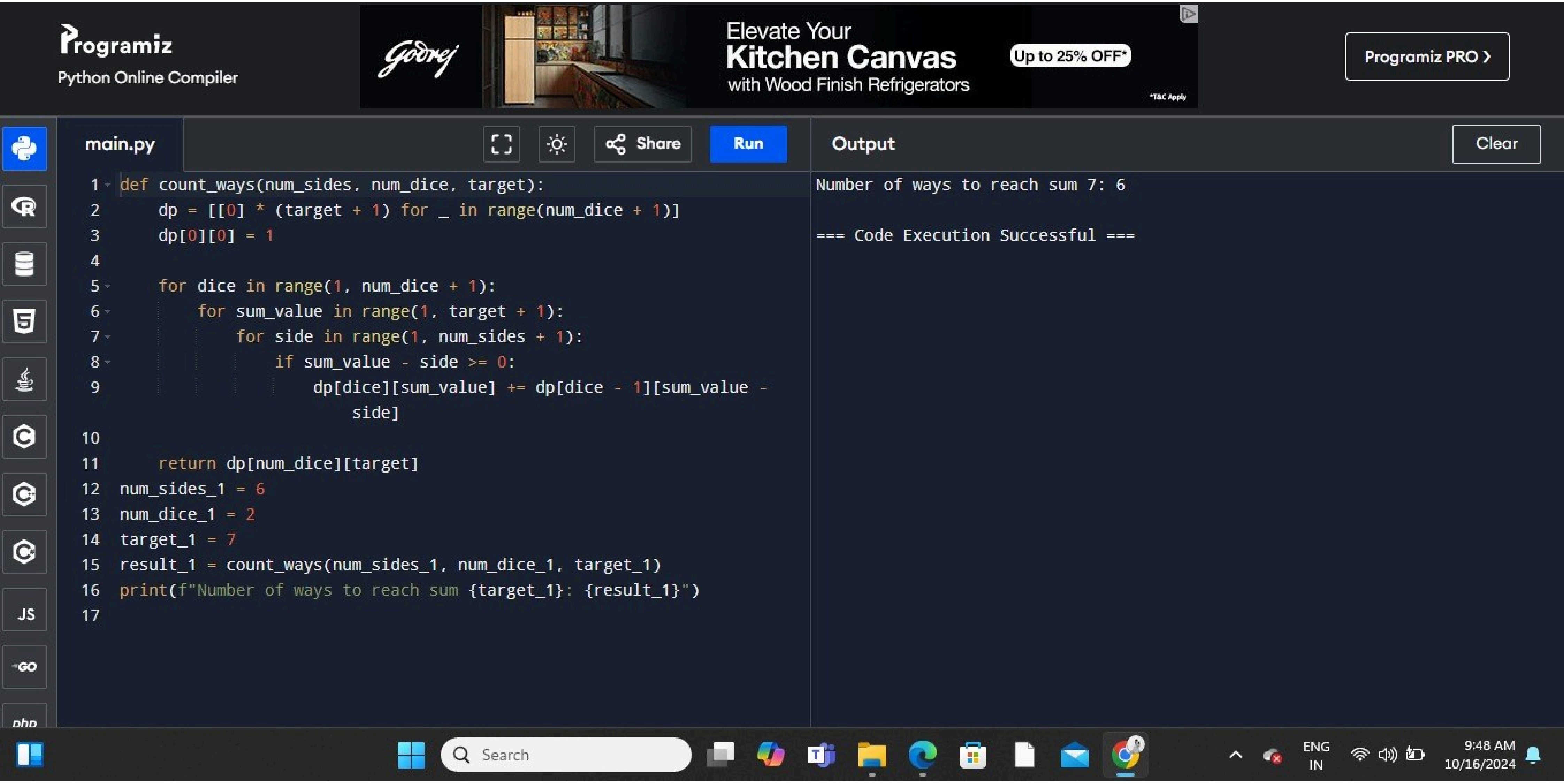
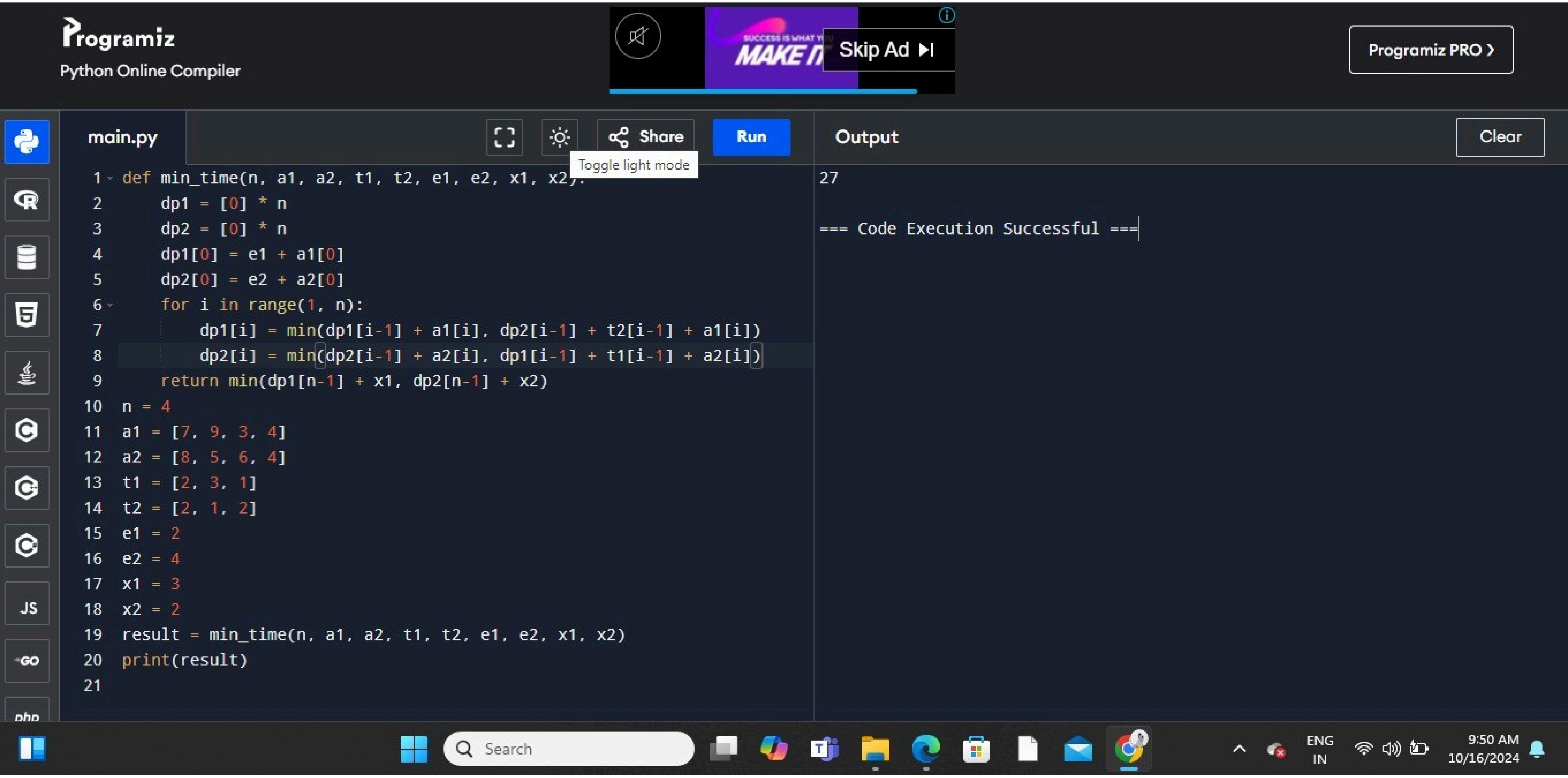


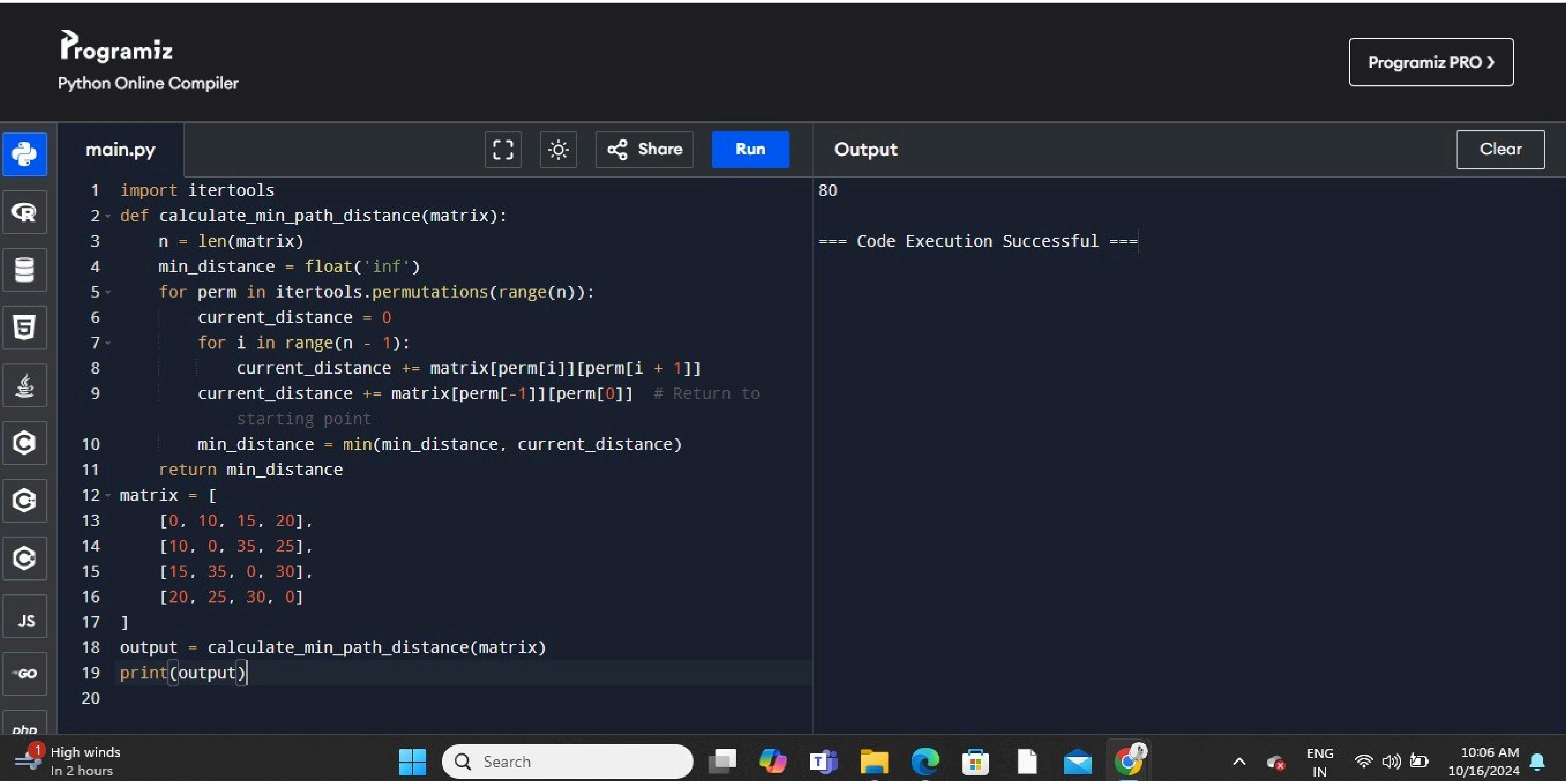
```python
def count_ways(num_sides, num_dice, target):
    dp = [[0] * (target + 1) for _ in range(num_dice + 1)]
    dp[0][0] = 1

    for dice in range(1, num_dice + 1):
        for sum_value in range(1, target + 1):
            for side in range(1, num_sides + 1):
                if sum_value - side >= 0:
                    dp[dice][sum_value] += dp[dice - 1][sum_value -
                        side]

    return dp[num_dice][target]
num_sides_1 = 6
num_dice_1 = 2
target_1 = 7
result_1 = count_ways(num_sides_1, num_dice_1, target_1)
print(f"Number of ways to reach sum {target_1}: {result_1}")
```

```
Number of ways to reach sum 7: 6

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



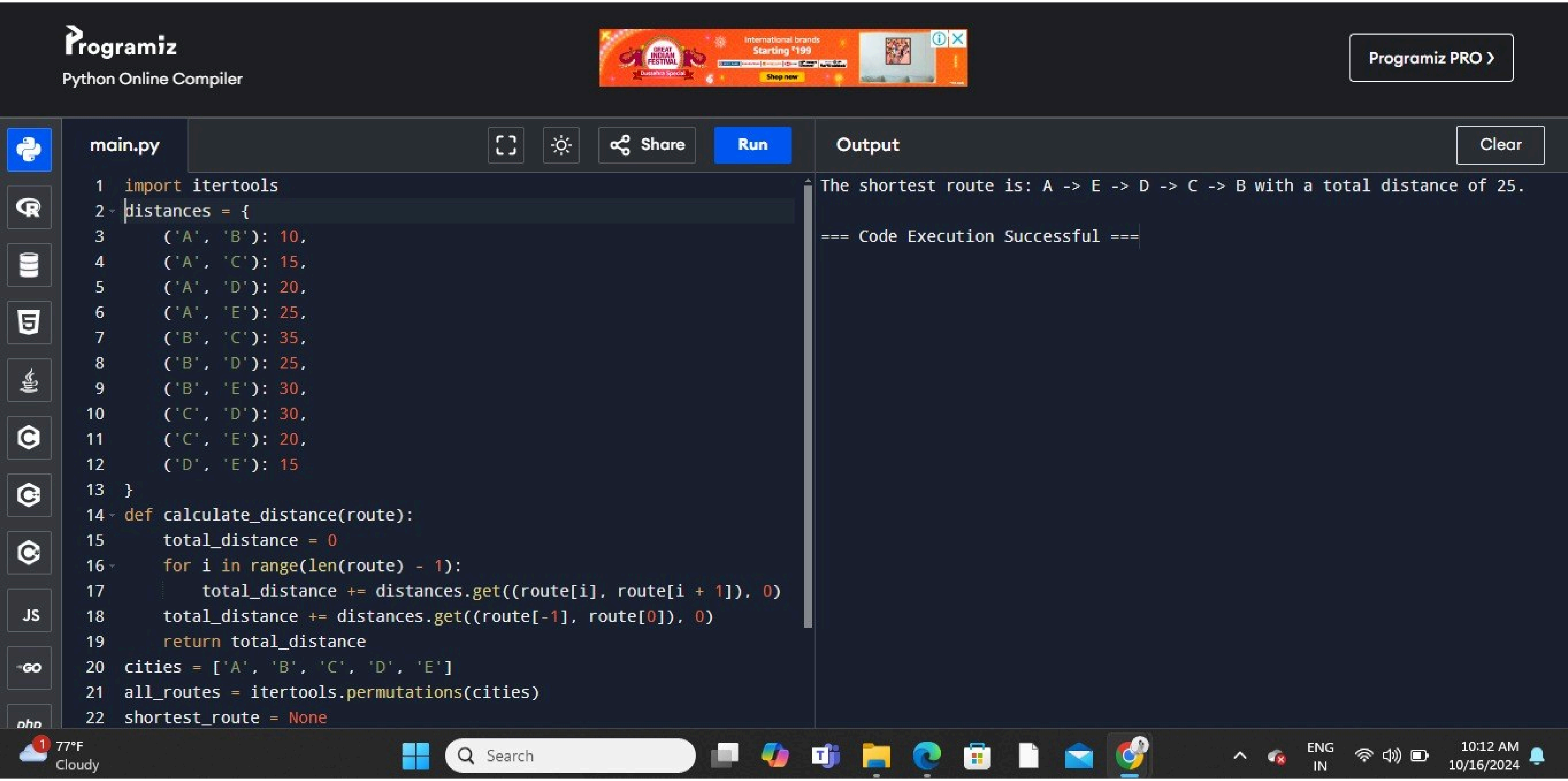
```python
def min_time(n, a1, a2, t1, t2, e1, e2, x1, x2):
    dp1 = [0] * n
    dp2 = [0] * n
    dp1[0] = e1 + a1[0]
    dp2[0] = e2 + a2[0]
    for i in range(1, n):
        dp1[i] = min(dp1[i-1] + a1[i], dp2[i-1] + t2[i-1] + a1[i])
        dp2[i] = min(dp2[i-1] + a2[i], dp1[i-1] + t1[i-1] + a2[i])
    return min(dp1[n-1] + x1, dp2[n-1] + x2)
n = 4
a1 = [7, 9, 3, 4]
a2 = [8, 5, 6, 4]
t1 = [2, 3, 1]
t2 = [2, 1, 2]
e1 = 2
e2 = 4
x1 = 3
x2 = 2
result = min_time(n, a1, a2, t1, t2, e1, e2, x1, x2)
print(result)
```

```
27

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



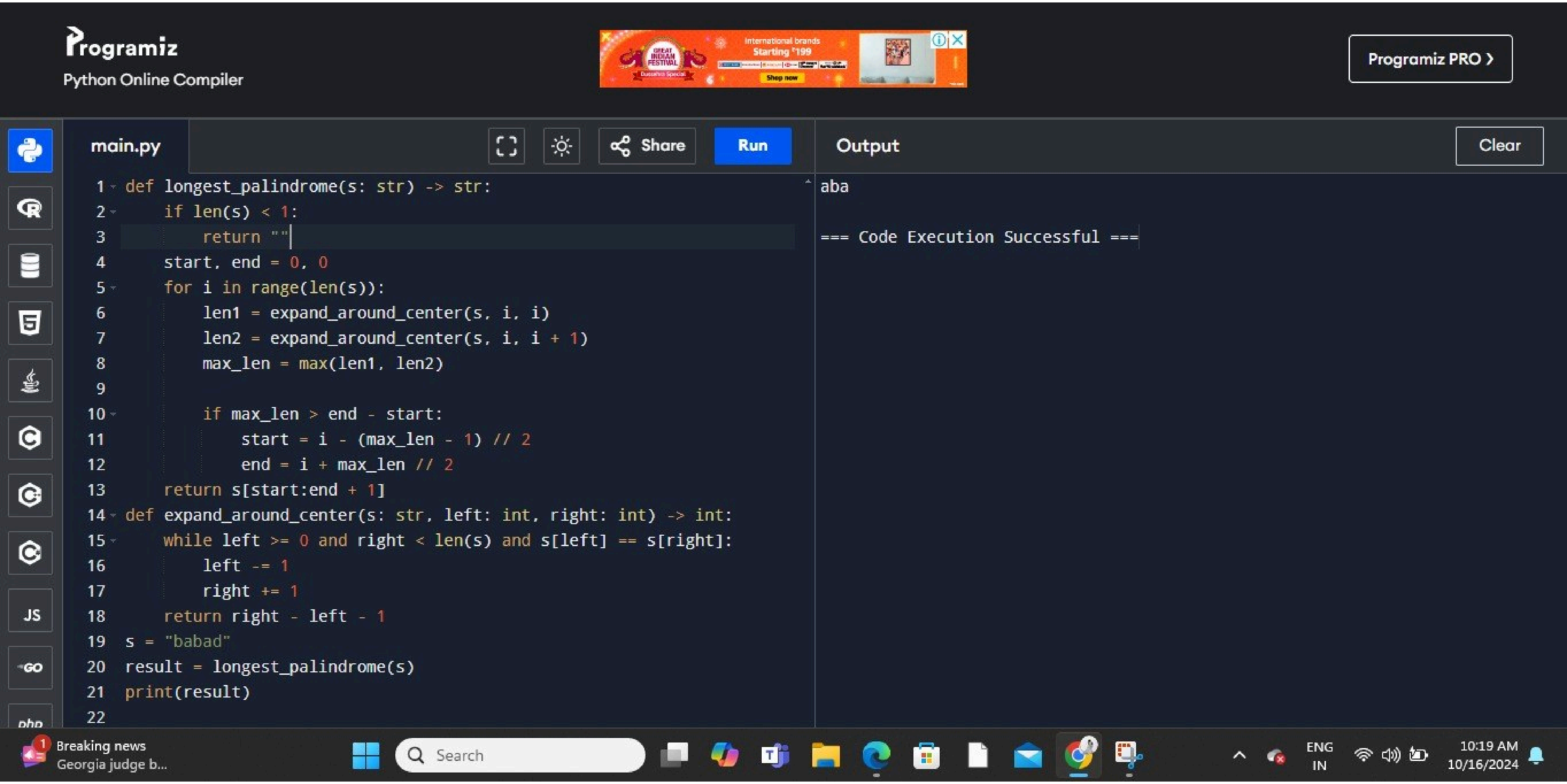
```python
import itertools
def calculate_min_path_distance(matrix):
    n = len(matrix)
    min_distance = float('inf')
    for perm in itertools.permutations(range(n)):
        current_distance = 0
        for i in range(n - 1):
            current_distance += matrix[perm[i]][perm[i + 1]]
        current_distance += matrix[perm[-1]][perm[0]]  # Return to starting point
        min_distance = min(min_distance, current_distance)
    return min_distance
matrix = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]
output = calculate_min_path_distance(matrix)
print(output)
```

```
80

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



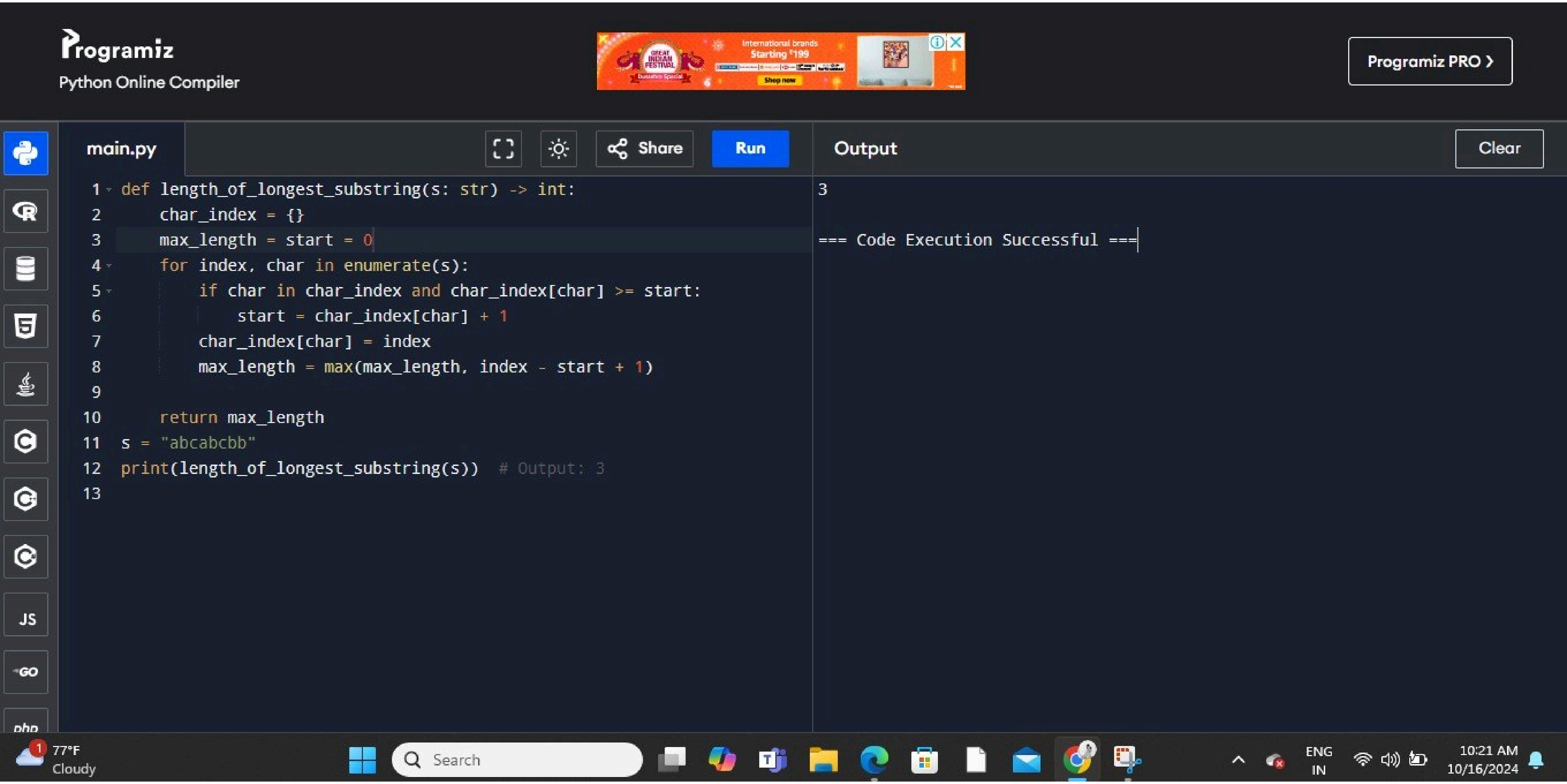
```python
import itertools
distances = {
    ('A', 'B'): 10,
    ('A', 'C'): 15,
    ('A', 'D'): 20,
    ('A', 'E'): 25,
    ('B', 'C'): 35,
    ('B', 'D'): 25,
    ('B', 'E'): 30,
    ('C', 'D'): 30,
    ('C', 'E'): 20,
    ('D', 'E'): 15
}
def calculate_distance(route):
    total_distance = 0
    for i in range(len(route) - 1):
        total_distance += distances.get((route[i], route[i + 1]), 0)
    total_distance += distances.get((route[-1], route[0]), 0)
    return total_distance
cities = ['A', 'B', 'C', 'D', 'E']
all_routes = itertools.permutations(cities)
shortest_route = None
```

```
The shortest route is: A -> E -> D -> C -> B with a total distance of 25.

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



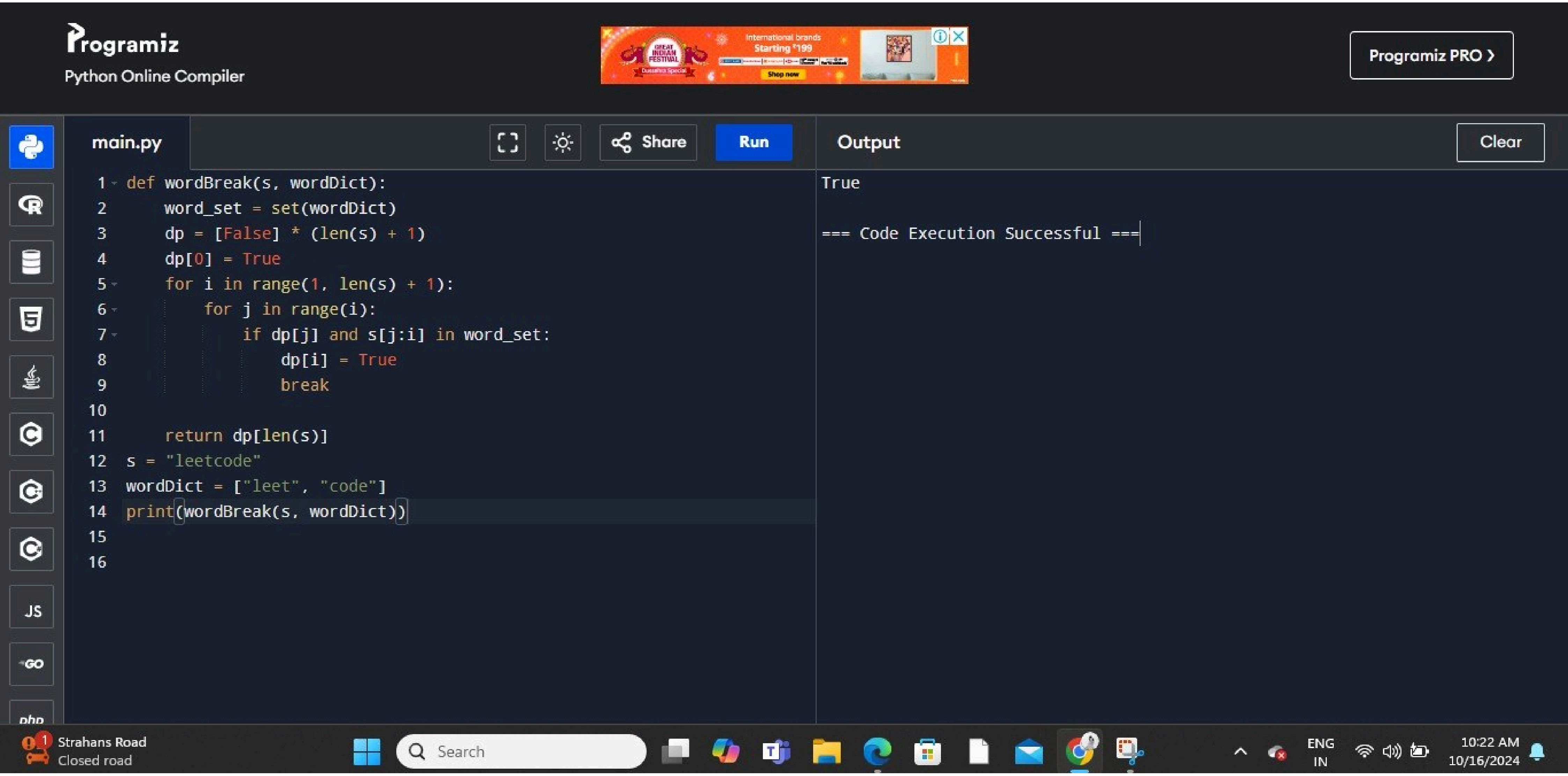
```python
def longest_palindrome(s: str) -> str:
    if len(s) < 1:
        return ""
    start, end = 0, 0
    for i in range(len(s)):
        len1 = expand_around_center(s, i, i)
        len2 = expand_around_center(s, i, i + 1)
        max_len = max(len1, len2)

        if max_len > end - start:
            start = i - (max_len - 1) // 2
            end = i + max_len // 2
    return s[start:end + 1]
def expand_around_center(s: str, left: int, right: int) -> int:
    while left >= 0 and right < len(s) and s[left] == s[right]:
        left -= 1
        right += 1
    return right - left - 1
s = "babad"
result = longest_palindrome(s)
print(result)

```

```
aba

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



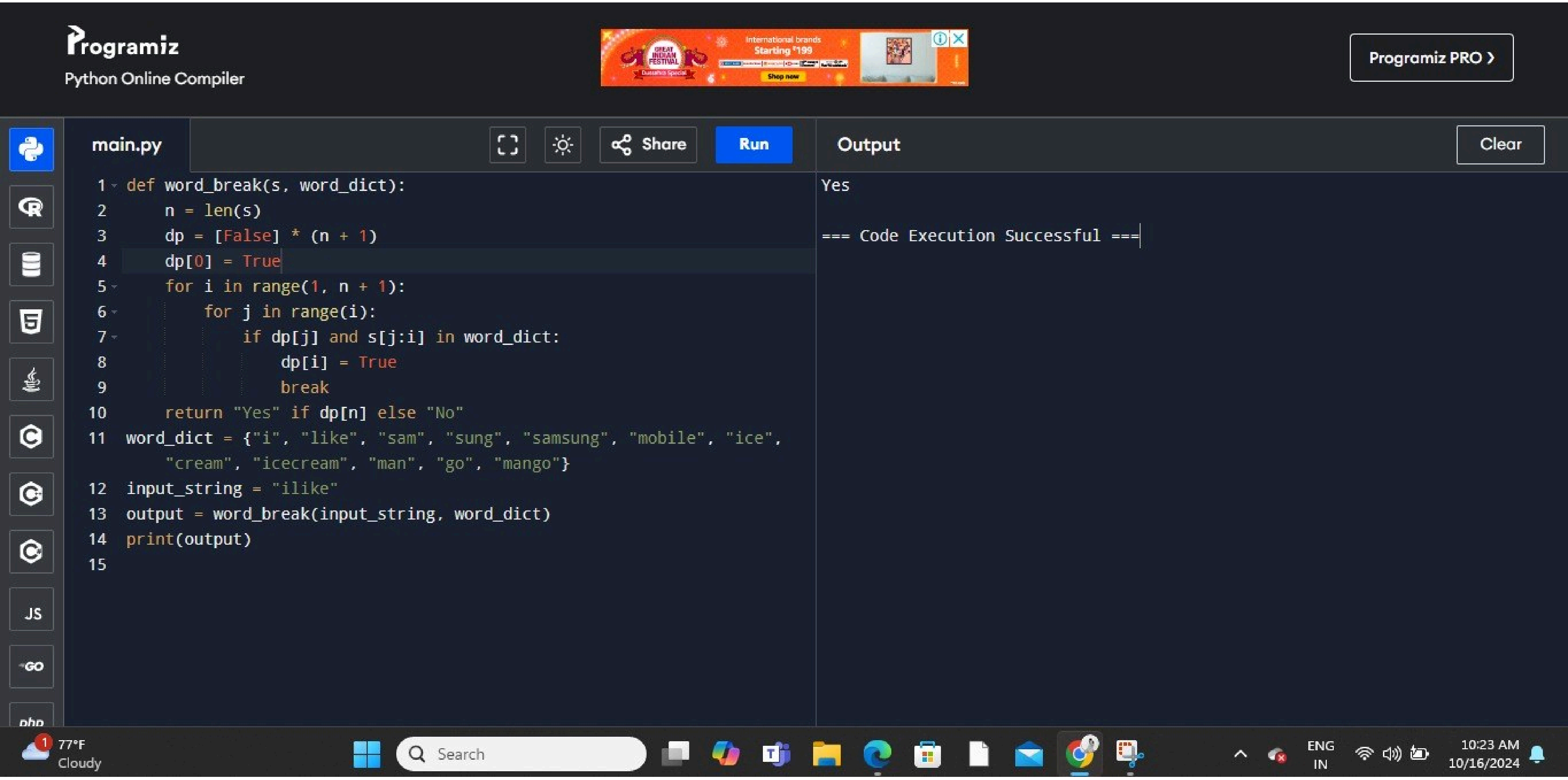
```python
def length_of_longest_substring(s: str) -> int:
    char_index = {}
    max_length = start = 0
    for index, char in enumerate(s):
        if char in char_index and char_index[char] >= start:
            start = char_index[char] + 1
        char_index[char] = index
        max_length = max(max_length, index - start + 1)

    return max_length
s = "abcabcbb"
print(length_of_longest_substring(s))  # Output: 3
```

```
3


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



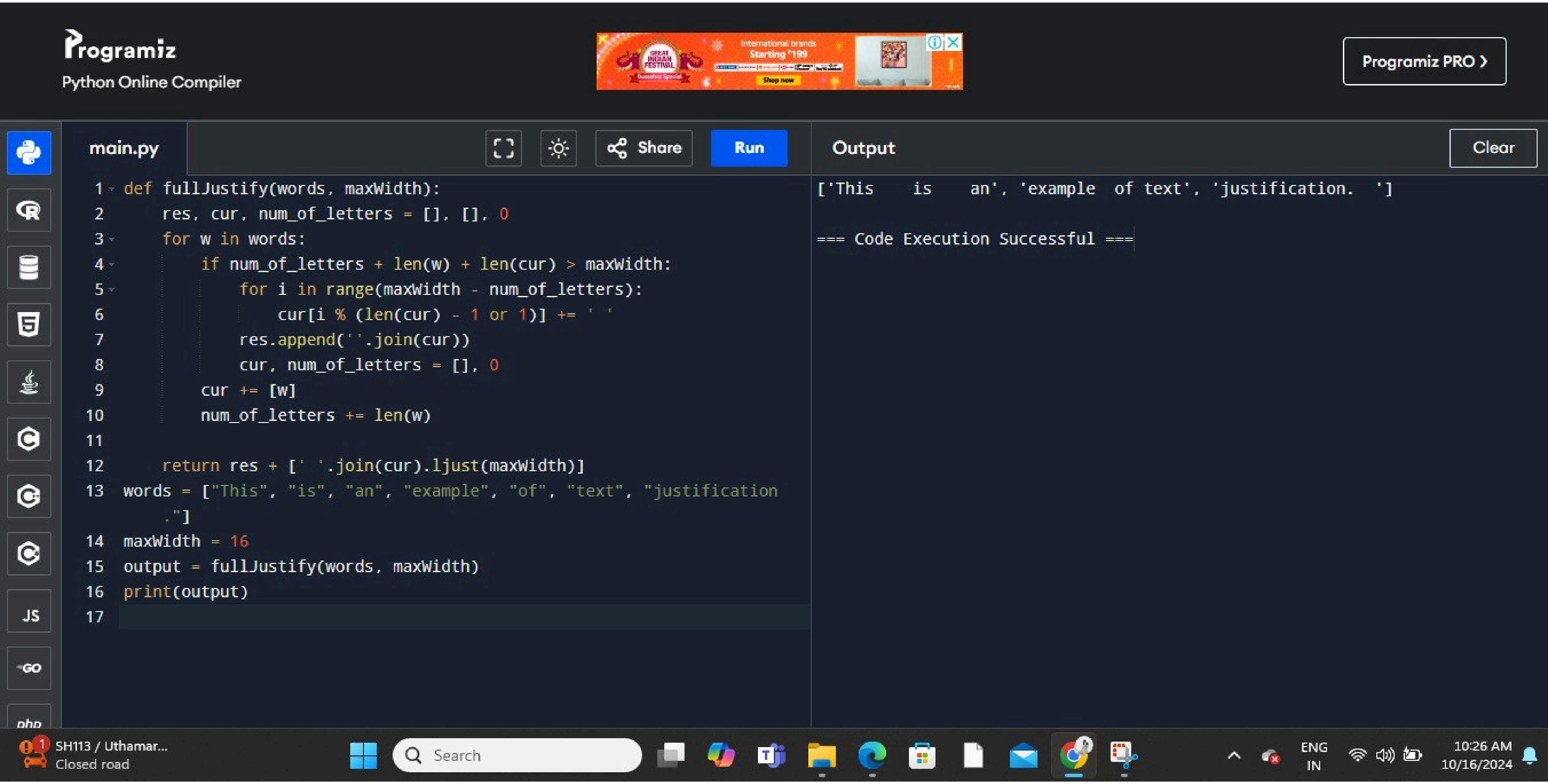
```python
def wordBreak(s, wordDict):
    word_set = set(wordDict)
    dp = [False] * (len(s) + 1)
    dp[0] = True
    for i in range(1, len(s) + 1):
        for j in range(i):
            if dp[j] and s[j:i] in word_set:
                dp[i] = True
                break

    return dp[len(s)]
s = "leetcode"
wordDict = ["leet", "code"]
print(wordBreak(s, wordDict))
```

```
True

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



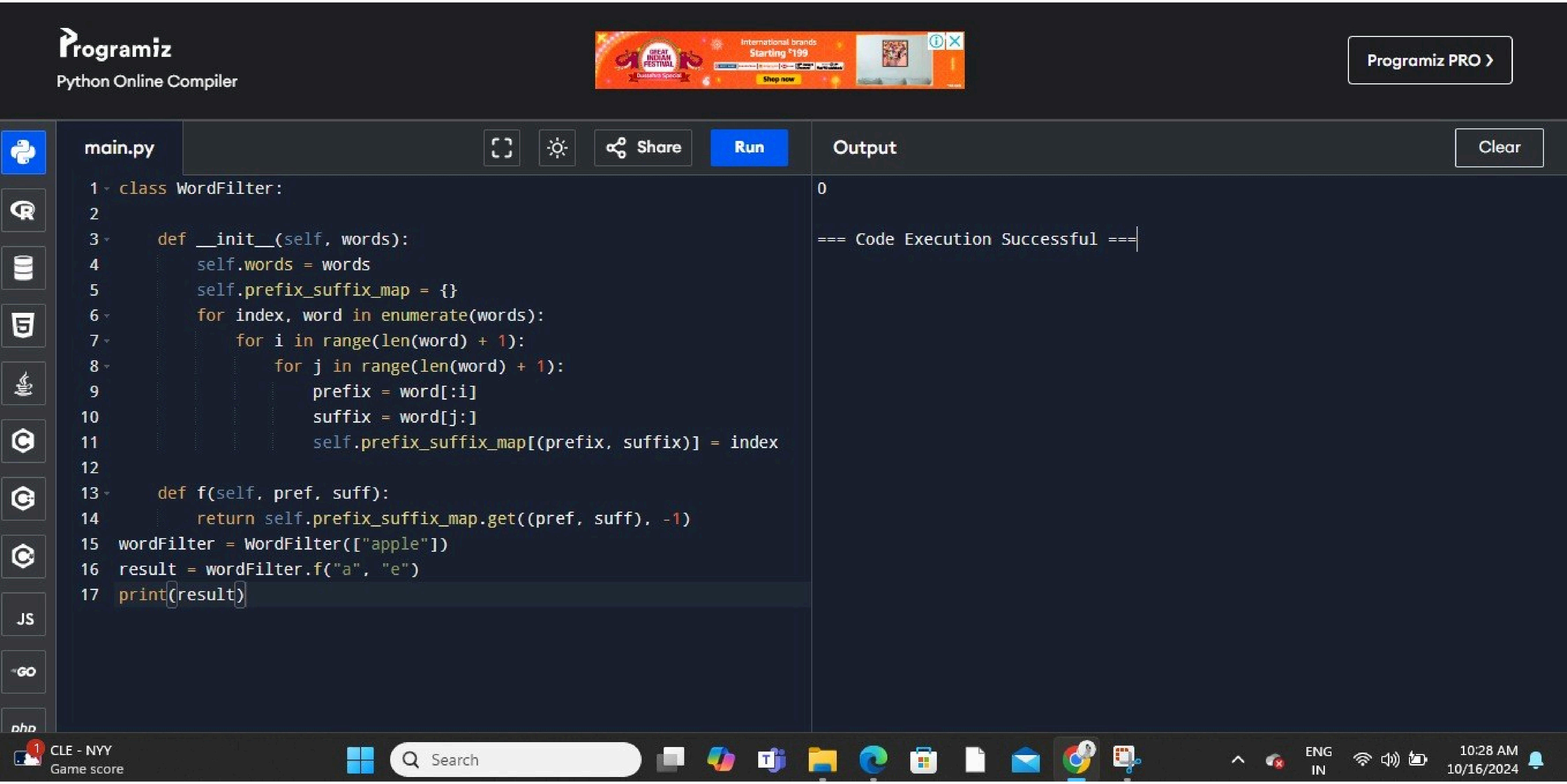
```python
def word_break(s, word_dict):
    n = len(s)
    dp = [False] * (n + 1)
    dp[0] = True
    for i in range(1, n + 1):
        for j in range(i):
            if dp[j] and s[j:i] in word_dict:
                dp[i] = True
                break
    return "Yes" if dp[n] else "No"
word_dict = {"i", "like", "sam", "sung", "samsung", "mobile", "ice",
    "cream", "icecream", "man", "go", "mango"}
input_string = "ilike"
output = word_break(input_string, word_dict)
print(output)
```

```
Yes


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



```python
def fullJustify(words, maxWidth):
    res, cur, num_of_letters = [], [], 0
    for w in words:
        if num_of_letters + len(w) + len(cur) > maxWidth:
            for i in range(maxWidth - num_of_letters):
                cur[i % (len(cur) - 1 or 1)] += ' '
            res.append(''.join(cur))
            cur, num_of_letters = [], 0
        cur += [w]
        num_of_letters += len(w)

    return res + [' '.join(cur).ljust(maxWidth)]
words = ["This", "is", "an", "example", "of", "text", "justification
    ."]
maxWidth = 16
output = fullJustify(words, maxWidth)
print(output)
```

```
['This    is    an', 'example  of text', 'justification.  ']

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



```python
class WordFilter:

    def __init__(self, words):
        self.words = words
        self.prefix_suffix_map = {}
        for index, word in enumerate(words):
            for i in range(len(word) + 1):
                for j in range(len(word) + 1):
                    prefix = word[:i]
                    suffix = word[j:]
                    self.prefix_suffix_map[(prefix, suffix)] = index

    def f(self, pref, suff):
        return self.prefix_suffix_map.get((pref, suff), -1)
wordFilter = WordFilter(["apple"])
result = wordFilter.f("a", "e")
print(result)
```

```
0

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