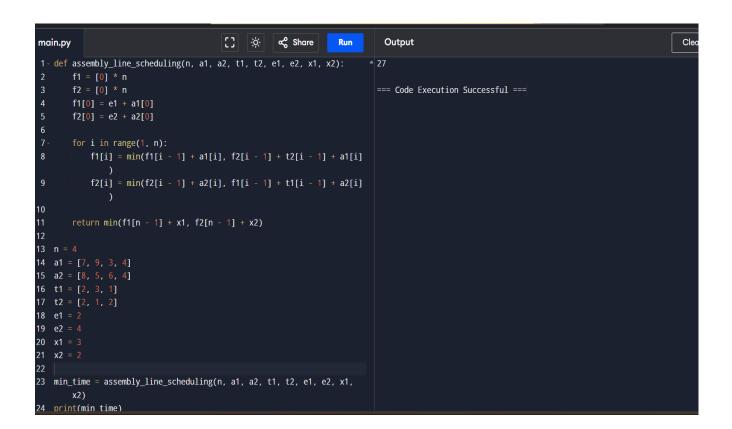
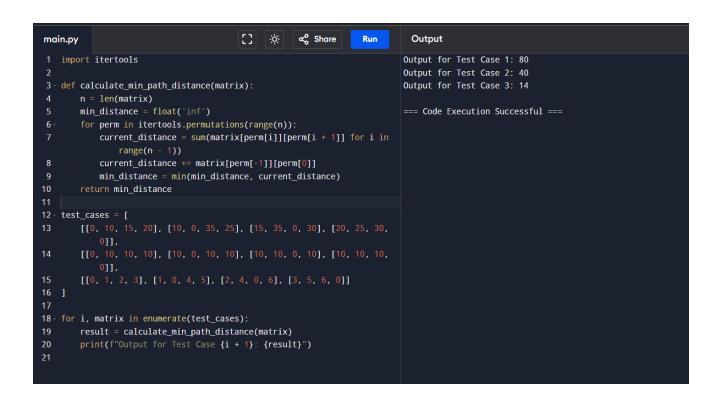
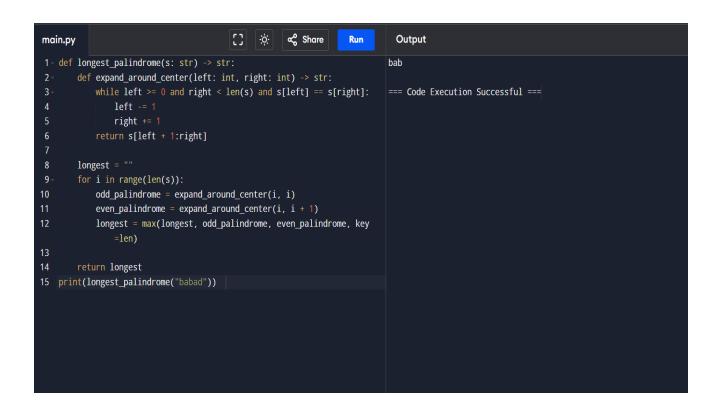
```
Output
                                                    ∝ Share
main.py
                                                                         Test Case 1: Number of ways to reach sum 7: 6
 1 - def count_ways(num_sides, num_dice, target):
                                                                         Test Case 2: Number of ways to reach sum 10: 6
       dp = [[0 for _ in range(target + 1)] for _ in range(num_dice + 1)]
       dp[0][0] = 1
                                                                         === Code Execution Successful ===
        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
10
       return dp[num_dice][target]
12 num_sides_1 = 6
13 num_dice_1 = 2
14 target_1 = 7
15 result_1 = count_ways(num_sides_1, num_dice_1, target_1)
16 print(f"Test Case 1: Number of ways to reach sum {target_1}:
        {result_1}")
17 num_sides_2 = 4
19 target_2 = 10
20 result_2 = count_ways(num_sides_2, num_dice_2, target_2)
21 print(f"Test Case 2: Number of ways to reach sum {target_2}:
        {result_2}")
```



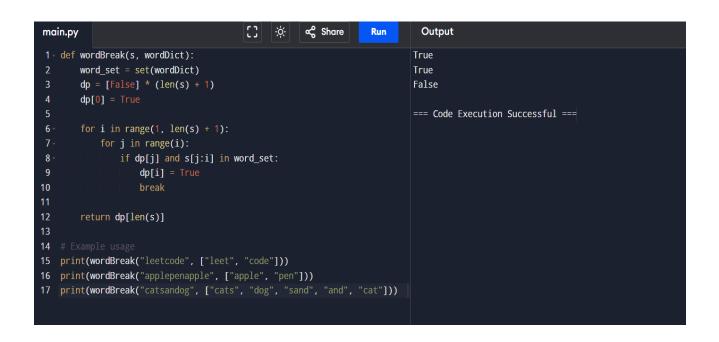
```
∝ Share
                                                                          Run
                                                                                     Output
main.py
                                                                                  Minimum Production Time: 17
 1 import numpy as np
 3 def min_production_time(station_times, transfer_times, dependencies)
                                                                                    === Code Execution Successful ===
         n = len(station_times[0])
        f1 = [0] * n
f2 = [0] * n
f3 = [0] * n
         f1[0] = station_times[0][0]
         f2[0] = station_times[1][0]
f3[0] = station_times[2][0]
         for i in range(1, n):
             f1[i] = min(f1[i - 1] + station_times[0][i], f2[i - 1] +
                 transfer_times[1][0] + station_times[0][i], f3[i - 1] +
                  transfer_times[2][0] + station_times[0][i])
             f2[i] = min(f2[i - 1] + station_times[1][i], f1[i - 1] +
                  transfer\_times[0][1] + station\_times[1][i], f3[i - 1] +
                  transfer_times[2][1] + station_times[1][i])
             f3[i] = min(f3[i - 1] + station_times[2][i], f1[i - 1] +
16
                   transfer\_times[0][2] + station\_times[2][i], \ f2[i-1] + transfer\_times[1][2] + station\_times[2][i]) 
         return min(f1[n - 1], f2[n - 1], f3[n - 1])
18
20 station times = [[5. 9. 3]. [6. 8. 4]. [7. 6. 5]]
```



```
[] 🔅
                                                         ∝ Share
                                                                                  Output
                                                                                                                                                        Clear
                                                                       Run
main.py
 1 from itertools import permutations
                                                                                Shortest route: A -> B -> D -> E -> C -> A, Total distance: 85
 3 - distances = {
                                                                                 === Code Execution Successful ===
       ('A', 'B'): 10, ('A', 'C'): 15, ('A', 'D'): 20, ('A', 'E'): 25, ('B', 'C'): 35, ('B', 'D'): 25, ('B', 'E'): 30, ('C', 'D'): 20,
10 def total_distance(route):
        return sum(distances.get((route[i], route[i + 1]), distances.get
            ((route[i + 1], route[i]))) for i in range(len(route) - 1))
15 shortest_route = min(permutations(cities), key=lambda route:
        total_distance(route + (route[0],)))
16 shortest_distance = total_distance(shortest_route + (shortest_route[0]
18 print(f'Shortest route: {" -> ".join(shortest_route)} ->
        {shortest_route[0]}, Total distance: {shortest_distance}')
```



```
∝ Share
                                                               Run
                                                                          Output
main.py
 1 def length_of_longest_substring(s: str) -> int:
       char_index_map = {}
       left = max_length = 0
       for right in range(len(s)):
                                                                        === Code Execution Successful ===
6
           if s[right] in char_index_map:
               left = max(left, char_index_map[s[right]] + 1)
           char_index_map[s[right]] = right
           max_length = max(max_length, right - left + 1)
       return max_length
13 print(length_of_longest_substring("abcabcbb"))
14 print(length_of_longest_substring("bbbbb"))
15 print(length_of_longest_substring("pwwkew"))
```



```
Output
                               [] 🔆 🗠 Share
main.py
                                                 Run
 1 def word_break_recursive(s, word_dict):
                                                        Input: ilike Output: Yes
      if not s:
                                                        Input: ilikesamsung Output: Yes
      for word in word_dict:
                                                        === Code Execution Successful ===
         if s.startswith(word):
           if word_break_recursive(s[len(word):], word_dict):
12 input_string2 = "ilikesamsung"
14 print("Input:", input_string1, "Output:", "Yes" if
      word\_break\_recursive(input\_string1, \ dictionary) \ else \ "No")
```

