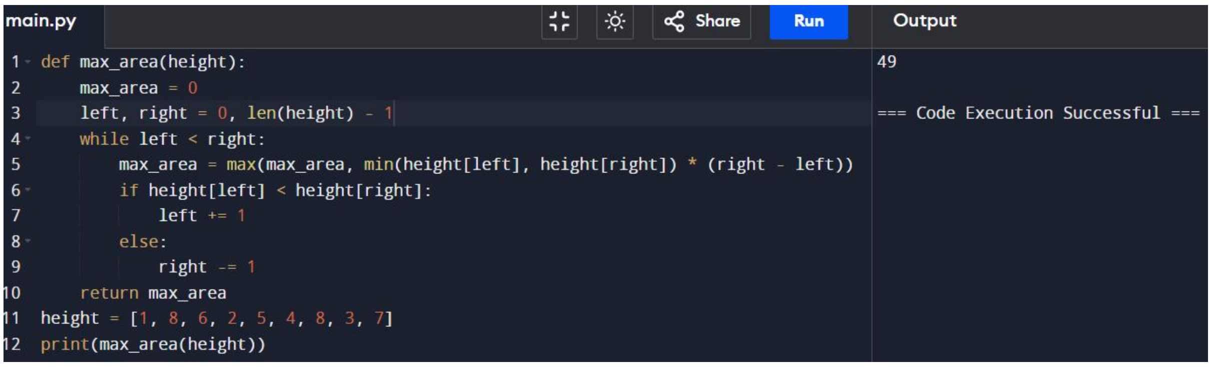
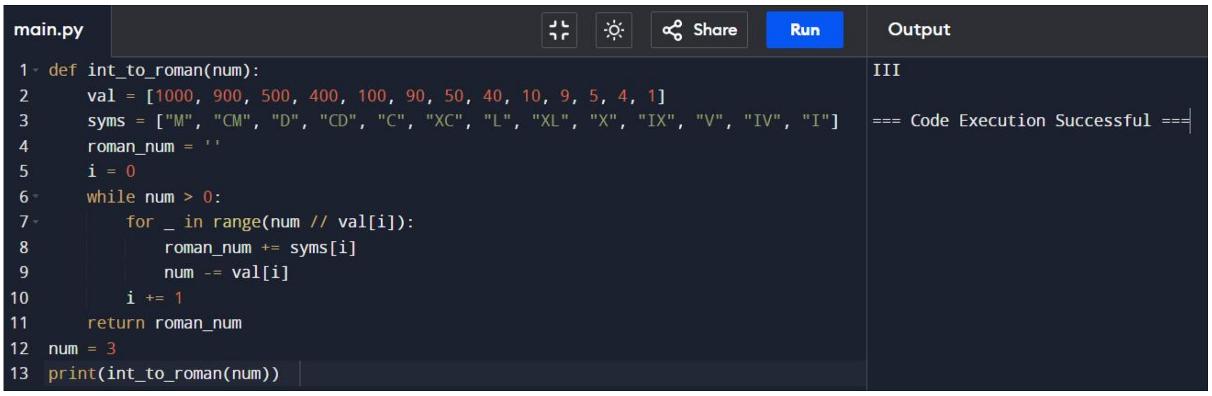
1



```python
def max_area(height):
    max_area = 0
    left, right = 0, len(height) - 1
    while left < right:
        max_area = max(max_area, min(height[left], height[right]) * (right - left))
        if height[left] < height[right]:
            left += 1
        else:
            right -= 1
    return max_area
height = [1, 8, 6, 2, 5, 4, 8, 3, 7]
print(max_area(height))
```

```
49

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

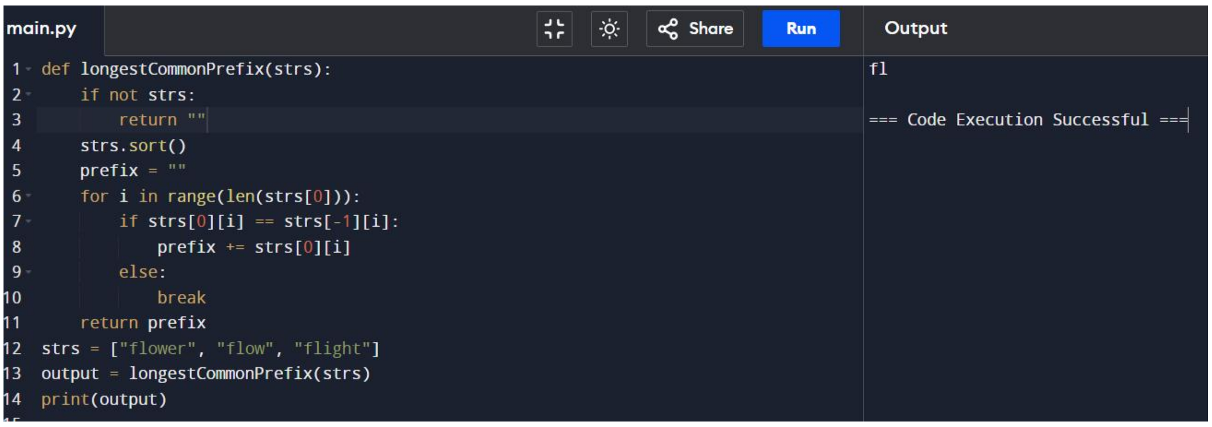
2



```python
def int_to_roman(num):
    val = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]
    syms = ["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 += syms[i]
            num -= val[i]
        i += 1
    return roman_num
num = 3
print(int_to_roman(num))
```

```
III

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

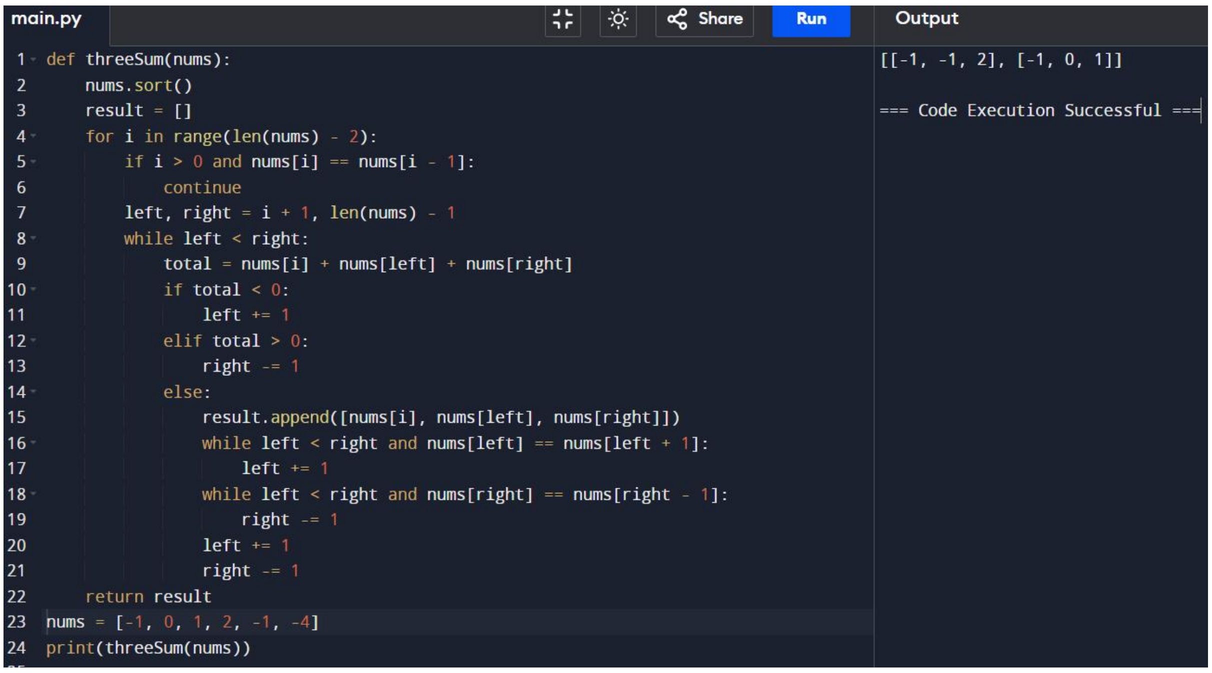
3



```python
def longestCommonPrefix(strs):
    if not strs:
        return ""
    strs.sort()
    prefix = ""
    for i in range(len(strs[0])):
        if strs[0][i] == strs[-1][i]:
            prefix += strs[0][i]
        else:
            break
    return prefix
strs = ["flower", "flow", "flight"]
output = longestCommonPrefix(strs)
print(output)
```

```
fl

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

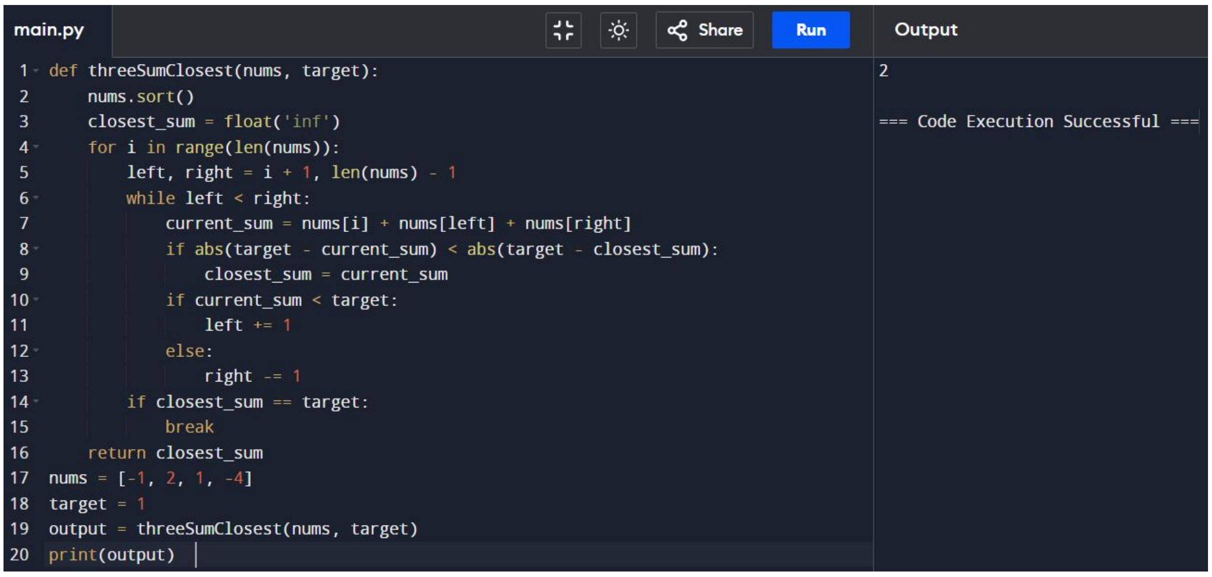
4



```python
def threeSum(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:
                left += 1
            elif total > 0:
                right -= 1
            else:
                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
    return result
nums = [-1, 0, 1, 2, -1, -4]
print(threeSum(nums))
```

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

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

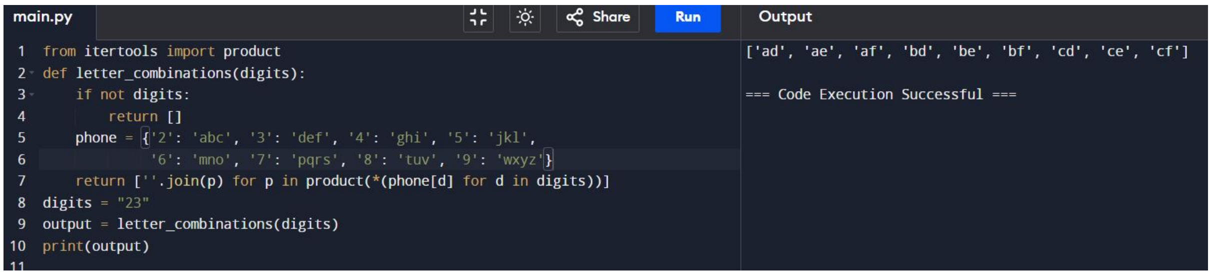
5



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

```
2

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

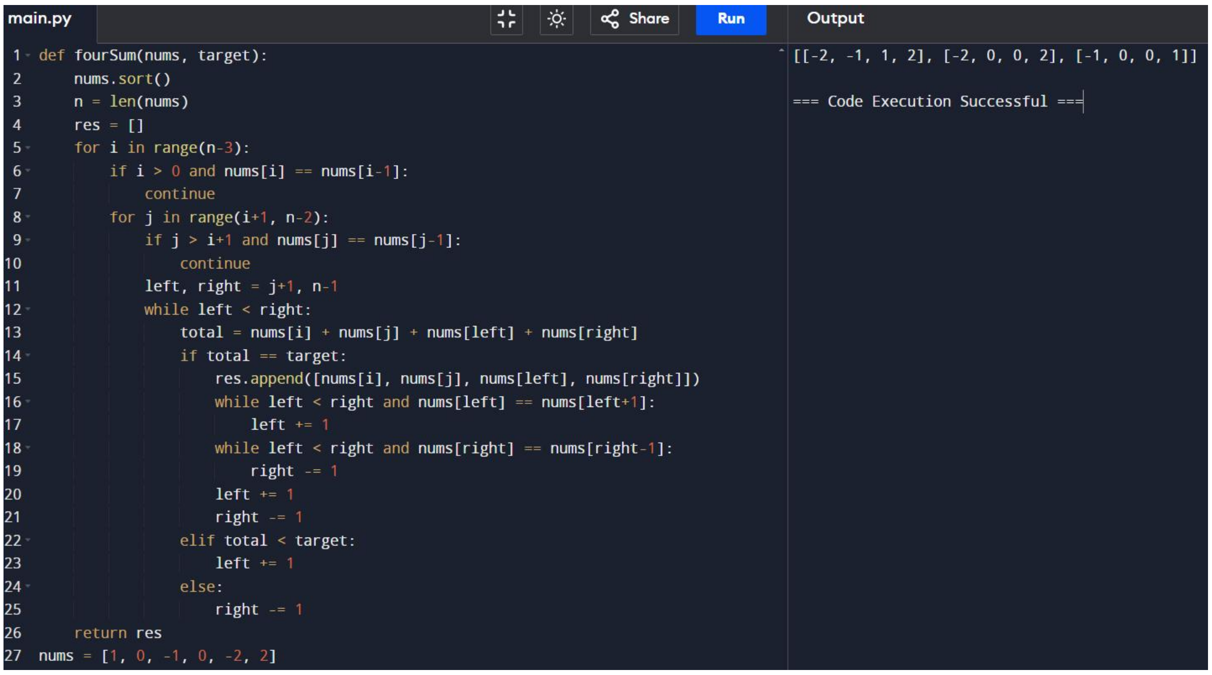
6



```python
from itertools import product
def letter_combinations(digits):
    if not digits:
        return []
    phone = {'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',
             '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'}
    return [''.join(p) for p in product(*(phone[d] for d in digits))]
digits = "23"
output = letter_combinations(digits)
print(output)
```

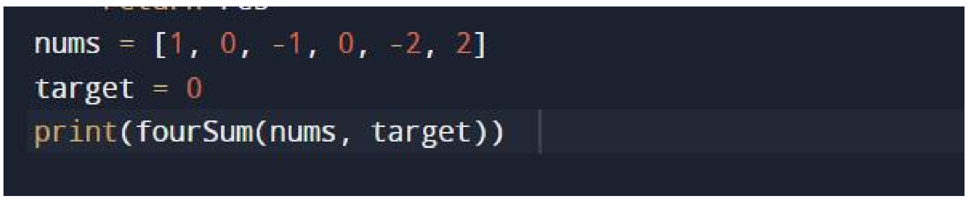
```
['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']

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

7



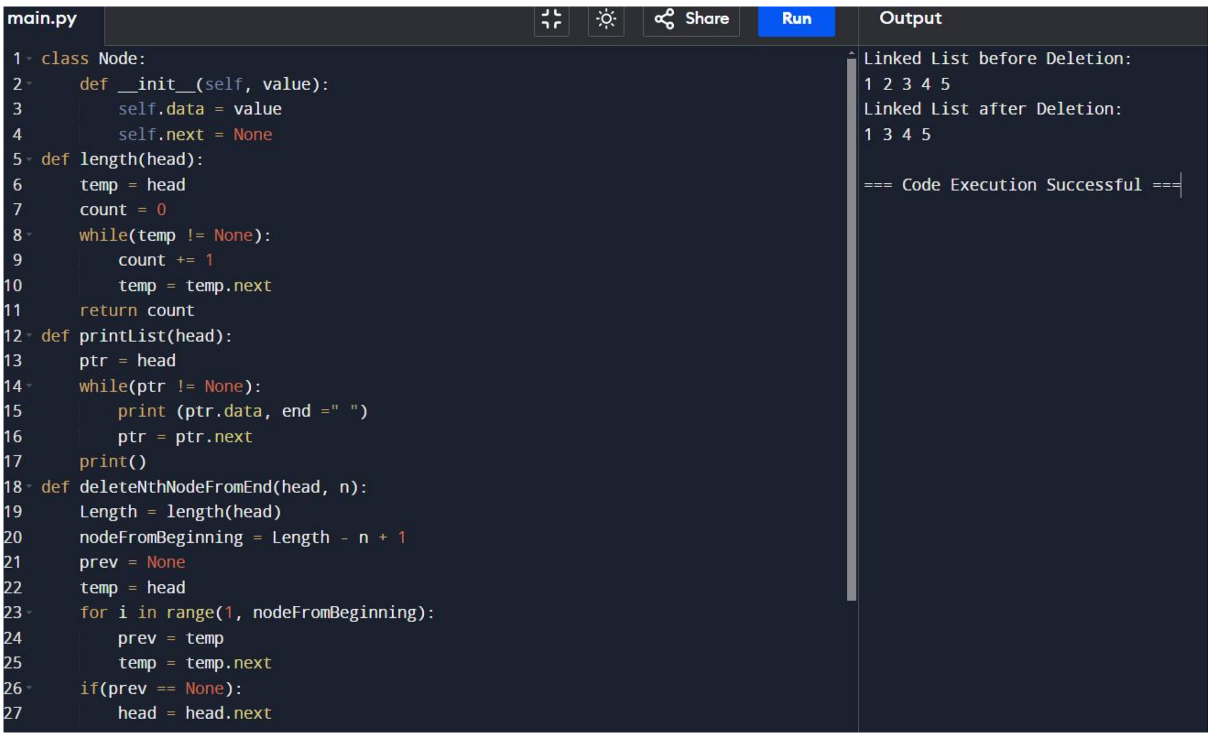
```python
def fourSum(nums, target):
    nums.sort()
    n = len(nums)
    res = []
    for i in range(n-3):
        if i > 0 and nums[i] == nums[i-1]:
            continue
        for j in range(i+1, n-2):
            if j > i+1 and nums[j] == nums[j-1]:
                continue
            left, right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

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

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



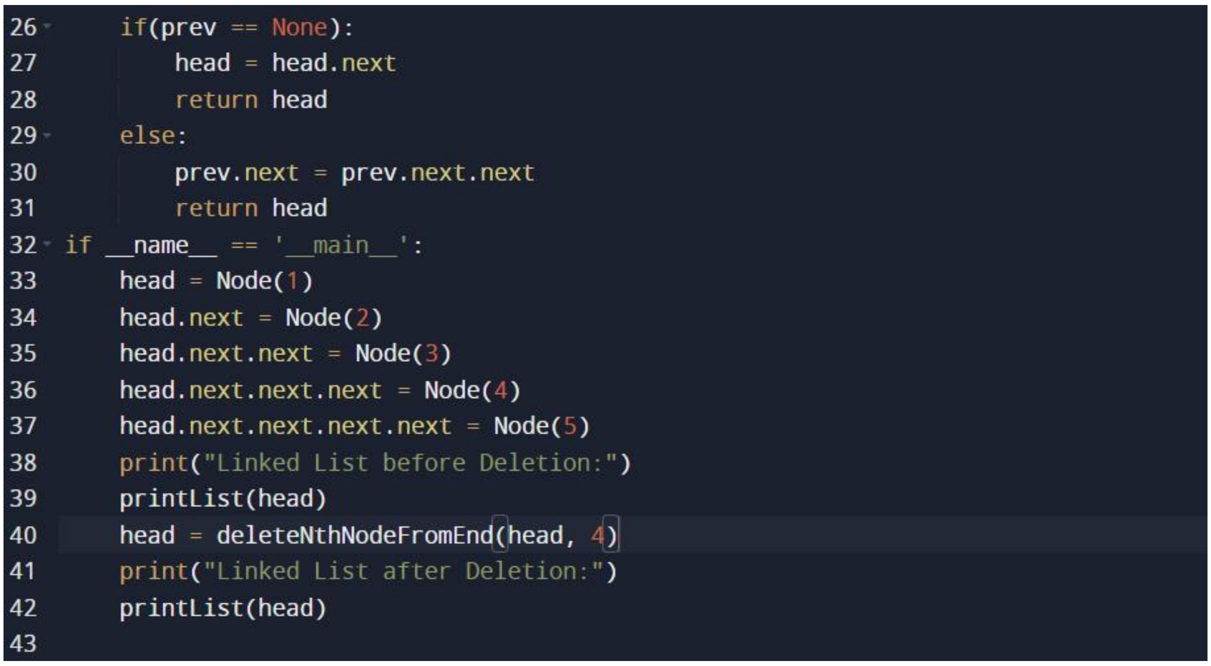
```python
nums = [1, 0, -1, 0, -2, 2]
target = 0
print(fourSum(nums, target))
```

8



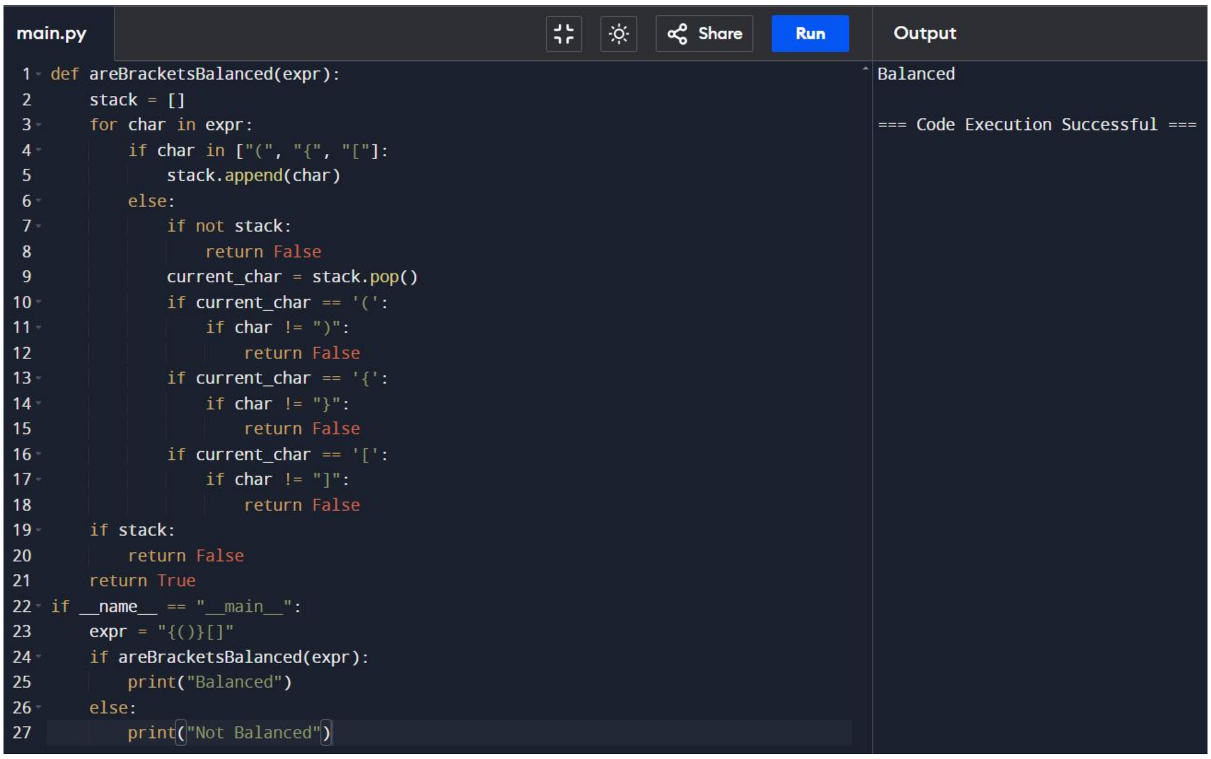
```python
class Node:
    def __init__(self, value):
        self.data = value
        self.next = None
def length(head):
    temp = head
    count = 0
    while(temp != None):
        count += 1
        temp = temp.next
    return count
def printList(head):
    ptr = head
    while(ptr != None):
        print (ptr.data, end =" ")
        ptr = ptr.next
    print()
def deleteNthNodeFromEnd(head, n):
    Length = length(head)
    nodeFromBeginning = Length - n + 1
    prev = None
    temp = head
    for i in range(1, nodeFromBeginning):
        prev = temp
        temp = temp.next
    if(prev == None):
        head = head.next
```

```
Linked List before Deletion:
1 2 3 4 5
Linked List after Deletion:
1 3 4 5

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



```python
    if(prev == None):
        head = head.next
        return head
    else:
        prev.next = prev.next.next
        return head
if __name__ == '__main__':
    head = Node(1)
    head.next = Node(2)
    head.next.next = Node(3)
    head.next.next.next = Node(4)
    head.next.next.next.next = Node(5)
    print("Linked List before Deletion:")
    printList(head)
    head = deleteNthNodeFromEnd(head, 4)
    print("Linked List after Deletion:")
    printList(head)

```

9



```python
def areBracketsBalanced(expr):
    stack = []
    for char in expr:
        if char in ["(", "{", "["]:
            stack.append(char)
        else:
            if not stack:
                return False
            current_char = stack.pop()
            if current_char == '(':
                if char != ")":
                    return False
            if current_char == '{':
                if char != "}":
                    return False
            if current_char == '[':
                if char != "]":
                    return False
    if stack:
        return False
    return True
if __name__ == "__main__":
    expr = "{()}[]"
    if areBracketsBalanced(expr):
        print("Balanced")
    else:
        print("Not Balanced")
```

```
Balanced

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