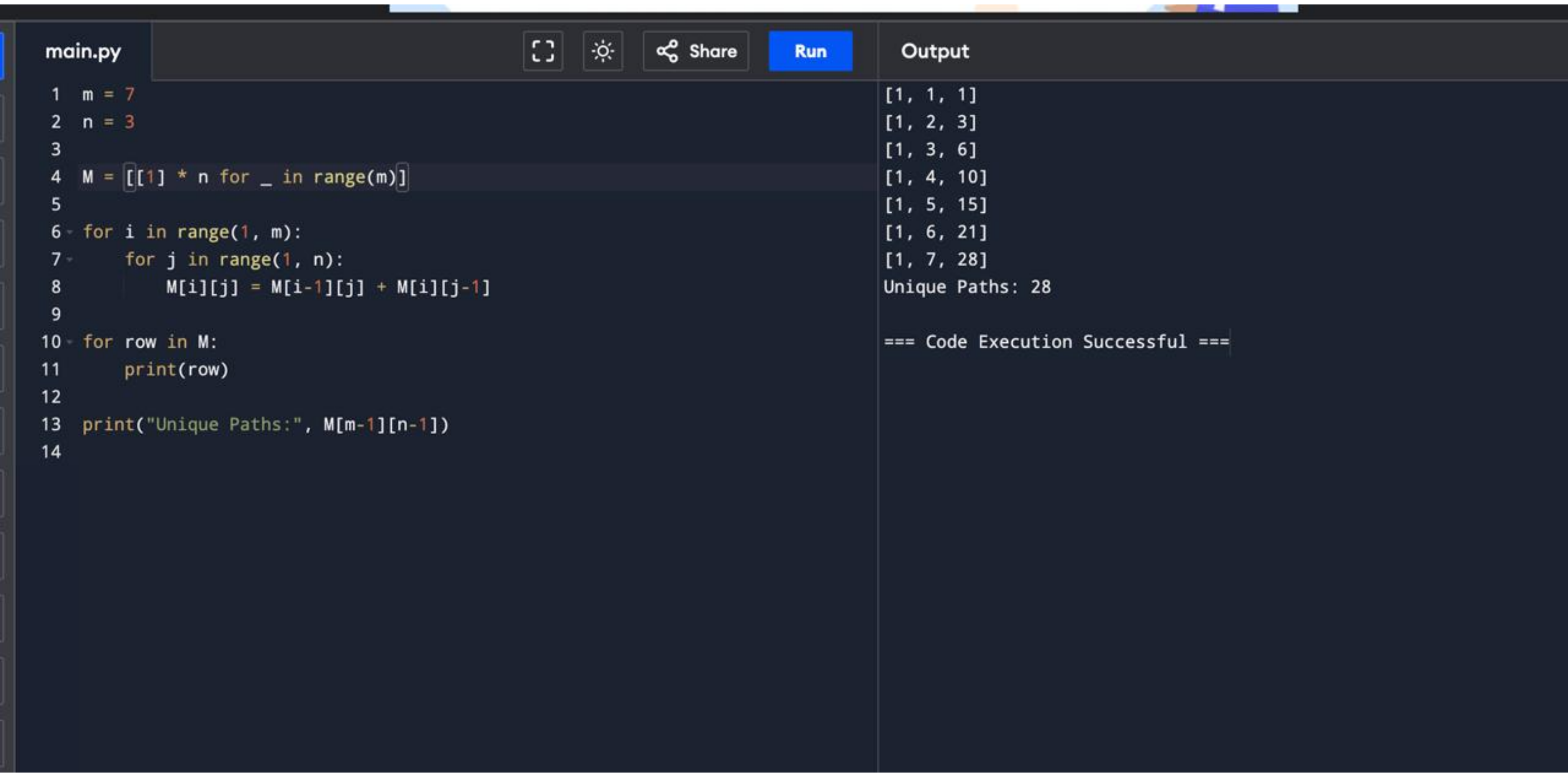
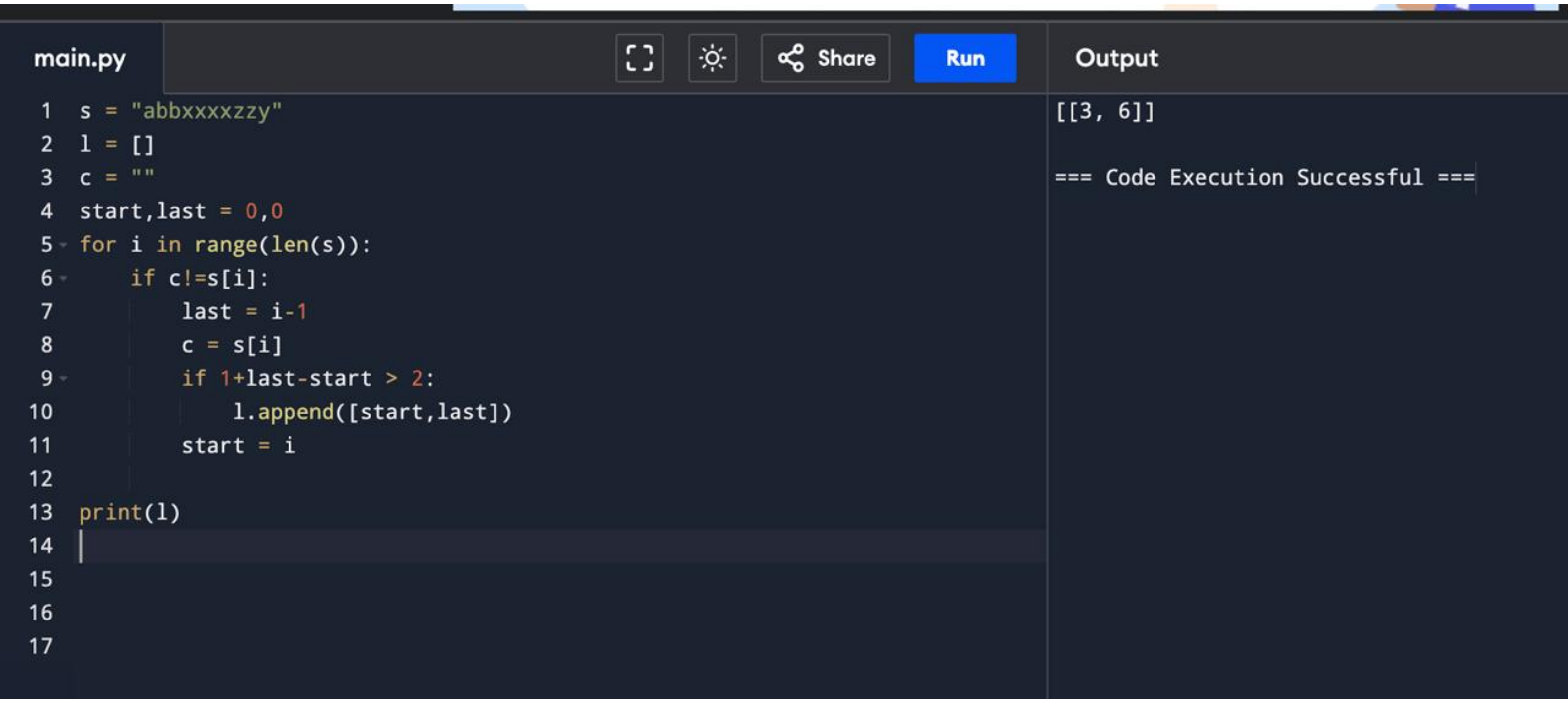


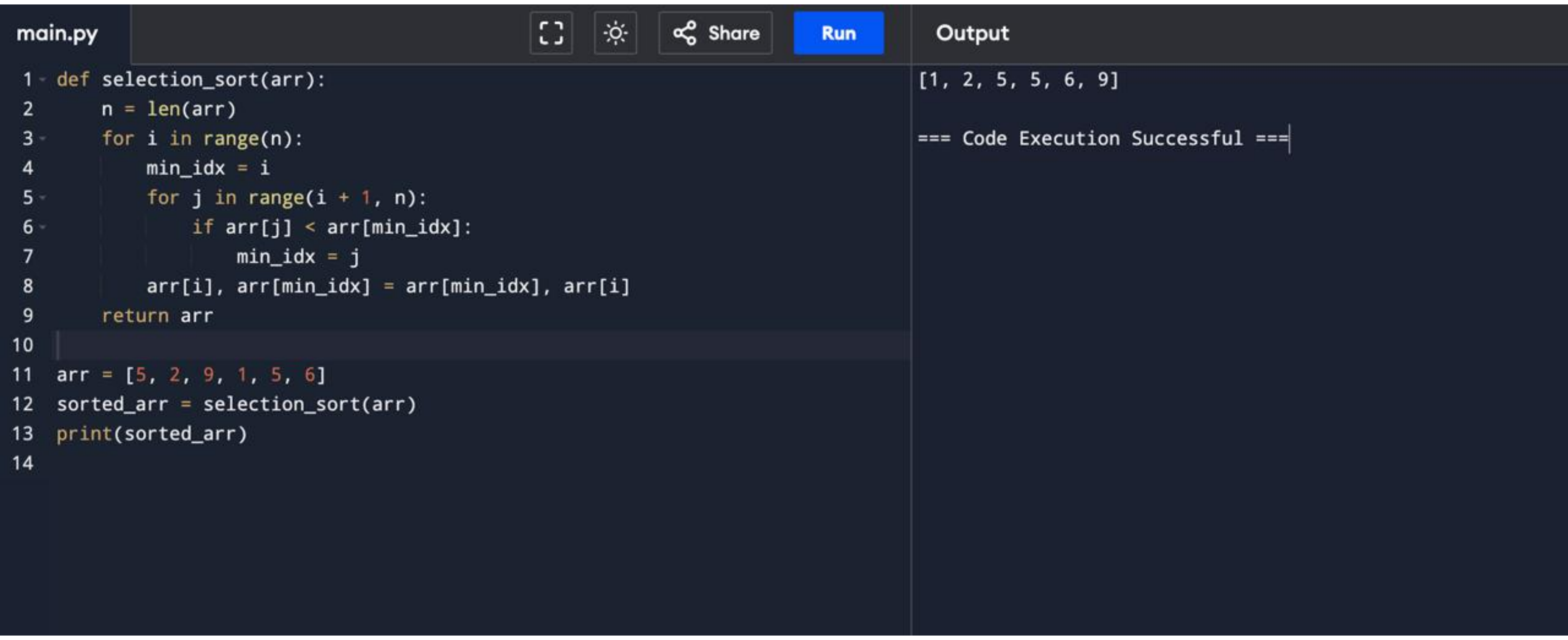
```python
m = 7
n = 3

M = [[1] * n for _ in range(m)]

for i in range(1, m):
    for j in range(1, n):
        M[i][j] = M[i-1][j] + M[i][j-1]

for row in M:
    print(row)

print("Unique Paths:", M[m-1][n-1])
```

```
[1, 1, 1]
[1, 2, 3]
[1, 3, 6]
[1, 4, 10]
[1, 5, 15]
[1, 6, 21]
[1, 7, 28]
Unique Paths: 28

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



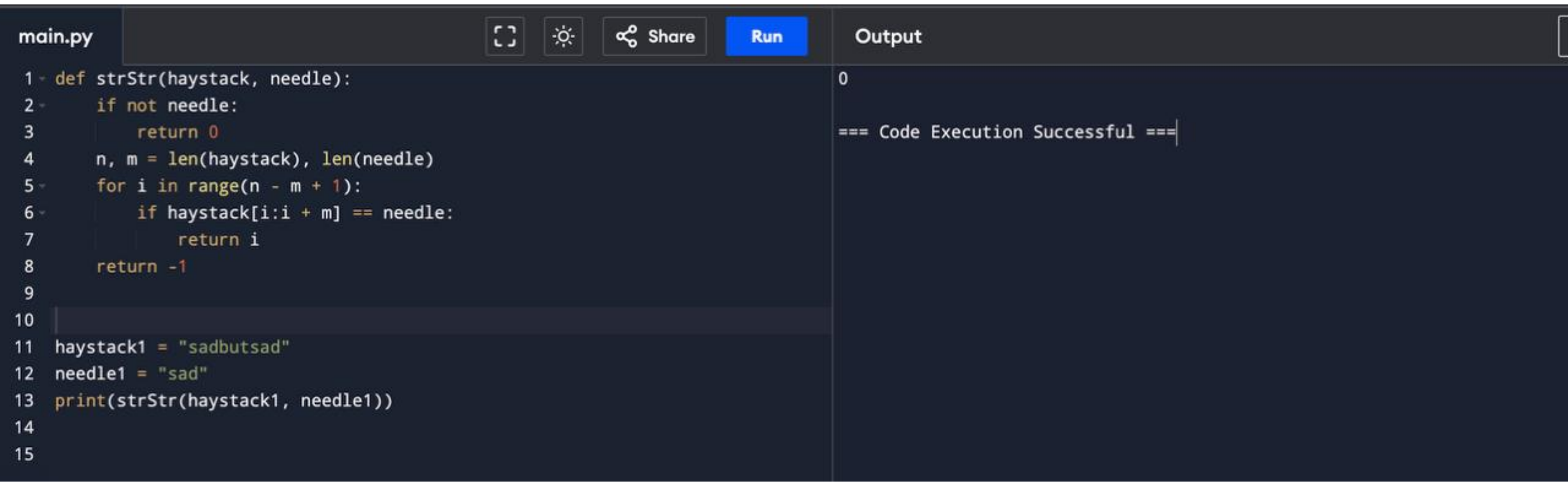
```python
s = "abbxxxxzzy"
l = []
c = ""
start,last = 0,0
for i in range(len(s)):
    if c!=s[i]:
        last = i-1
        c = s[i]
        if 1+last-start > 2:
            l.append([start,last])
        start = i

print(l)
```

```
[[3, 6]]

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



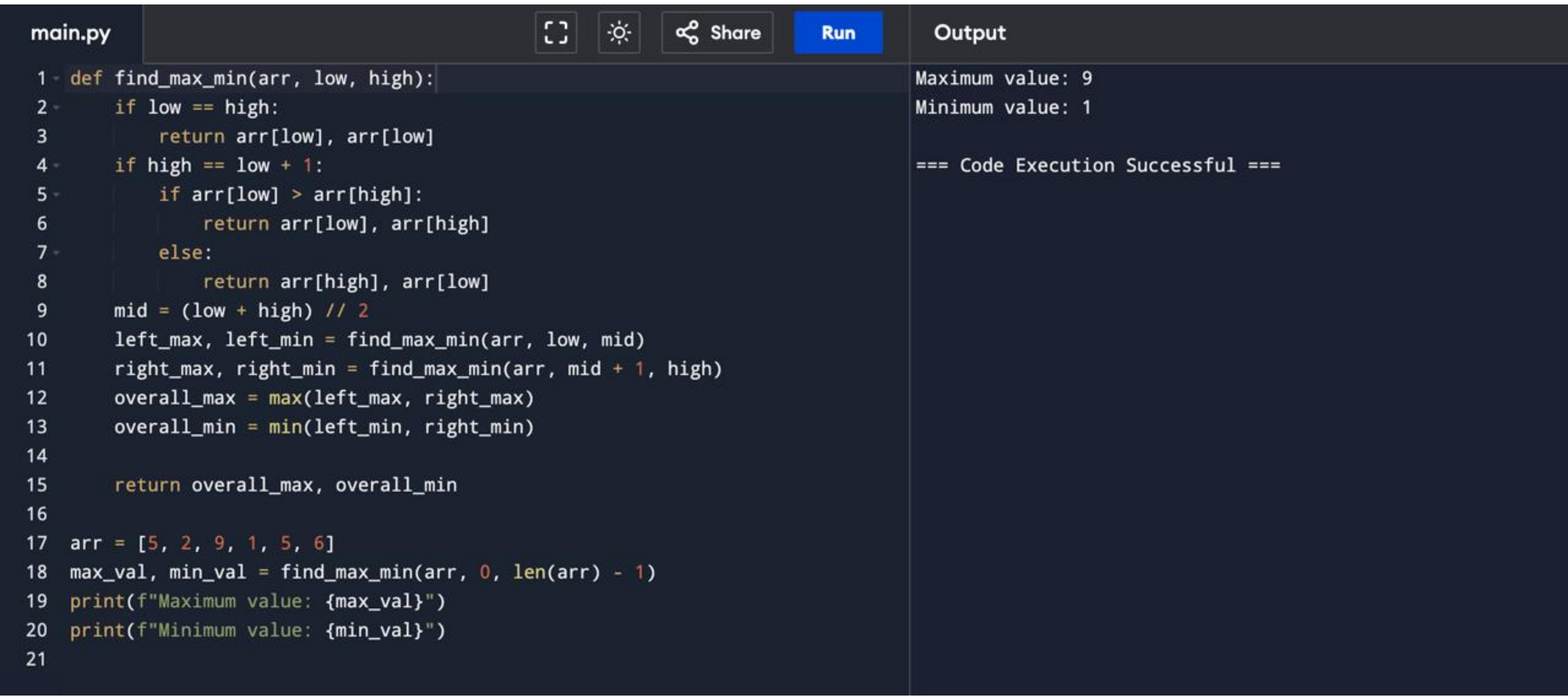
```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

arr = [5, 2, 9, 1, 5, 6]
sorted_arr = selection_sort(arr)
print(sorted_arr)
```

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

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



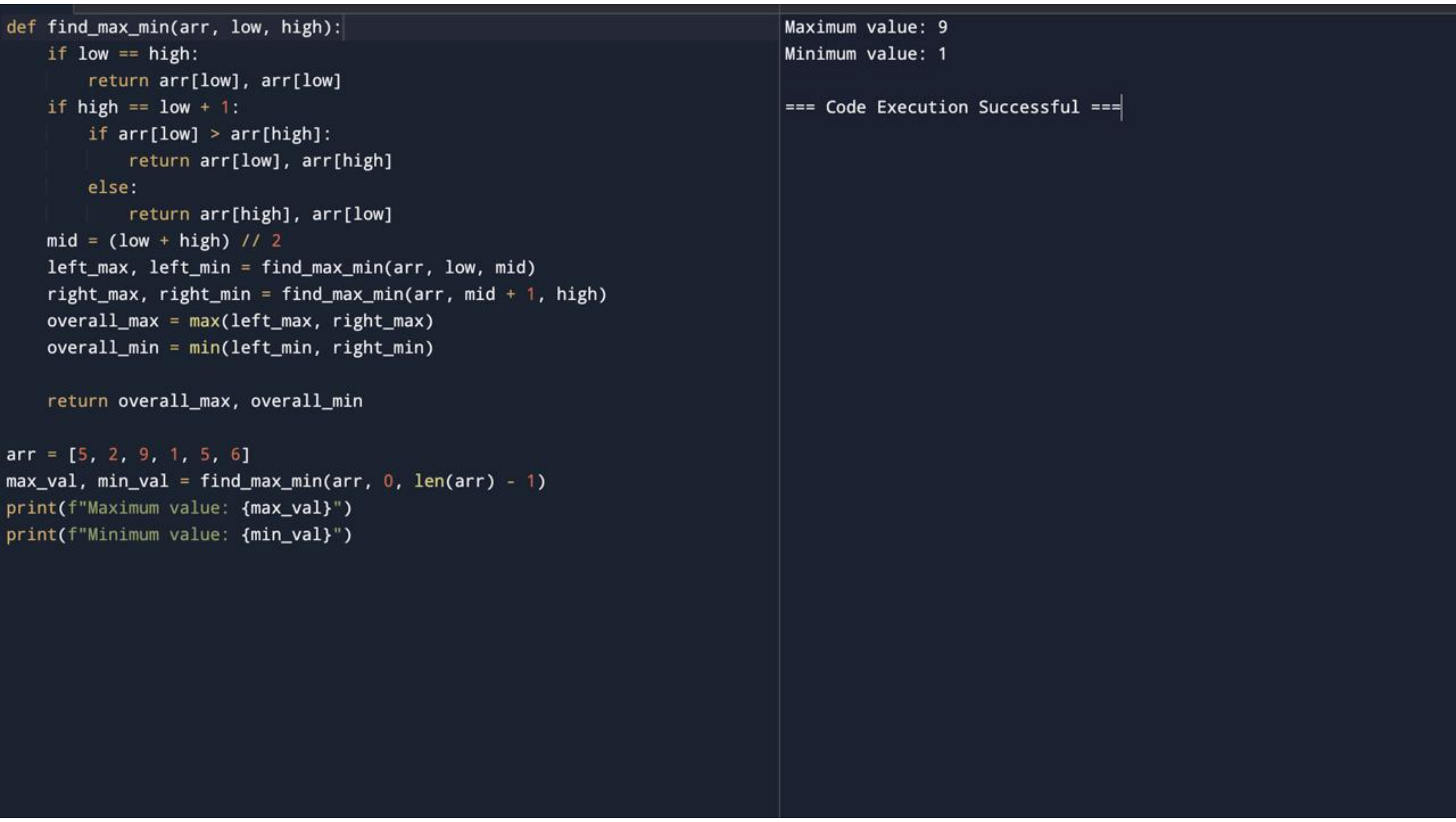
```python
def strStr(haystack, needle):
    if not needle:
        return 0
    n, m = len(haystack), len(needle)
    for i in range(n - m + 1):
        if haystack[i:i + m] == needle:
            return i
    return -1


haystack1 = "sadbutsad"
needle1 = "sad"
print(strStr(haystack1, needle1))
```

```
0

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



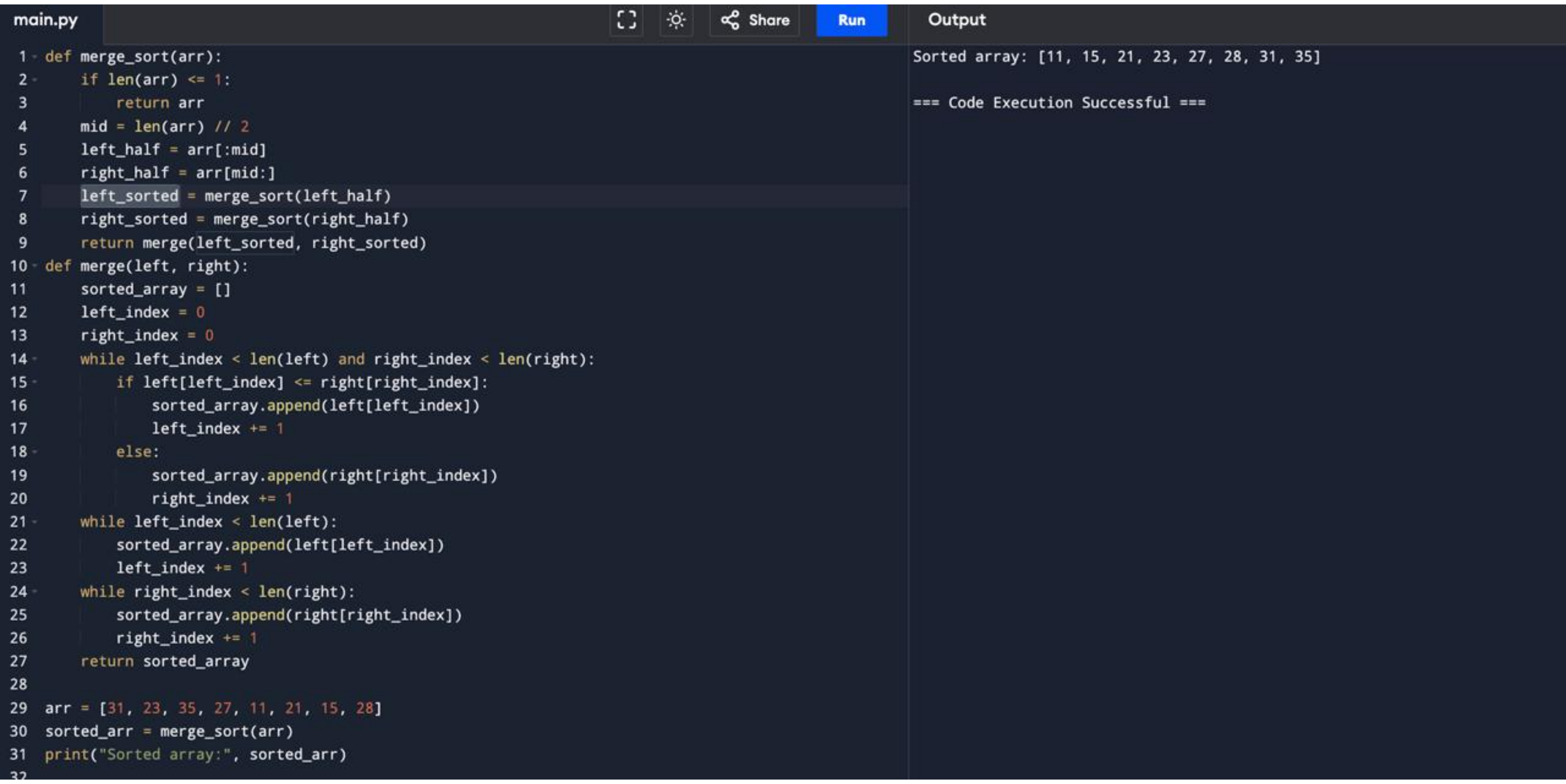
```python
def find_max_min(arr, low, high):
    if low == high:
        return arr[low], arr[low]
    if high == low + 1:
        if arr[low] > arr[high]:
            return arr[low], arr[high]
        else:
            return arr[high], arr[low]
    mid = (low + high) // 2
    left_max, left_min = find_max_min(arr, low, mid)
    right_max, right_min = find_max_min(arr, mid + 1, high)
    overall_max = max(left_max, right_max)
    overall_min = min(left_min, right_min)

    return overall_max, overall_min

arr = [5, 2, 9, 1, 5, 6]
max_val, min_val = find_max_min(arr, 0, len(arr) - 1)
print(f"Maximum value: {max_val}")
print(f"Minimum value: {min_val}")
```

```
Maximum value: 9
Minimum value: 1

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



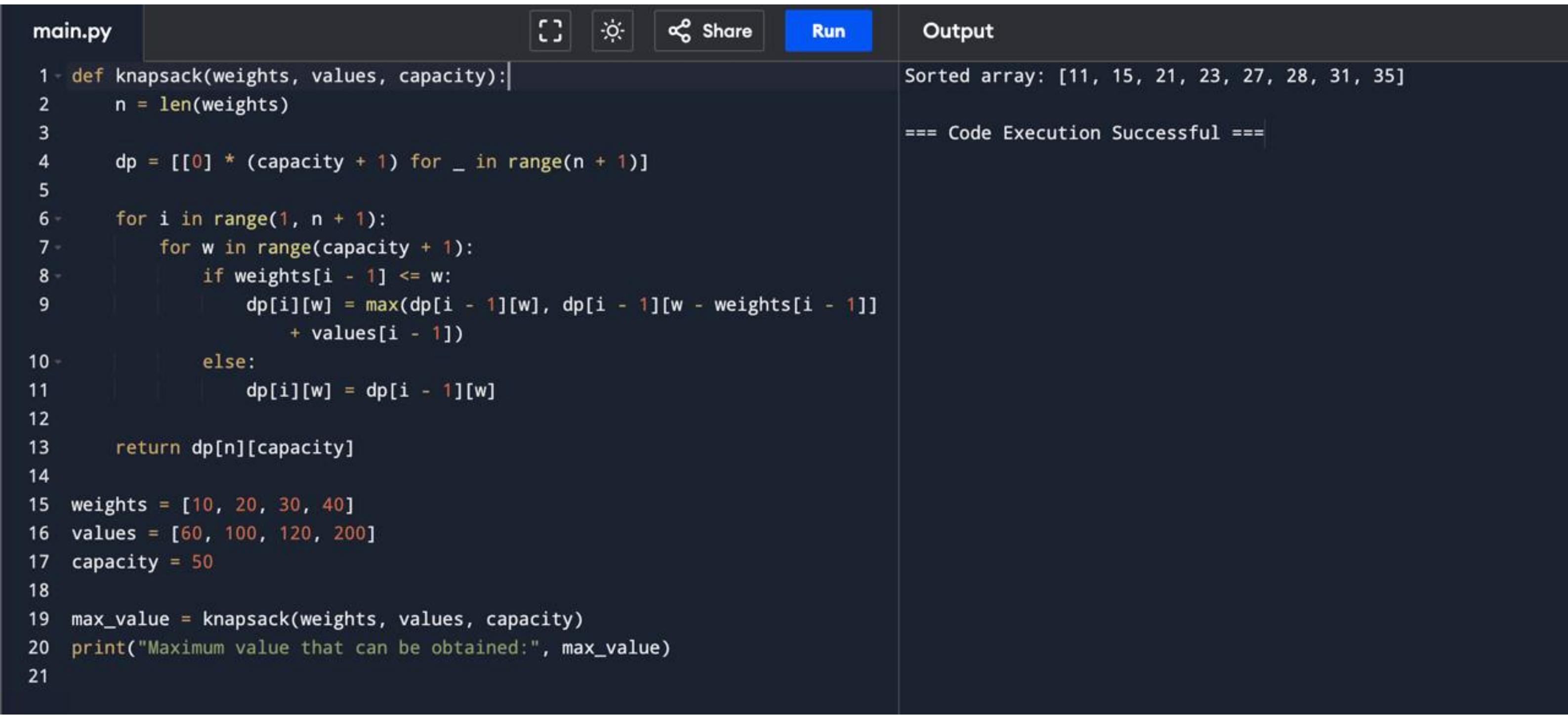
```
def find_max_min(arr, low, high):
    if low == high:
        return arr[low], arr[low]
    if high == low + 1:
        if arr[low] > arr[high]:
            return arr[low], arr[high]
        else:
            return arr[high], arr[low]
    mid = (low + high) // 2
    left_max, left_min = find_max_min(arr, low, mid)
    right_max, right_min = find_max_min(arr, mid + 1, high)
    overall_max = max(left_max, right_max)
    overall_min = min(left_min, right_min)

    return overall_max, overall_min

arr = [5, 2, 9, 1, 5, 6]
max_val, min_val = find_max_min(arr, 0, len(arr) - 1)
print(f"Maximum value: {max_val}")
print(f"Minimum value: {min_val}")
```

```
Maximum value: 9
Minimum value: 1

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



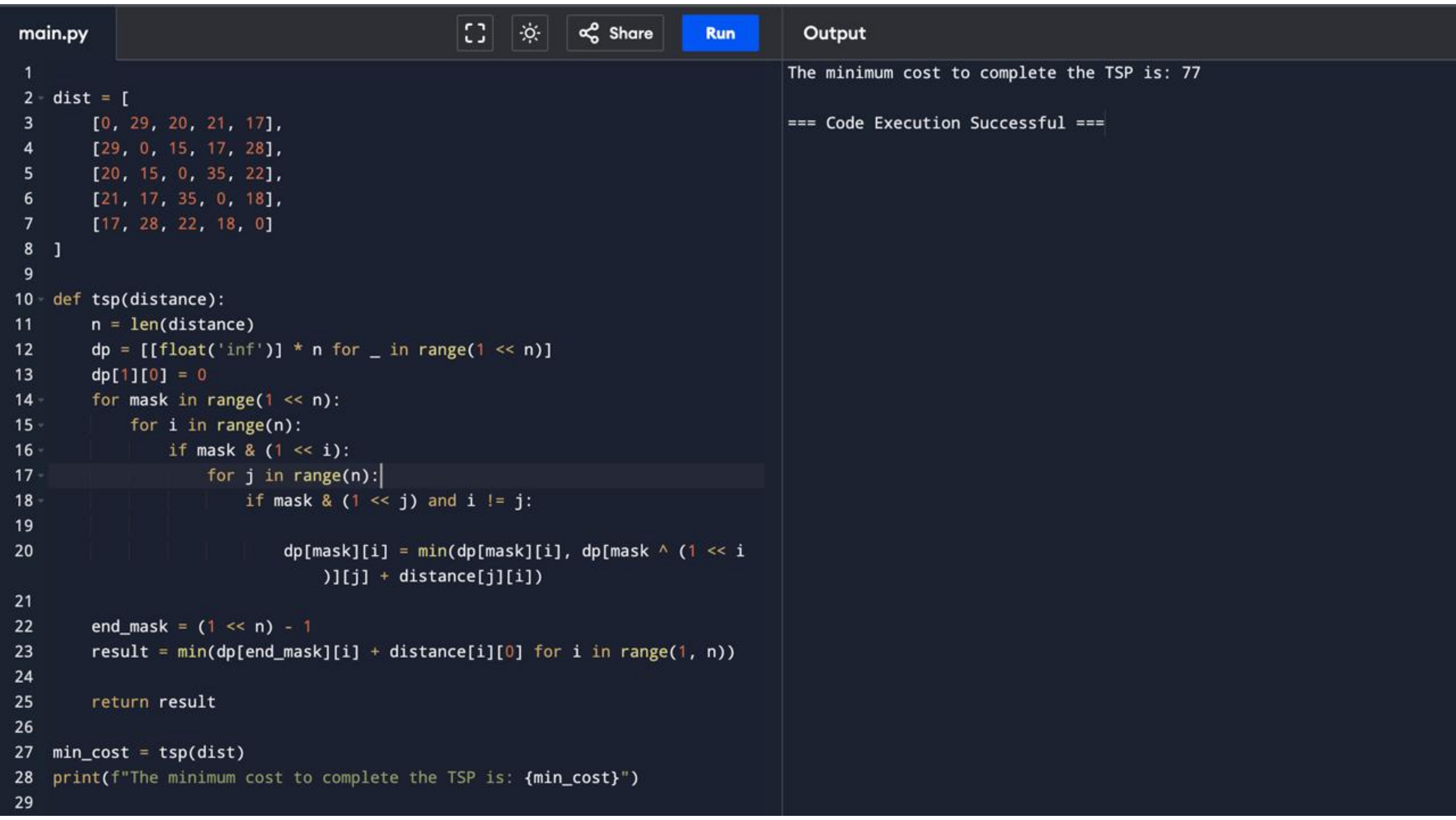
```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]
    left_sorted = merge_sort(left_half)
    right_sorted = merge_sort(right_half)
    return merge(left_sorted, right_sorted)
def merge(left, right):
    sorted_array = []
    left_index = 0
    right_index = 0
    while left_index < len(left) and right_index < len(right):
        if left[left_index] <= right[right_index]:
            sorted_array.append(left[left_index])
            left_index += 1
        else:
            sorted_array.append(right[right_index])
            right_index += 1
    while left_index < len(left):
        sorted_array.append(left[left_index])
        left_index += 1
    while right_index < len(right):
        sorted_array.append(right[right_index])
        right_index += 1
    return sorted_array

arr = [31, 23, 35, 27, 11, 21, 15, 28]
sorted_arr = merge_sort(arr)
print("Sorted array:", sorted_arr)
```

```
Sorted array: [11, 15, 21, 23, 27, 28, 31, 35]

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



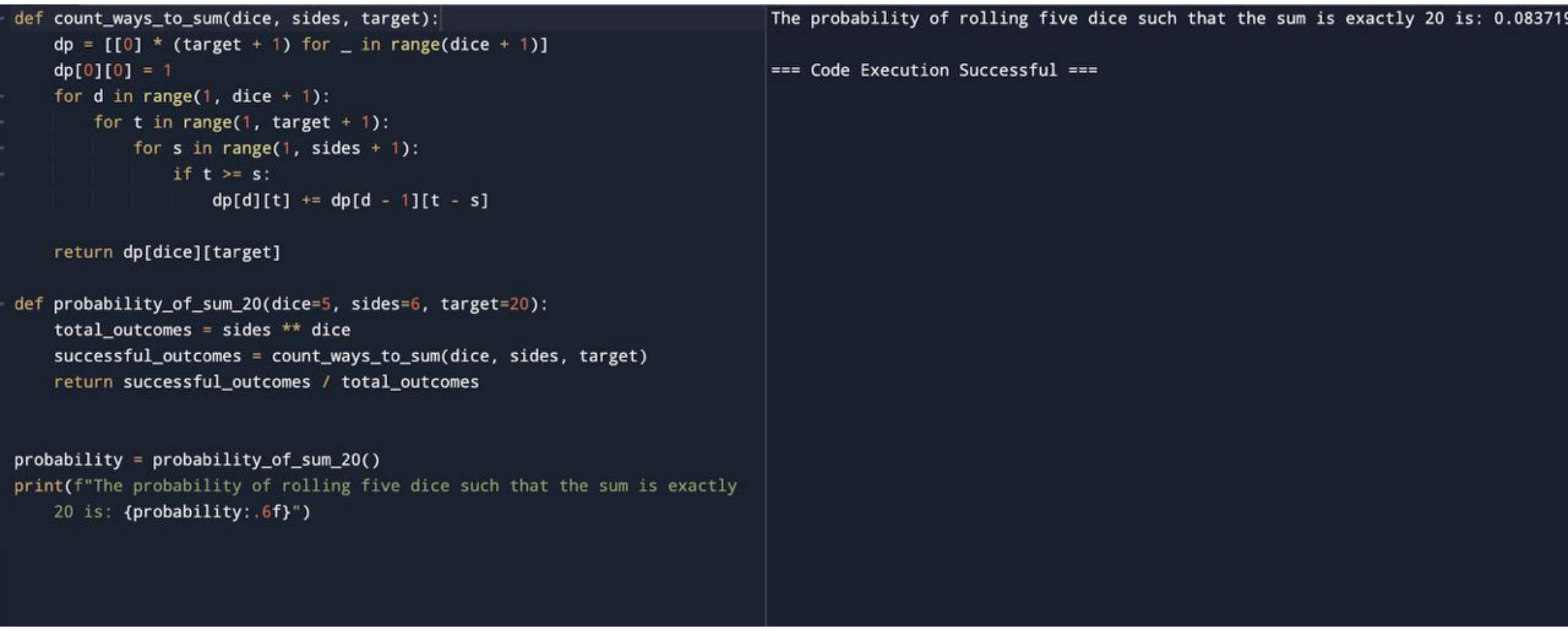
```python
def knapsack(weights, values, capacity):
    n = len(weights)

    dp = [[0] * (capacity + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(capacity + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]]
                    + values[i - 1])
            else:
                dp[i][w] = dp[i - 1][w]

    return dp[n][capacity]

weights = [10, 20, 30, 40]
values = [60, 100, 120, 200]
capacity = 50

max_value = knapsack(weights, values, capacity)
print("Maximum value that can be obtained:", max_value)
```

```
Sorted array: [11, 15, 21, 23, 27, 28, 31, 35]

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



```python

dist = [
    [0, 29, 20, 21, 17],
    [29, 0, 15, 17, 28],
    [20, 15, 0, 35, 22],
    [21, 17, 35, 0, 18],
    [17, 28, 22, 18, 0]
]

def tsp(distance):
    n = len(distance)
    dp = [[float('inf')] * n for _ in range(1 << n)]
    dp[1][0] = 0
    for mask in range(1 << n):
        for i in range(n):
            if mask & (1 << i):
                for j in range(n):
                    if mask & (1 << j) and i != j:

                        dp[mask][i] = min(dp[mask][i], dp[mask ^ (1 << i)][j] + distance[j][i])

    end_mask = (1 << n) - 1
    result = min(dp[end_mask][i] + distance[i][0] for i in range(1, n))

    return result

min_cost = tsp(dist)
print(f"The minimum cost to complete the TSP is: {min_cost}")

```

```
The minimum cost to complete the TSP is: 77

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



```python
def count_ways_to_sum(dice, sides, target):
    dp = [[0] * (target + 1) for _ in range(dice + 1)]
    dp[0][0] = 1
    for d in range(1, dice + 1):
        for t in range(1, target + 1):
            for s in range(1, sides + 1):
                if t >= s:
                    dp[d][t] += dp[d - 1][t - s]

    return dp[dice][target]

def probability_of_sum_20(dice=5, sides=6, target=20):
    total_outcomes = sides ** dice
    successful_outcomes = count_ways_to_sum(dice, sides, target)
    return successful_outcomes / total_outcomes


probability = probability_of_sum_20()
print(f"The probability of rolling five dice such that the sum is exactly
    20 is: {probability:.6f}")
```

```
The probability of rolling five dice such that the sum is exactly 20 is: 0.083719

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