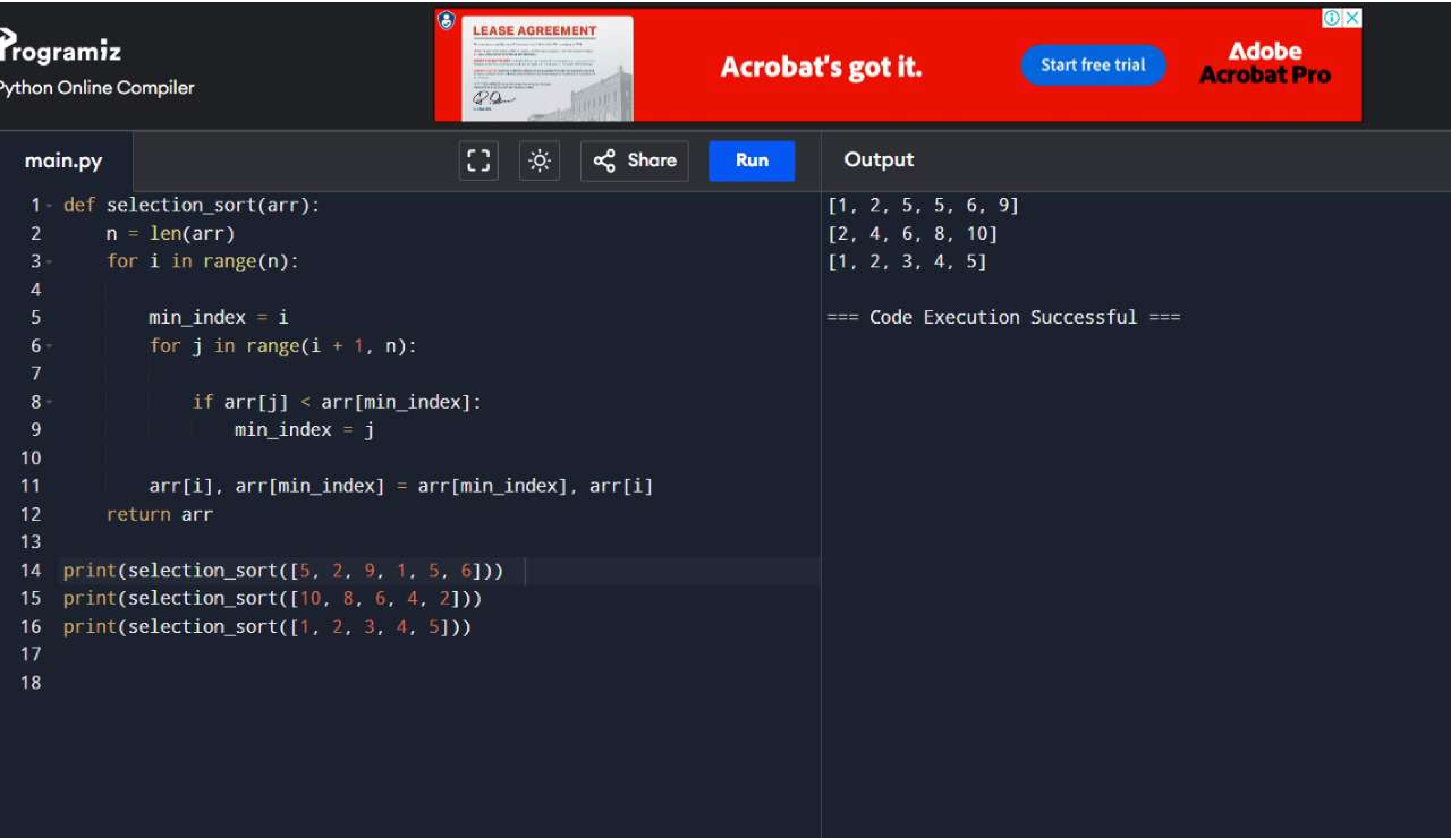
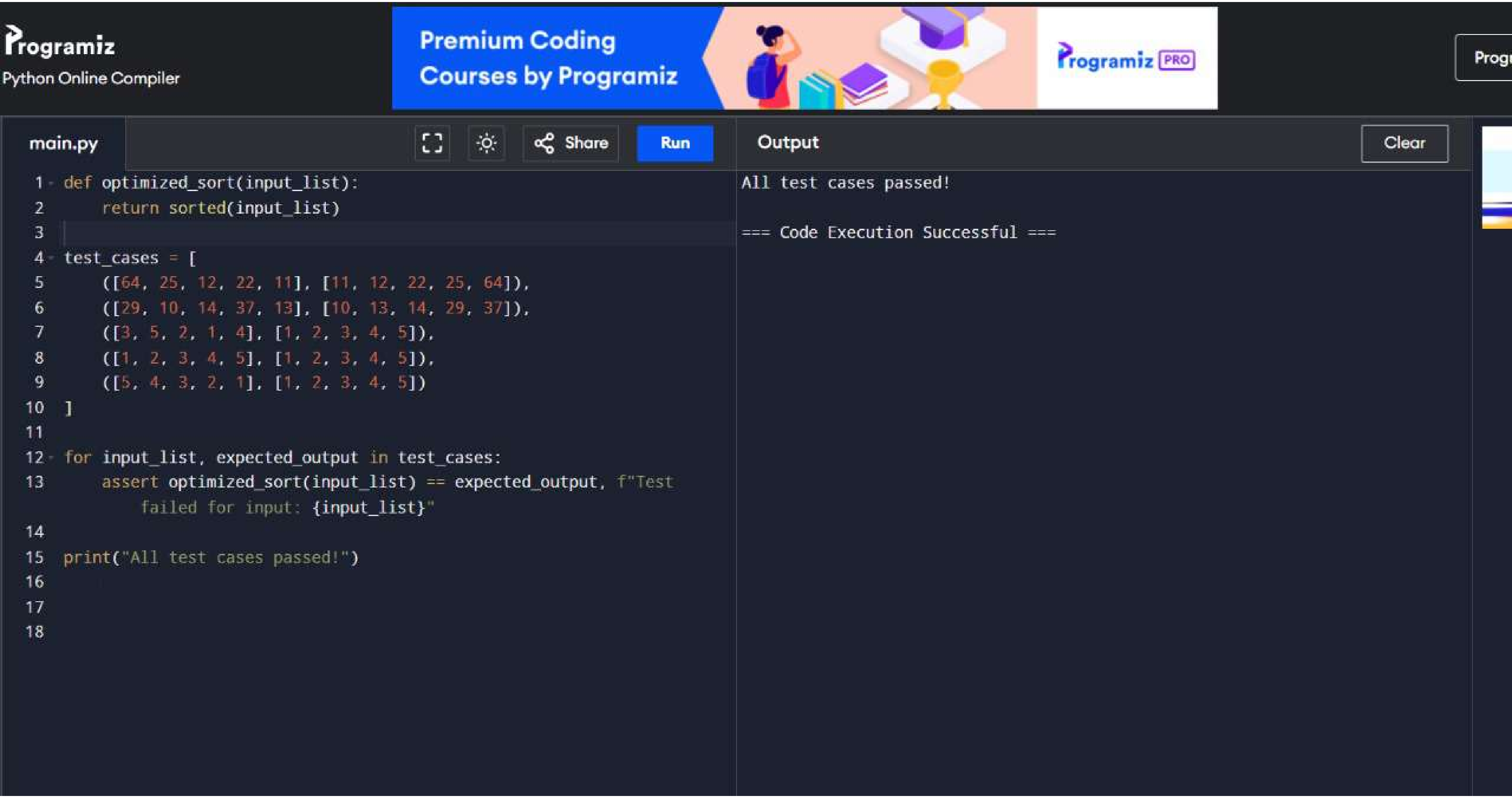


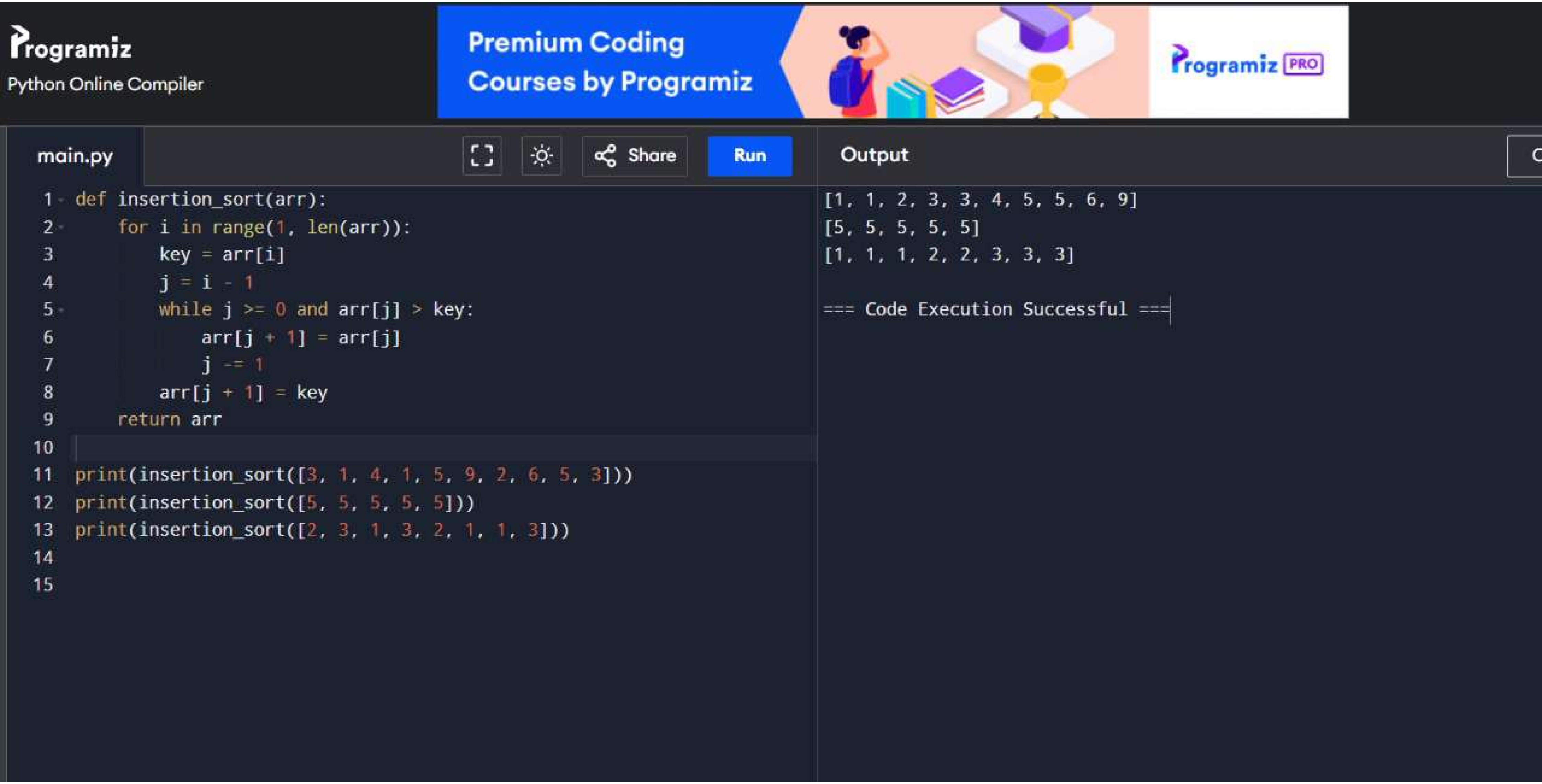
```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):

        min_index = i
        for j in range(i + 1, n):

            if arr[j] < arr[min_index]:
                min_index = j

        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr

print(selection_sort([5, 2, 9, 1, 5, 6]))
print(selection_sort([10, 8, 6, 4, 2]))
print(selection_sort([1, 2, 3, 4, 5]))
```

```
[1, 2, 5, 5, 6, 9]
[2, 4, 6, 8, 10]
[1, 2, 3, 4, 5]

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



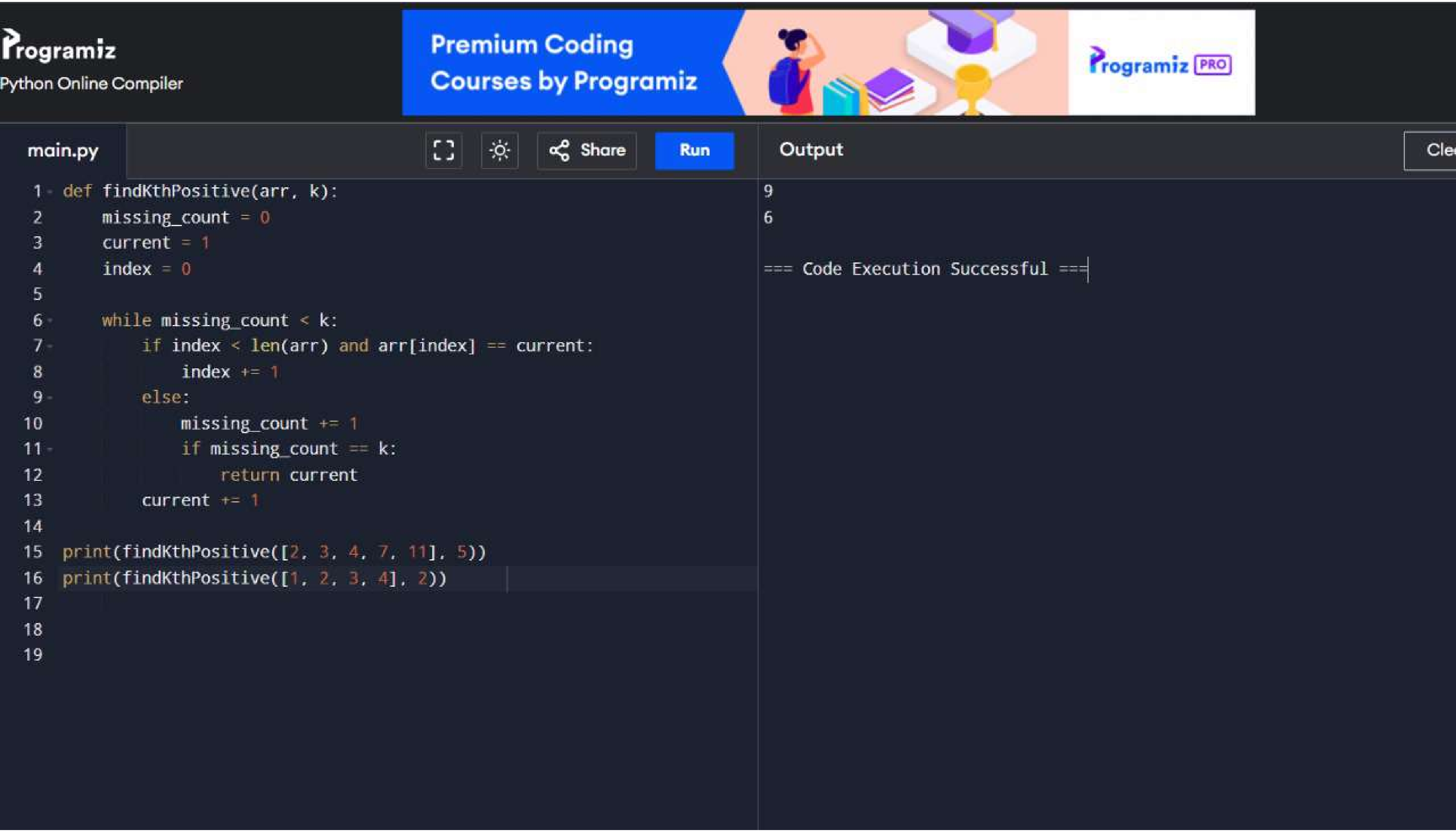
```python
def optimized_sort(input_list):
    return sorted(input_list)

test_cases = [
    ([64, 25, 12, 22, 11], [11, 12, 22, 25, 64]),
    ([29, 10, 14, 37, 13], [10, 13, 14, 29, 37]),
    ([3, 5, 2, 1, 4], [1, 2, 3, 4, 5]),
    ([1, 2, 3, 4, 5], [1, 2, 3, 4, 5]),
    ([5, 4, 3, 2, 1], [1, 2, 3, 4, 5])
]

for input_list, expected_output in test_cases:
    assert optimized_sort(input_list) == expected_output, f"Test failed for input: {input_list}"

print("All test cases passed!")
```

```
All test cases passed!

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



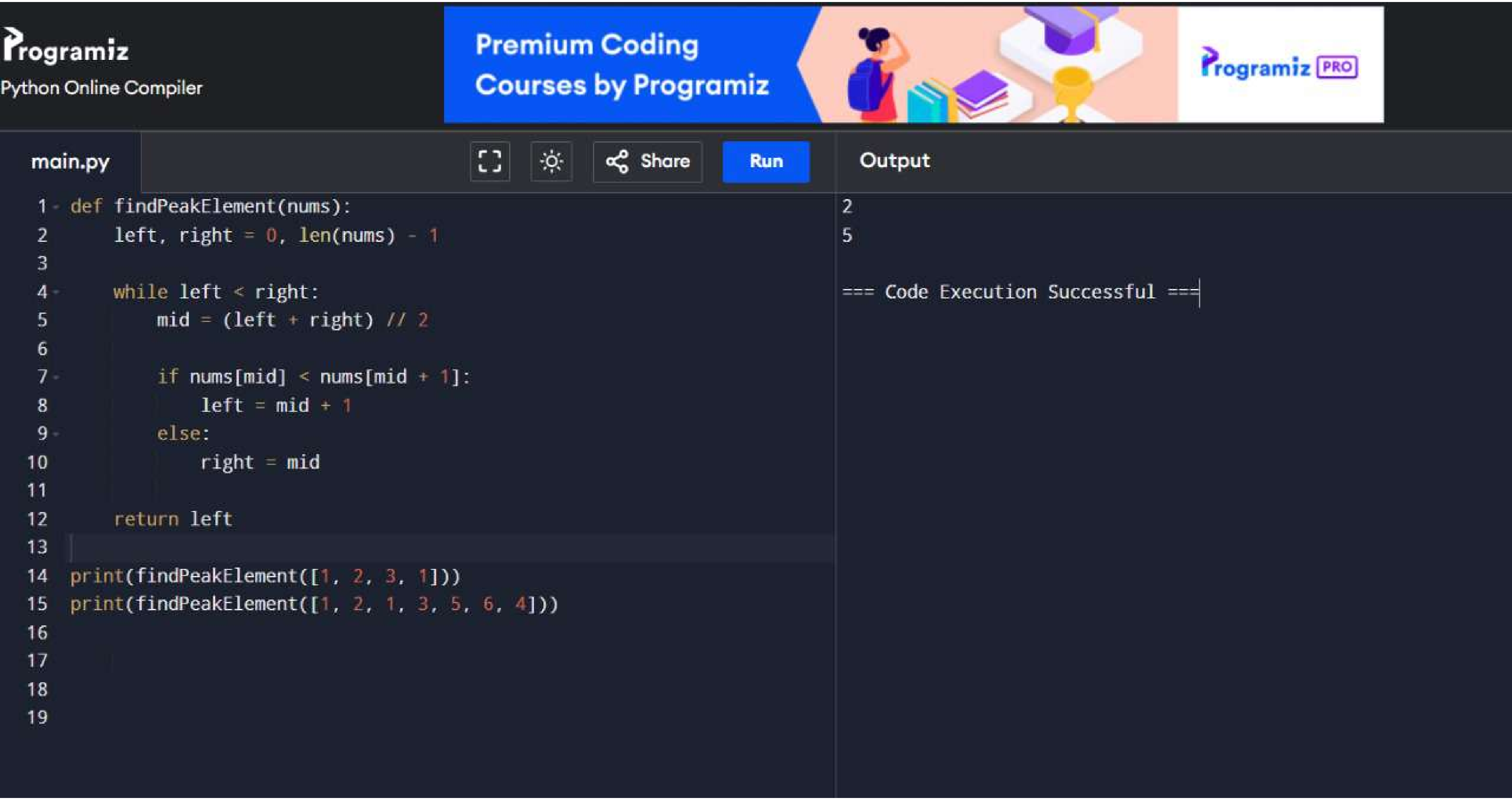
```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

print(insertion_sort([3, 1, 4, 1, 5, 9, 2, 6, 5, 3]))
print(insertion_sort([5, 5, 5, 5, 5]))
print(insertion_sort([2, 3, 1, 3, 2, 1, 1, 3]))
```

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

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



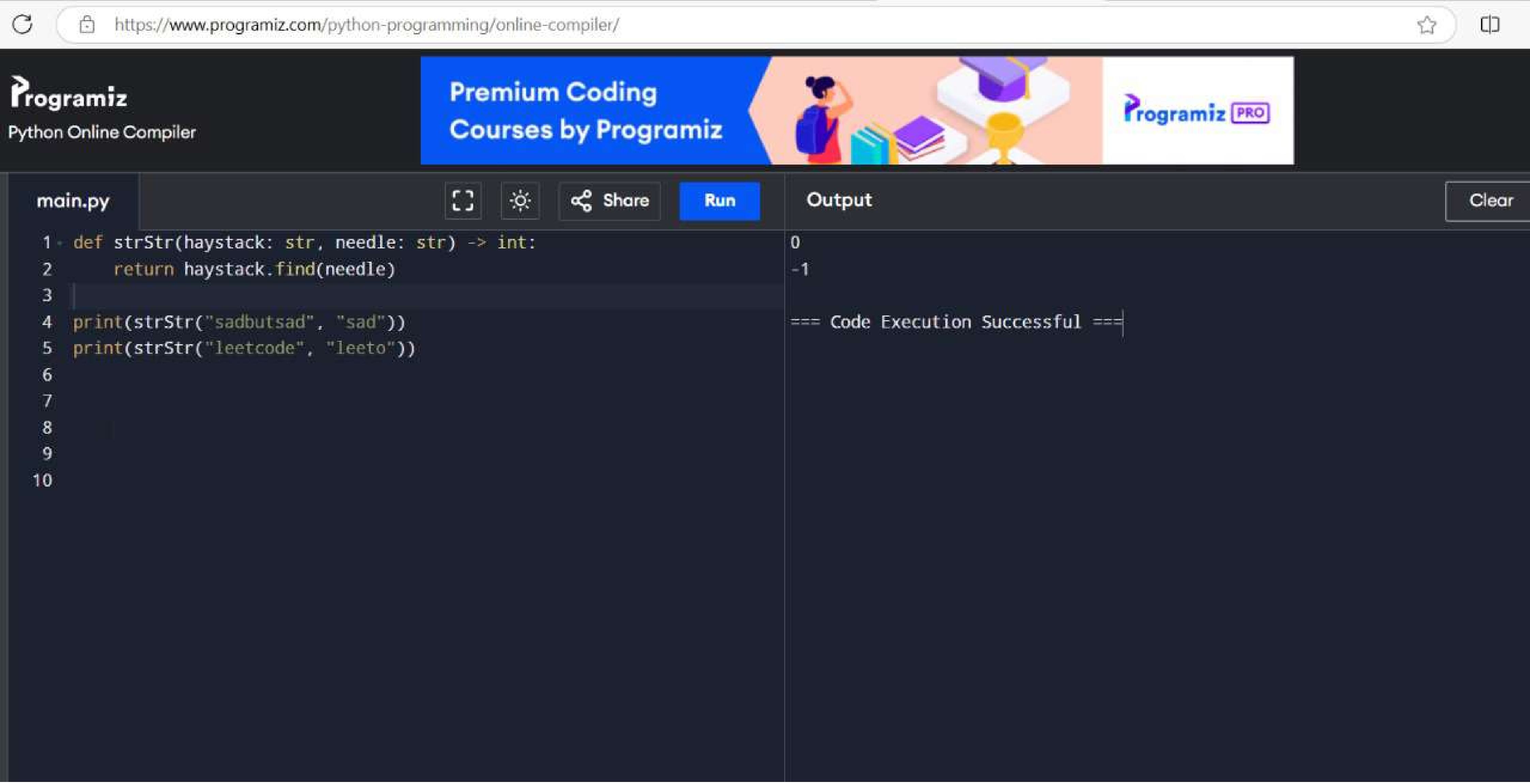
```python
def findKthPositive(arr, k):
    missing_count = 0
    current = 1
    index = 0

    while missing_count < k:
        if index < len(arr) and arr[index] == current:
            index += 1
        else:
            missing_count += 1
            if missing_count == k:
                return current
        current += 1

print(findKthPositive([2, 3, 4, 7, 11], 5))
print(findKthPositive([1, 2, 3, 4], 2))
```

```
9
6

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



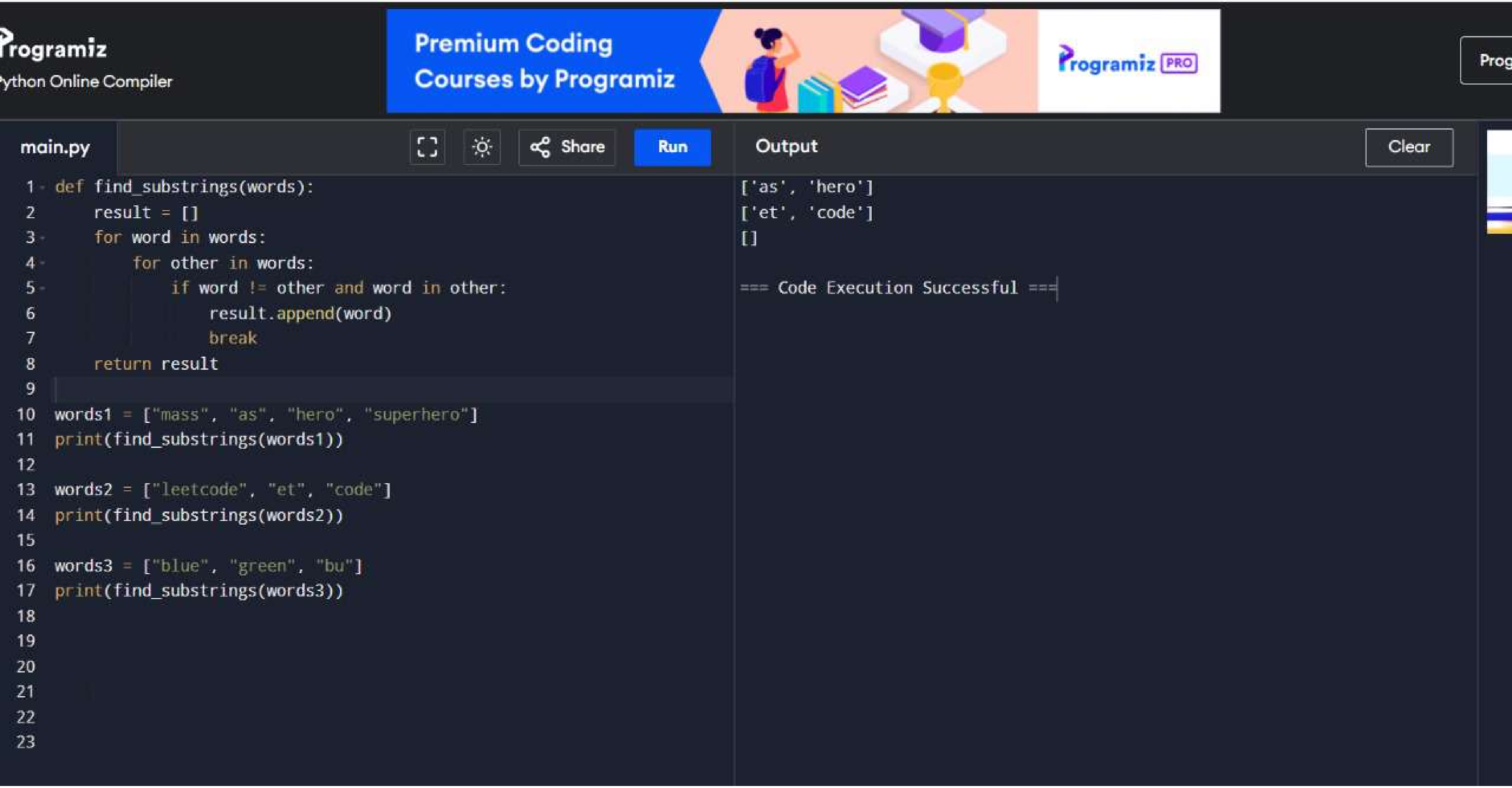
```python
def findPeakElement(nums):
    left, right = 0, len(nums) - 1

    while left < right:
        mid = (left + right) // 2

        if nums[mid] < nums[mid + 1]:
            left = mid + 1
        else:
            right = mid

    return left

print(findPeakElement([1, 2, 3, 1]))
print(findPeakElement([1, 2, 1, 3, 5, 6, 4]))
```

```
2
5

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



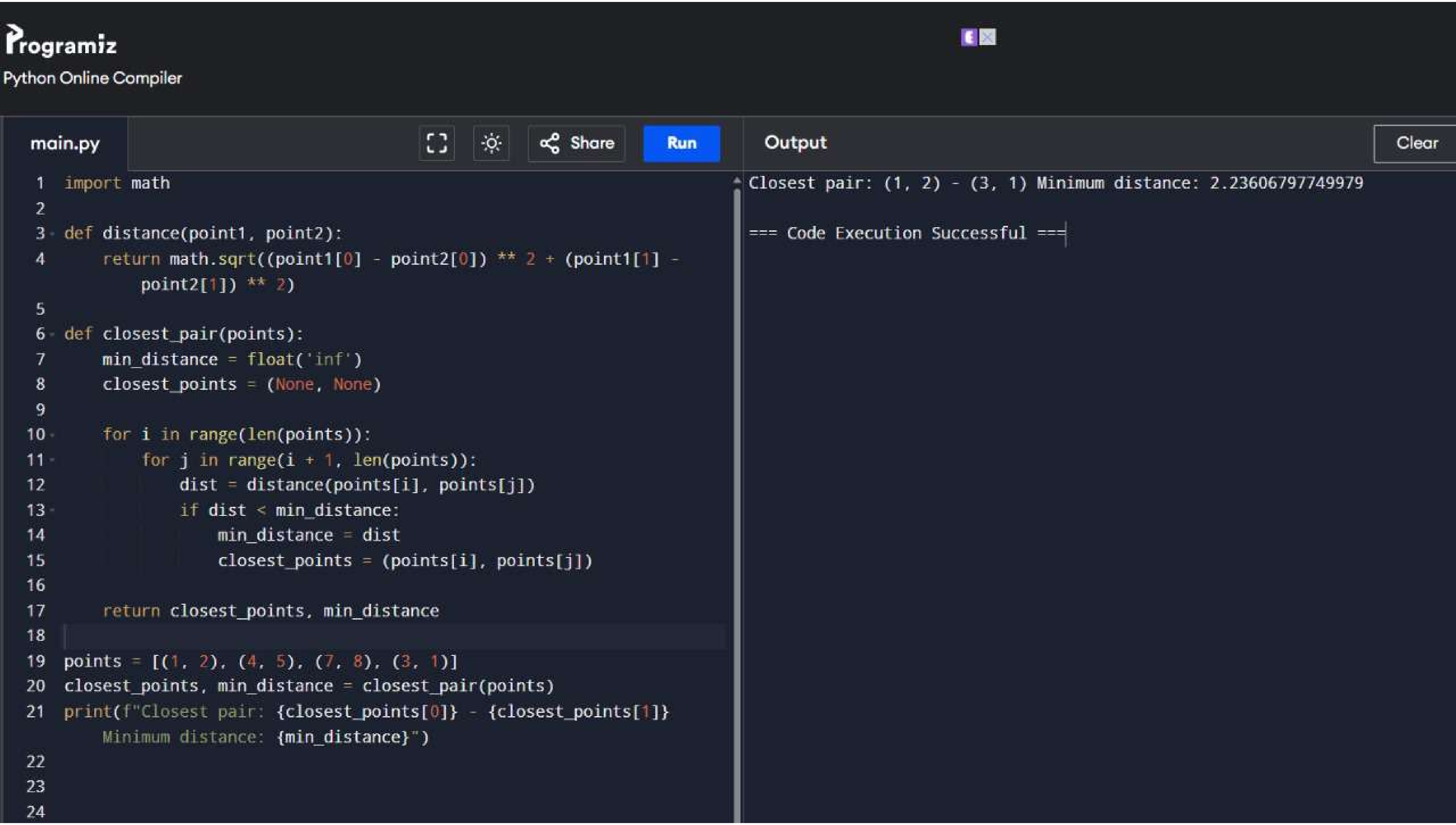
```python
def strStr(haystack: str, needle: str) -> int:
    return haystack.find(needle)

print(strStr("sadbutsad", "sad"))
print(strStr("leetcode", "leeto"))
```

```
0
-1

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



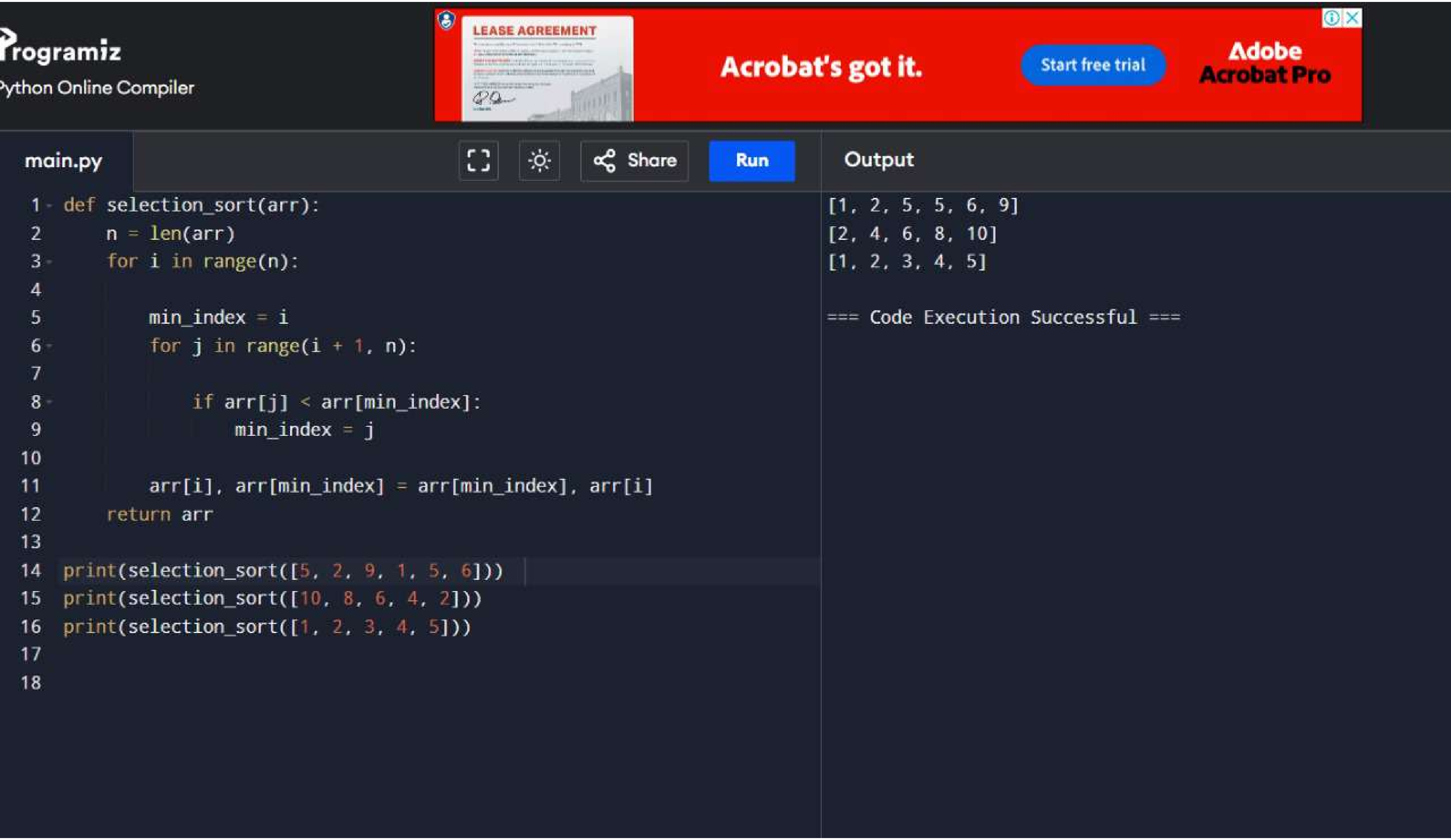
```python
def find_substrings(words):
    result = []
    for word in words:
        for other in words:
            if word != other and word in other:
                result.append(word)
                break
    return result

words1 = ["mass", "as", "hero", "superhero"]
print(find_substrings(words1))

words2 = ["leetcode", "et", "code"]
print(find_substrings(words2))

words3 = ["blue", "green", "bu"]
print(find_substrings(words3))
```

```
['as', 'hero']
['et', 'code']
[]

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



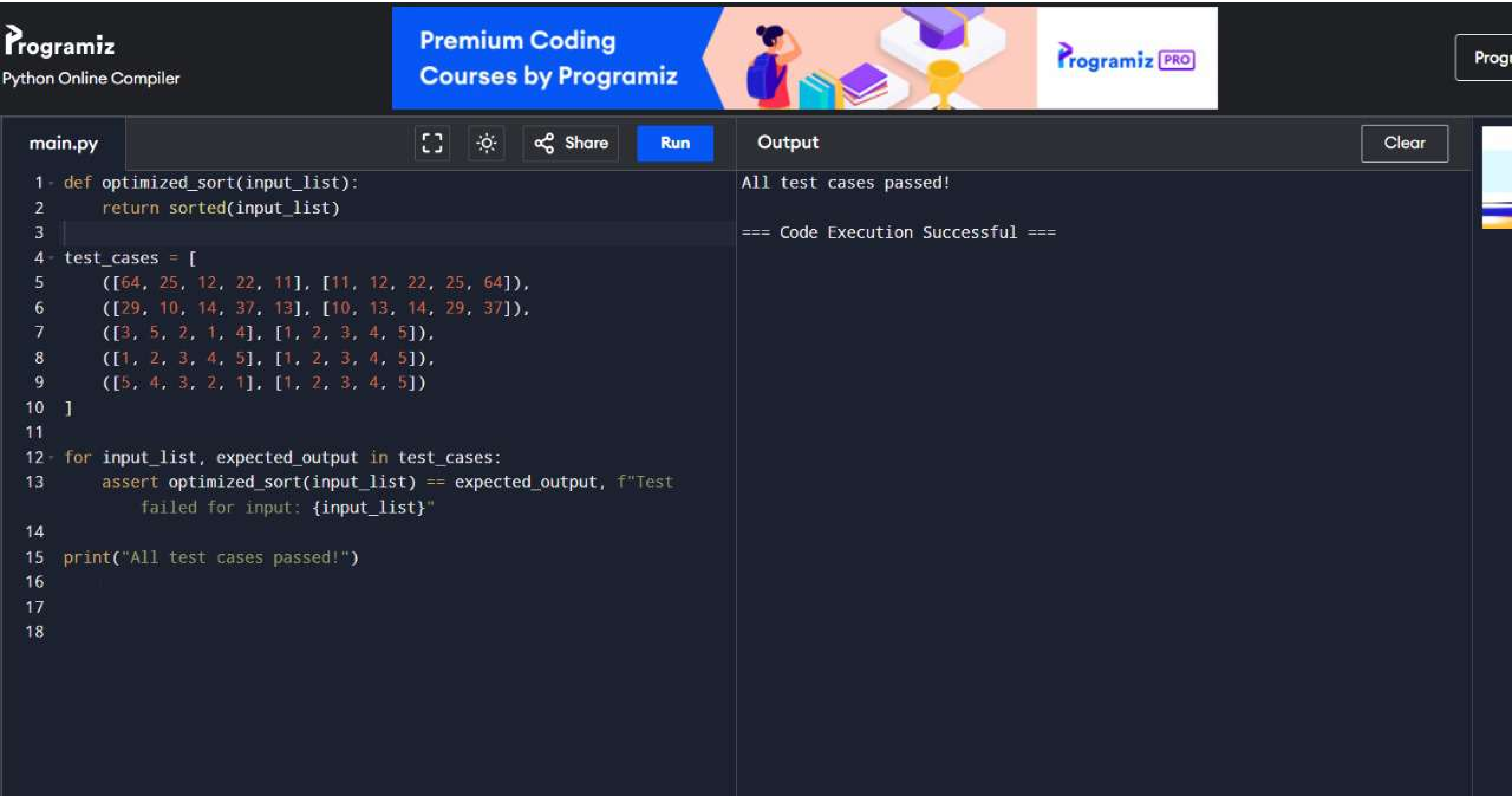
```python
import math

def distance(point1, point2):
    return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)

def closest_pair(points):
    min_distance = float('inf')
    closest_points = (None, None)

    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            dist = distance(points[i], points[j])
            if dist < min_distance:
                min_distance = dist
                closest_points = (points[i], points[j])

    return closest_points, min_distance

points = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_points, min_distance = closest_pair(points)
print(f"Closest pair: {closest_points[0]} - {closest_points[1]} Minimum distance: {min_distance}")
```

```
Closest pair: (1, 2) - (3, 1) Minimum distance: 2.23606797749979

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



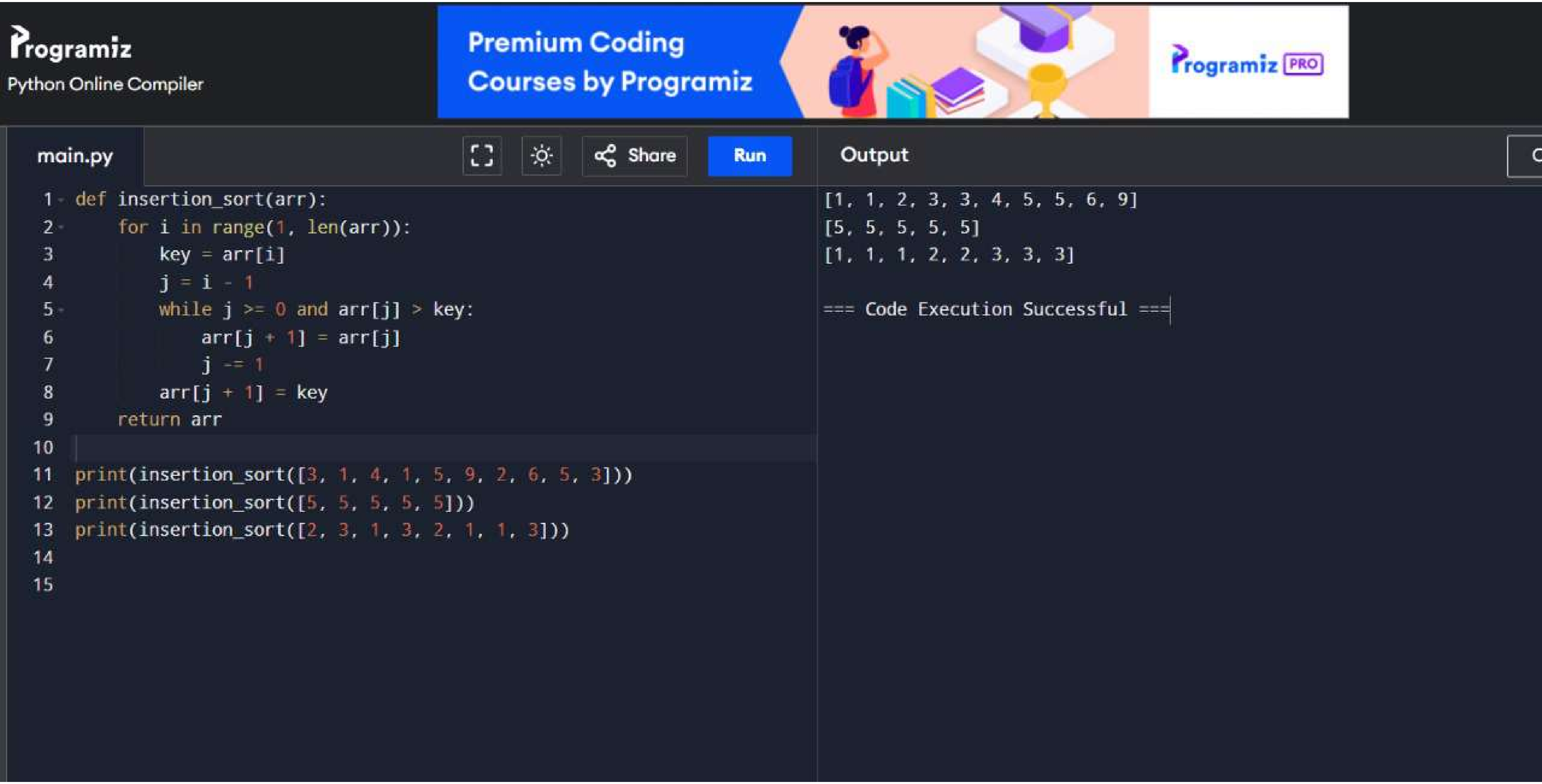
```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):

        min_index = i
        for j in range(i + 1, n):

            if arr[j] < arr[min_index]:
                min_index = j

        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr

print(selection_sort([5, 2, 9, 1, 5, 6]))
print(selection_sort([10, 8, 6, 4, 2]))
print(selection_sort([1, 2, 3, 4, 5]))
```

```
[1, 2, 5, 5, 6, 9]
[2, 4, 6, 8, 10]
[1, 2, 3, 4, 5]

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



```python
def optimized_sort(input_list):
    return sorted(input_list)

test_cases = [
    ([64, 25, 12, 22, 11], [11, 12, 22, 25, 64]),
    ([29, 10, 14, 37, 13], [10, 13, 14, 29, 37]),
    ([3, 5, 2, 1, 4], [1, 2, 3, 4, 5]),
    ([1, 2, 3, 4, 5], [1, 2, 3, 4, 5]),
    ([5, 4, 3, 2, 1], [1, 2, 3, 4, 5])
]

for input_list, expected_output in test_cases:
    assert optimized_sort(input_list) == expected_output, f"Test failed for input: {input_list}"

print("All test cases passed!")
```

```
All test cases passed!

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



```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

print(insertion_sort([3, 1, 4, 1, 5, 9, 2, 6, 5, 3]))
print(insertion_sort([5, 5, 5, 5, 5]))
print(insertion_sort([2, 3, 1, 3, 2, 1, 1, 3]))
```

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

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