ASSIGNMENT-6

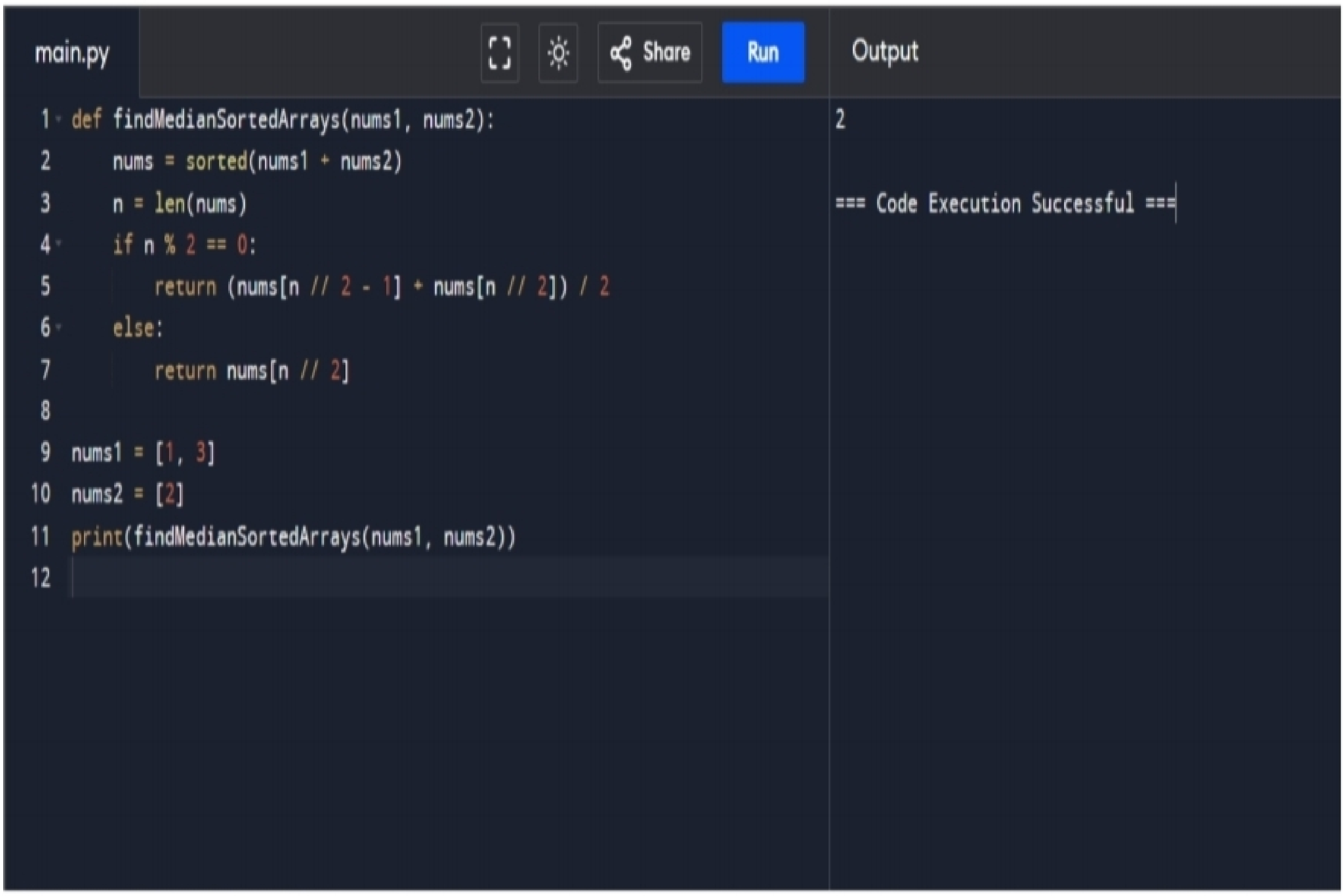
NAME: K. Gowtham

REG-NO: 192372078

SUBJECT: Python

CODE: CSA0898

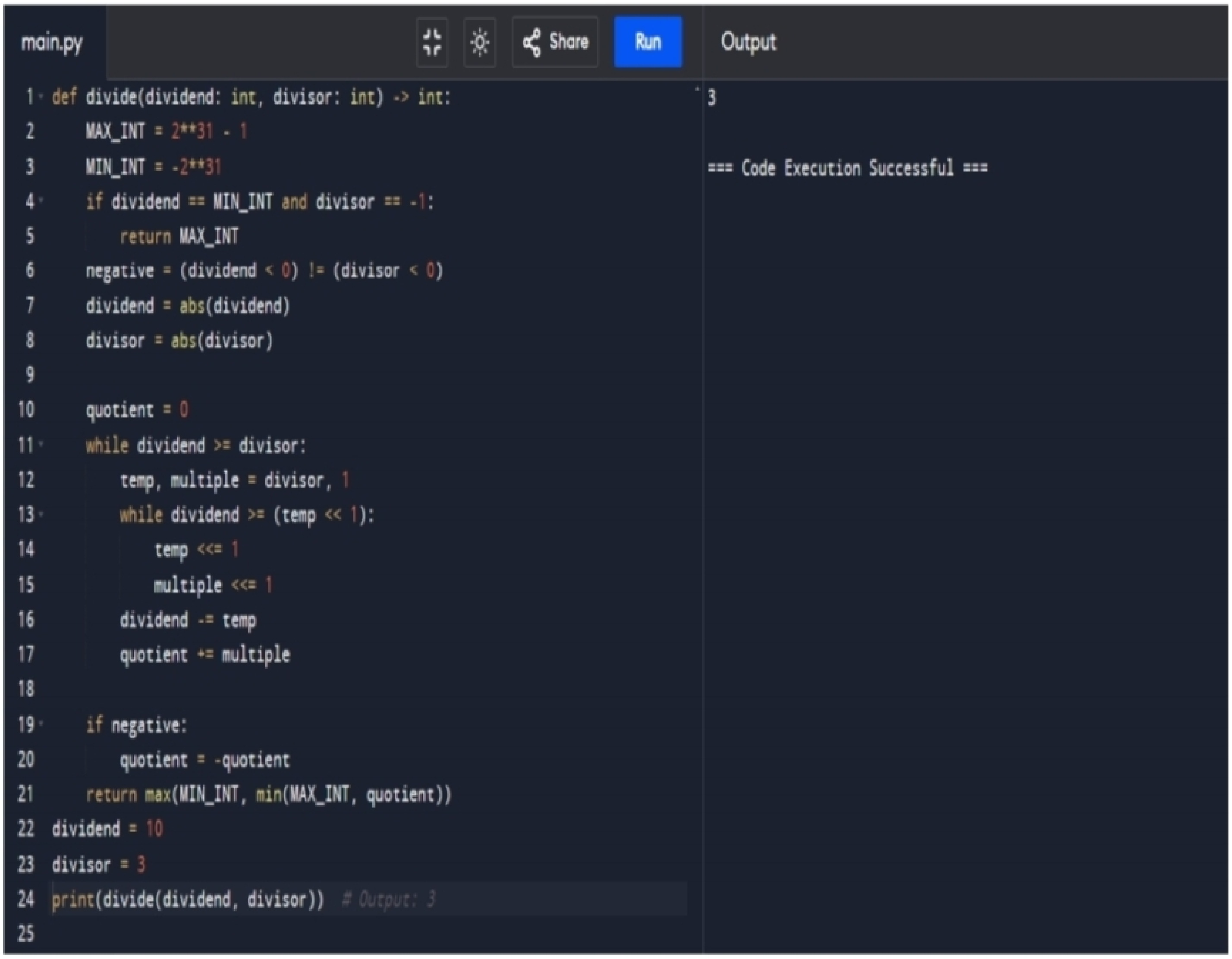
1.Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)). Example 1: Input: nums1 = [1,3], nums2 = [2] Output: 2.00000



```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


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

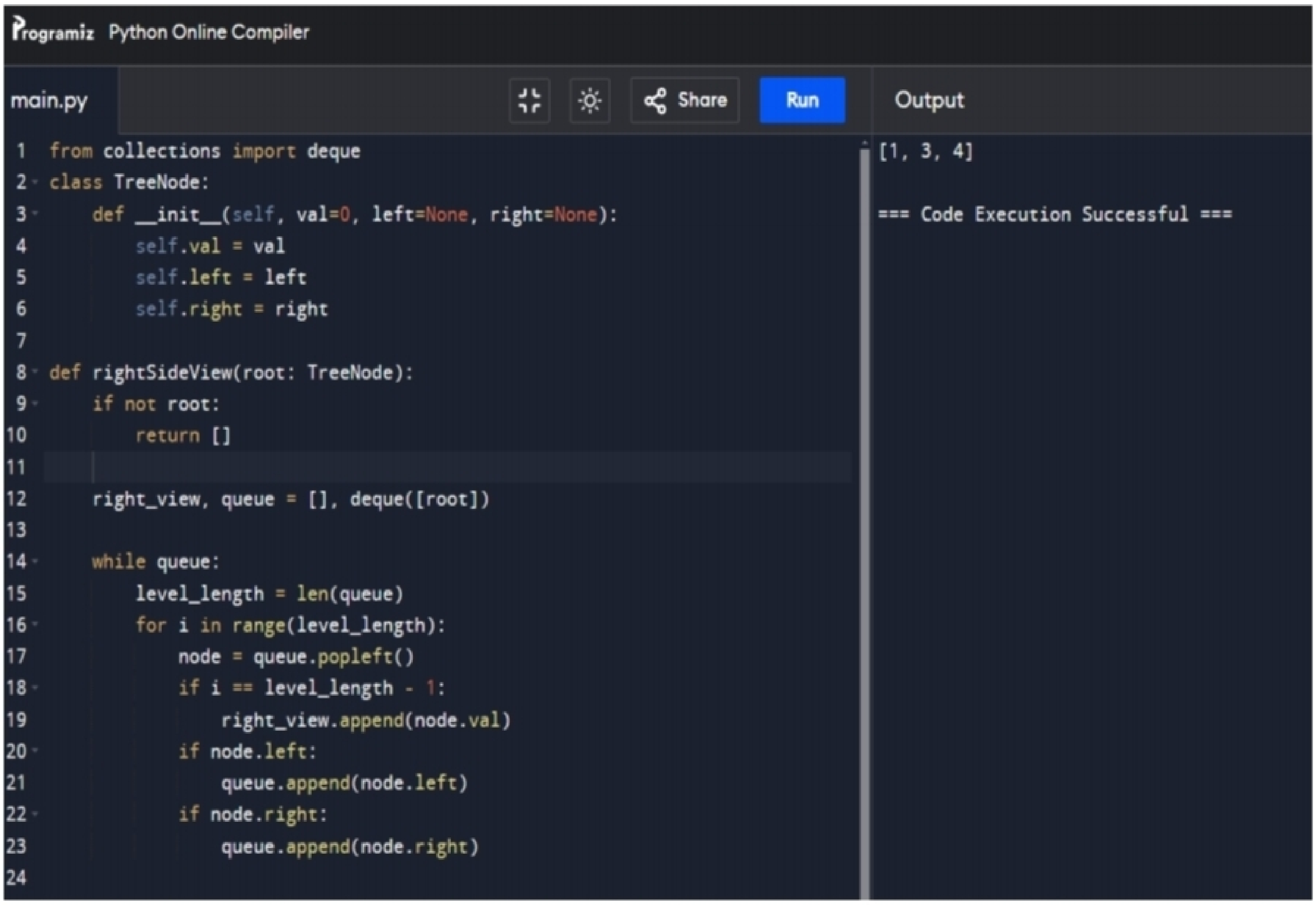
2.Given two integers dividend and divisor, divide two integers without using multiplication, division, and mod operator. The integer division should truncate toward zero, which means losing its fractional part. For example, 8.345 would be truncated to 8, and -2.7335 would be truncated to -2. Return the quotient after dividing dividend by divisor. Note: Assume we are dealing with an environment that could only store integers within the 32-bit signed integer range: [−231, 231 − 1]. For this problem, if the quotient is strictly greater than 231 - 1, then return 231 - 1, and if the quotient is strictly less than -231, then return -231.



```python
def divide(dividend: int, divisor: int) -> int:
    MAX_INT = 2**31 - 1
    MIN_INT = -2**31
    if dividend == MIN_INT and divisor == -1:
        return MAX_INT
    negative = (dividend < 0) != (divisor < 0)
    dividend = abs(dividend)
    divisor = abs(divisor)

    quotient = 0
    while dividend >= divisor:
        temp, multiple = divisor, 1
        while dividend >= (temp << 1):
            temp <<= 1
            multiple <<= 1
        dividend -= temp
        quotient += multiple

    if negative:
        quotient = -quotient
    return max(MIN_INT, min(MAX_INT, quotient))
dividend = 10
divisor = 3
print(divide(dividend, divisor)) # Output: 3
```

```
3

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

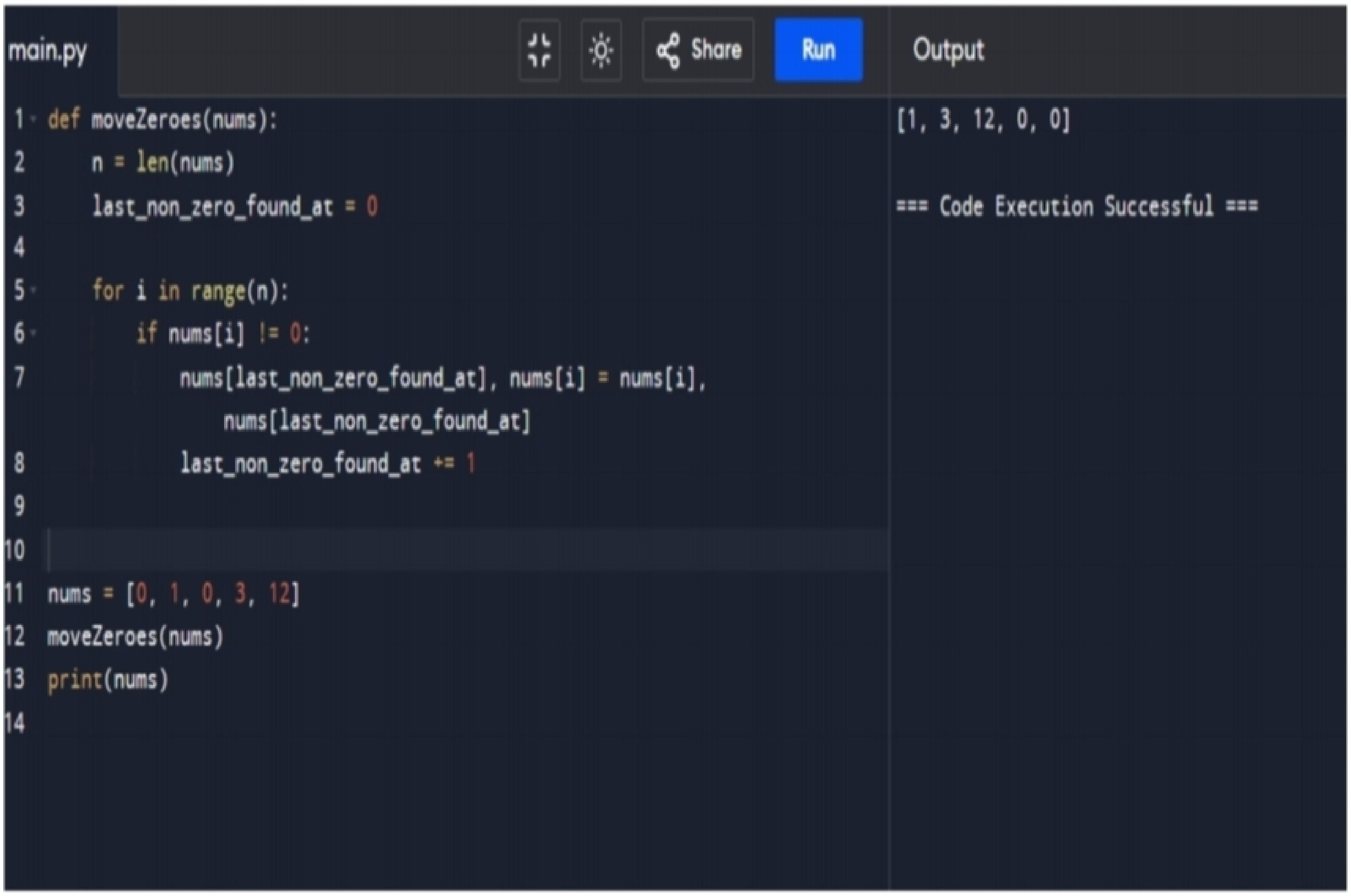
3. Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.



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

def rightSideView(root: TreeNode):
    if not root:
        return []

    right_view, queue = [], deque([root])

    while queue:
        level_length = len(queue)
        for i in range(level_length):
            node = queue.popleft()
            if i == level_length - 1:
                right_view.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

```

```
[1, 3, 4]

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

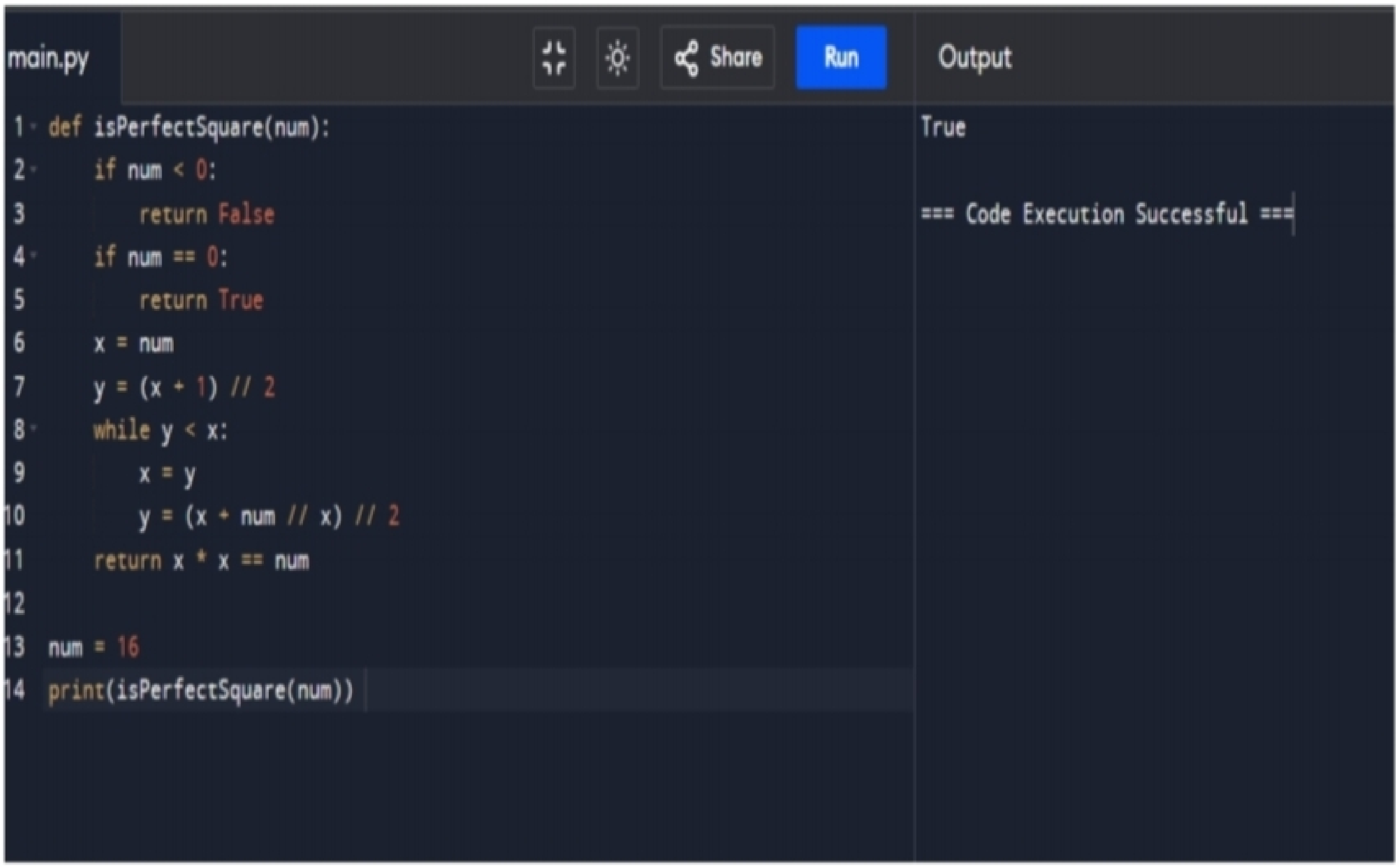
4. Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements. Note that you must do this in-place without making a copy of the array. Example 1: Input: nums = [0,1,0,3,12] Output: [1,3,12,0,0]



```python
def moveZeroes(nums):
    n = len(nums)
    last_non_zero_found_at = 0

    for i in range(n):
        if nums[i] != 0:
            nums[last_non_zero_found_at], nums[i] = nums[i],
                nums[last_non_zero_found_at]
            last_non_zero_found_at += 1


nums = [0, 1, 0, 3, 12]
moveZeroes(nums)
print(nums)
```

```
[1, 3, 12, 0, 0]

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

5. . Given a positive integer num, return true if num is a perfect square or false otherwise. A perfect square is an integer that is the square of an integer. In other words, it is the product of some integer with itself. You must not use any built-in library function, such as sqrt. Example 1: Input: num = 16 Output: true



```python
def isPerfectSquare(num):
    if num < 0:
        return False
    if num == 0:
        return True
    x = num
    y = (x + 1) // 2
    while y < x:
        x = y
        y = (x + num // x) // 2
    return x * x == num

num = 16
print(isPerfectSquare(num))
```

```
True

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