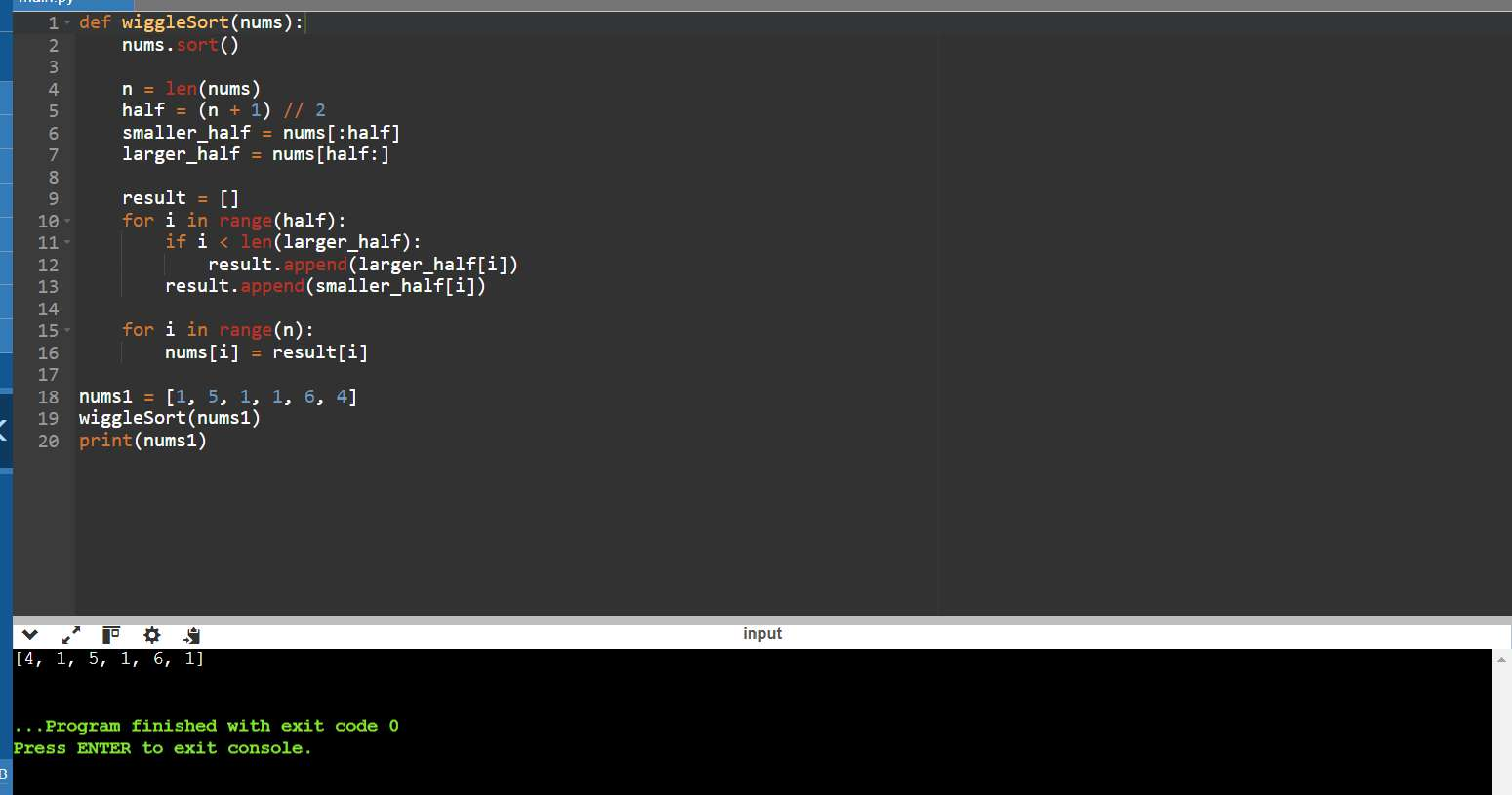
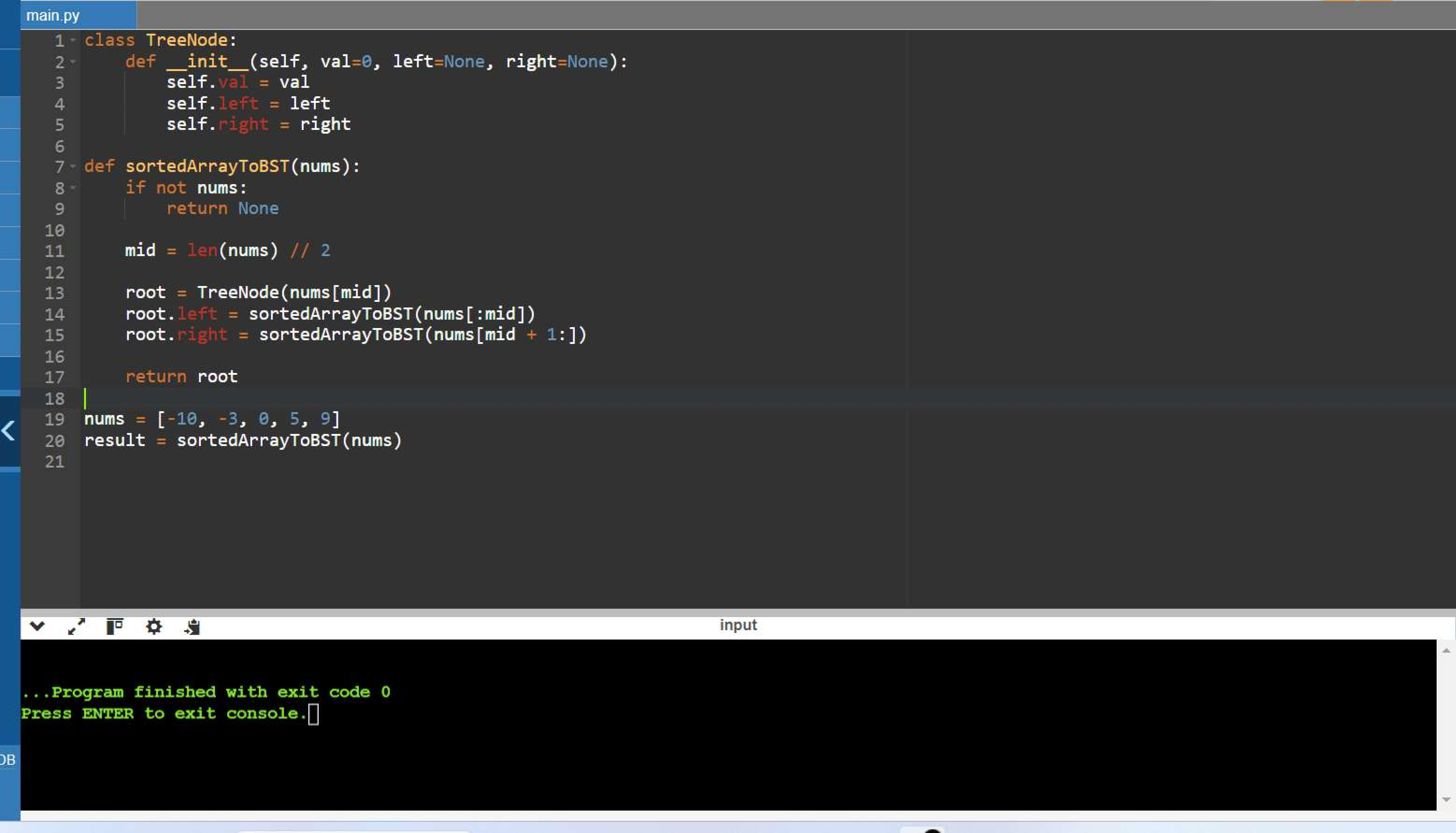


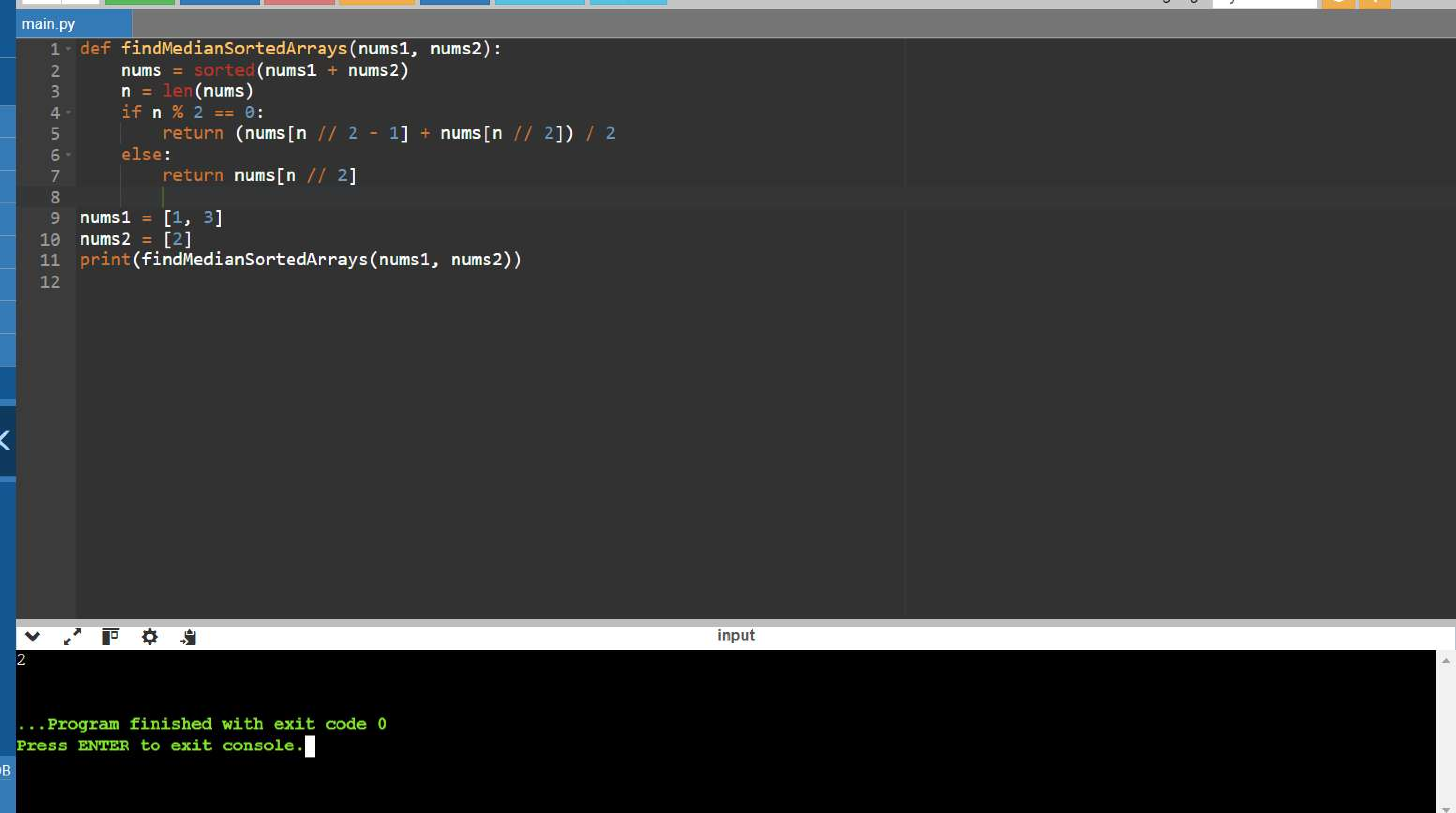
```python
def wiggleSort(nums):
    nums.sort()

    n = len(nums)
    half = (n + 1) // 2
    smaller_half = nums[:half]
    larger_half = nums[half:]

    result = []
    for i in range(half):
        if i < len(larger_half):
            result.append(larger_half[i])
        result.append(smaller_half[i])

    for i in range(n):
        nums[i] = result[i]

nums1 = [1, 5, 1, 1, 6, 4]
wiggleSort(nums1)
print(nums1)
```

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


...Program finished with exit code 0
Press ENTER to exit console.
```



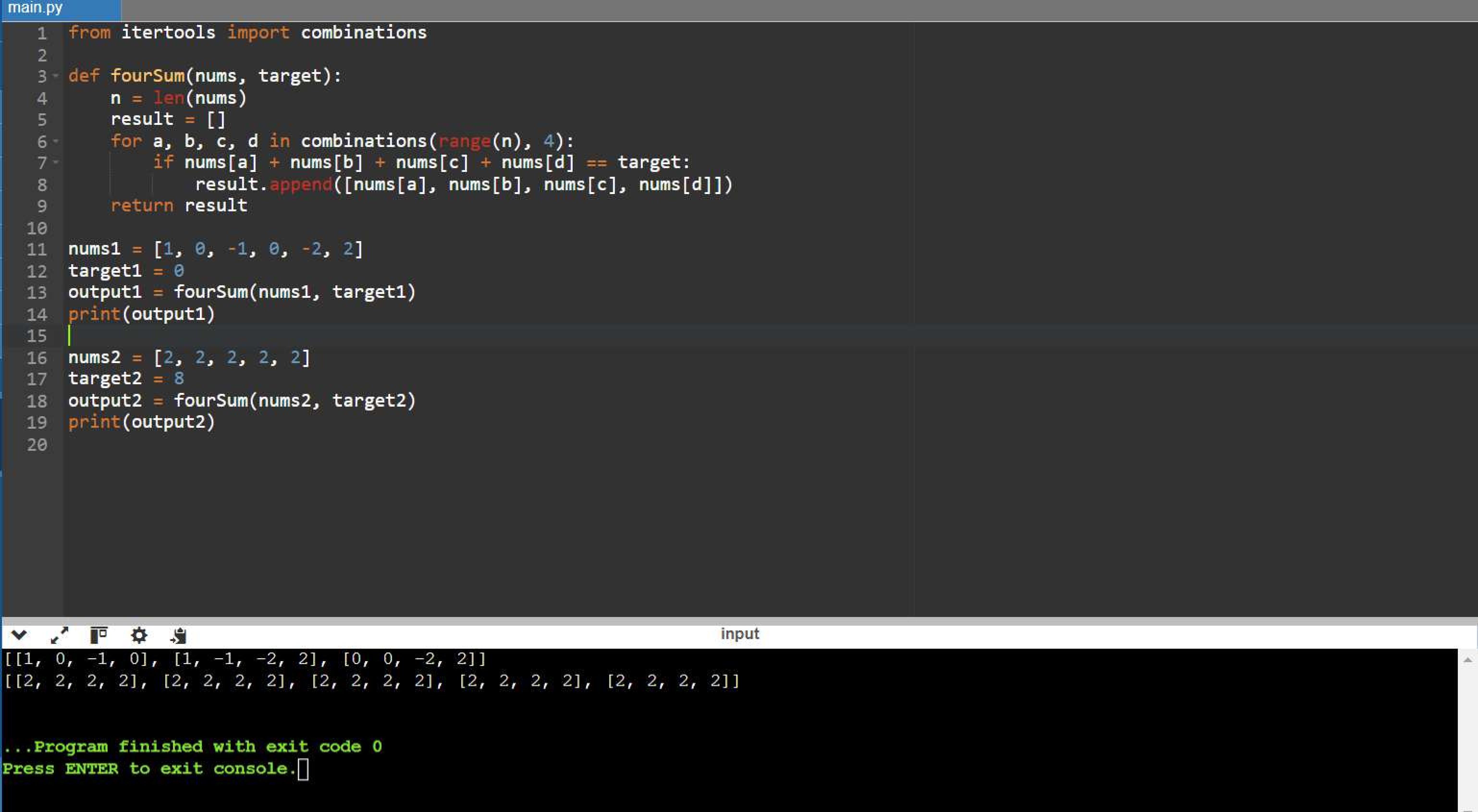
```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def sortedArrayToBST(nums):
    if not nums:
        return None

    mid = len(nums) // 2

    root = TreeNode(nums[mid])
    root.left = sortedArrayToBST(nums[:mid])
    root.right = sortedArrayToBST(nums[mid + 1:])

    return root

nums = [-10, -3, 0, 5, 9]
result = sortedArrayToBST(nums)
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```


main.py

```python
def findMedianSortedArrays(nums1, nums2):
    nums = sorted(nums1 + nums2)
    n = len(nums)
    if n % 2 == 0:
        return (nums[n // 2 - 1] + nums[n // 2]) / 2
    else:
        return nums[n // 2]

nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))
```

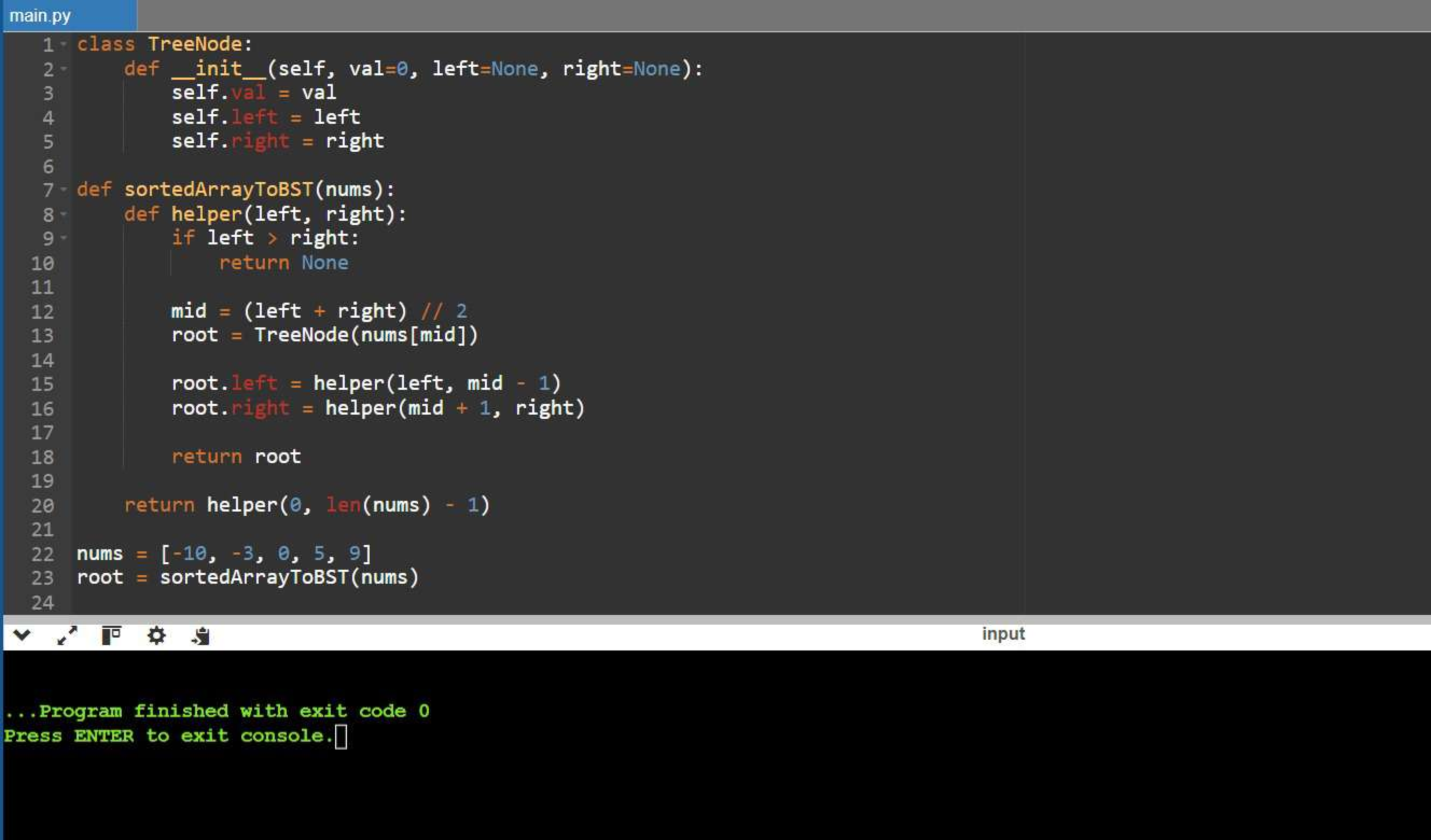
```
2


...Program finished with exit code 0
Press ENTER to exit console.
```



main.py

```python
from itertools import combinations

def fourSum(nums, target):
    n = len(nums)
    result = []
    for a, b, c, d in combinations(range(n), 4):
        if nums[a] + nums[b] + nums[c] + nums[d] == target:
            result.append([nums[a], nums[b], nums[c], nums[d]])
    return result

nums1 = [1, 0, -1, 0, -2, 2]
target1 = 0
output1 = fourSum(nums1, target1)
print(output1)

nums2 = [2, 2, 2, 2, 2]
target2 = 8
output2 = fourSum(nums2, target2)
print(output2)
```

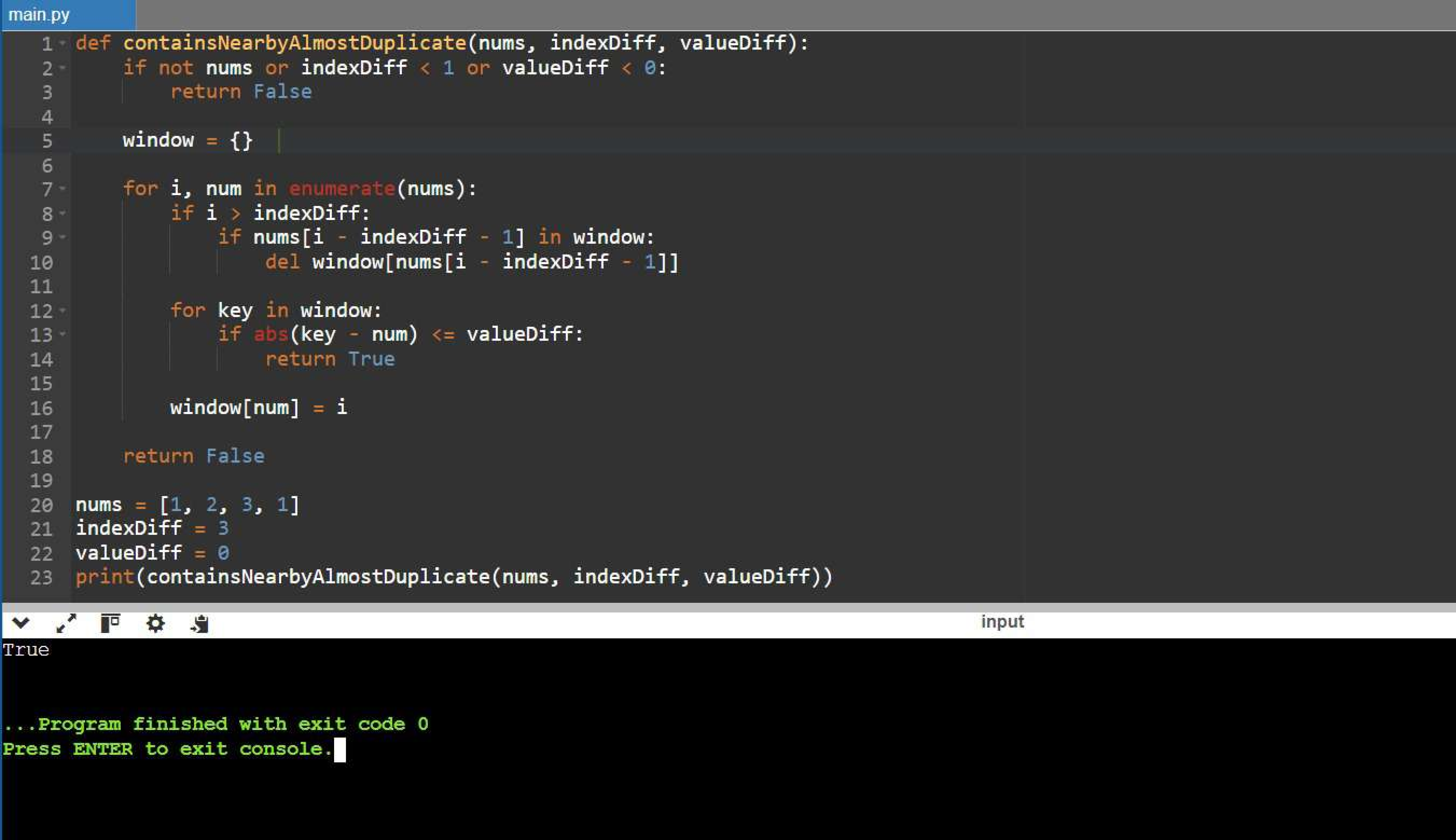
```
[[1, 0, -1, 0], [1, -1, -2, 2], [0, 0, -2, 2]]
[[2, 2, 2, 2], [2, 2, 2, 2], [2, 2, 2, 2], [2, 2, 2, 2], [2, 2, 2, 2]]


...Program finished with exit code 0
Press ENTER to exit console.
```



main.py

```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 helper(left, right):
        if left > right:
            return None

        mid = (left + right) // 2
        root = TreeNode(nums[mid])

        root.left = helper(left, mid - 1)
        root.right = helper(mid + 1, right)

        return root

    return helper(0, len(nums) - 1)

nums = [-10, -3, 0, 5, 9]
root = sortedArrayToBST(nums)

```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

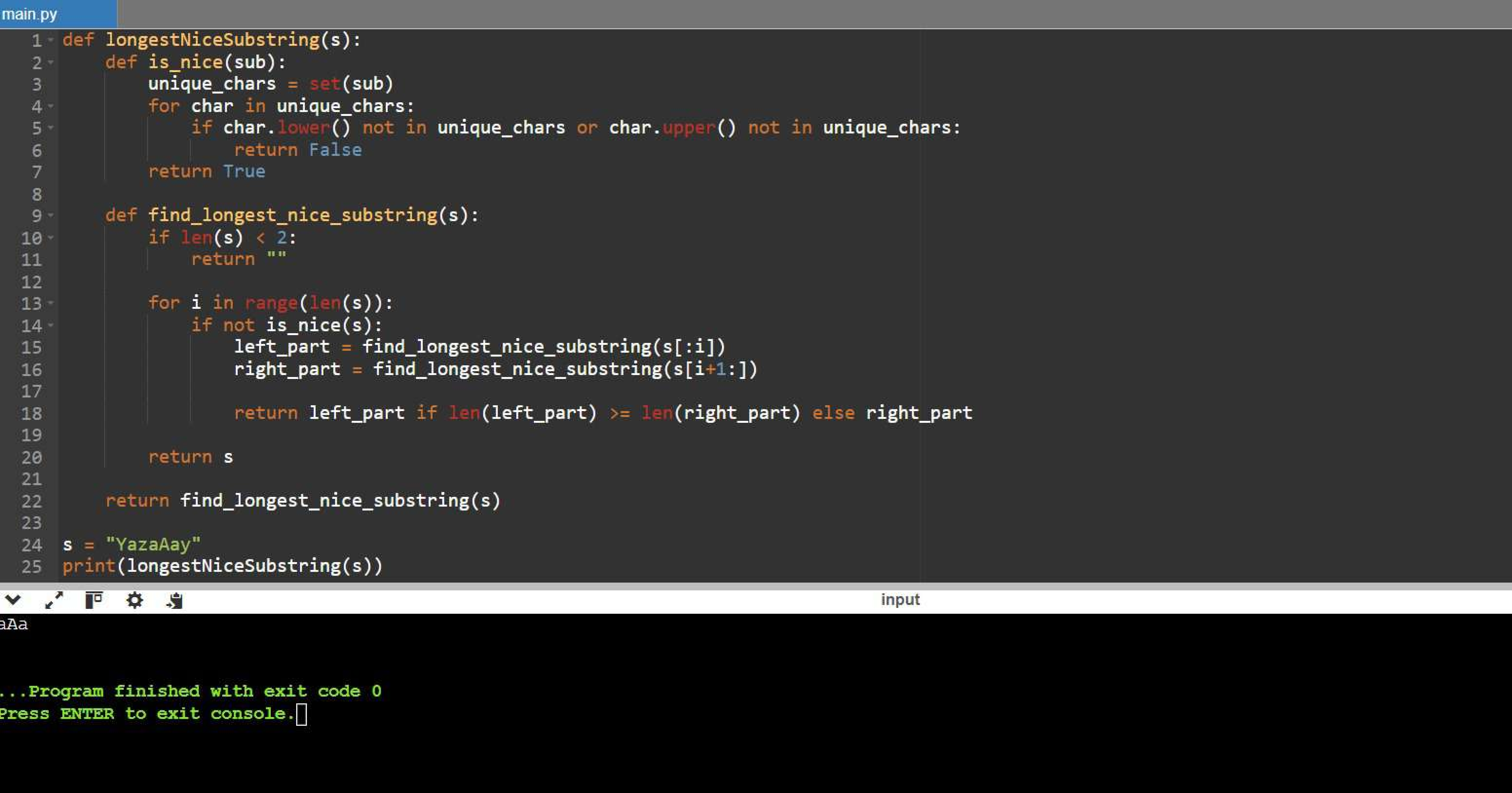


main.py

```python
def containsNearbyAlmostDuplicate(nums, indexDiff, valueDiff):
    if not nums or indexDiff < 1 or valueDiff < 0:
        return False

    window = {}

    for i, num in enumerate(nums):
        if i > indexDiff:
            if nums[i - indexDiff - 1] in window:
                del window[nums[i - indexDiff - 1]]

        for key in window:
            if abs(key - num) <= valueDiff:
                return True

        window[num] = i

    return False

nums = [1, 2, 3, 1]
indexDiff = 3
valueDiff = 0
print(containsNearbyAlmostDuplicate(nums, indexDiff, valueDiff))
```

input

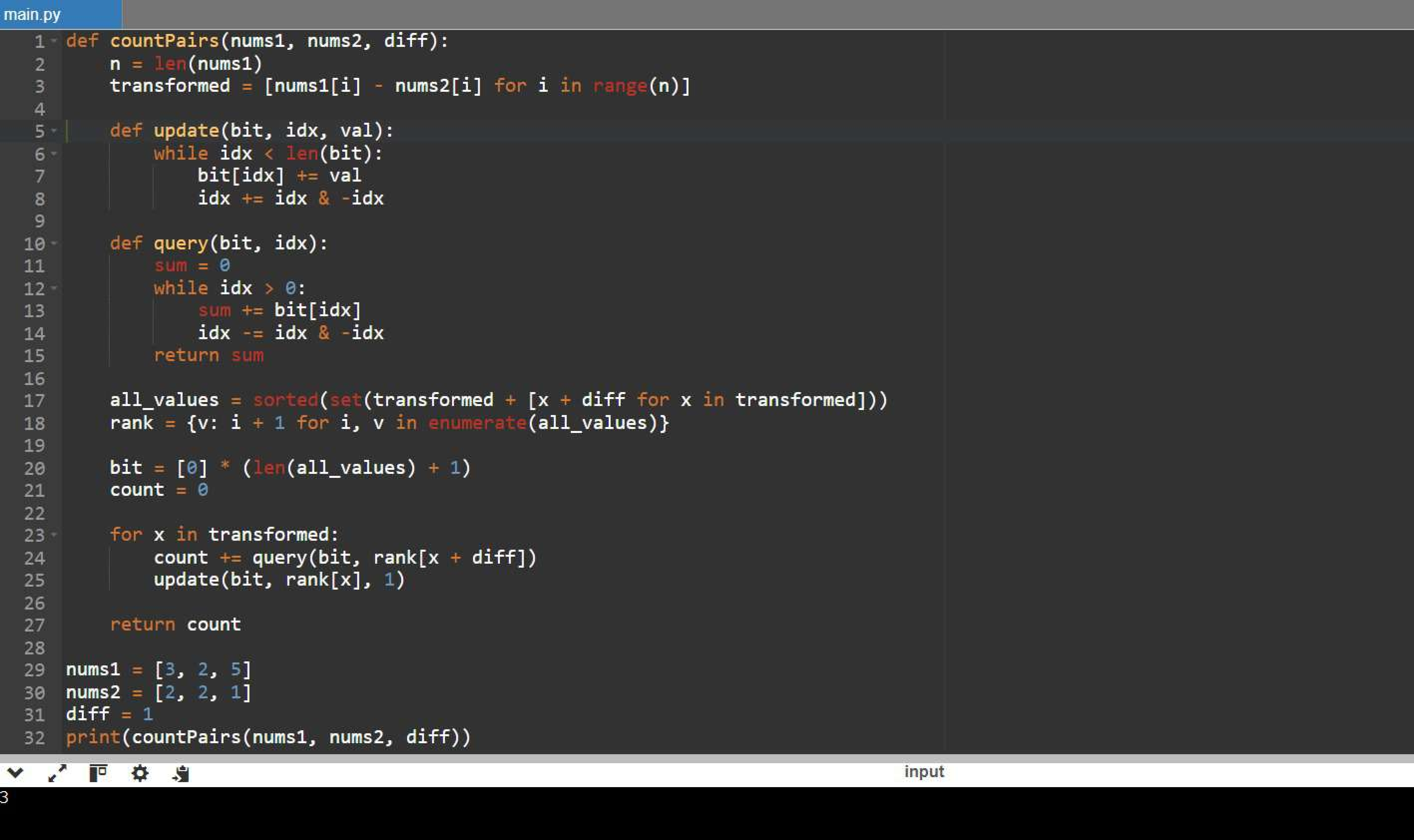
```
True


...Program finished with exit code 0
Press ENTER to exit console.
```



main.py

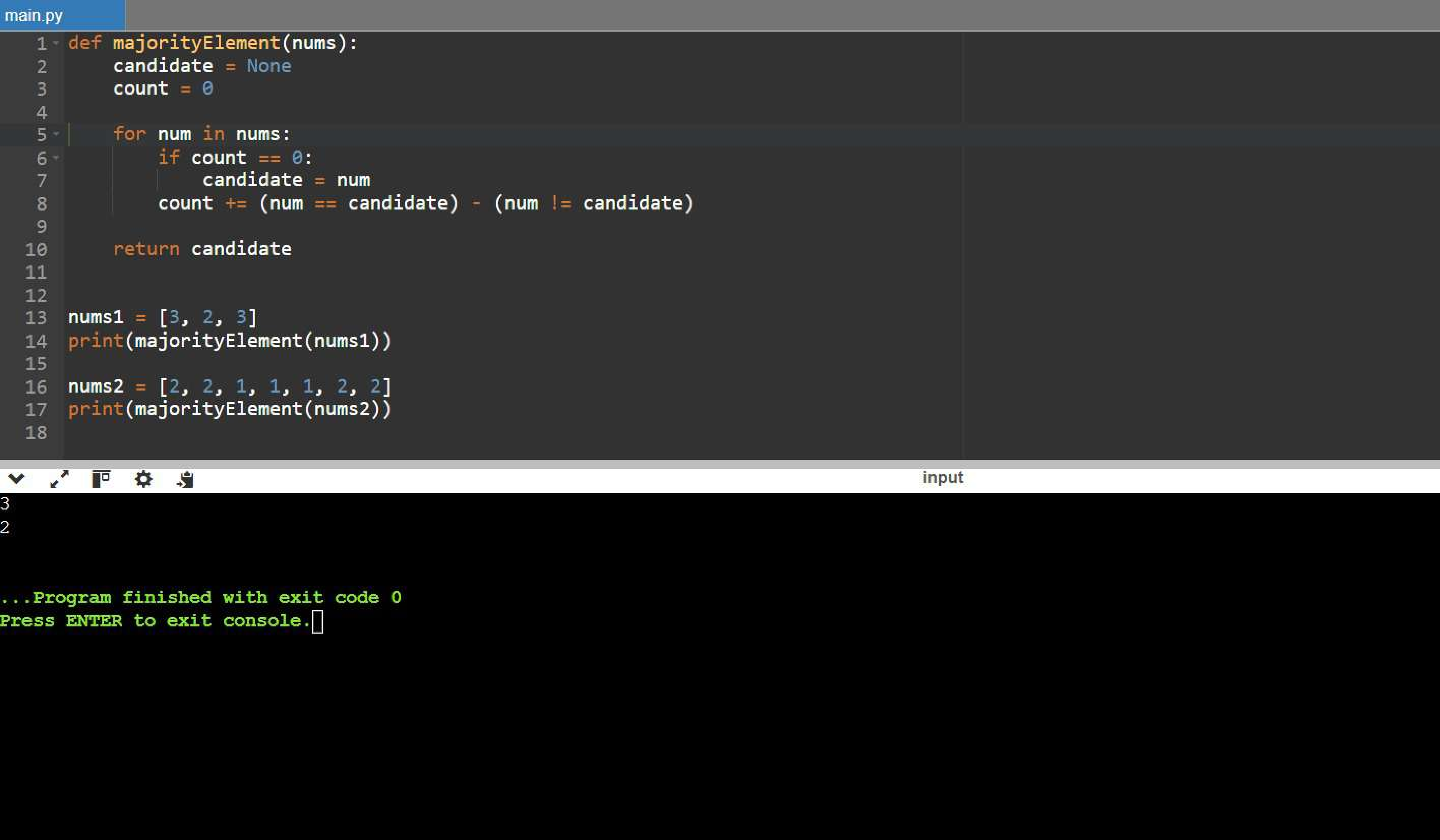
```python
def longestNiceSubstring(s):
    def is_nice(sub):
        unique_chars = set(sub)
        for char in unique_chars:
            if char.lower() not in unique_chars or char.upper() not in unique_chars:
                return False
        return True

    def find_longest_nice_substring(s):
        if len(s) < 2:
            return ""

        for i in range(len(s)):
            if not is_nice(s):
                left_part = find_longest_nice_substring(s[:i])
                right_part = find_longest_nice_substring(s[i+1:])

                return left_part if len(left_part) >= len(right_part) else right_part

        return s

    return find_longest_nice_substring(s)

s = "YazaAay"
print(longestNiceSubstring(s))
```

```
aAa



...Program finished with exit code 0
Press ENTER to exit console.
```


main.py

```python
def countPairs(nums1, nums2, diff):
    n = len(nums1)
    transformed = [nums1[i] - nums2[i] for i in range(n)]

    def update(bit, idx, val):
        while idx < len(bit):
            bit[idx] += val
            idx += idx & -idx

    def query(bit, idx):
        sum = 0
        while idx > 0:
            sum += bit[idx]
            idx -= idx & -idx
        return sum

    all_values = sorted(set(transformed + [x + diff for x in transformed]))
    rank = {v: i + 1 for i, v in enumerate(all_values)}

    bit = [0] * (len(all_values) + 1)
    count = 0

    for x in transformed:
        count += query(bit, rank[x + diff])
        update(bit, rank[x], 1)

    return count

nums1 = [3, 2, 5]
nums2 = [2, 2, 1]
diff = 1
print(countPairs(nums1, nums2, diff))
```

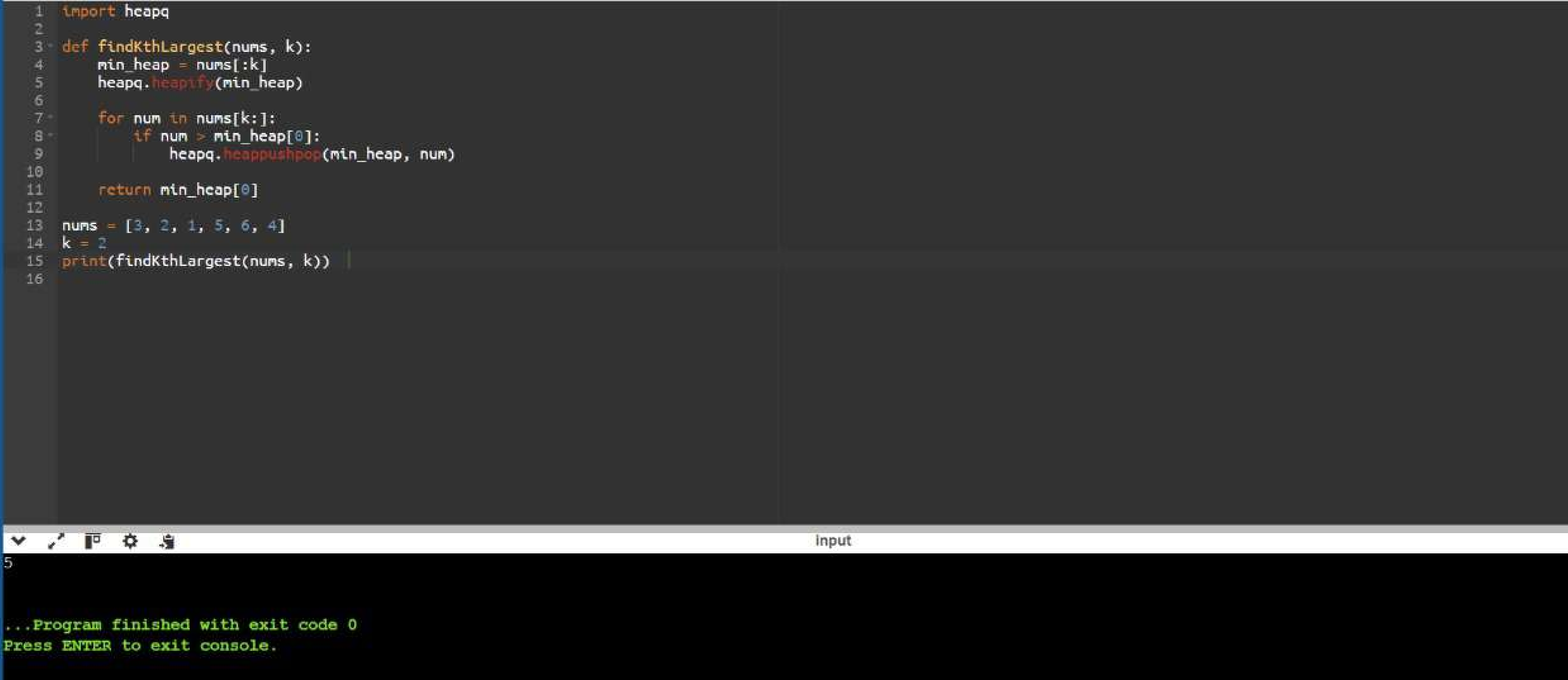
```
3
```



main.py

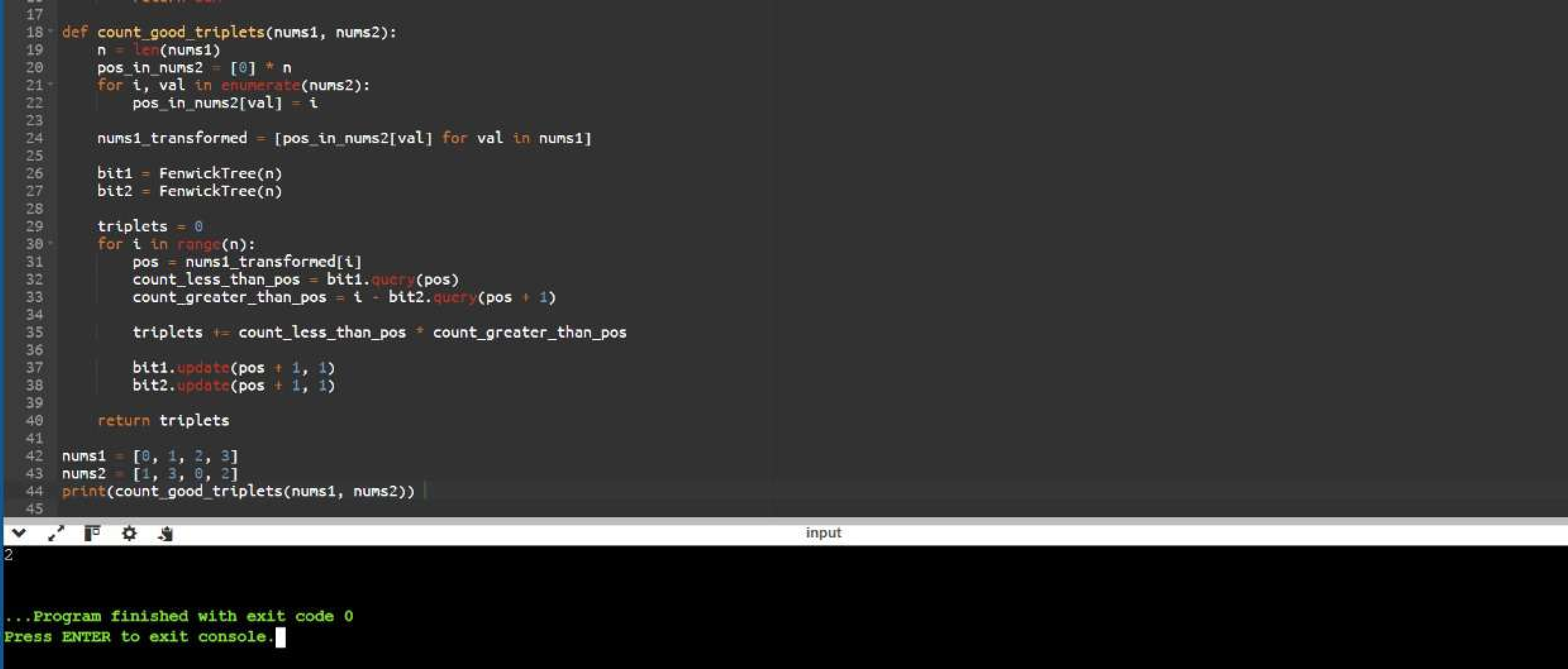
```python
def majorityElement(nums):
    candidate = None
    count = 0

    for num in nums:
        if count == 0:
            candidate = num
        count += (num == candidate) - (num != candidate)

    return candidate


nums1 = [3, 2, 3]
print(majorityElement(nums1))

nums2 = [2, 2, 1, 1, 1, 2, 2]
print(majorityElement(nums2))

```

```
3
2


...Program finished with exit code 0
Press ENTER to exit console.
```



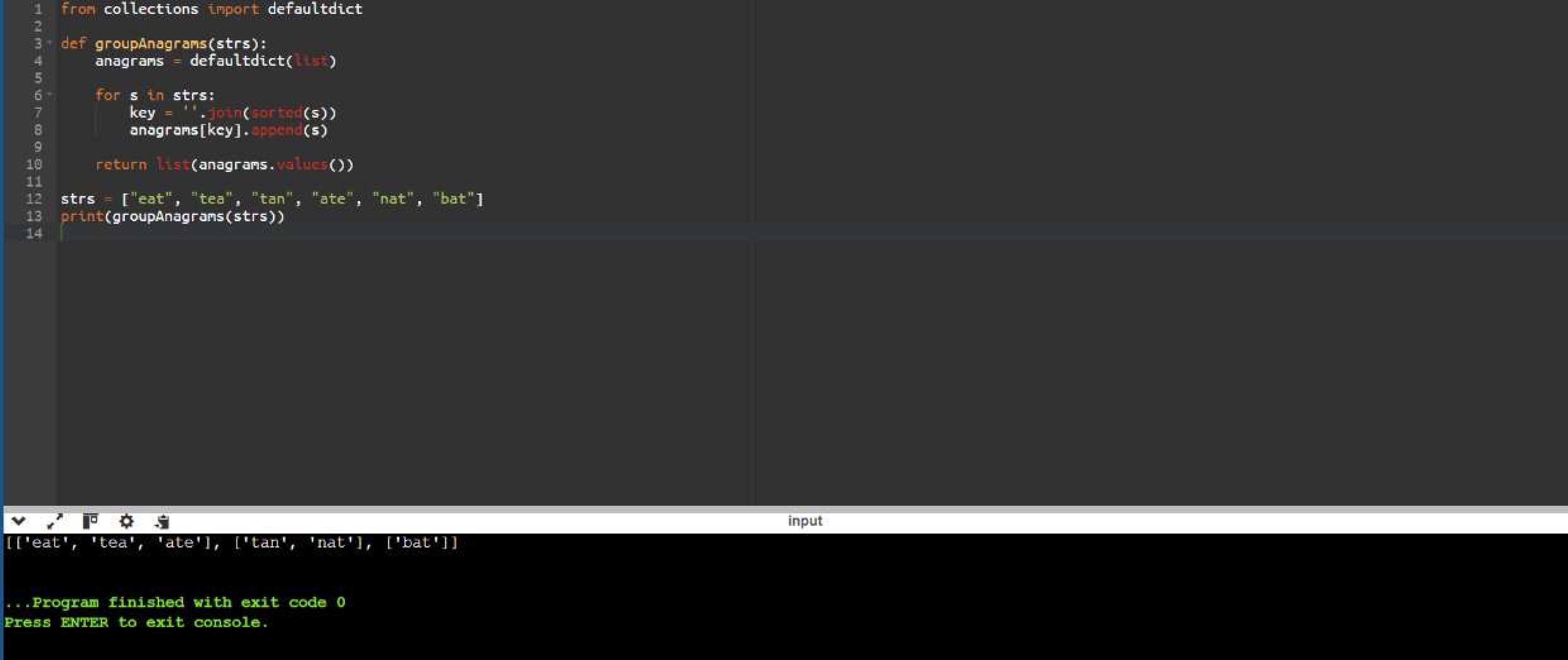
```python
import heapq

def findKthLargest(nums, k):
    min_heap = nums[:k]
    heapq.heapify(min_heap)

    for num in nums[k:]:
        if num > min_heap[0]:
            heapq.heappushpop(min_heap, num)

    return min_heap[0]

nums = [3, 2, 1, 5, 6, 4]
k = 2
print(findKthLargest(nums, k))
```

```
5


...Program finished with exit code 0
Press ENTER to exit console.
```



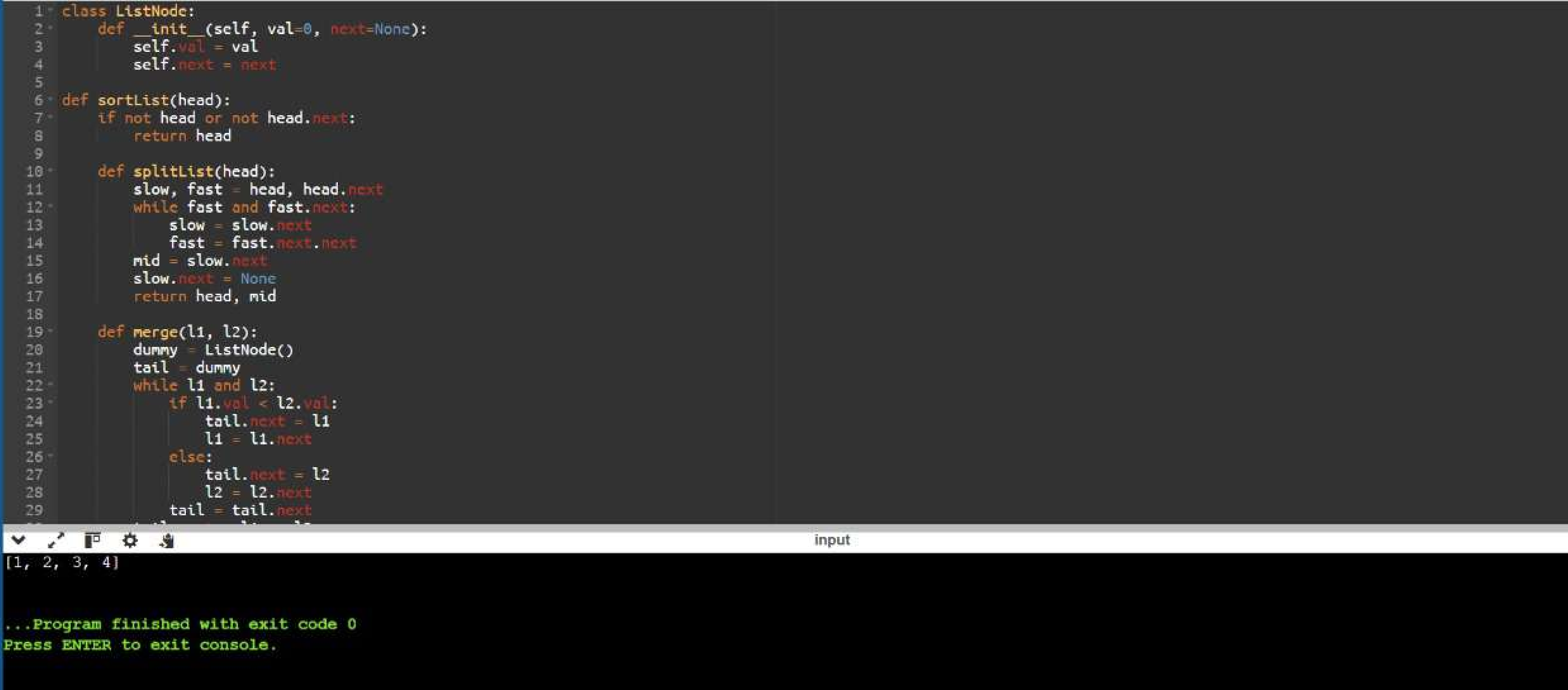
```python
def count_good_triplets(nums1, nums2):
    n = len(nums1)
    pos_in_nums2 = [0] * n
    for i, val in enumerate(nums2):
        pos_in_nums2[val] = i

    nums1_transformed = [pos_in_nums2[val] for val in nums1]

    bit1 = FenwickTree(n)
    bit2 = FenwickTree(n)

    triplets = 0
    for i in range(n):
        pos = nums1_transformed[i]
        count_less_than_pos = bit1.query(pos)
        count_greater_than_pos = i - bit2.query(pos + 1)

        triplets += count_less_than_pos * count_greater_than_pos

        bit1.update(pos + 1, 1)
        bit2.update(pos + 1, 1)

    return triplets

nums1 = [0, 1, 2, 3]
nums2 = [1, 3, 0, 2]
print(count_good_triplets(nums1, nums2))
```

```
2


...Program finished with exit code 0
Press ENTER to exit console.
```



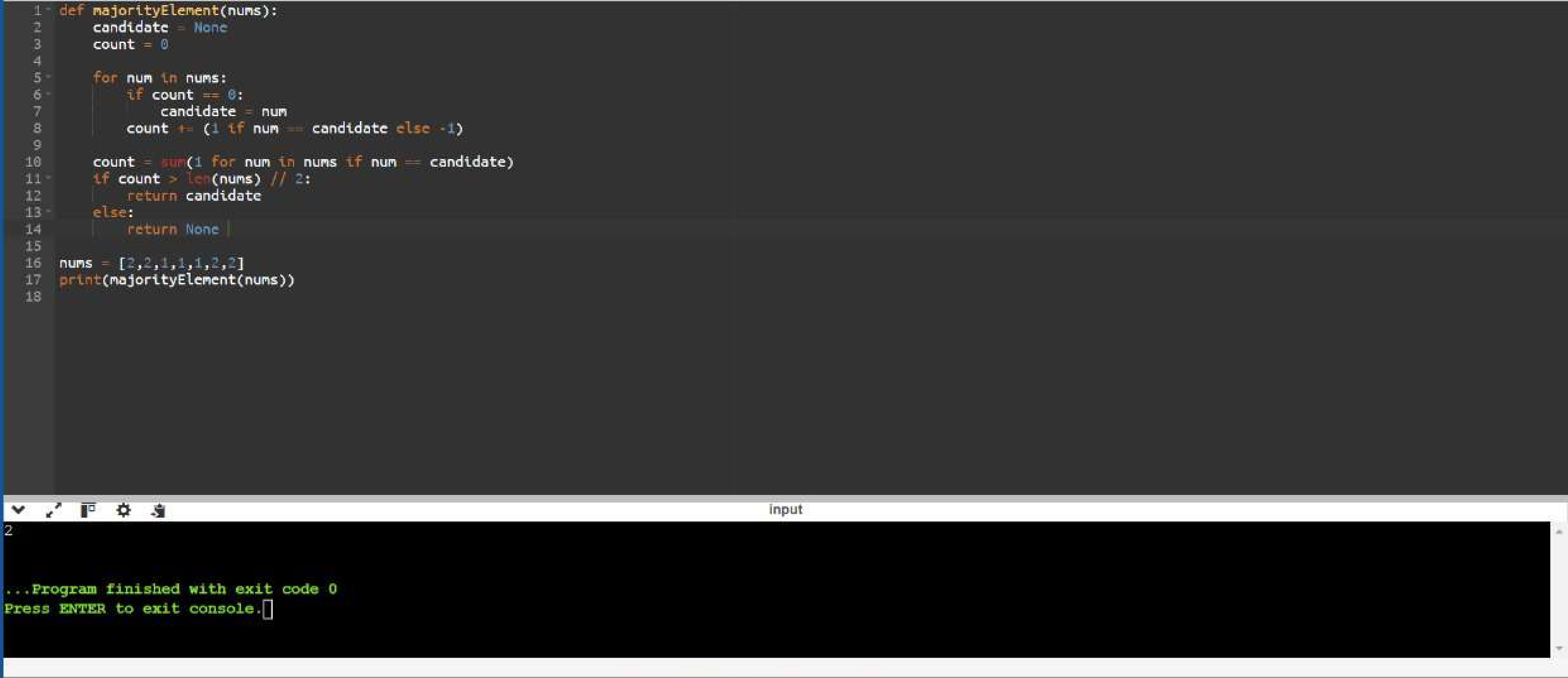
```python
from collections import defaultdict

def groupAnagrams(strs):
    anagrams = defaultdict(list)

    for s in strs:
        key = ''.join(sorted(s))
        anagrams[key].append(s)

    return list(anagrams.values())

strs = ["eat", "tea", "tan", "ate", "nat", "bat"]
print(groupAnagrams(strs))

```

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


...Program finished with exit code 0
Press ENTER to exit console.
```



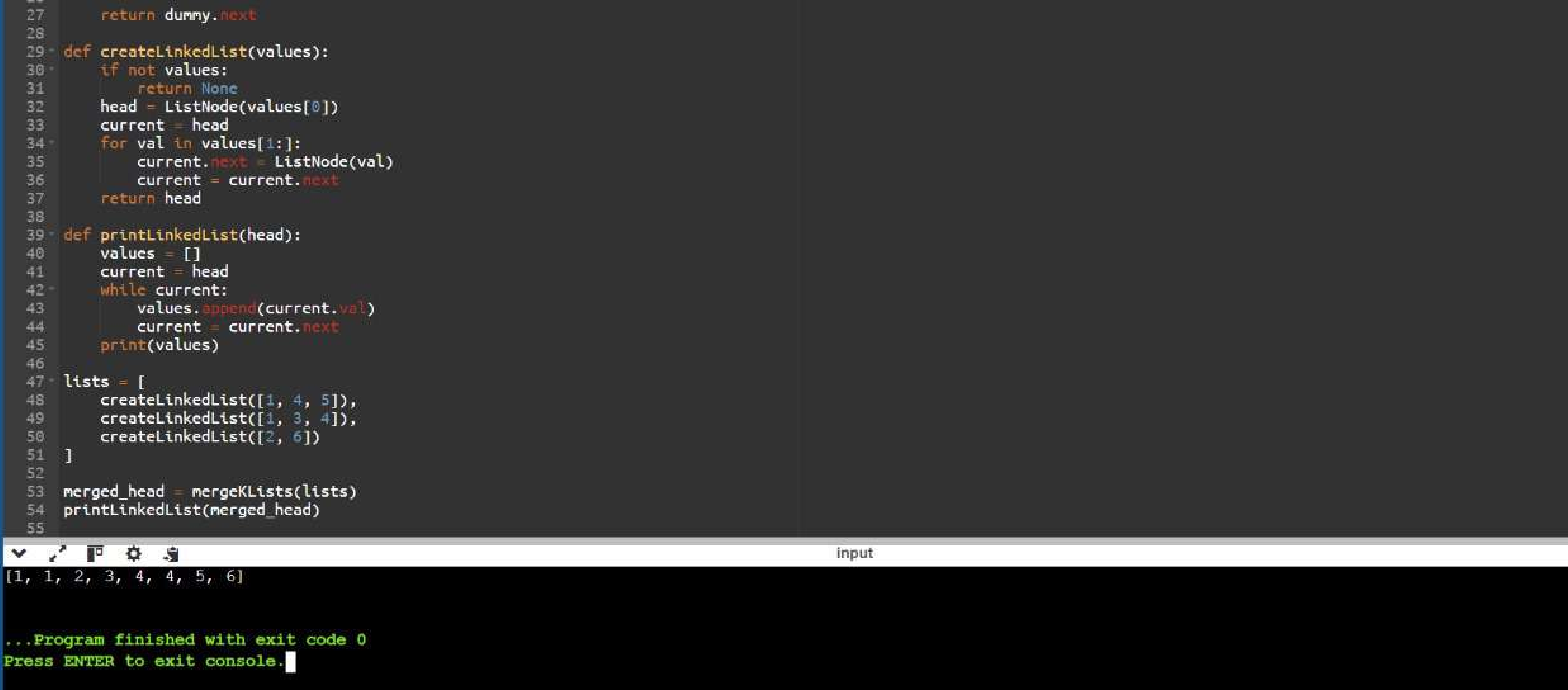
```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def sortList(head):
    if not head or not head.next:
        return head

    def splitList(head):
        slow, fast = head, head.next
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
        mid = slow.next
        slow.next = None
        return head, mid

    def merge(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
```

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


...Program finished with exit code 0
Press ENTER to exit console.
```



```python
def majorityElement(nums):
    candidate = None
    count = 0

    for num in nums:
        if count == 0:
            candidate = num
        count += (1 if num == candidate else -1)

    count = sum(1 for num in nums if num == candidate)
    if count > len(nums) // 2:
        return candidate
    else:
        return None

nums = [2,2,1,1,1,2,2]
print(majorityElement(nums))
```

```
2


...Program finished with exit code 0
Press ENTER to exit console.
```



```
    return dummy.next

def createLinkedList(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for val in values[1:]:
        current.next = ListNode(val)
        current = current.next
    return head

def printLinkedList(head):
    values = []
    current = head
    while current:
        values.append(current.val)
        current = current.next
    print(values)

lists = [
    createLinkedList([1, 4, 5]),
    createLinkedList([1, 3, 4]),
    createLinkedList([2, 6])
]

merged_head = mergeKLists(lists)
printLinkedList(merged_head)

```

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


...Program finished with exit code 0
Press ENTER to exit console.
```