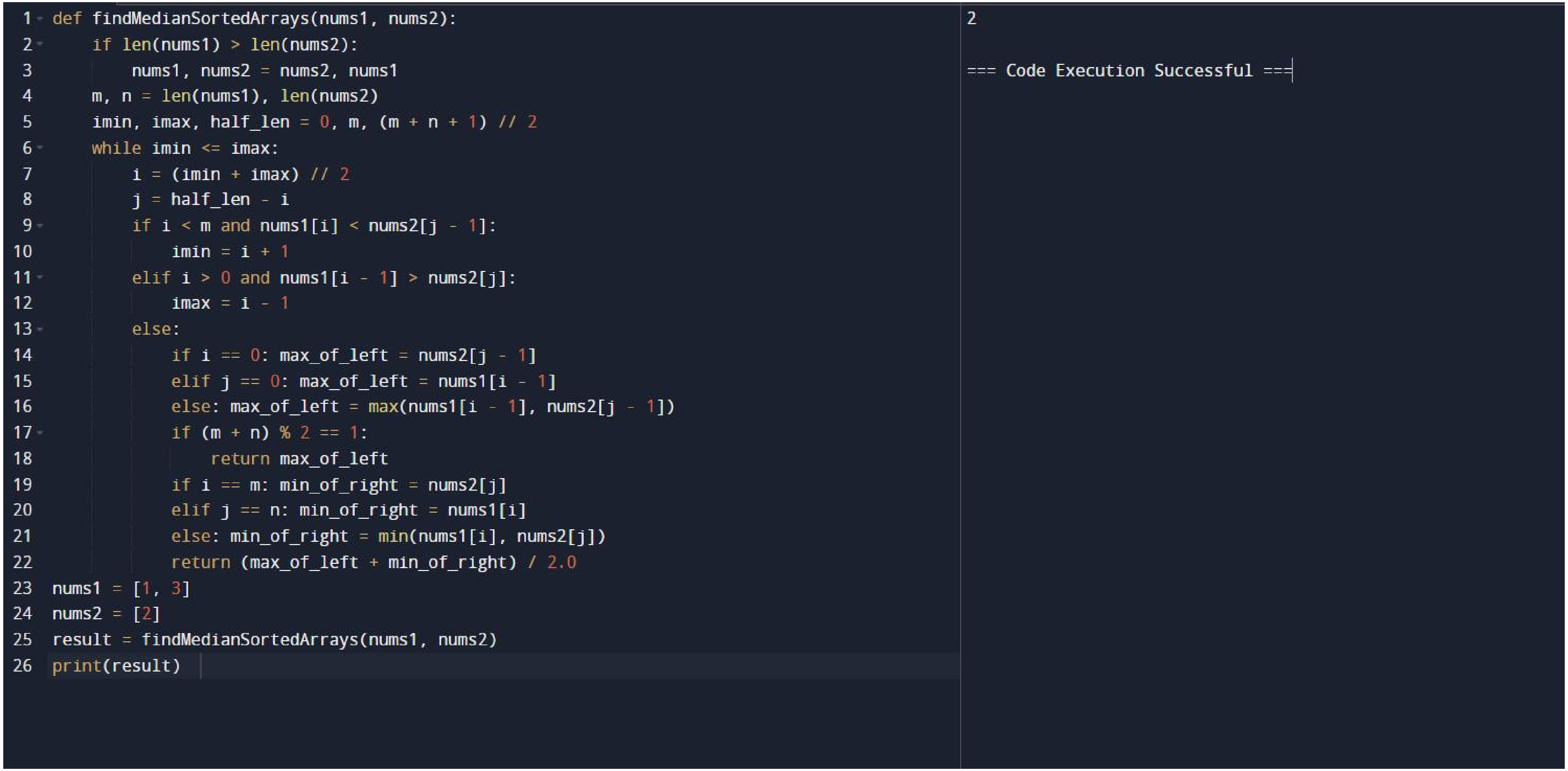
```python
from heapq import heappush, heappop
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
    def __lt__(self, other):
        return self.val < other.val
def mergeKLists(lists):
    heap = []
    for l in lists:
        if l:
            heappush(heap, l)
        dummy = ListNode()
    current = dummy
    while heap:
        smallest = heappop(heap)
        current.next = smallest
        current = current.next
        if smallest.next:
            heappush(heap, smallest.next)
    return dummy.next
def to_linked_list(lst):
    dummy = ListNode()
    current = dummy
    for val in lst:
        current.next = ListNode(val)
        current = current.next
    return dummy.next
def to_list(node):
    lst = []
    while node:
        lst.append(node.val)
        node = node.next
    return lst
lists = [
    to_linked_list([1, 4, 5]),
    to_linked_list([1, 3, 4]),
    to_linked_list([2, 6])]
merged_head = mergeKLists(lists)
merged_list = to_list(merged_head)
print(merged_list)
```

```
[1, 1, 2, 3, 4, 4, 5, 6]

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



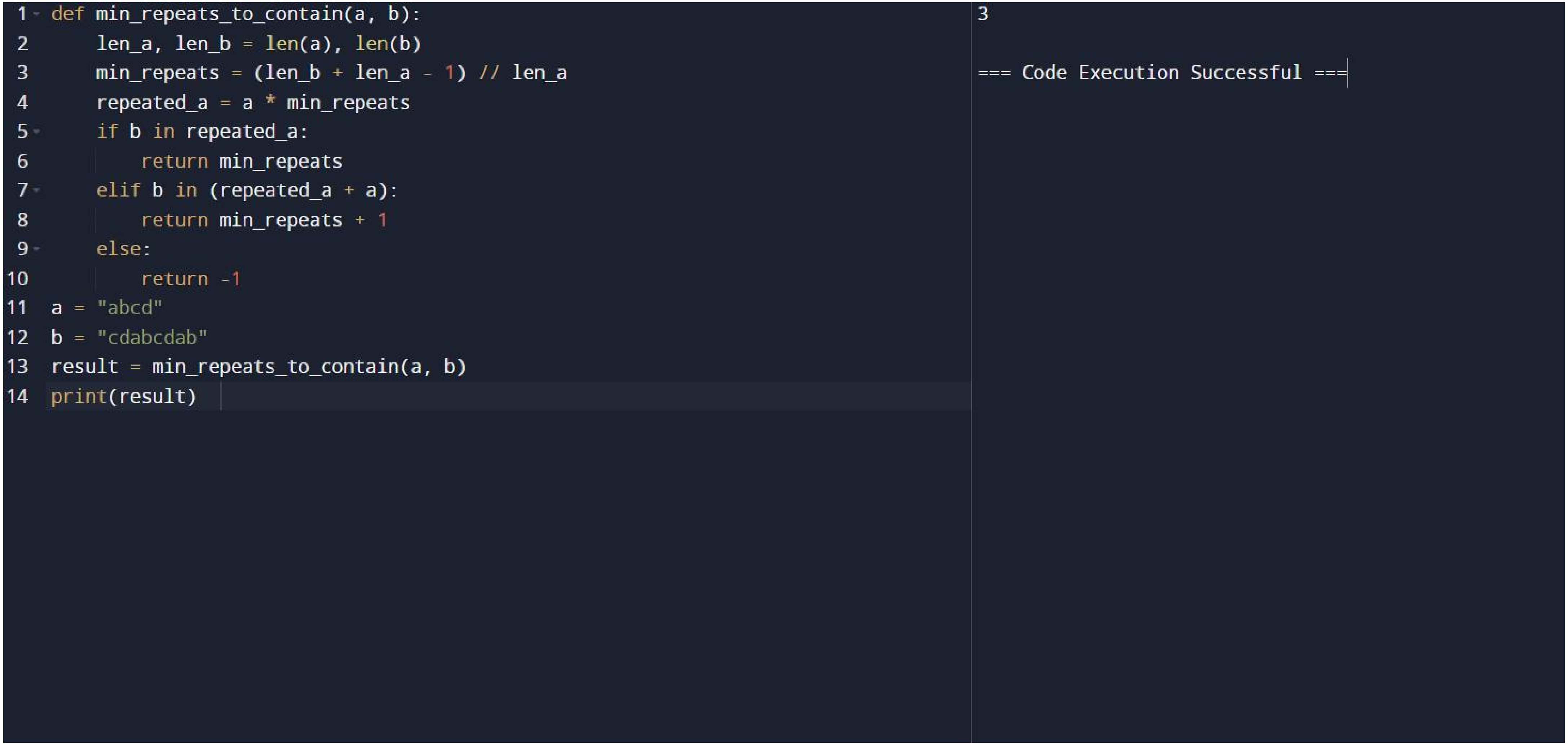
```python
from bisect import bisect_right
from collections import defaultdict
def min_operations_to_make_increasing(arr1, arr2):
    arr2 = sorted(set(arr2))
    dp = {-1: 0}
    for num in arr1:
        new_dp = defaultdict(lambda: float('inf'))
        for key in dp:
            if num > key:
                new_dp[num] = min(new_dp[num], dp[key])
            idx = bisect_right(arr2, key)
            if idx < len(arr2):
                new_dp[arr2[idx]] = min(new_dp[arr2[idx]], dp[key] + 1)
        dp = new_dp
    if dp:
        return min(dp.values())
    else:
        return -1
arr1 = [1, 5, 3, 6, 7]
arr2 = [1, 3, 2, 4]
result = min_operations_to_make_increasing(arr1, arr2)
print(result)
```

```
1


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



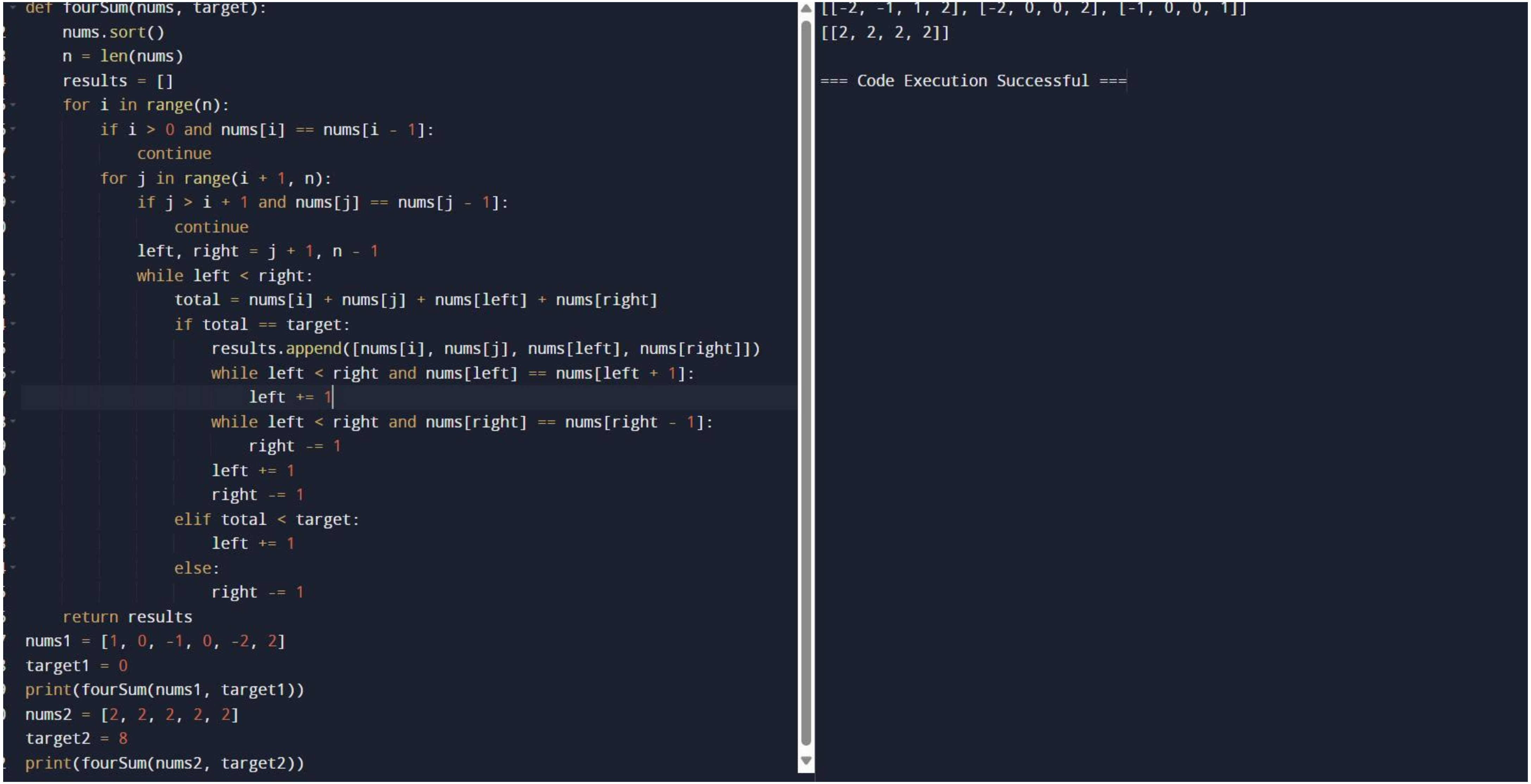
```python
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    m, n = len(nums1), len(nums2)
    imin, imax, half_len = 0, m, (m + n + 1) // 2
    while imin <= imax:
        i = (imin + imax) // 2
        j = half_len - i
        if i < m and nums1[i] < nums2[j - 1]:
            imin = i + 1
        elif i > 0 and nums1[i - 1] > nums2[j]:
            imax = i - 1
        else:
            if i == 0: max_of_left = nums2[j - 1]
            elif j == 0: max_of_left = nums1[i - 1]
            else: max_of_left = max(nums1[i - 1], nums2[j - 1])
            if (m + n) % 2 == 1:
                return max_of_left
            if i == m: min_of_right = nums2[j]
            elif j == n: min_of_right = nums1[i]
            else: min_of_right = min(nums1[i], nums2[j])
            return (max_of_left + min_of_right) / 2.0
nums1 = [1, 3]
nums2 = [2]
result = findMedianSortedArrays(nums1, nums2)
print(result)
```

```
2

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



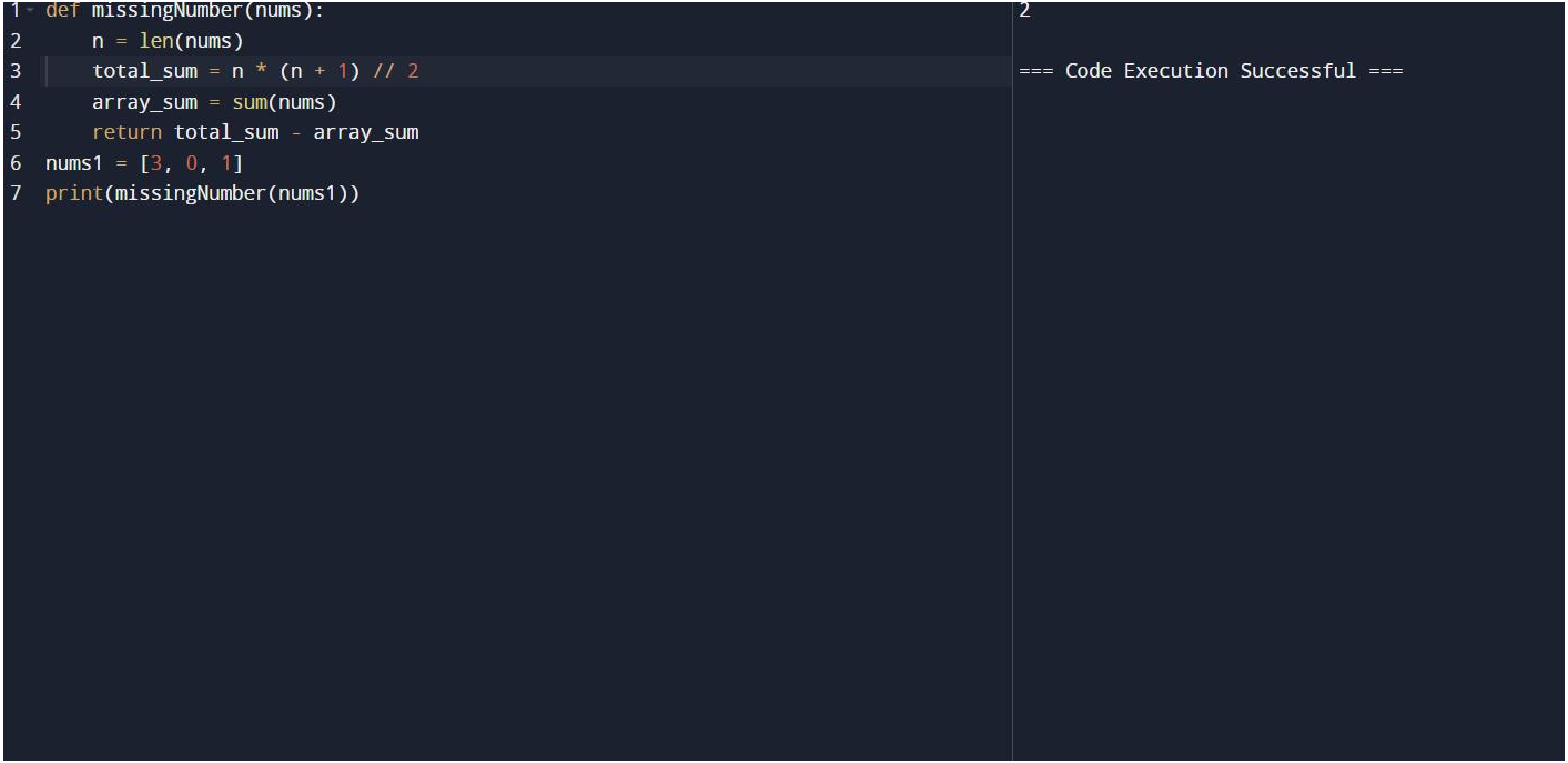
```python
def min_repeats_to_contain(a, b):
    len_a, len_b = len(a), len(b)
    min_repeats = (len_b + len_a - 1) // len_a
    repeated_a = a * min_repeats
    if b in repeated_a:
        return min_repeats
    elif b in (repeated_a + a):
        return min_repeats + 1
    else:
        return -1
a = "abcd"
b = "cdabcdab"
result = min_repeats_to_contain(a, b)
print(result)
```

```
3

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



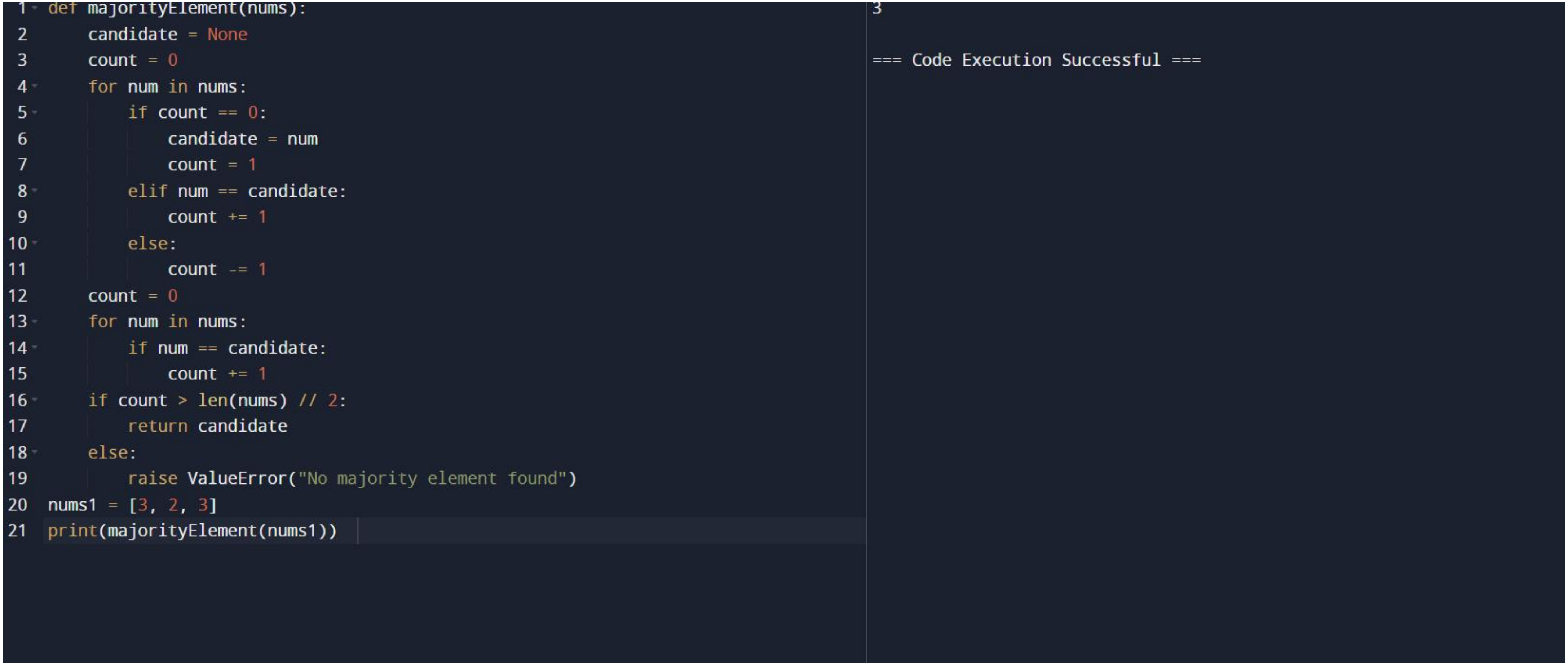
```python
def fourSum(nums, target):
    nums.sort()
    n = len(nums)
    results = []
    for i in range(n):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        for j in range(i + 1, n):
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue
            left, right = j + 1, n - 1
            while left < right:
                total = nums[i] + nums[j] + nums[left] + nums[right]
                if total == target:
                    results.append([nums[i], nums[j], nums[left], nums[right]])
                    while left < right and nums[left] == nums[left + 1]:
                        left += 1
                    while left < right and nums[right] == nums[right - 1]:
                        right -= 1
                    left += 1
                    right -= 1
                elif total < target:
                    left += 1
                else:
                    right -= 1
    return results
nums1 = [1, 0, -1, 0, -2, 2]
target1 = 0
print(fourSum(nums1, target1))
nums2 = [2, 2, 2, 2, 2]
target2 = 8
print(fourSum(nums2, target2))
```

```
[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
[[2, 2, 2, 2]]

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



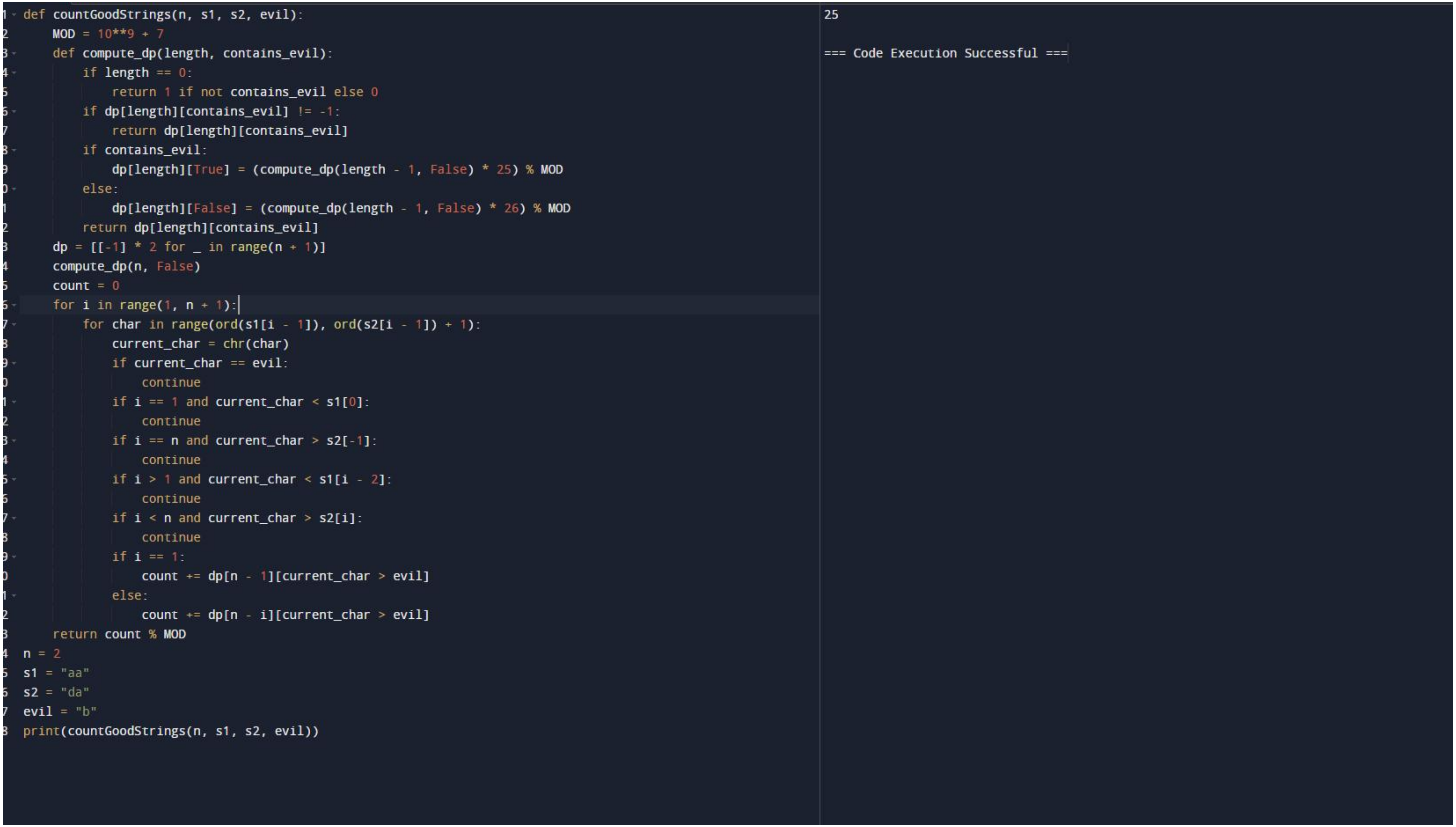
```python
def missingNumber(nums):
    n = len(nums)
    total_sum = n * (n + 1) // 2
    array_sum = sum(nums)
    return total_sum - array_sum
nums1 = [3, 0, 1]
print(missingNumber(nums1))
```

```
2


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



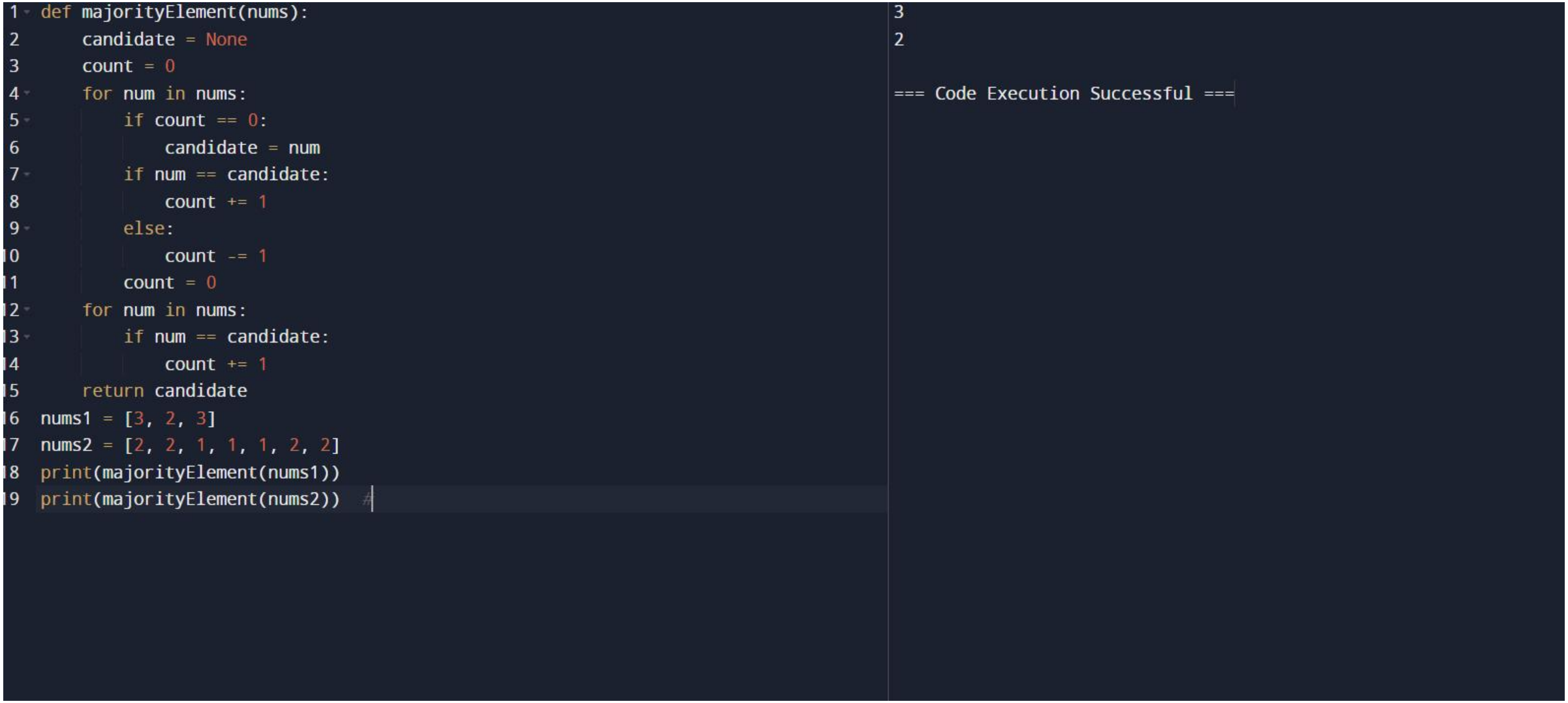
```
def majorityElement(nums):
    candidate = None
    count = 0
    for num in nums:
        if count == 0:
            candidate = num
            count = 1
        elif num == candidate:
            count += 1
        else:
            count -= 1
    count = 0
    for num in nums:
        if num == candidate:
            count += 1
    if count > len(nums) // 2:
        return candidate
    else:
        raise ValueError("No majority element found")
nums1 = [3, 2, 3]
print(majorityElement(nums1))
```

```
3

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



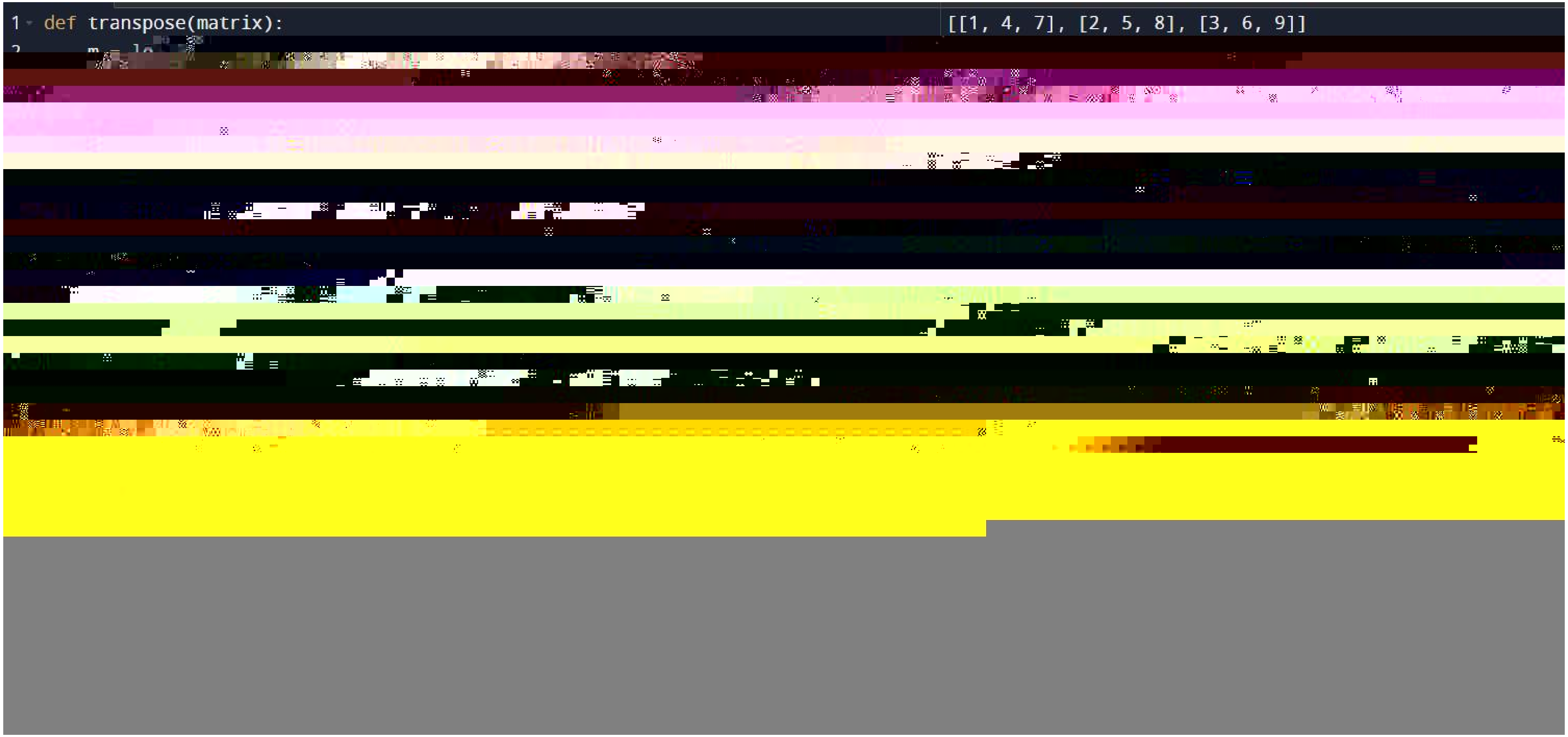
```python
def countGoodStrings(n, s1, s2, evil):
    MOD = 10**9 + 7
    def compute_dp(length, contains_evil):
        if length == 0:
            return 1 if not contains_evil else 0
        if dp[length][contains_evil] != -1:
            return dp[length][contains_evil]
        if contains_evil:
            dp[length][True] = (compute_dp(length - 1, False) * 25) % MOD
        else:
            dp[length][False] = (compute_dp(length - 1, False) * 26) % MOD
        return dp[length][contains_evil]
    dp = [[-1] * 2 for _ in range(n + 1)]
    compute_dp(n, False)
    count = 0
    for i in range(1, n + 1):
        for char in range(ord(s1[i - 1]), ord(s2[i - 1]) + 1):
            current_char = chr(char)
            if current_char == evil:
                continue
            if i == 1 and current_char < s1[0]:
                continue
            if i == n and current_char > s2[-1]:
                continue
            if i > 1 and current_char < s1[i - 2]:
                continue
            if i < n and current_char > s2[i]:
                continue
            if i == 1:
                count += dp[n - 1][current_char > evil]
            else:
                count += dp[n - i][current_char > evil]
    return count % MOD
n = 2
s1 = "aa"
s2 = "da"
evil = "b"
print(countGoodStrings(n, s1, s2, evil))
```

```
25

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



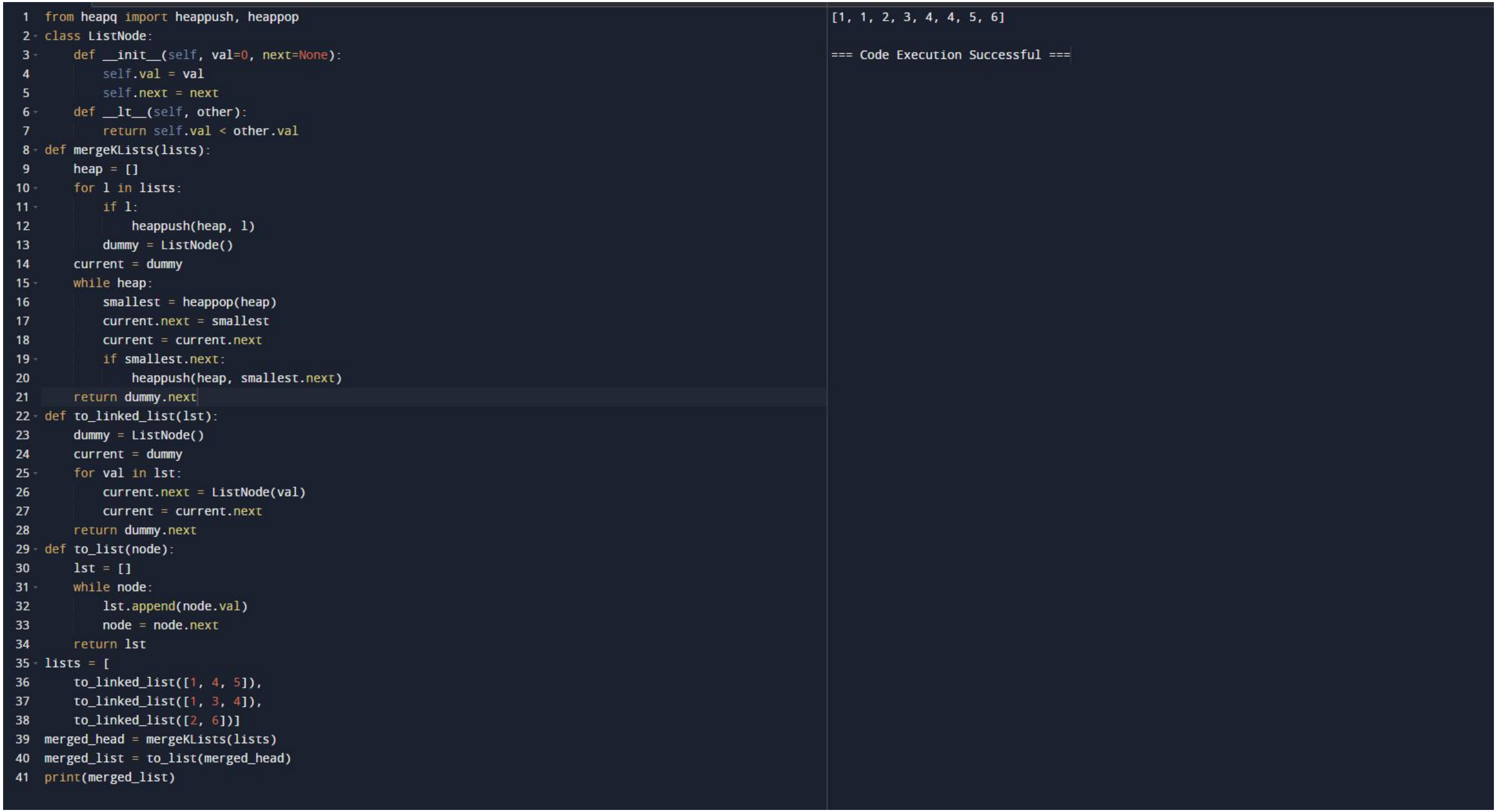
```python
def majorityElement(nums):
    candidate = None
    count = 0
    for num in nums:
        if count == 0:
            candidate = num
        if num == candidate:
            count += 1
        else:
            count -= 1
        count = 0
    for num in nums:
        if num == candidate:
            count += 1
    return candidate
nums1 = [3, 2, 3]
nums2 = [2, 2, 1, 1, 1, 2, 2]
print(majorityElement(nums1))
print(majorityElement(nums2))  #
```

```
3
2

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



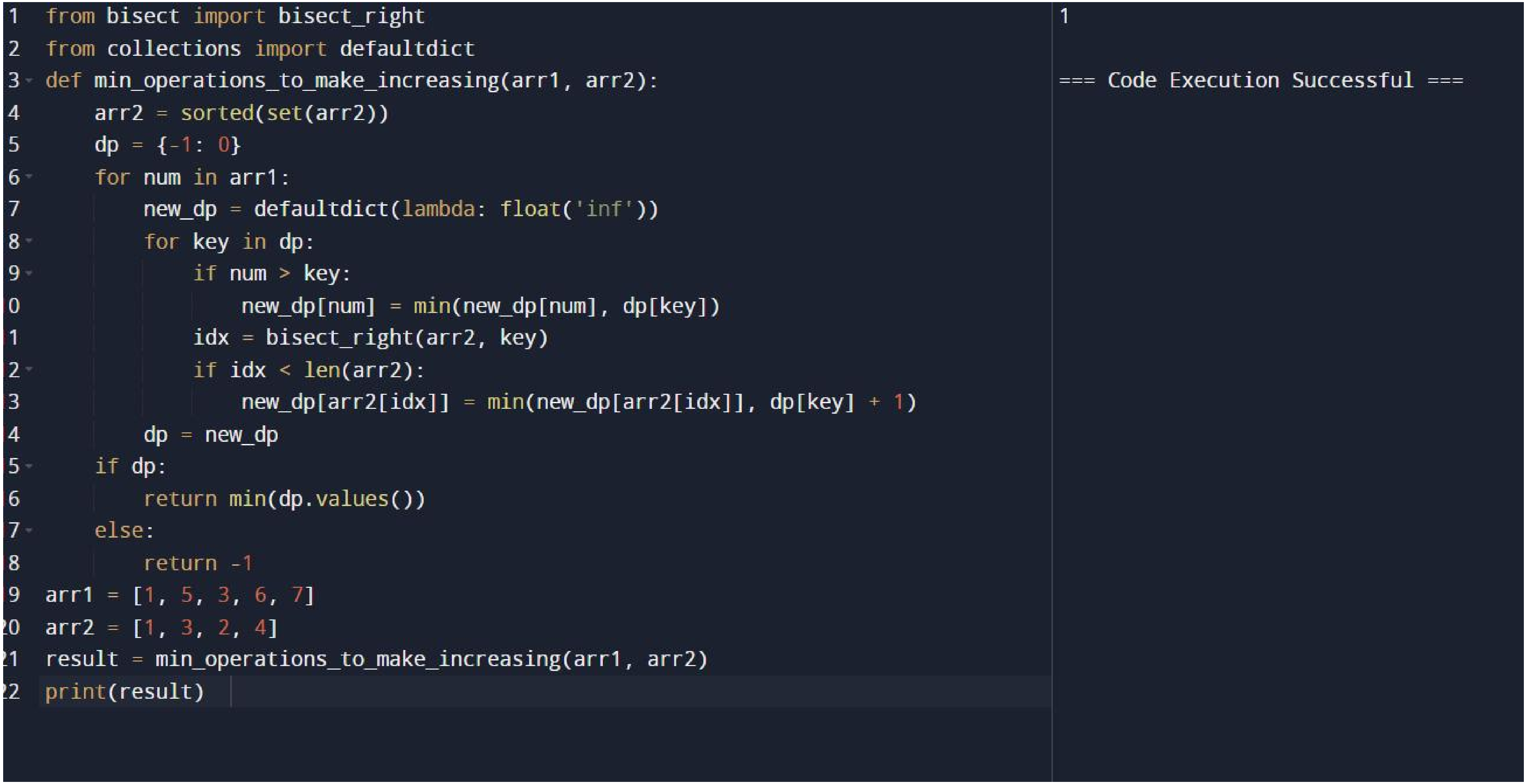
```
def transpose(matrix):
    m = le
```

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```



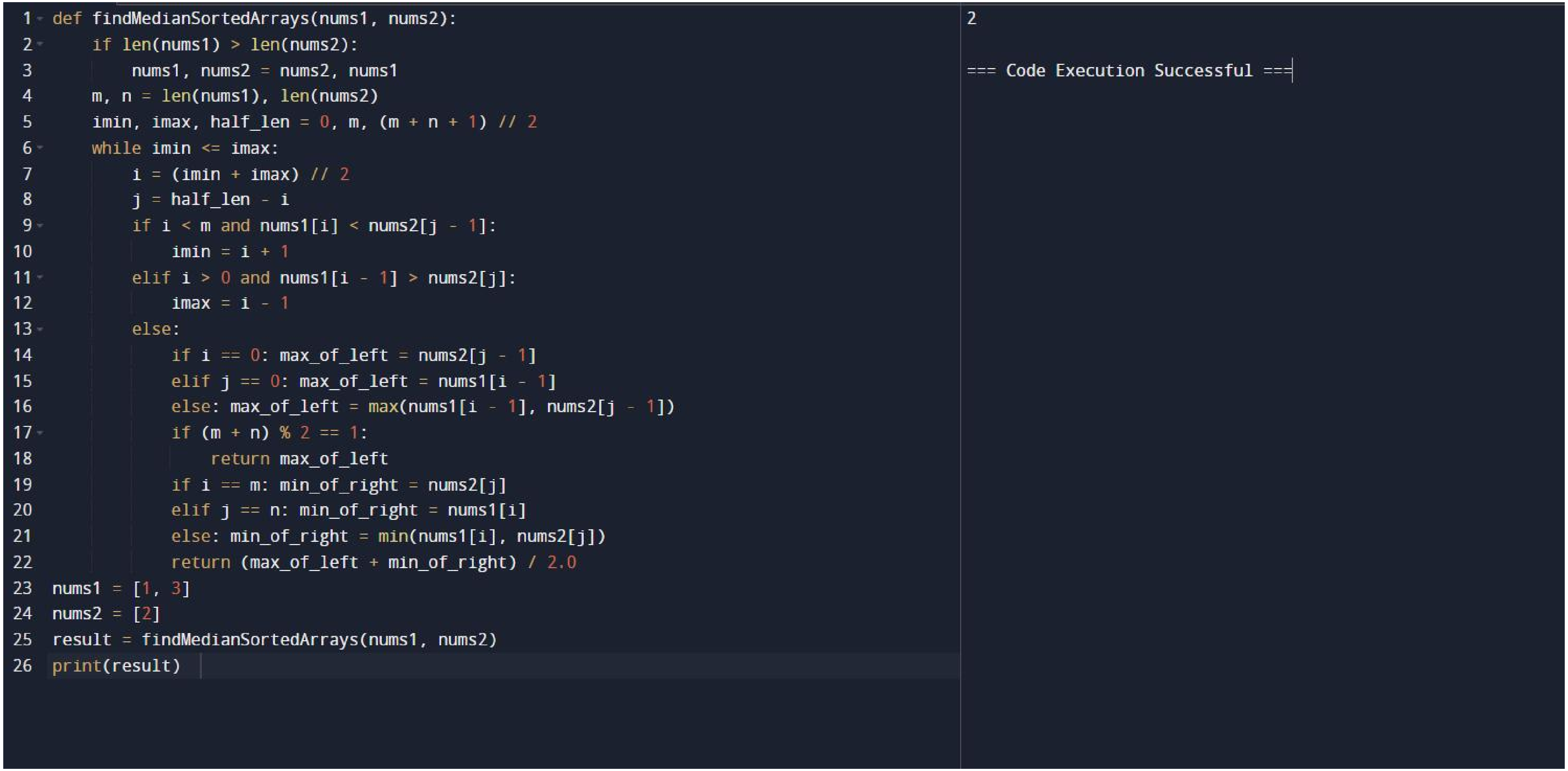
```python
from heapq import heappush, heappop
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
    def __lt__(self, other):
        return self.val < other.val
def mergeKLists(lists):
    heap = []
    for l in lists:
        if l:
            heappush(heap, l)
        dummy = ListNode()
    current = dummy
    while heap:
        smallest = heappop(heap)
        current.next = smallest
        current = current.next
        if smallest.next:
            heappush(heap, smallest.next)
    return dummy.next
def to_linked_list(lst):
    dummy = ListNode()
    current = dummy
    for val in lst:
        current.next = ListNode(val)
        current = current.next
    return dummy.next
def to_list(node):
    lst = []
    while node:
        lst.append(node.val)
        node = node.next
    return lst
lists = [
    to_linked_list([1, 4, 5]),
    to_linked_list([1, 3, 4]),
    to_linked_list([2, 6])]
merged_head = mergeKLists(lists)
merged_list = to_list(merged_head)
print(merged_list)
```

```
[1, 1, 2, 3, 4, 4, 5, 6]

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



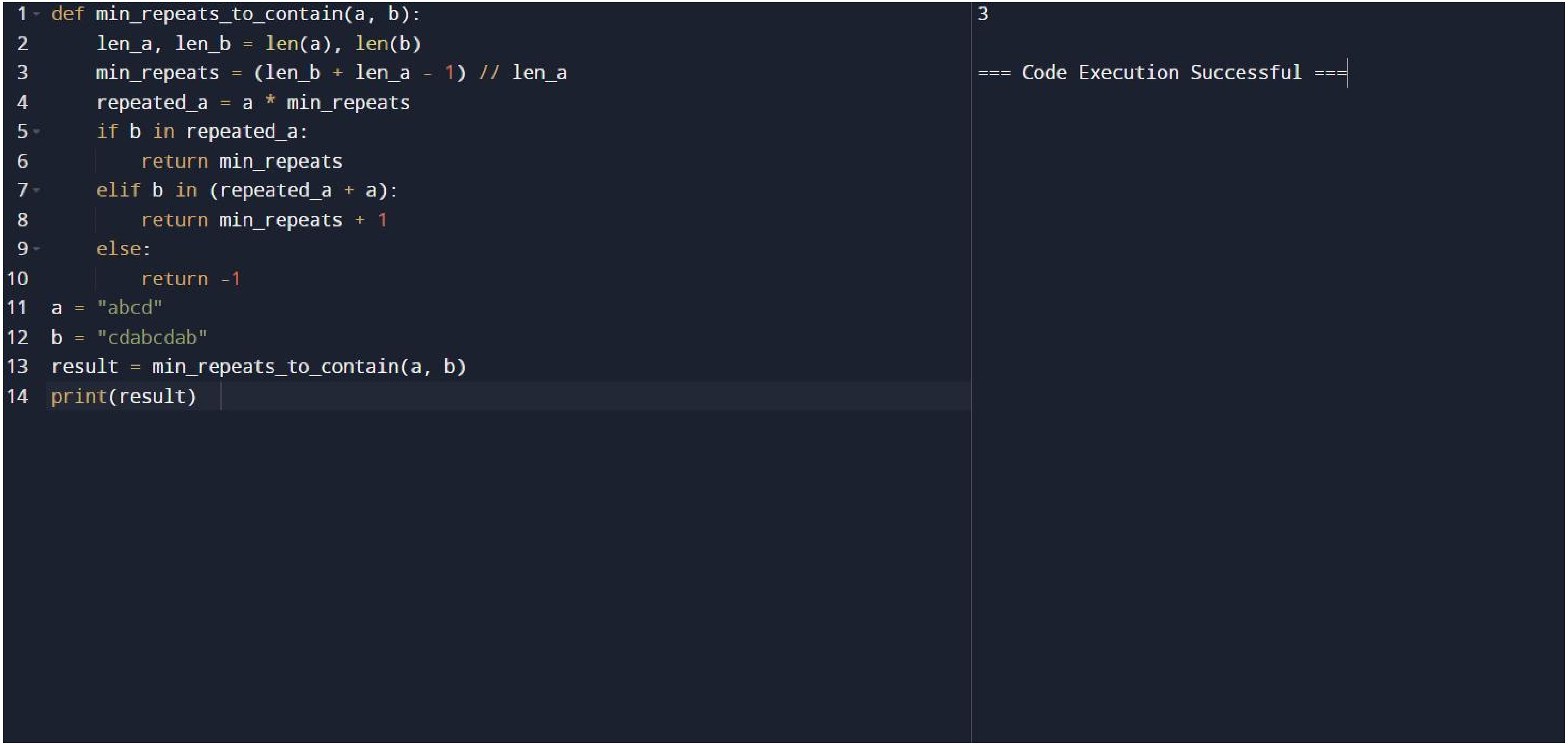
```python
from bisect import bisect_right
from collections import defaultdict
def min_operations_to_make_increasing(arr1, arr2):
    arr2 = sorted(set(arr2))
    dp = {-1: 0}
    for num in arr1:
        new_dp = defaultdict(lambda: float('inf'))
        for key in dp:
            if num > key:
                new_dp[num] = min(new_dp[num], dp[key])
            idx = bisect_right(arr2, key)
            if idx < len(arr2):
                new_dp[arr2[idx]] = min(new_dp[arr2[idx]], dp[key] + 1)
        dp = new_dp
    if dp:
        return min(dp.values())
    else:
        return -1
arr1 = [1, 5, 3, 6, 7]
arr2 = [1, 3, 2, 4]
result = min_operations_to_make_increasing(arr1, arr2)
print(result)
```

```
1


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



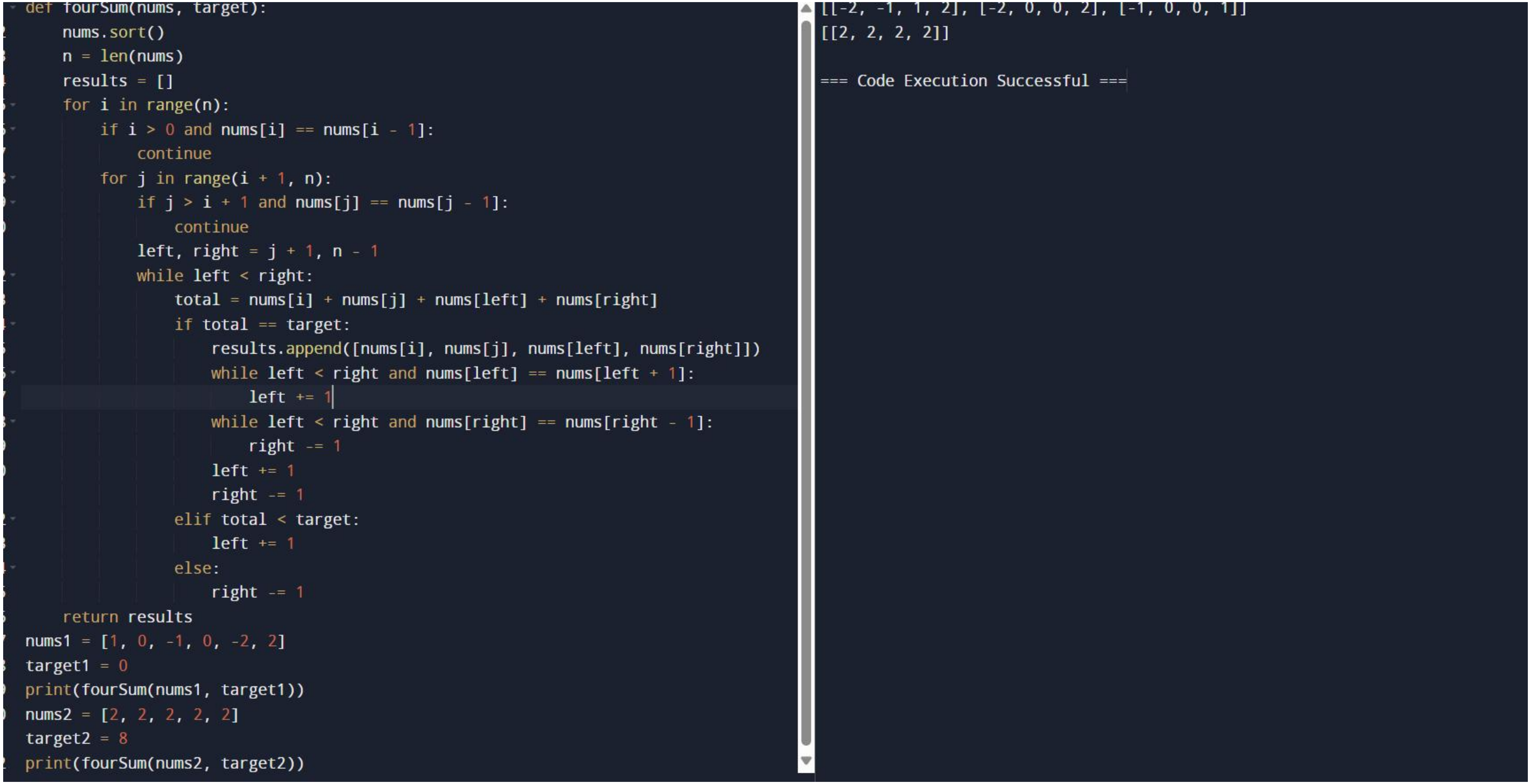
```python
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    m, n = len(nums1), len(nums2)
    imin, imax, half_len = 0, m, (m + n + 1) // 2
    while imin <= imax:
        i = (imin + imax) // 2
        j = half_len - i
        if i < m and nums1[i] < nums2[j - 1]:
            imin = i + 1
        elif i > 0 and nums1[i - 1] > nums2[j]:
            imax = i - 1
        else:
            if i == 0: max_of_left = nums2[j - 1]
            elif j == 0: max_of_left = nums1[i - 1]
            else: max_of_left = max(nums1[i - 1], nums2[j - 1])
            if (m + n) % 2 == 1:
                return max_of_left
            if i == m: min_of_right = nums2[j]
            elif j == n: min_of_right = nums1[i]
            else: min_of_right = min(nums1[i], nums2[j])
            return (max_of_left + min_of_right) / 2.0
nums1 = [1, 3]
nums2 = [2]
result = findMedianSortedArrays(nums1, nums2)
print(result)
```

```
2

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



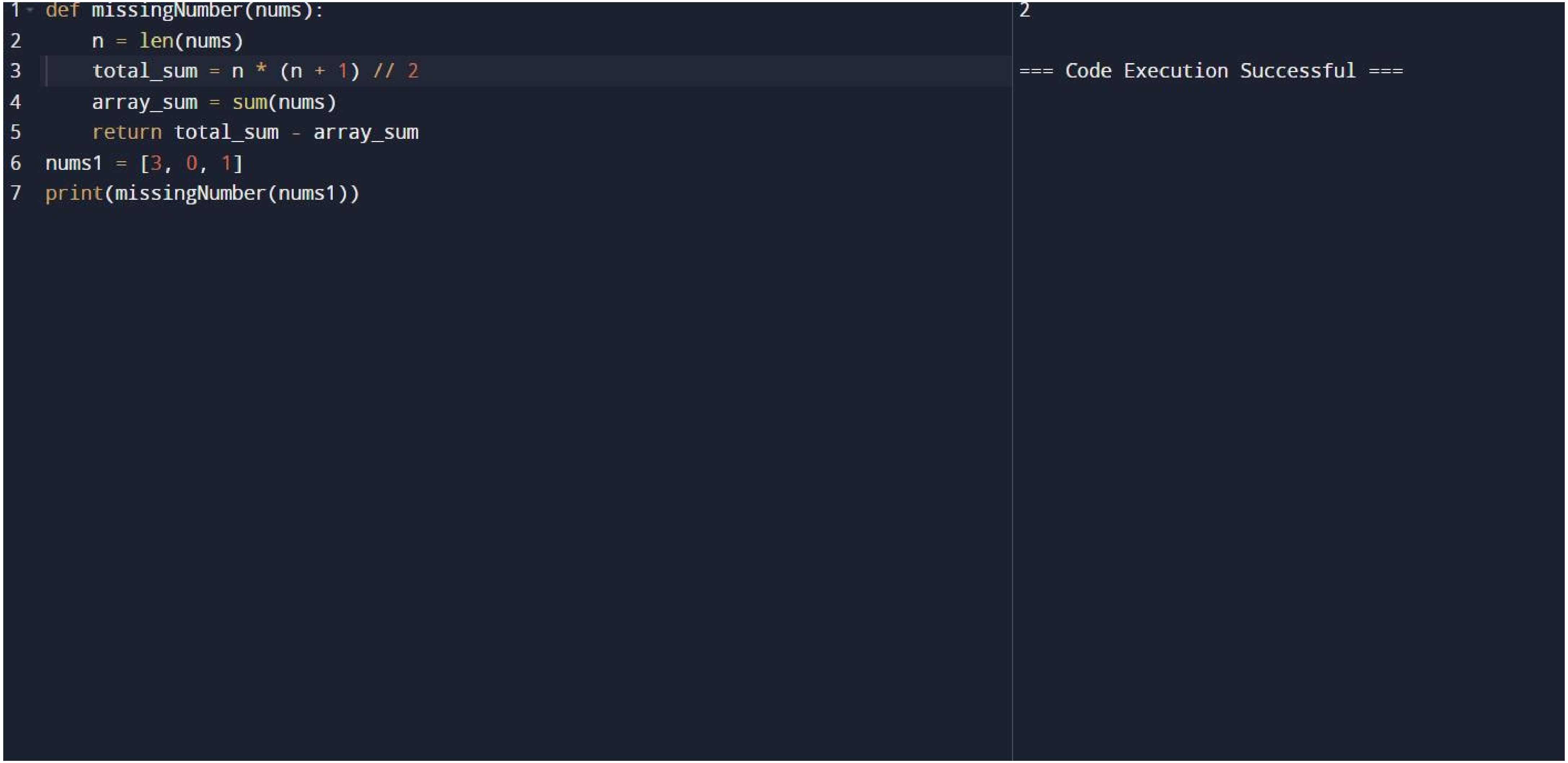
```python
def min_repeats_to_contain(a, b):
    len_a, len_b = len(a), len(b)
    min_repeats = (len_b + len_a - 1) // len_a
    repeated_a = a * min_repeats
    if b in repeated_a:
        return min_repeats
    elif b in (repeated_a + a):
        return min_repeats + 1
    else:
        return -1
a = "abcd"
b = "cdabcdab"
result = min_repeats_to_contain(a, b)
print(result)
```

```
3

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



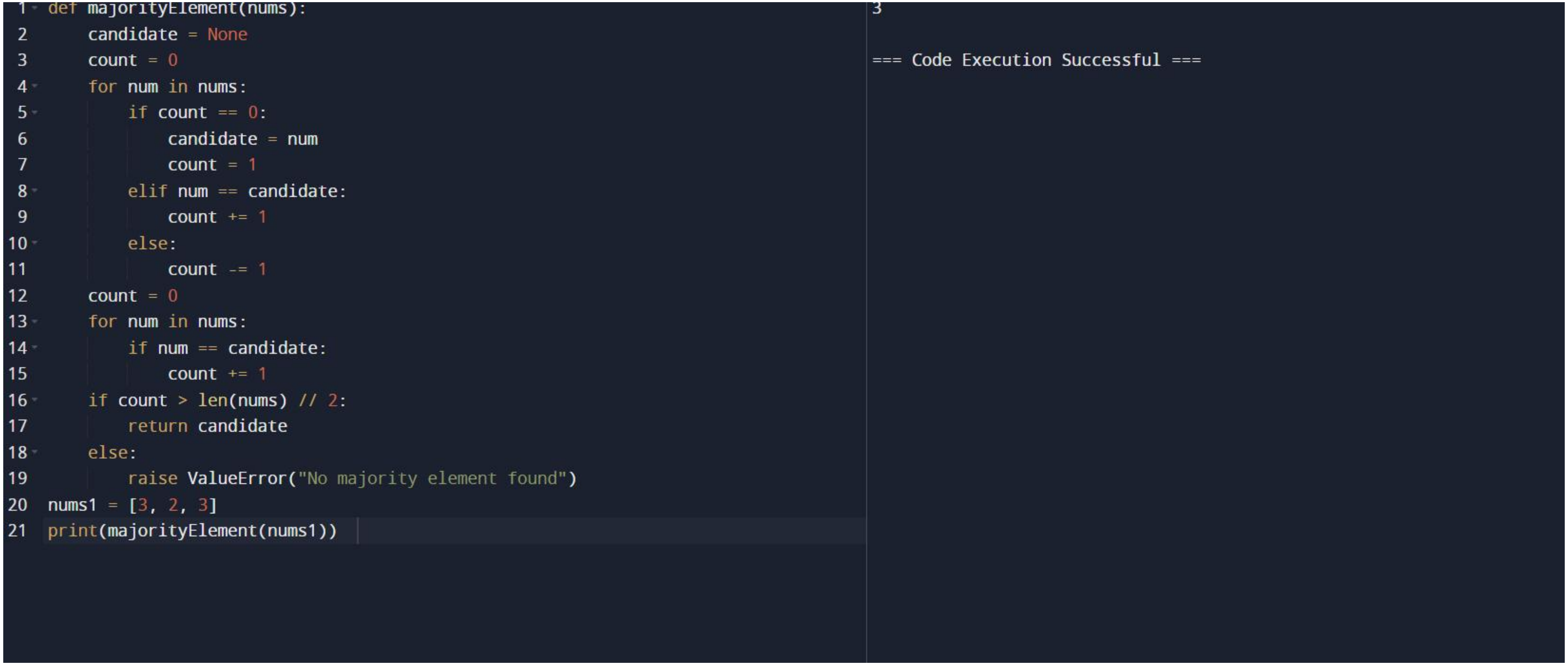
```
def fourSum(nums, target):
    nums.sort()
    n = len(nums)
    results = []
    for i in range(n):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        for j in range(i + 1, n):
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue
            left, right = j + 1, n - 1
            while left < right:
                total = nums[i] + nums[j] + nums[left] + nums[right]
                if total == target:
                    results.append([nums[i], nums[j], nums[left], nums[right]])
                    while left < right and nums[left] == nums[left + 1]:
                        left += 1
                    while left < right and nums[right] == nums[right - 1]:
                        right -= 1
                    left += 1
                    right -= 1
                elif total < target:
                    left += 1
                else:
                    right -= 1
    return results
nums1 = [1, 0, -1, 0, -2, 2]
target1 = 0
print(fourSum(nums1, target1))
nums2 = [2, 2, 2, 2, 2]
target2 = 8
print(fourSum(nums2, target2))
```

```
[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
[[2, 2, 2, 2]]

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



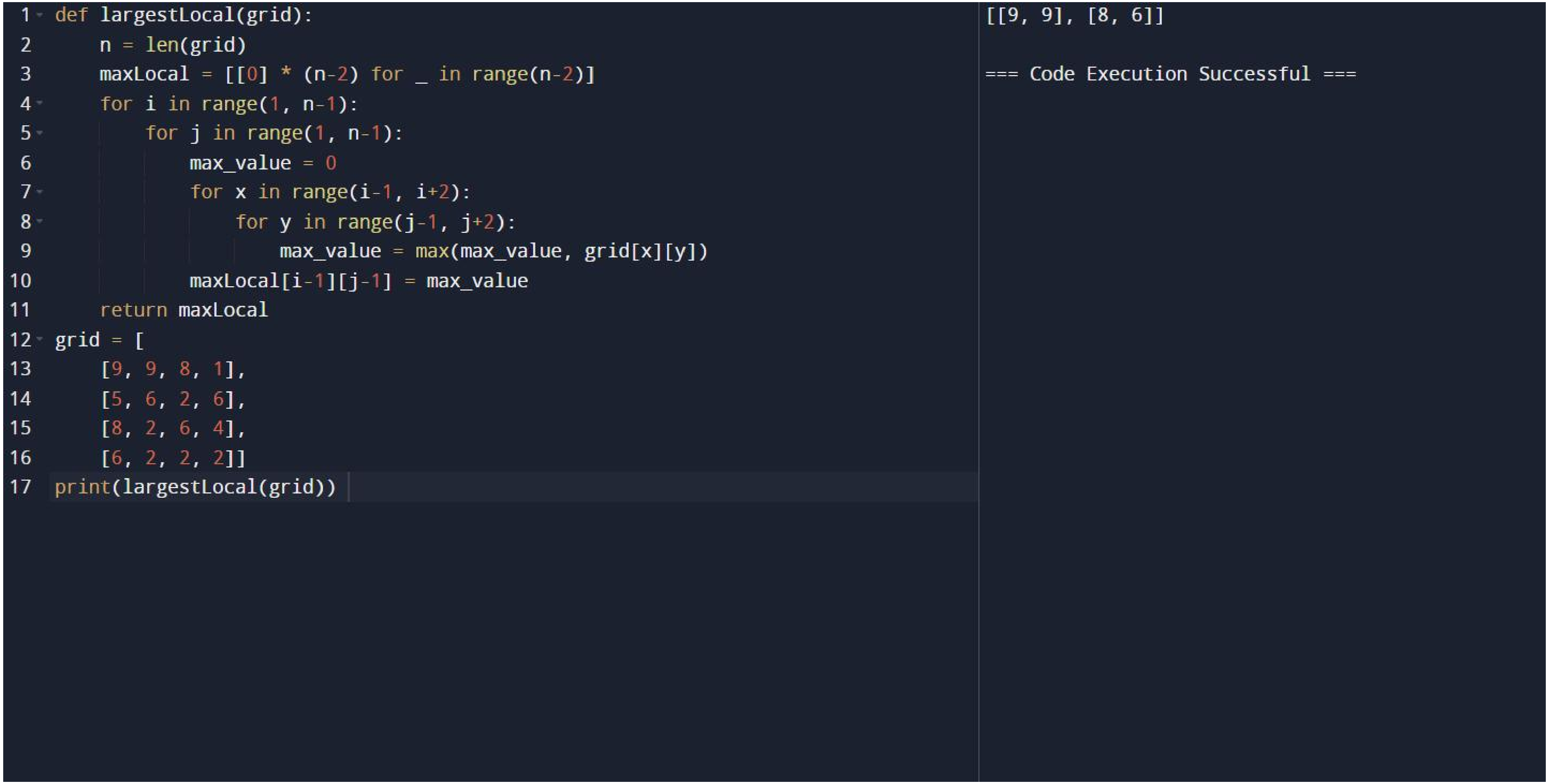
```python
def missingNumber(nums):
    n = len(nums)
    total_sum = n * (n + 1) // 2
    array_sum = sum(nums)
    return total_sum - array_sum
nums1 = [3, 0, 1]
print(missingNumber(nums1))
```

```
2


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



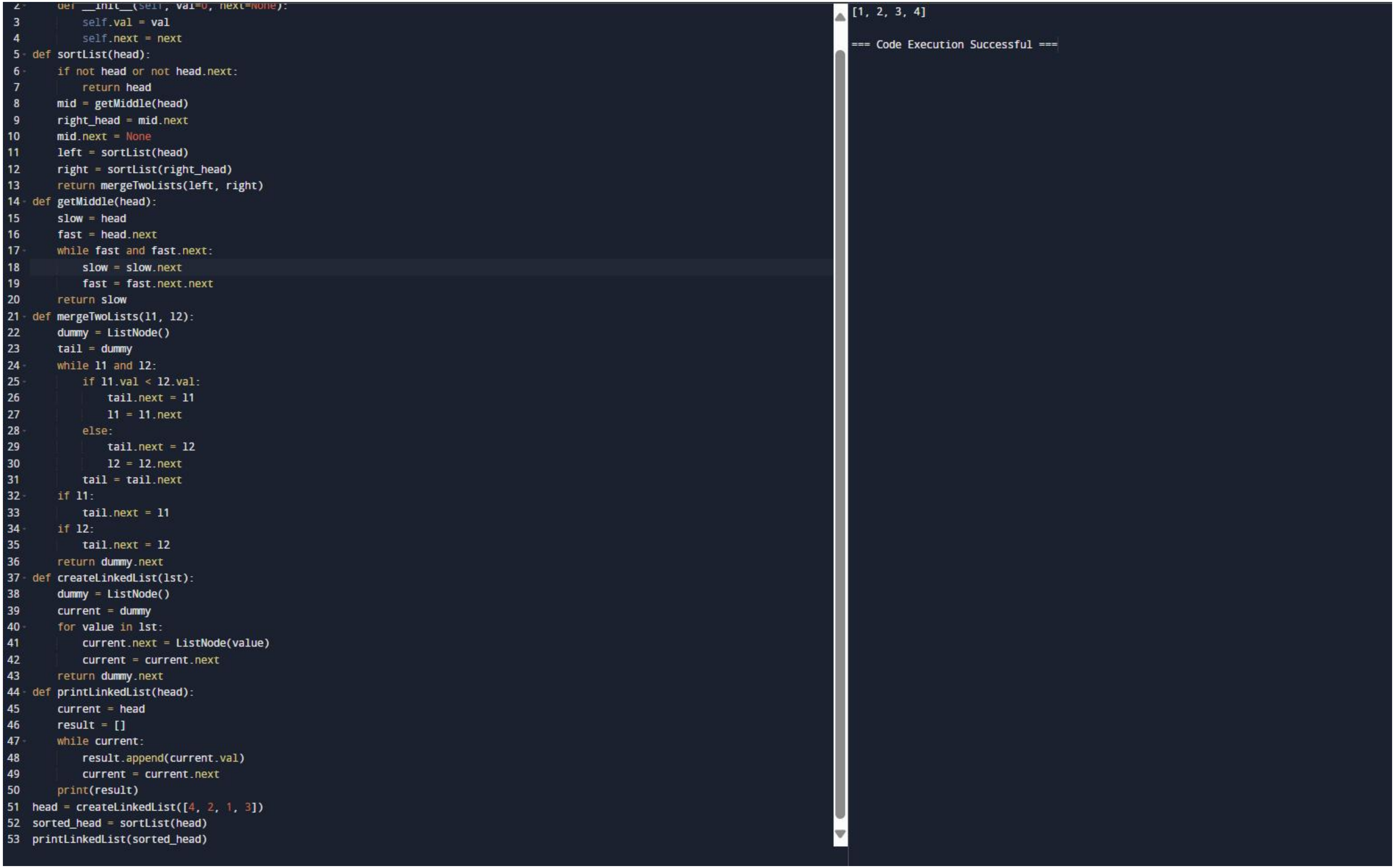
```python
def majorityElement(nums):
    candidate = None
    count = 0
    for num in nums:
        if count == 0:
            candidate = num
            count = 1
        elif num == candidate:
            count += 1
        else:
            count -= 1
    count = 0
    for num in nums:
        if num == candidate:
            count += 1
    if count > len(nums) // 2:
        return candidate
    else:
        raise ValueError("No majority element found")
nums1 = [3, 2, 3]
print(majorityElement(nums1))
```

```
3

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



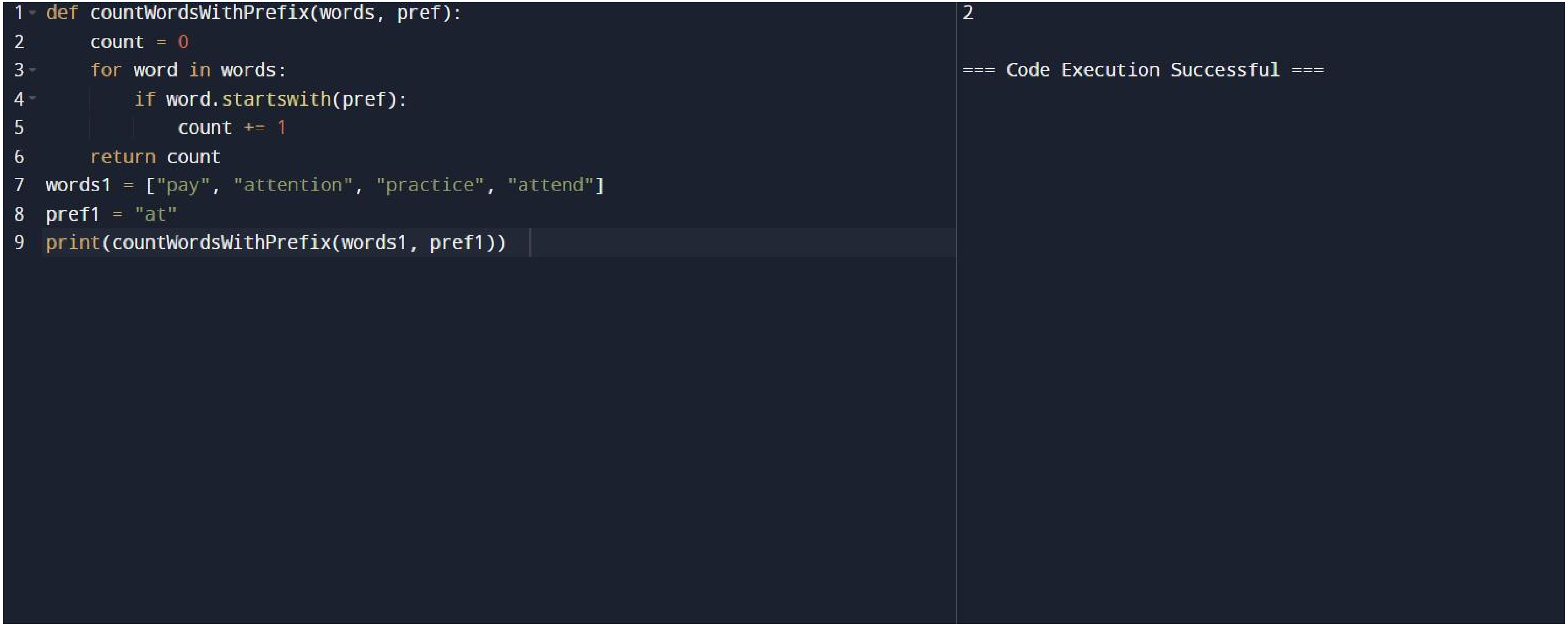
```python
def largestLocal(grid):
    n = len(grid)
    maxLocal = [[0] * (n-2) for _ in range(n-2)]
    for i in range(1, n-1):
        for j in range(1, n-1):
            max_value = 0
            for x in range(i-1, i+2):
                for y in range(j-1, j+2):
                    max_value = max(max_value, grid[x][y])
            maxLocal[i-1][j-1] = max_value
    return maxLocal
grid = [
    [9, 9, 8, 1],
    [5, 6, 2, 6],
    [8, 2, 6, 4],
    [6, 2, 2, 2]]
print(largestLocal(grid))
```

```
[[9, 9], [8, 6]]

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



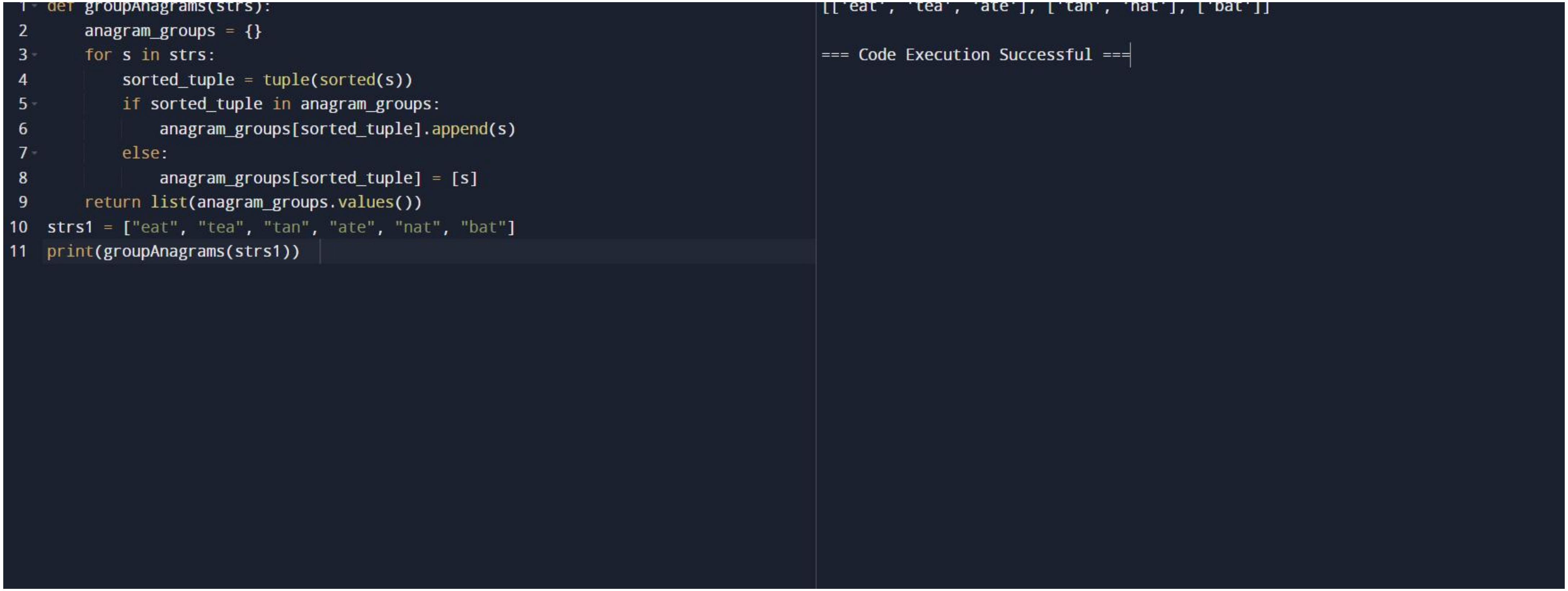
```
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def sortList(head):
    if not head or not head.next:
        return head
    mid = getMiddle(head)
    right_head = mid.next
    mid.next = None
    left = sortList(head)
    right = sortList(right_head)
    return mergeTwoLists(left, right)
def getMiddle(head):
    slow = head
    fast = head.next
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
    return slow
def mergeTwoLists(l1, l2):
    dummy = ListNode()
    tail = dummy
    while l1 and l2:
        if l1.val < l2.val:
            tail.next = l1
            l1 = l1.next
        else:
            tail.next = l2
            l2 = l2.next
        tail = tail.next
    if l1:
        tail.next = l1
    if l2:
        tail.next = l2
    return dummy.next
def createLinkedList(lst):
    dummy = ListNode()
    current = dummy
    for value in lst:
        current.next = ListNode(value)
        current = current.next
    return dummy.next
def printLinkedList(head):
    current = head
    result = []
    while current:
        result.append(current.val)
        current = current.next
    print(result)
head = createLinkedList([4, 2, 1, 3])
sorted_head = sortList(head)
printLinkedList(sorted_head)
```

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

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



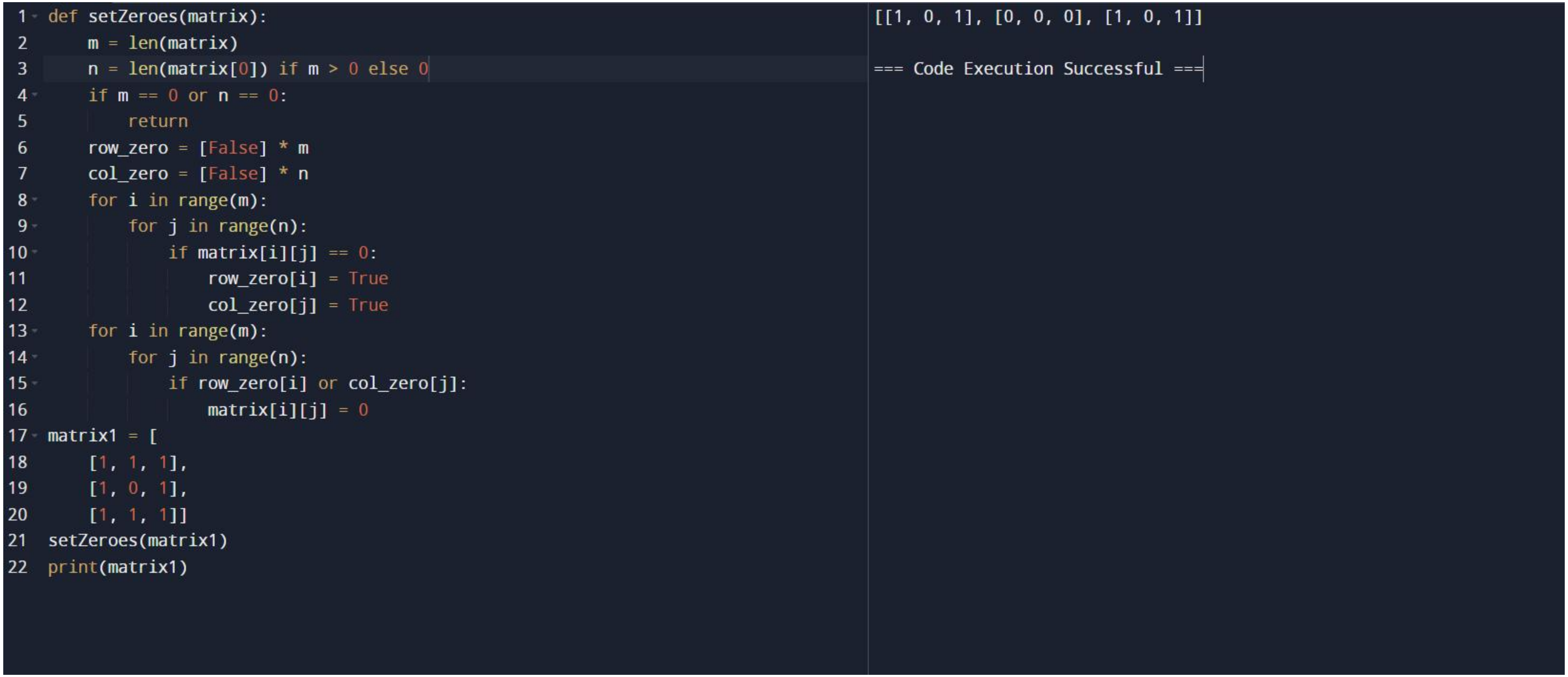
```python
def countWordsWithPrefix(words, pref):
    count = 0
    for word in words:
        if word.startswith(pref):
            count += 1
    return count
words1 = ["pay", "attention", "practice", "attend"]
pref1 = "at"
print(countWordsWithPrefix(words1, pref1))
```

```
2

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



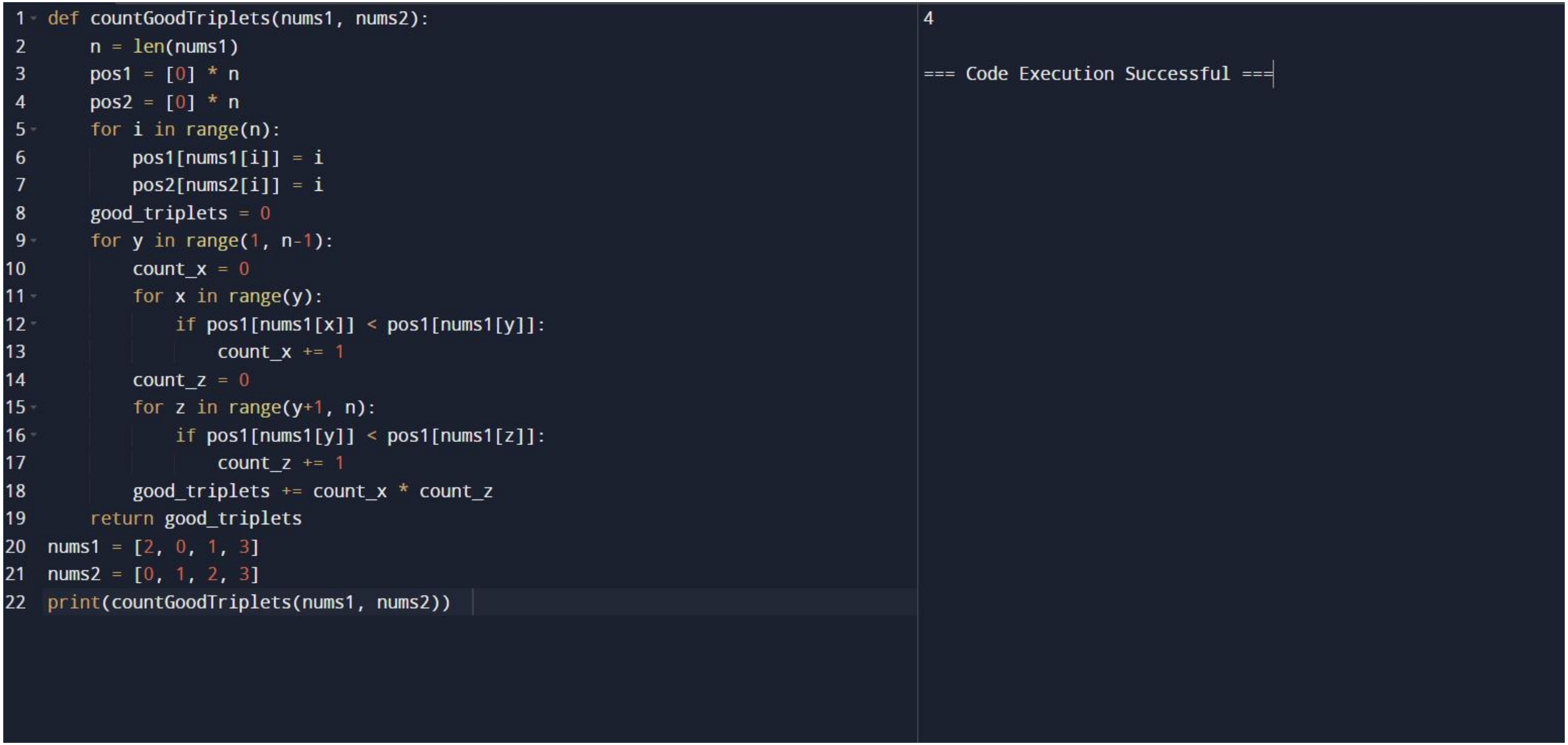
```python
def groupAnagrams(strs):
    anagram_groups = {}
    for s in strs:
        sorted_tuple = tuple(sorted(s))
        if sorted_tuple in anagram_groups:
            anagram_groups[sorted_tuple].append(s)
        else:
            anagram_groups[sorted_tuple] = [s]
    return list(anagram_groups.values())
strs1 = ["eat", "tea", "tan", "ate", "nat", "bat"]
print(groupAnagrams(strs1))
```

```
[['eat', 'tea', 'ate'], ['tan', 'nat'], ['bat']]

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



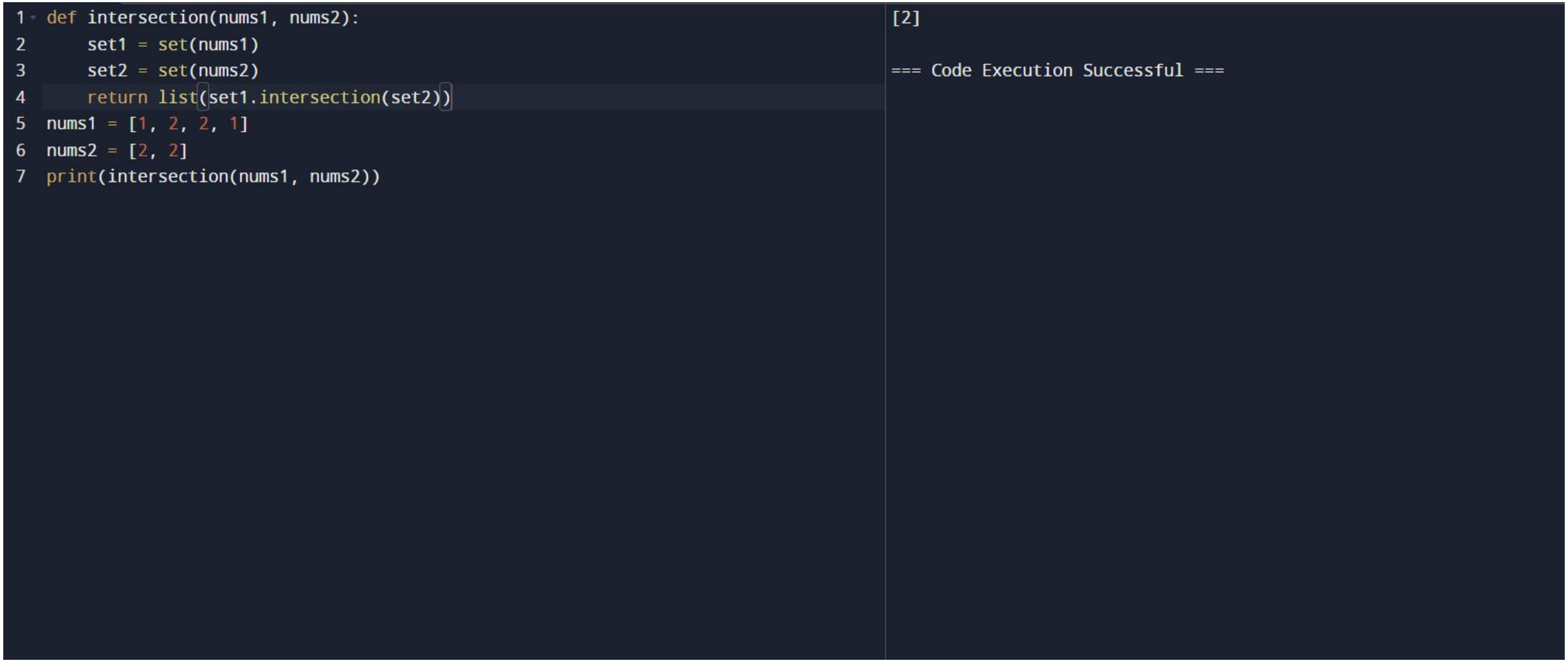
```python
def setZeroes(matrix):
    m = len(matrix)
    n = len(matrix[0]) if m > 0 else 0
    if m == 0 or n == 0:
        return
    row_zero = [False] * m
    col_zero = [False] * n
    for i in range(m):
        for j in range(n):
            if matrix[i][j] == 0:
                row_zero[i] = True
                col_zero[j] = True
    for i in range(m):
        for j in range(n):
            if row_zero[i] or col_zero[j]:
                matrix[i][j] = 0
matrix1 = [
    [1, 1, 1],
    [1, 0, 1],
    [1, 1, 1]]
setZeroes(matrix1)
print(matrix1)
```

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

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



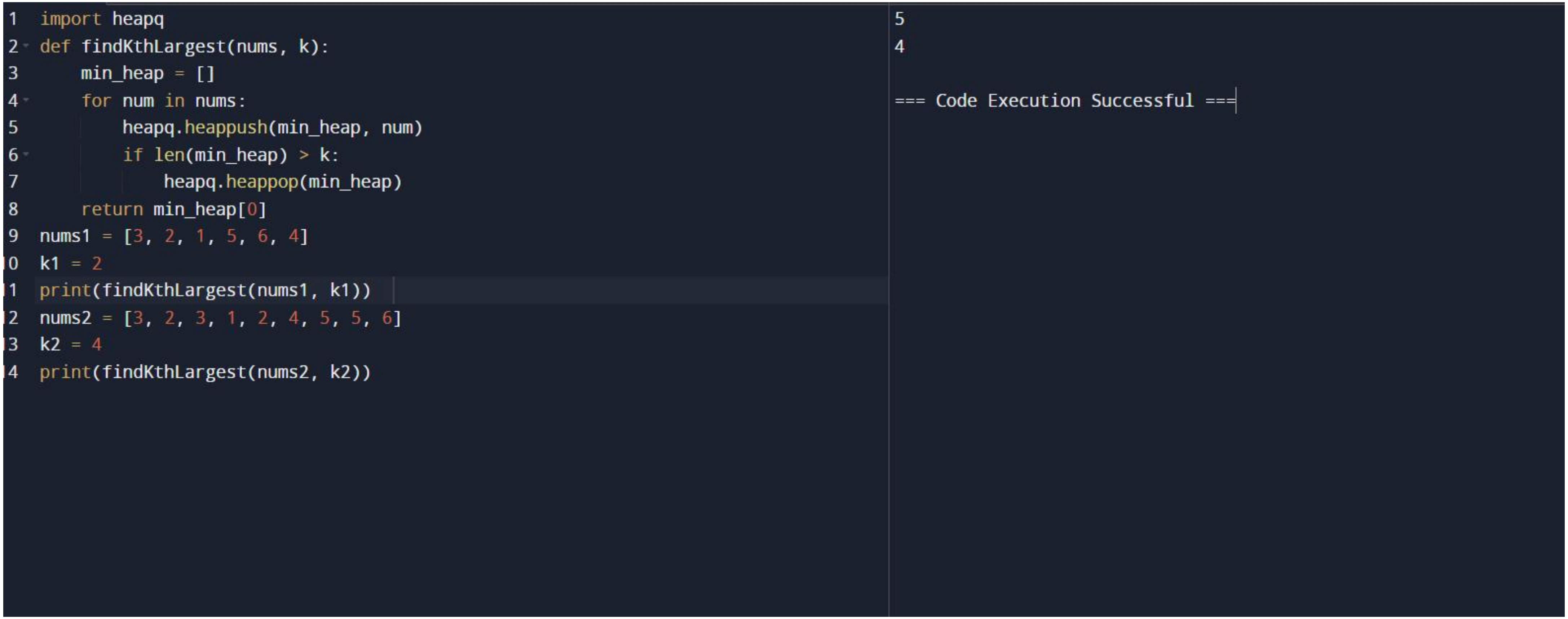
```python
def countGoodTriplets(nums1, nums2):
    n = len(nums1)
    pos1 = [0] * n
    pos2 = [0] * n
    for i in range(n):
        pos1[nums1[i]] = i
        pos2[nums2[i]] = i
    good_triplets = 0
    for y in range(1, n-1):
        count_x = 0
        for x in range(y):
            if pos1[nums1[x]] < pos1[nums1[y]]:
                count_x += 1
        count_z = 0
        for z in range(y+1, n):
            if pos1[nums1[y]] < pos1[nums1[z]]:
                count_z += 1
        good_triplets += count_x * count_z
    return good_triplets
nums1 = [2, 0, 1, 3]
nums2 = [0, 1, 2, 3]
print(countGoodTriplets(nums1, nums2))
```

```
4

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



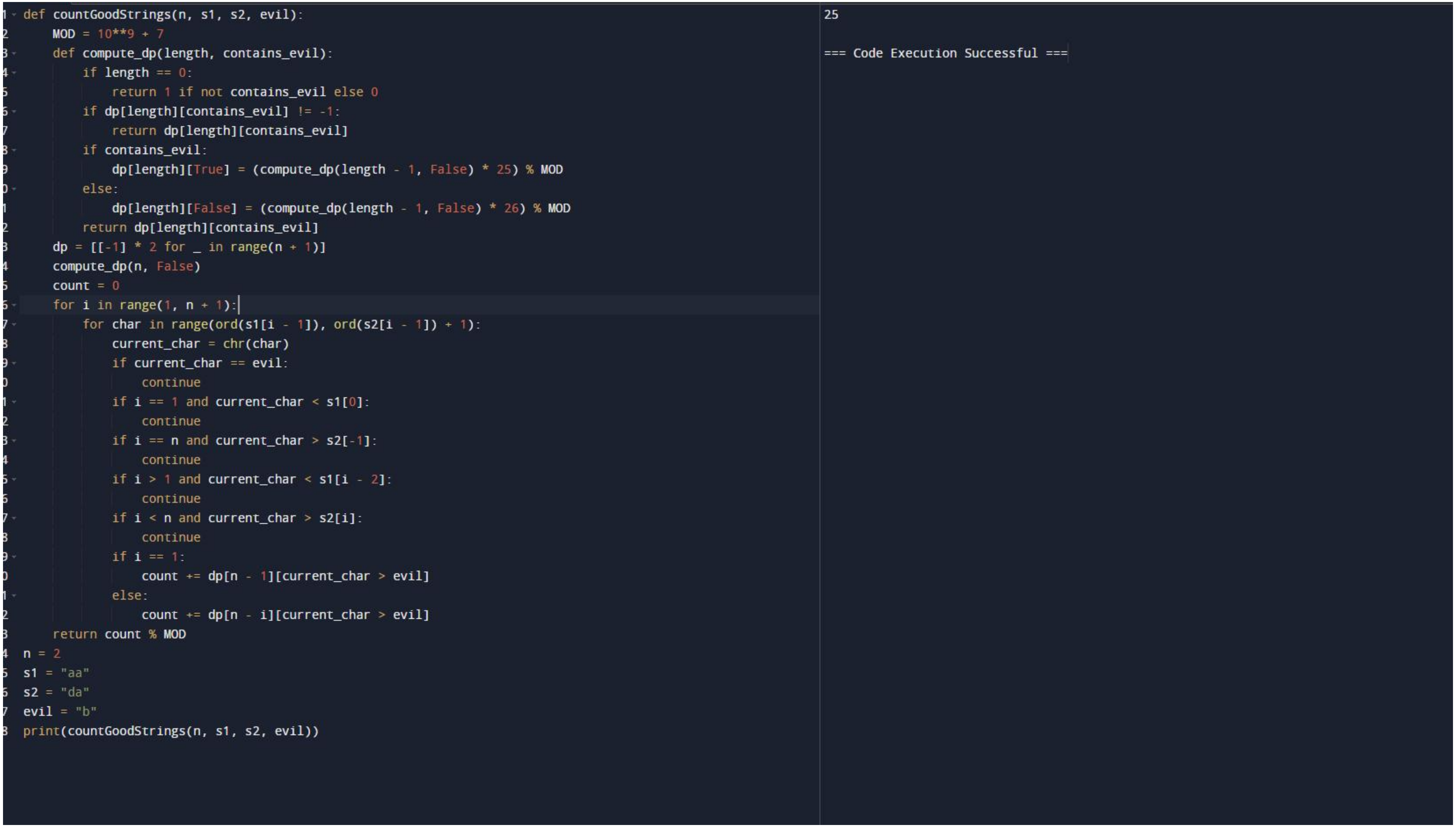
```python
def intersection(nums1, nums2):
    set1 = set(nums1)
    set2 = set(nums2)
    return list(set1.intersection(set2))
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
print(intersection(nums1, nums2))
```

```
[2]

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



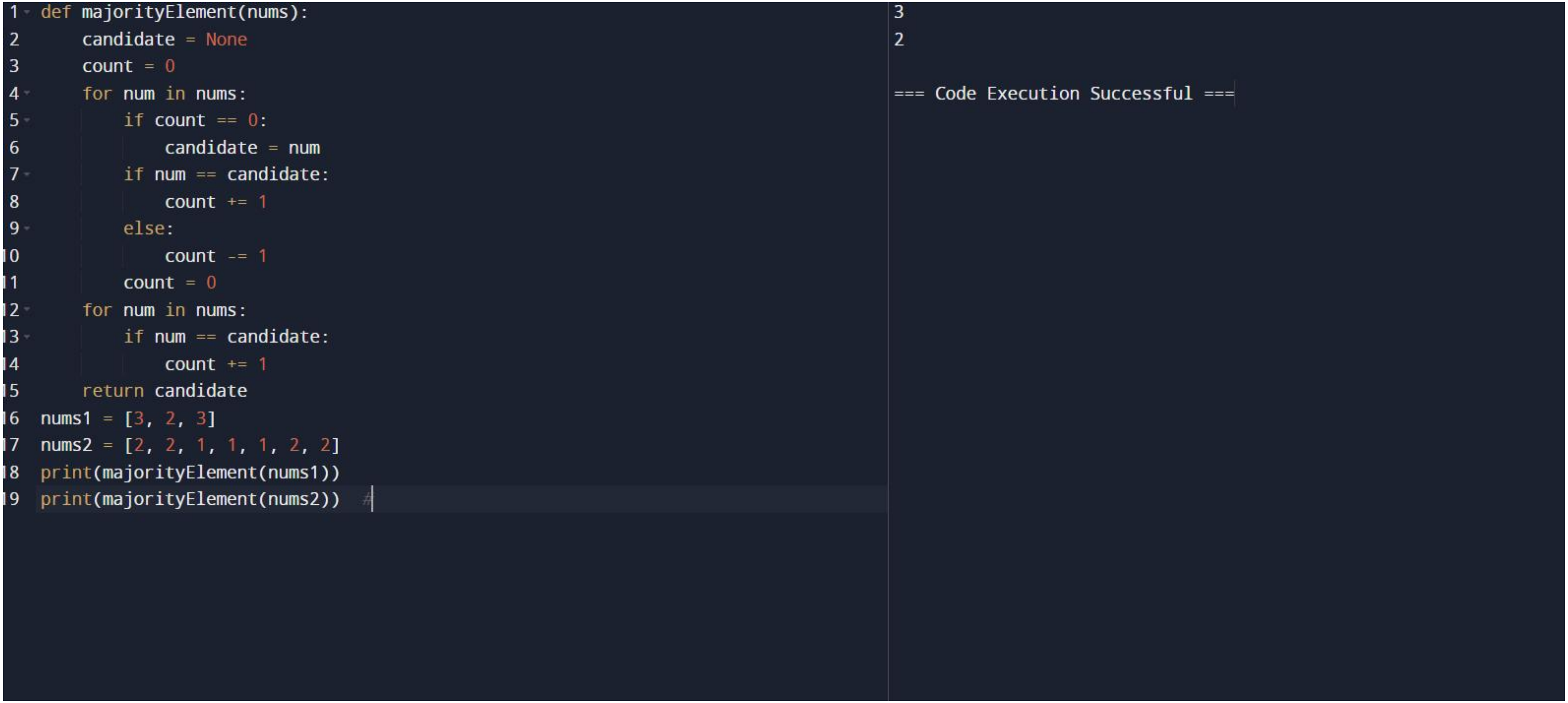
```python
import heapq
def findKthLargest(nums, k):
    min_heap = []
    for num in nums:
        heapq.heappush(min_heap, num)
        if len(min_heap) > k:
            heapq.heappop(min_heap)
    return min_heap[0]
nums1 = [3, 2, 1, 5, 6, 4]
k1 = 2
print(findKthLargest(nums1, k1))
nums2 = [3, 2, 3, 1, 2, 4, 5, 5, 6]
k2 = 4
print(findKthLargest(nums2, k2))
```

```
5
4

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



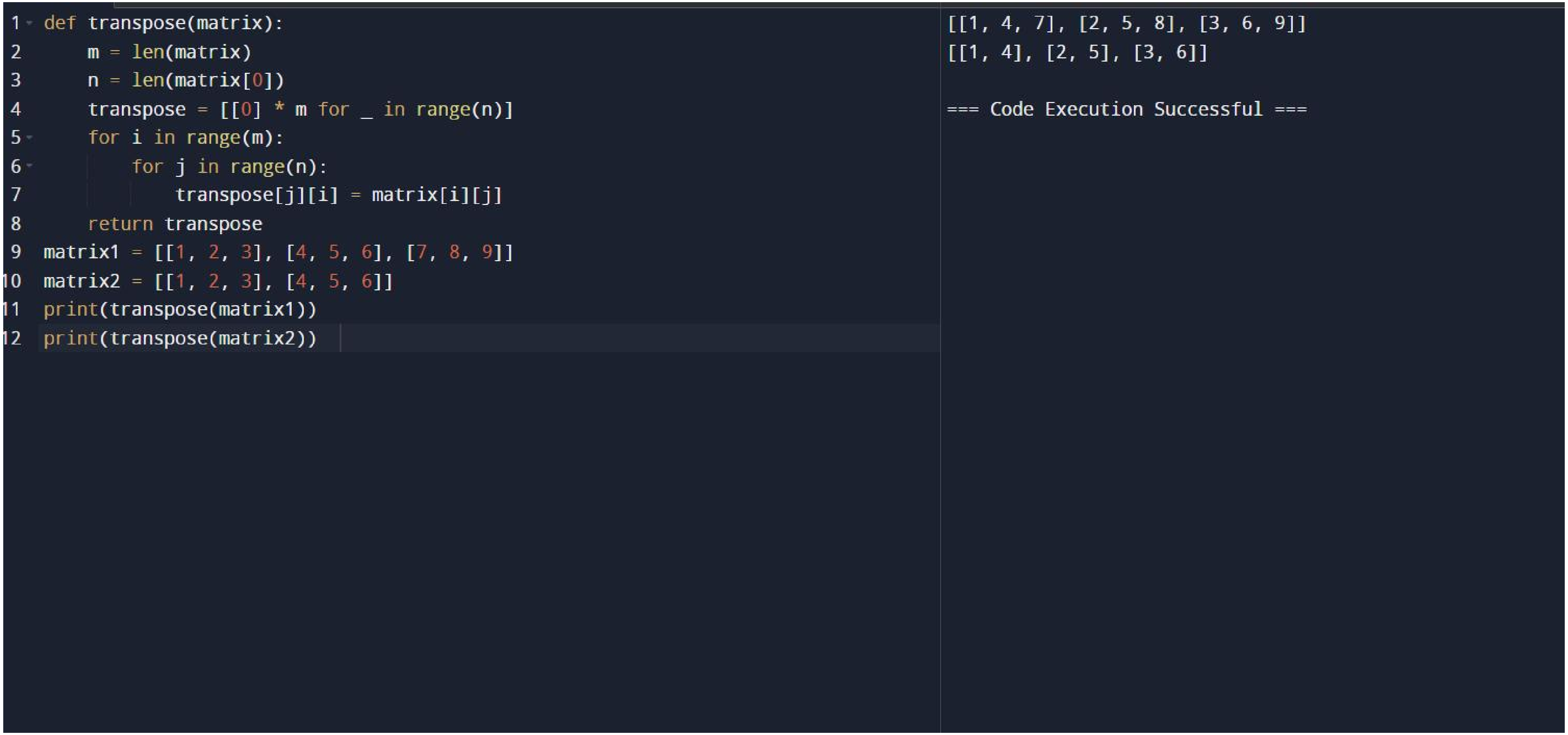
```python
def countGoodStrings(n, s1, s2, evil):
    MOD = 10**9 + 7
    def compute_dp(length, contains_evil):
        if length == 0:
            return 1 if not contains_evil else 0
        if dp[length][contains_evil] != -1:
            return dp[length][contains_evil]
        if contains_evil:
            dp[length][True] = (compute_dp(length - 1, False) * 25) % MOD
        else:
            dp[length][False] = (compute_dp(length - 1, False) * 26) % MOD
        return dp[length][contains_evil]
    dp = [[-1] * 2 for _ in range(n + 1)]
    compute_dp(n, False)
    count = 0
    for i in range(1, n + 1):
        for char in range(ord(s1[i - 1]), ord(s2[i - 1]) + 1):
            current_char = chr(char)
            if current_char == evil:
                continue
            if i == 1 and current_char < s1[0]:
                continue
            if i == n and current_char > s2[-1]:
                continue
            if i > 1 and current_char < s1[i - 2]:
                continue
            if i < n and current_char > s2[i]:
                continue
            if i == 1:
                count += dp[n - 1][current_char > evil]
            else:
                count += dp[n - i][current_char > evil]
    return count % MOD
n = 2
s1 = "aa"
s2 = "da"
evil = "b"
print(countGoodStrings(n, s1, s2, evil))
```

```
25

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



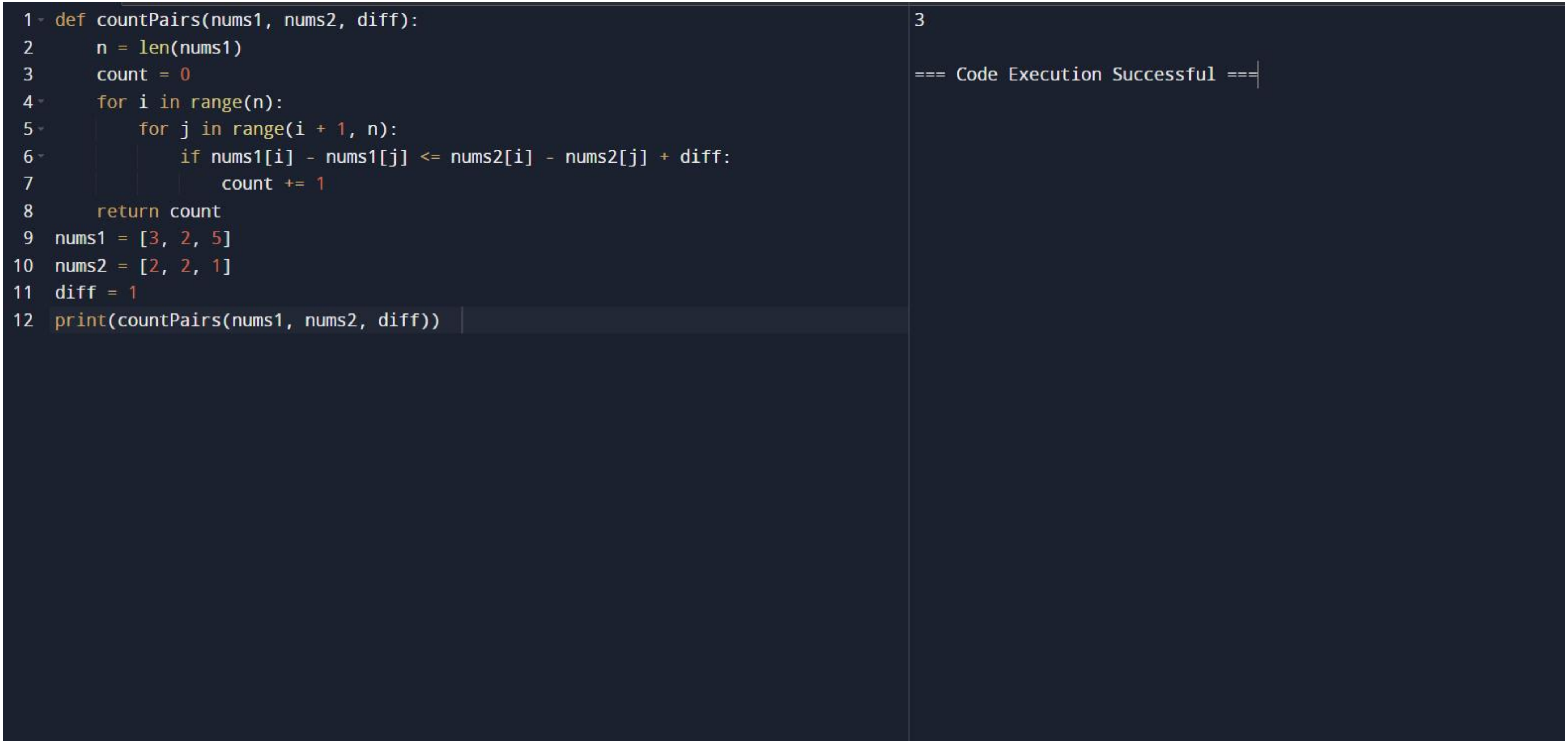
```python
def majorityElement(nums):
    candidate = None
    count = 0
    for num in nums:
        if count == 0:
            candidate = num
        if num == candidate:
            count += 1
        else:
            count -= 1
        count = 0
    for num in nums:
        if num == candidate:
            count += 1
    return candidate
nums1 = [3, 2, 3]
nums2 = [2, 2, 1, 1, 1, 2, 2]
print(majorityElement(nums1))
print(majorityElement(nums2))  #
```

```
3
2

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



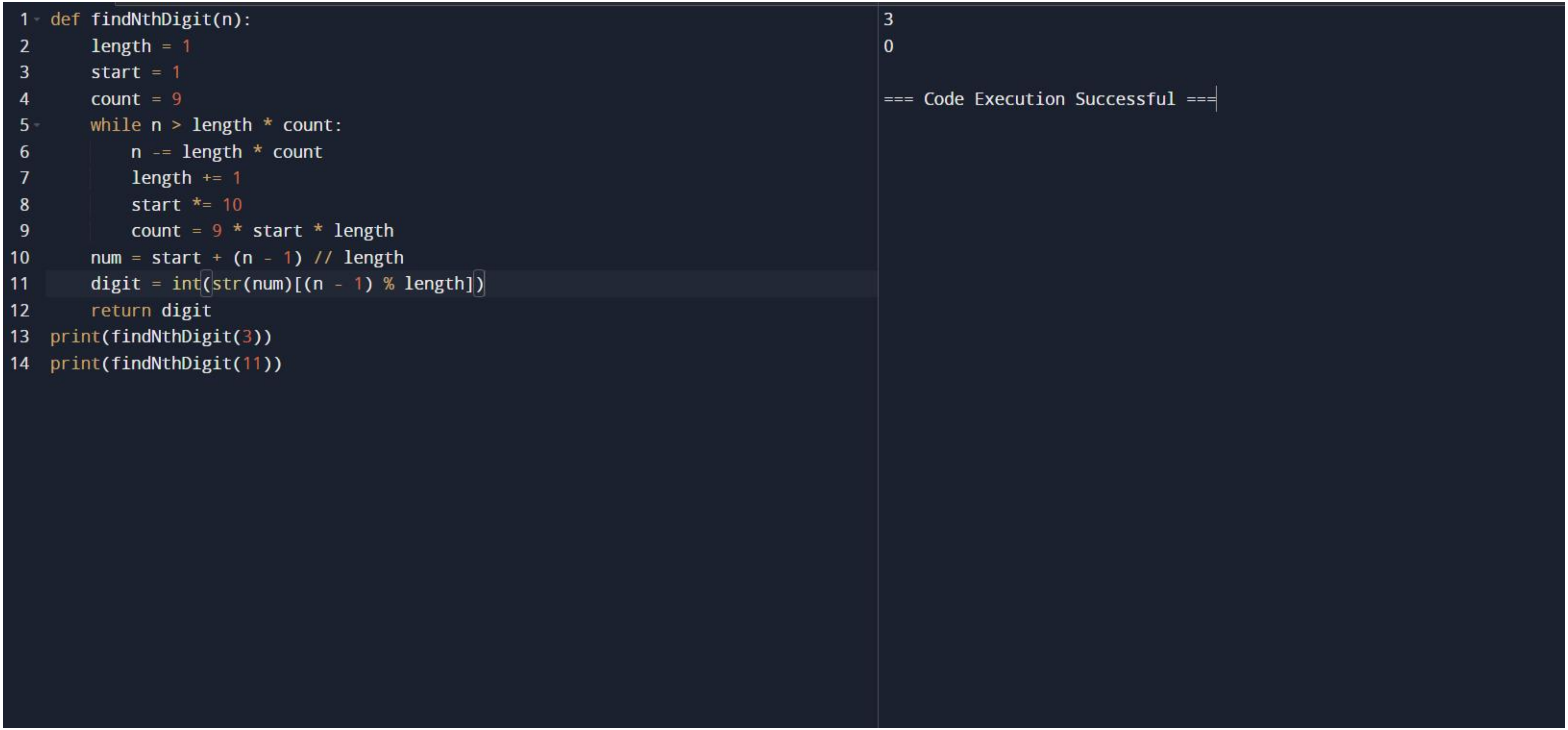
```python
def transpose(matrix):
    m = len(matrix)
    n = len(matrix[0])
    transpose = [[0] * m for _ in range(n)]
    for i in range(m):
        for j in range(n):
            transpose[j][i] = matrix[i][j]
    return transpose
matrix1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix2 = [[1, 2, 3], [4, 5, 6]]
print(transpose(matrix1))
print(transpose(matrix2))
```

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
[[1, 4], [2, 5], [3, 6]]

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



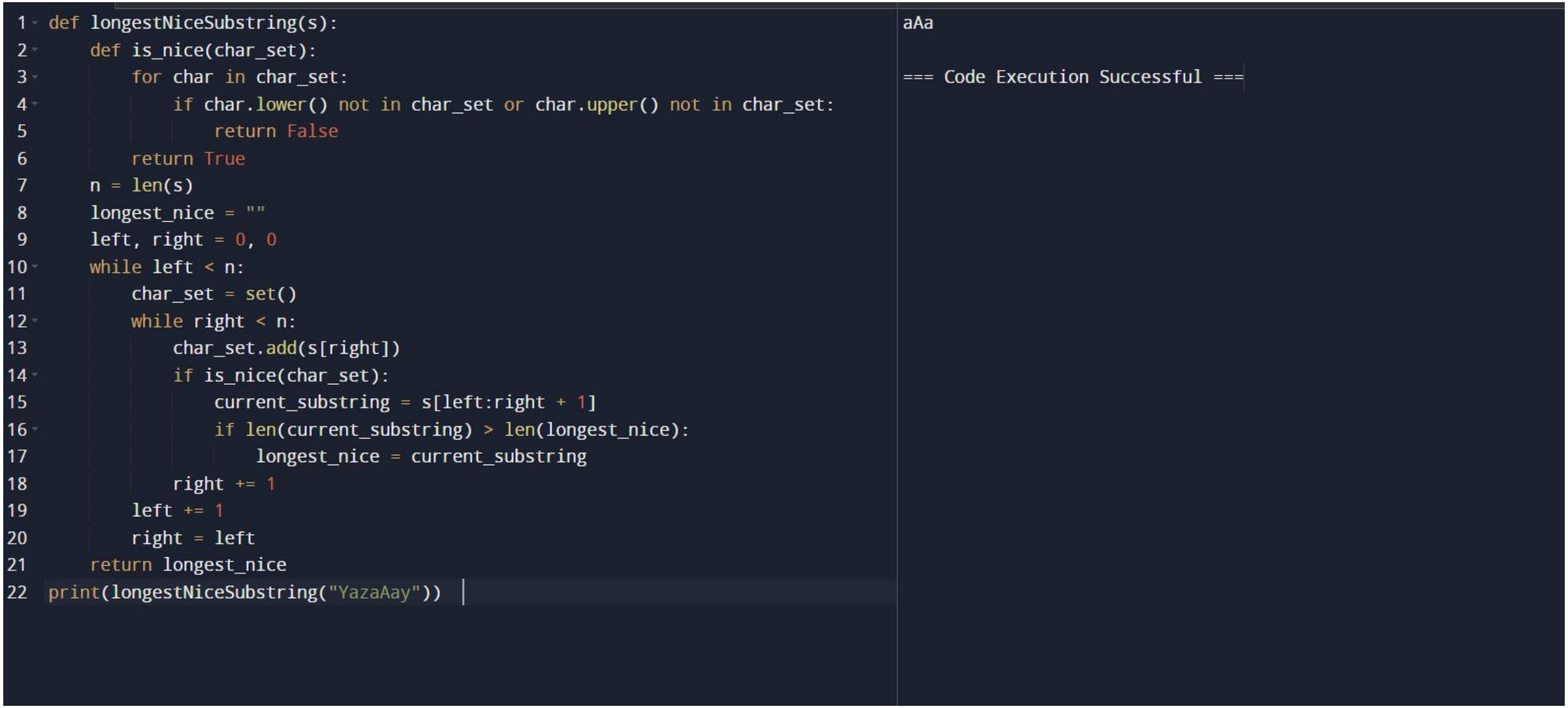
```
def countPairs(nums1, nums2, diff):
    n = len(nums1)
    count = 0
    for i in range(n):
        for j in range(i + 1, n):
            if nums1[i] - nums1[j] <= nums2[i] - nums2[j] + diff:
                count += 1
    return count
nums1 = [3, 2, 5]
nums2 = [2, 2, 1]
diff = 1
print(countPairs(nums1, nums2, diff))
```

```
3

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



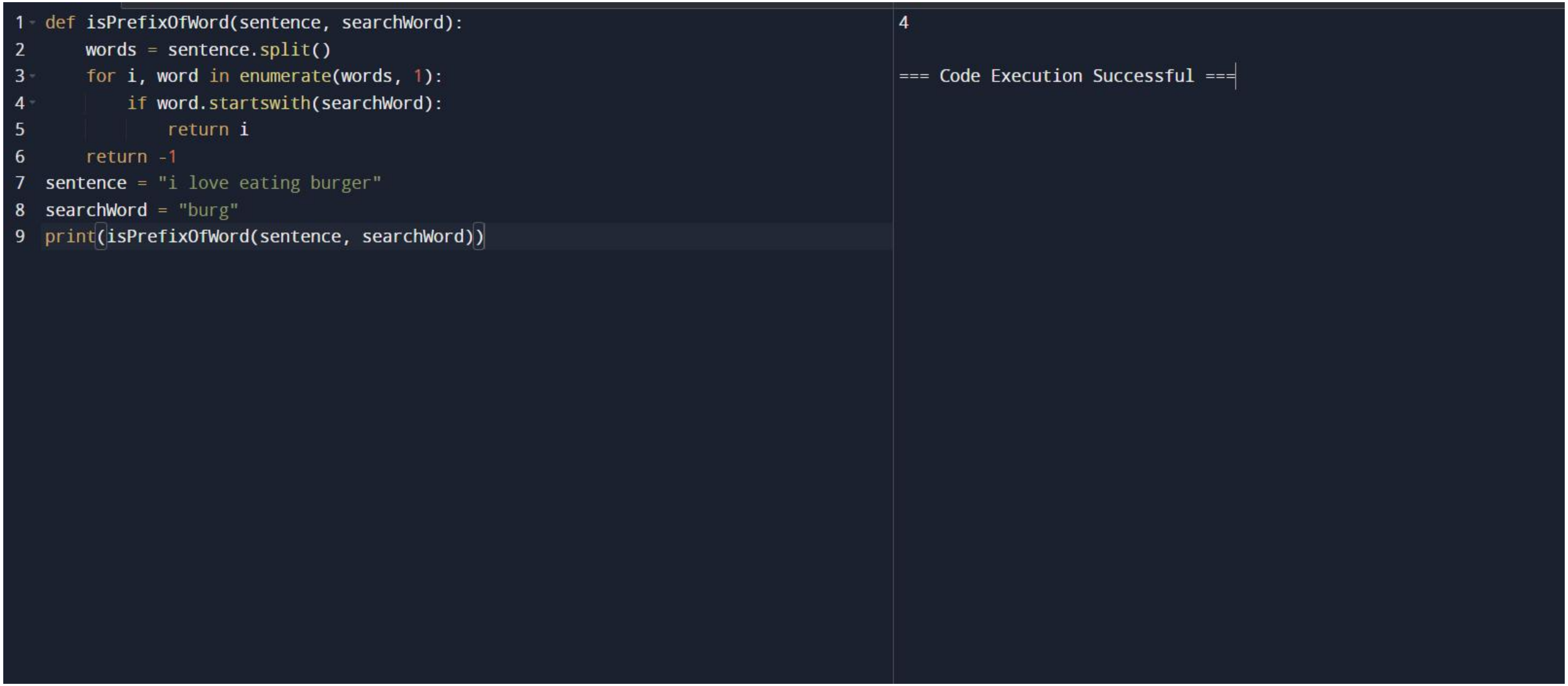
```python
def findNthDigit(n):
    length = 1
    start = 1
    count = 9
    while n > length * count:
        n -= length * count
        length += 1
        start *= 10
        count = 9 * start * length
    num = start + (n - 1) // length
    digit = int(str(num)[(n - 1) % length])
    return digit
print(findNthDigit(3))
print(findNthDigit(11))
```

```
3
0

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



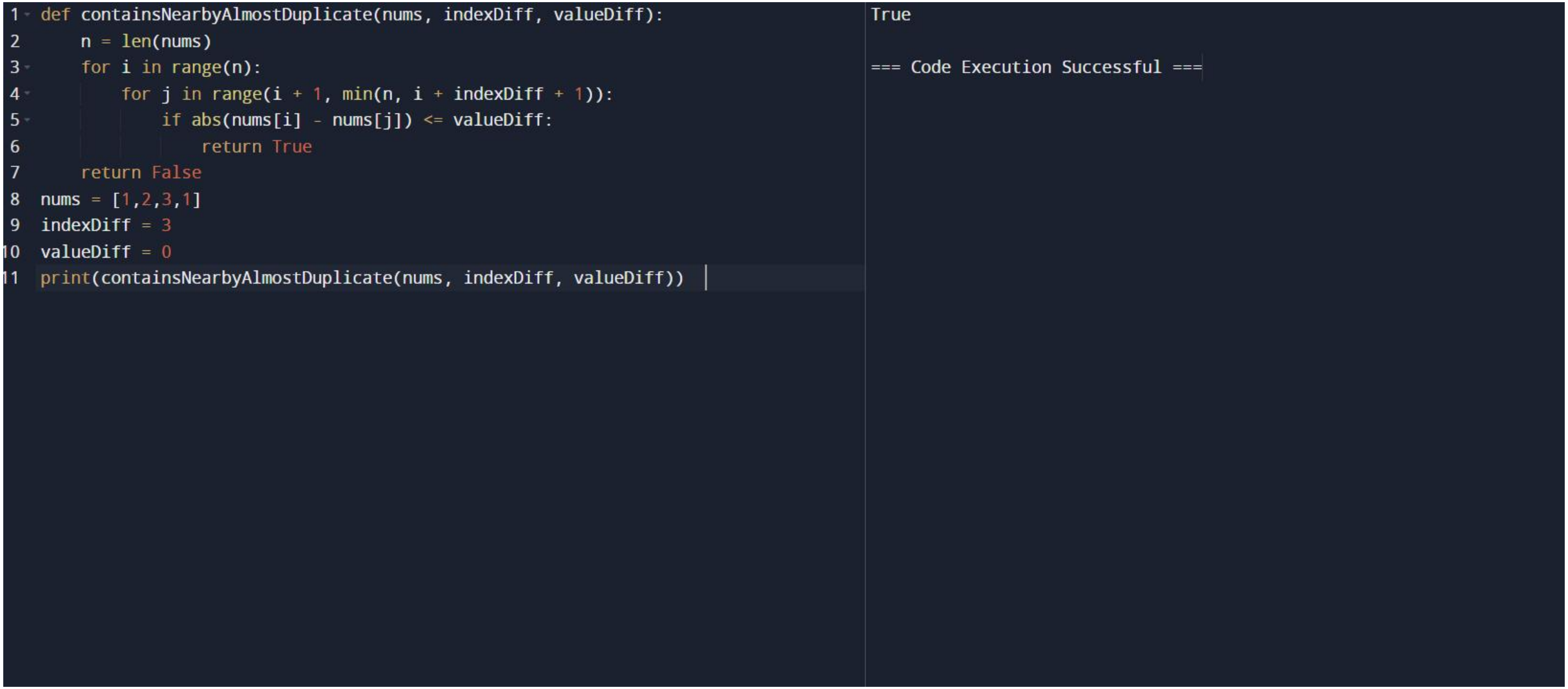
```python
def longestNiceSubstring(s):
    def is_nice(char_set):
        for char in char_set:
            if char.lower() not in char_set or char.upper() not in char_set:
                return False
        return True
    n = len(s)
    longest_nice = ""
    left, right = 0, 0
    while left < n:
        char_set = set()
        while right < n:
            char_set.add(s[right])
            if is_nice(char_set):
                current_substring = s[left:right + 1]
                if len(current_substring) > len(longest_nice):
                    longest_nice = current_substring
            right += 1
        left += 1
        right = left
    return longest_nice
print(longestNiceSubstring("YazaAay"))
```

```
aAa

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



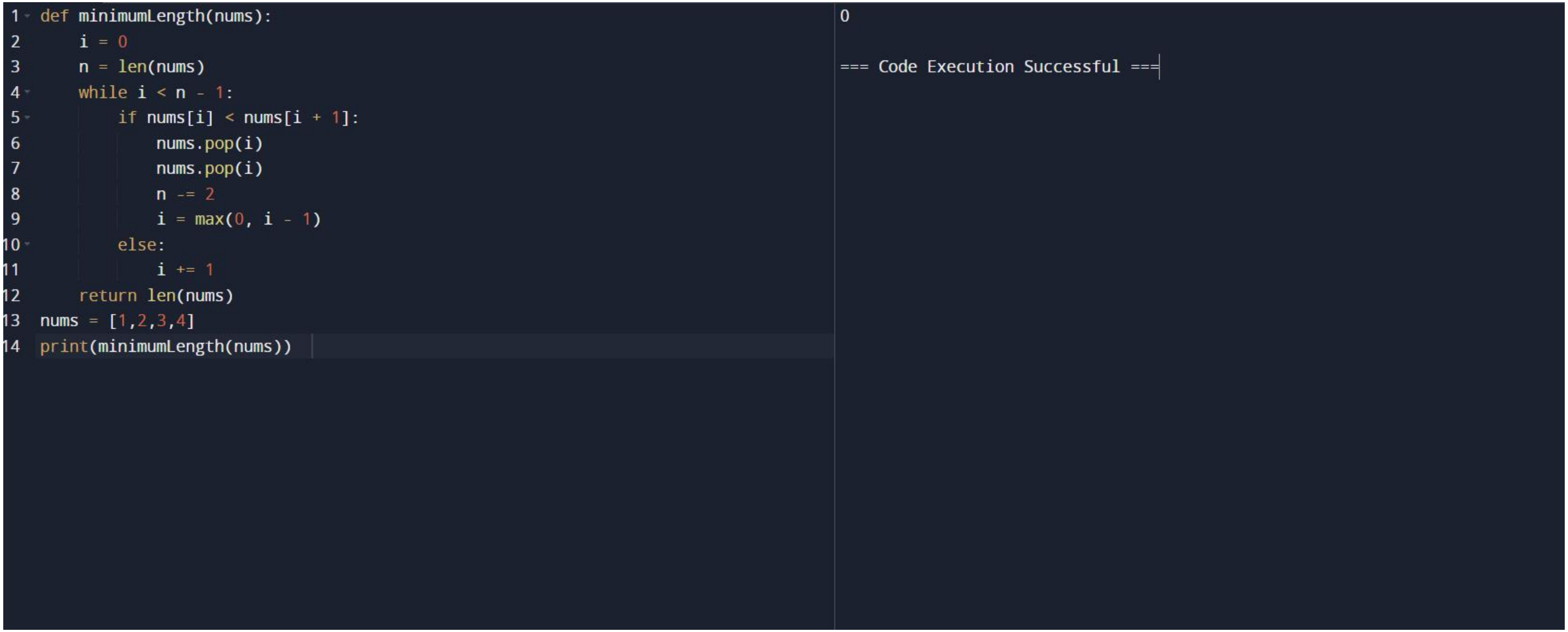
```python
def isPrefixOfWord(sentence, searchWord):
    words = sentence.split()
    for i, word in enumerate(words, 1):
        if word.startswith(searchWord):
            return i
    return -1
sentence = "i love eating burger"
searchWord = "burg"
print(isPrefixOfWord(sentence, searchWord))
```

```
4

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



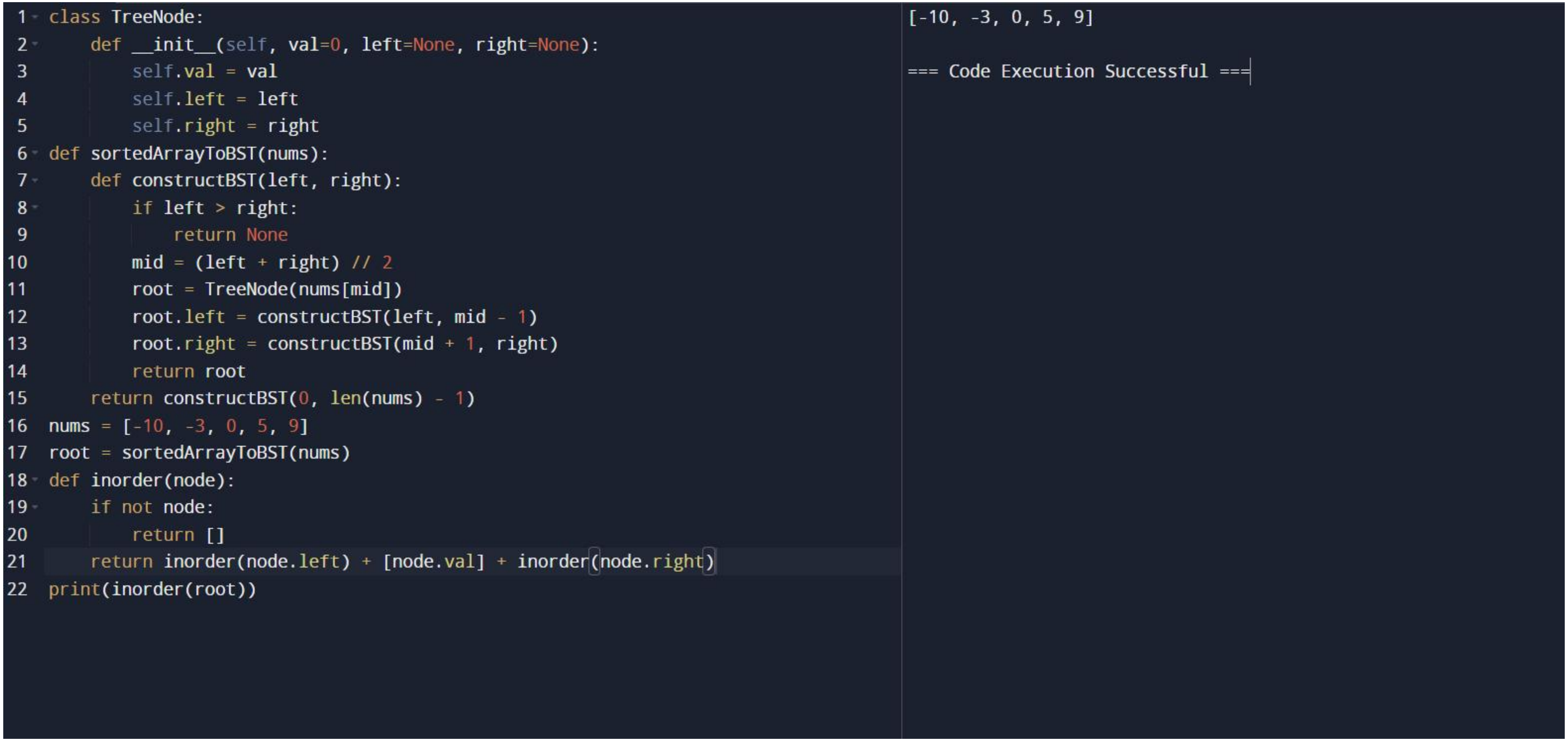
```python
def containsNearbyAlmostDuplicate(nums, indexDiff, valueDiff):
    n = len(nums)
    for i in range(n):
        for j in range(i + 1, min(n, i + indexDiff + 1)):
            if abs(nums[i] - nums[j]) <= valueDiff:
                return True
    return False
nums = [1,2,3,1]
indexDiff = 3
valueDiff = 0
print(containsNearbyAlmostDuplicate(nums, indexDiff, valueDiff))
```

```
True

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



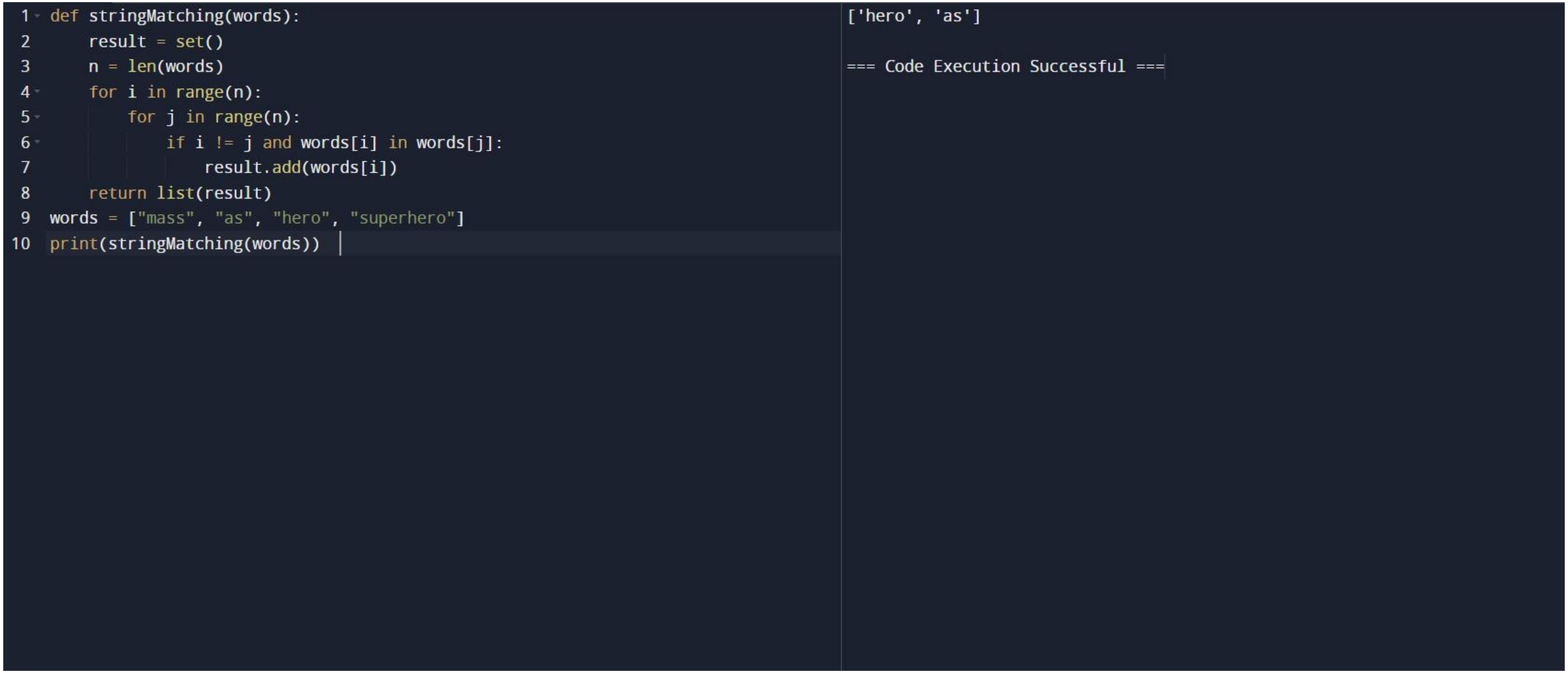
```python
def minimumLength(nums):
    i = 0
    n = len(nums)
    while i < n - 1:
        if nums[i] < nums[i + 1]:
            nums.pop(i)
            nums.pop(i)
            n -= 2
            i = max(0, i - 1)
        else:
            i += 1
    return len(nums)
nums = [1,2,3,4]
print(minimumLength(nums))
```

```
0

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



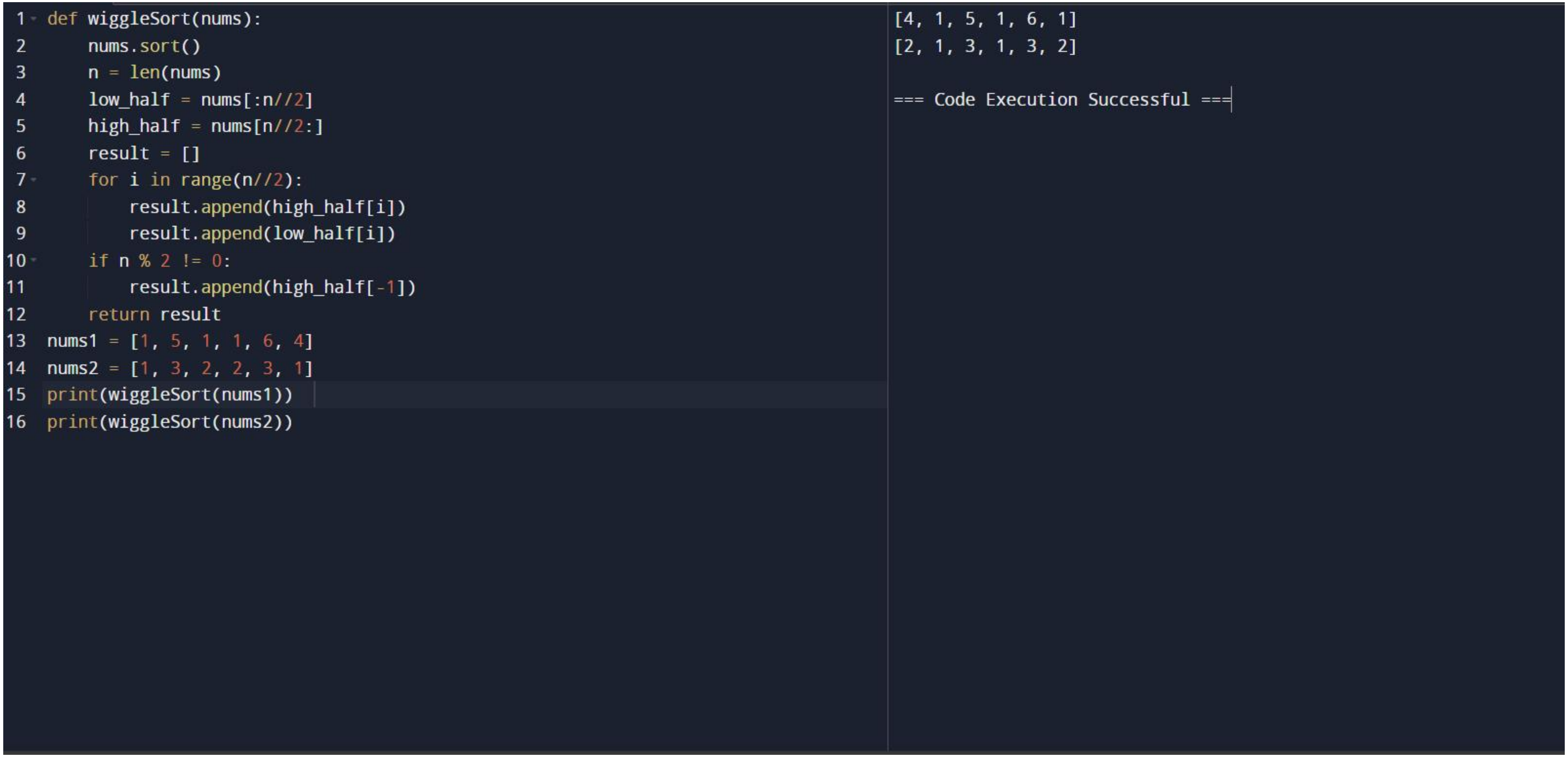
```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def sortedArrayToBST(nums):
    def constructBST(left, right):
        if left > right:
            return None
        mid = (left + right) // 2
        root = TreeNode(nums[mid])
        root.left = constructBST(left, mid - 1)
        root.right = constructBST(mid + 1, right)
        return root
    return constructBST(0, len(nums) - 1)
nums = [-10, -3, 0, 5, 9]
root = sortedArrayToBST(nums)
def inorder(node):
    if not node:
        return []
    return inorder(node.left) + [node.val] + inorder(node.right)
print(inorder(root))
```

```
[-10, -3, 0, 5, 9]

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



```python
def stringMatching(words):
    result = set()
    n = len(words)
    for i in range(n):
        for j in range(n):
            if i != j and words[i] in words[j]:
                result.add(words[i])
    return list(result)
words = ["mass", "as", "hero", "superhero"]
print(stringMatching(words))
```

```
['hero', 'as']

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



```python
def wiggleSort(nums):
    nums.sort()
    n = len(nums)
    low_half = nums[:n//2]
    high_half = nums[n//2:]
    result = []
    for i in range(n//2):
        result.append(high_half[i])
        result.append(low_half[i])
    if n % 2 != 0:
        result.append(high_half[-1])
    return result
nums1 = [1, 5, 1, 1, 6, 4]
nums2 = [1, 3, 2, 2, 3, 1]
print(wiggleSort(nums1))
print(wiggleSort(nums2))
```

```
[4, 1, 5, 1, 6, 1]
[2, 1, 3, 1, 3, 2]

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