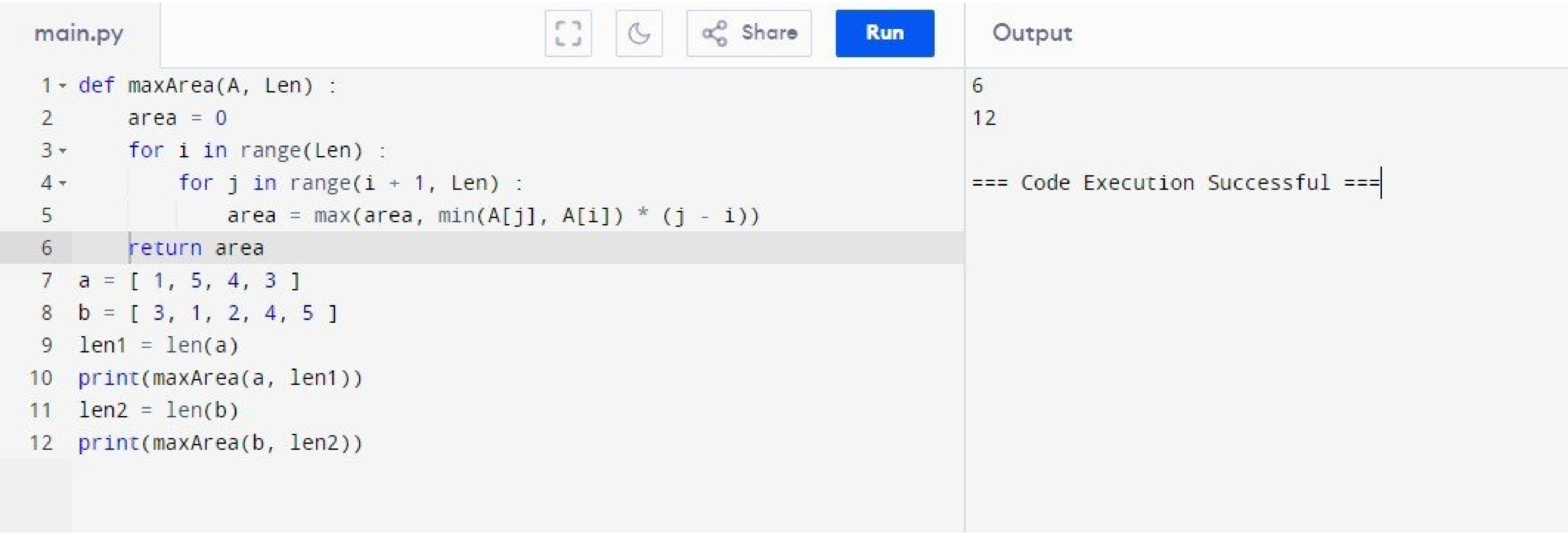
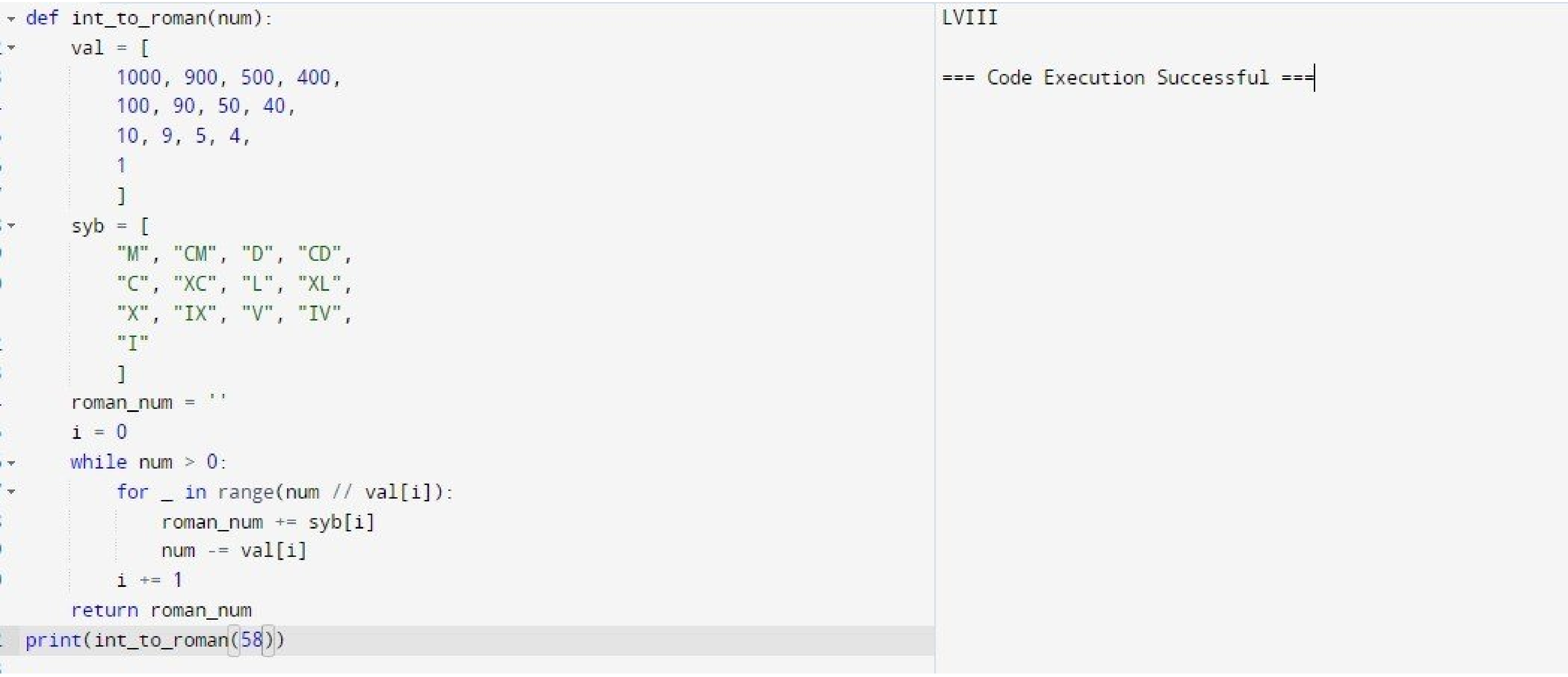


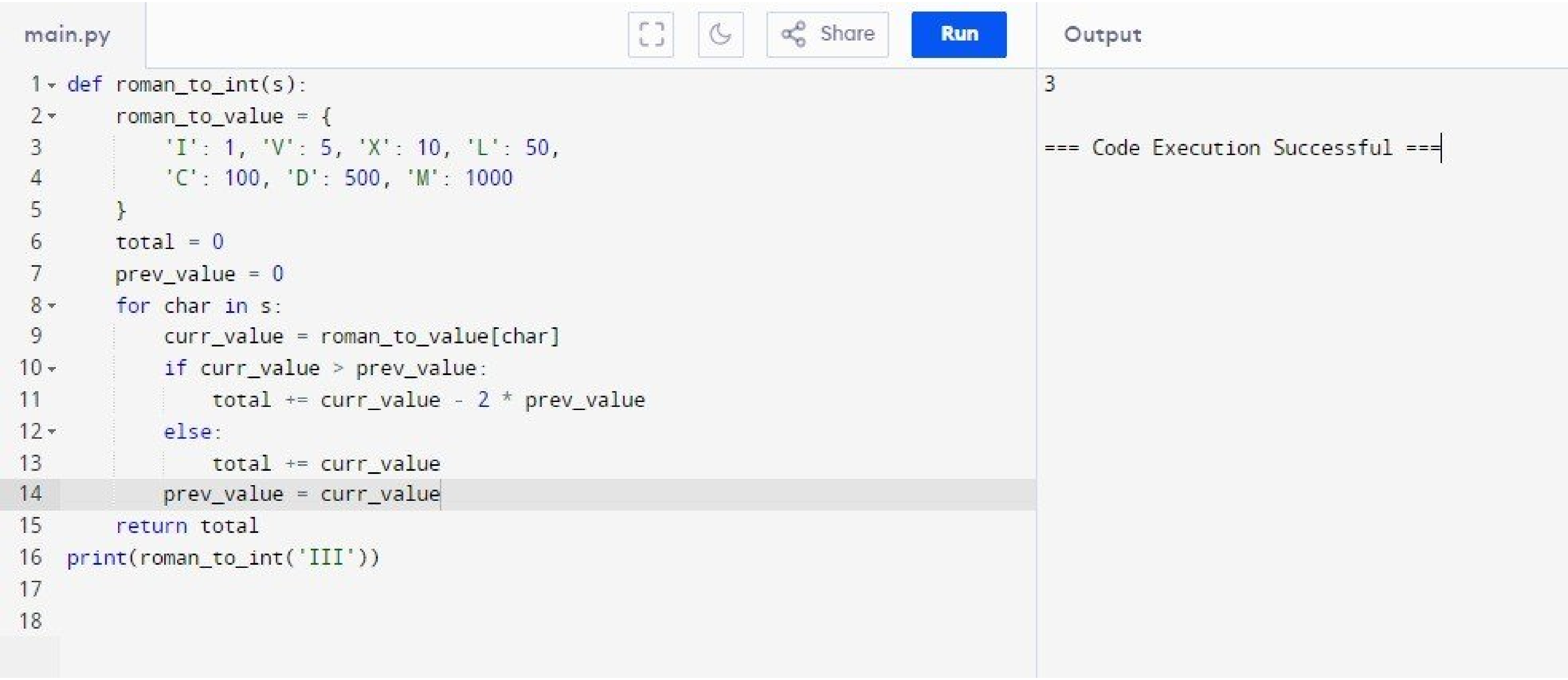
```python
def maxArea(A, Len) :
    area = 0
    for i in range(Len) :
        for j in range(i + 1, Len) :
            area = max(area, min(A[j], A[i]) * (j - i))
    return area
a = [ 1, 5, 4, 3 ]
b = [ 3, 1, 2, 4, 5 ]
len1 = len(a)
print(maxArea(a, len1))
len2 = len(b)
print(maxArea(b, len2))
```

```
6
12

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



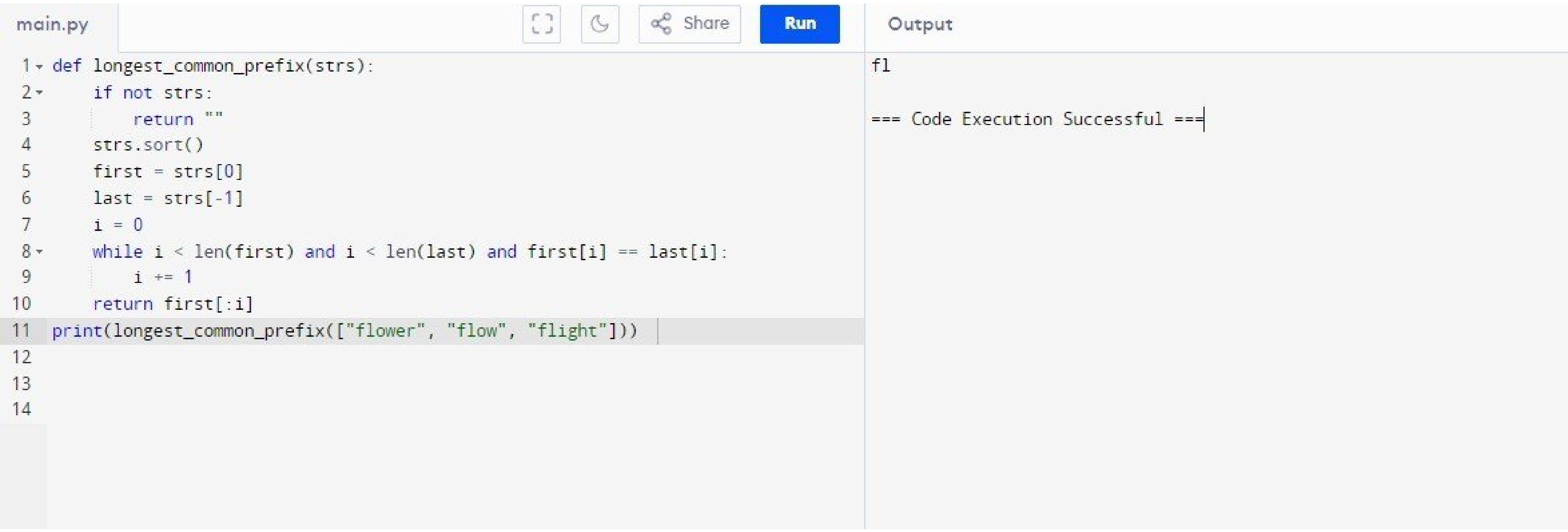
```python
def int_to_roman(num):
    val = [
        1000, 900, 500, 400,
        100, 90, 50, 40,
        10, 9, 5, 4,
        1
        ]
    syb = [
        "M", "CM", "D", "CD",
        "C", "XC", "L", "XL",
        "X", "IX", "V", "IV",
        "I"
        ]
    roman_num = ''
    i = 0
    while num > 0:
        for _ in range(num // val[i]):
            roman_num += syb[i]
            num -= val[i]
        i += 1
    return roman_num
print(int_to_roman(58))
```

```
LVIII

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



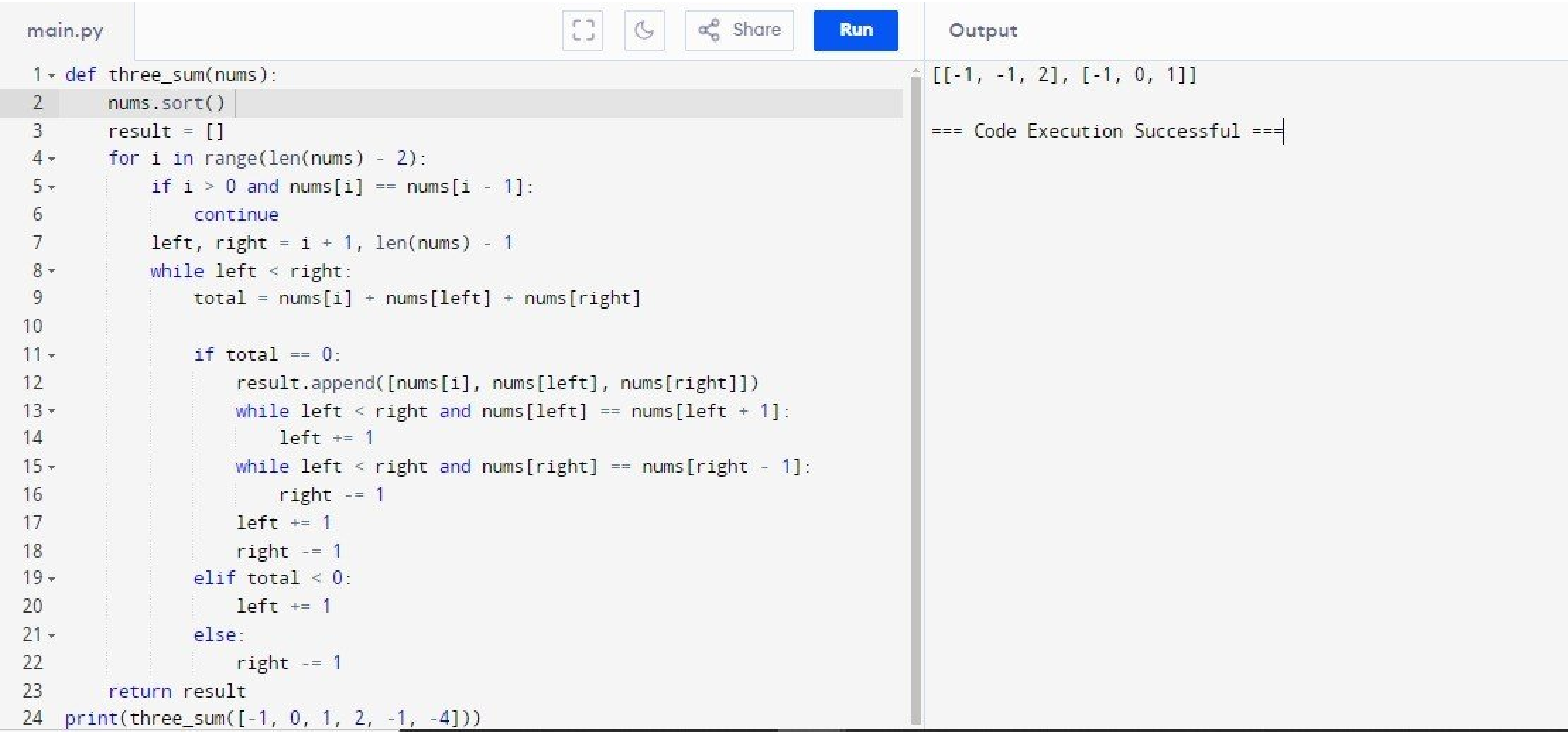
```python
def roman_to_int(s):
    roman_to_value = {
        'I': 1, 'V': 5, 'X': 10, 'L': 50,
        'C': 100, 'D': 500, 'M': 1000
    }
    total = 0
    prev_value = 0
    for char in s:
        curr_value = roman_to_value[char]
        if curr_value > prev_value:
            total += curr_value - 2 * prev_value
        else:
            total += curr_value
        prev_value = curr_value
    return total
print(roman_to_int('III'))
```

```
3


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



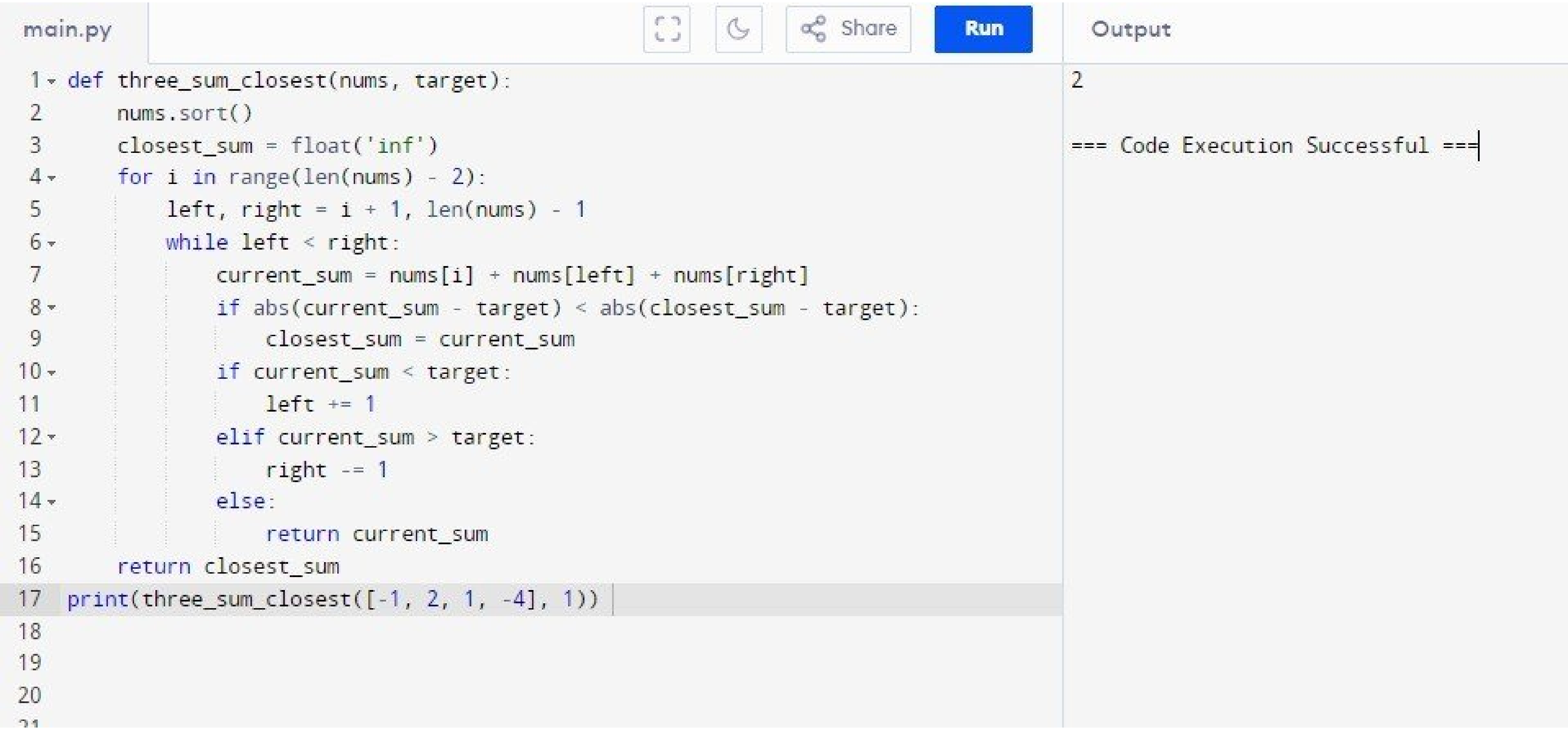
```python
def longest_common_prefix(strs):
    if not strs:
        return ""
    strs.sort()
    first = strs[0]
    last = strs[-1]
    i = 0
    while i < len(first) and i < len(last) and first[i] == last[i]:
        i += 1
    return first[:i]
print(longest_common_prefix(["flower", "flow", "flight"]))
```

```
fl

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



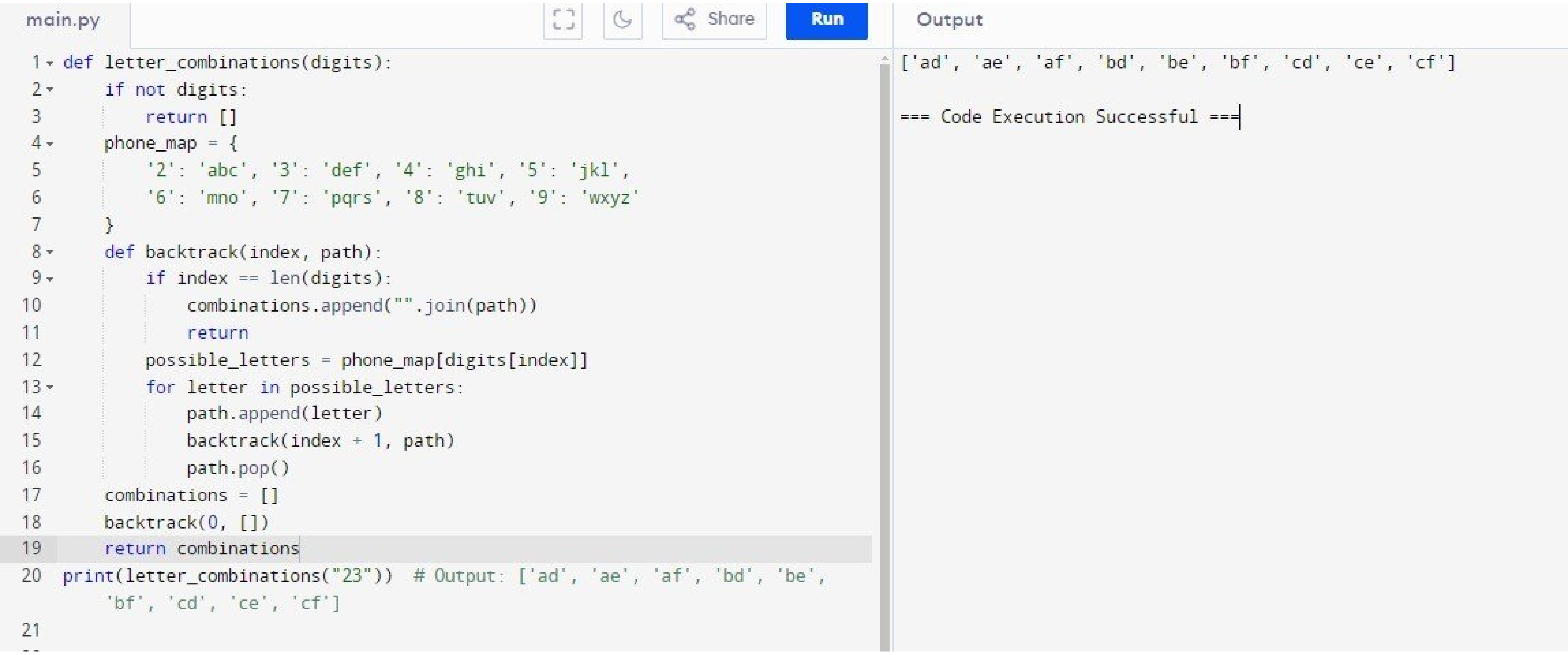
```python
def three_sum(nums):
    nums.sort()
    result = []
    for i in range(len(nums) - 2):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        left, right = i + 1, len(nums) - 1
        while left < right:
            total = nums[i] + nums[left] + nums[right]

            if total == 0:
                result.append([nums[i], 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 < 0:
                left += 1
            else:
                right -= 1
    return result
print(three_sum([-1, 0, 1, 2, -1, -4]))
```

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

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



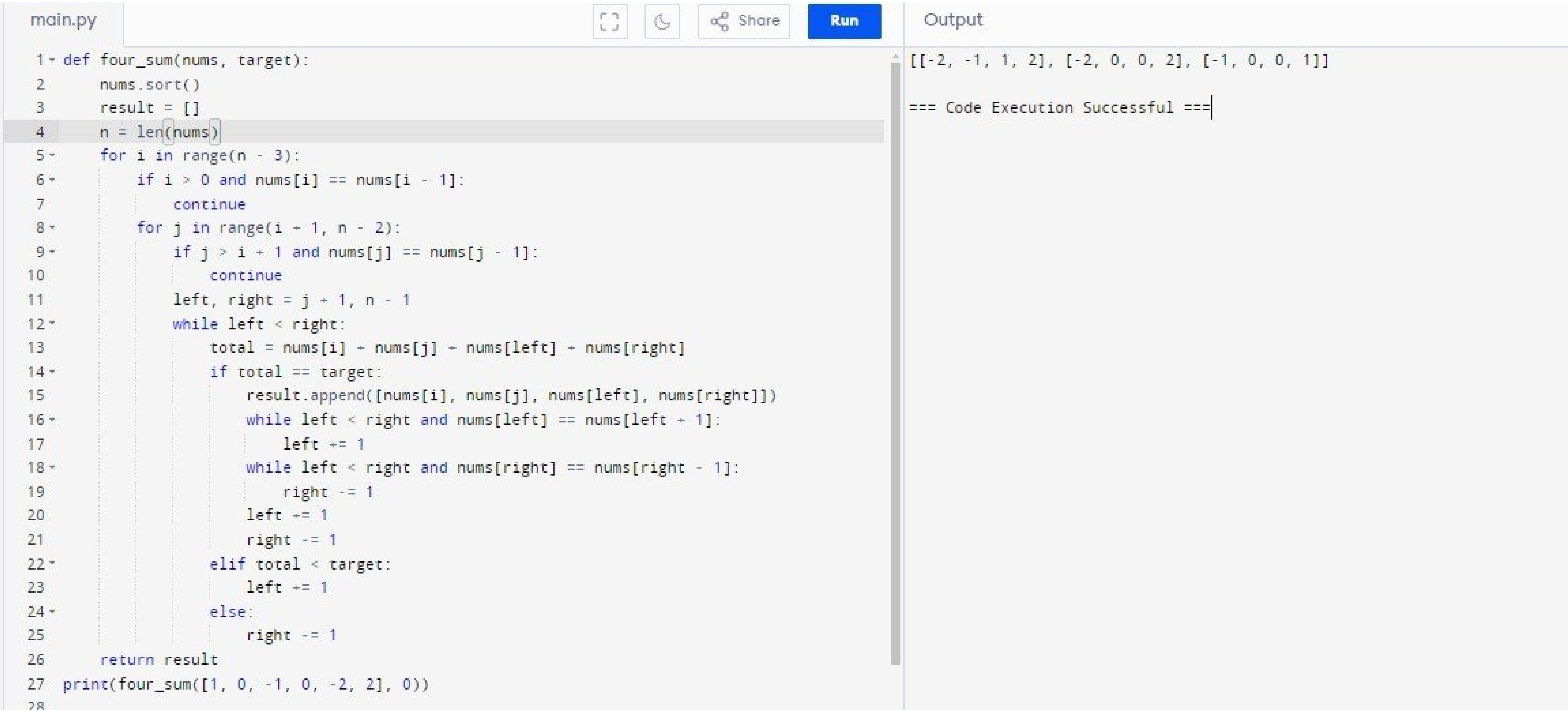
```python
def three_sum_closest(nums, target):
    nums.sort()
    closest_sum = float('inf')
    for i in range(len(nums) - 2):
        left, right = i + 1, len(nums) - 1
        while left < right:
            current_sum = nums[i] + nums[left] + nums[right]
            if abs(current_sum - target) < abs(closest_sum - target):
                closest_sum = current_sum
            if current_sum < target:
                left += 1
            elif current_sum > target:
                right -= 1
            else:
                return current_sum
    return closest_sum
print(three_sum_closest([-1, 2, 1, -4], 1))
```

```
2


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



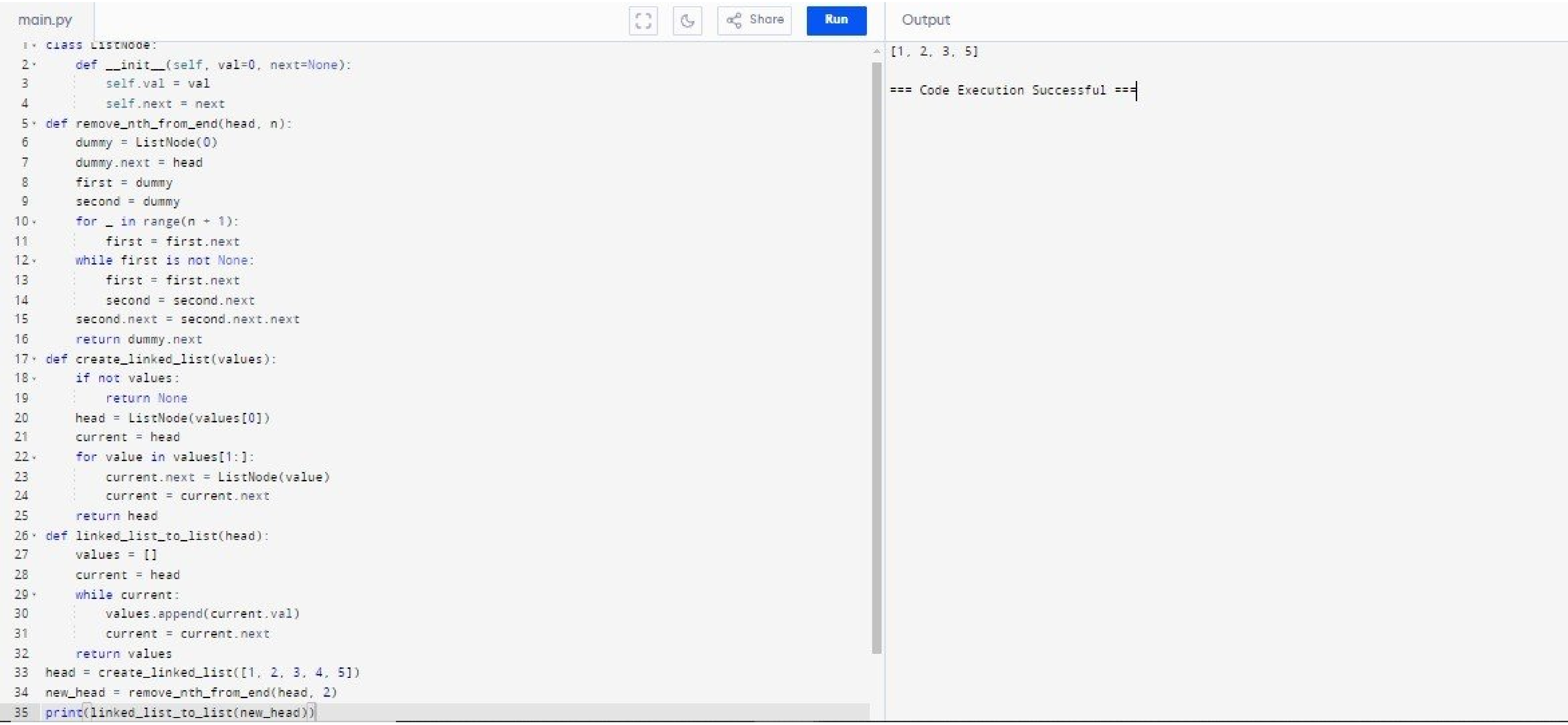
```python
def letter_combinations(digits):
    if not digits:
        return []
    phone_map = {
        '2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',
        '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'
    }
    def backtrack(index, path):
        if index == len(digits):
            combinations.append("".join(path))
            return
        possible_letters = phone_map[digits[index]]
        for letter in possible_letters:
            path.append(letter)
            backtrack(index + 1, path)
            path.pop()
    combinations = []
    backtrack(0, [])
    return combinations
print(letter_combinations("23"))  # Output: ['ad', 'ae', 'af', 'bd', 'be',
    'bf', 'cd', 'ce', 'cf']
```

```
['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']

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



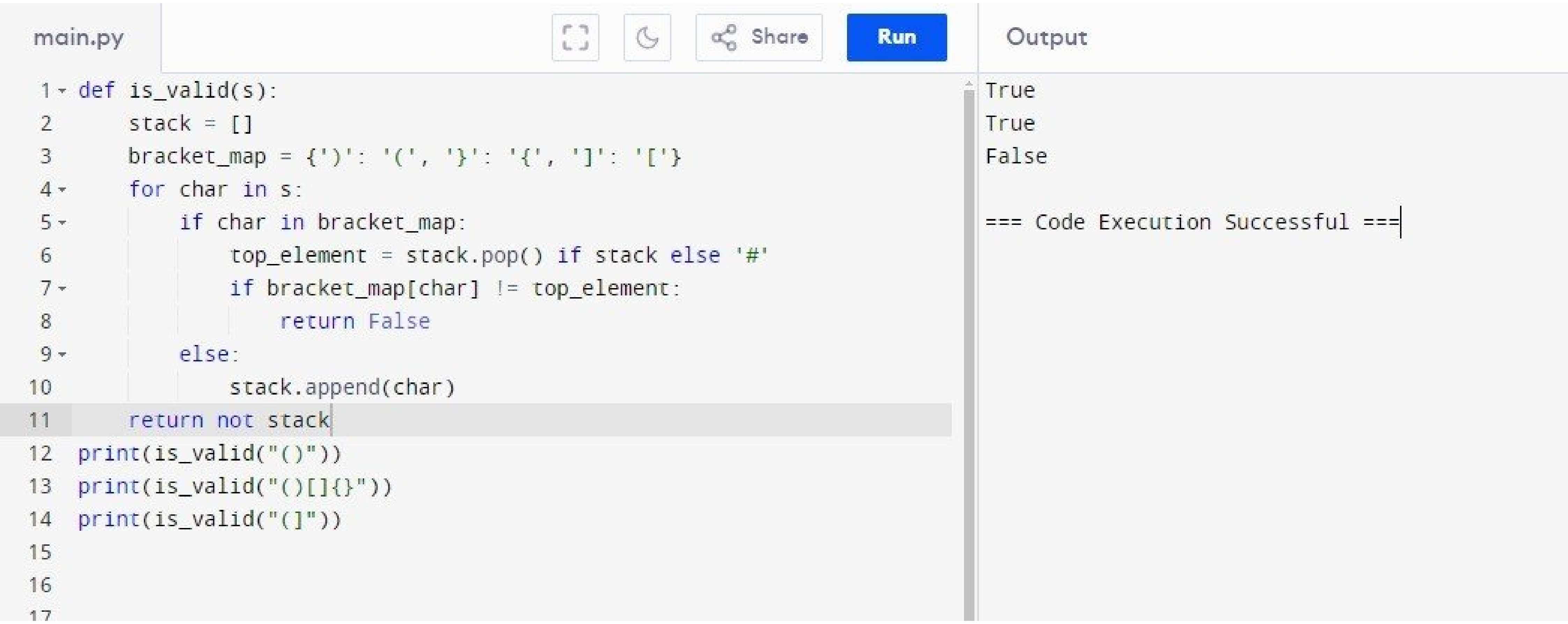
```python
def four_sum(nums, target):
    nums.sort()
    result = []
    n = len(nums)
    for i in range(n - 3):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        for j in range(i + 1, n - 2):
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue
            left, right = j + 1, n - 1
            while left < right:
                total = nums[i] + nums[j] + nums[left] + nums[right]
                if total == target:
                    result.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 result
print(four_sum([1, 0, -1, 0, -2, 2], 0))
```

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

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



```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def remove_nth_from_end(head, n):
    dummy = ListNode(0)
    dummy.next = head
    first = dummy
    second = dummy
    for _ in range(n + 1):
        first = first.next
    while first is not None:
        first = first.next
        second = second.next
    second.next = second.next.next
    return dummy.next
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head
def linked_list_to_list(head):
    values = []
    current = head
    while current:
        values.append(current.val)
        current = current.next
    return values
head = create_linked_list([1, 2, 3, 4, 5])
new_head = remove_nth_from_end(head, 2)
print(linked_list_to_list(new_head))
```

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

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



```python
def is_valid(s):
    stack = []
    bracket_map = {')': '(', '}': '{', ']': '['}
    for char in s:
        if char in bracket_map:
            top_element = stack.pop() if stack else '#'
            if bracket_map[char] != top_element:
                return False
        else:
            stack.append(char)
    return not stack
print(is_valid("()"))
print(is_valid("()[]{}"))
print(is_valid("(]"))
```

```
True
True
False

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