main.py

Share Run Output

```python
def count_ways(num_sides, num_dice, target):

    dp = [[0 for _ in range(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"Test Case 1: Number of ways to reach sum {target_1}:
    {result_1}")
num_sides_2 = 4
num_dice_2 = 3
target_2 = 10
result_2 = count_ways(num_sides_2, num_dice_2, target_2)
print(f"Test Case 2: Number of ways to reach sum {target_2}:
    {result_2}")
```

```
Test Case 1: Number of ways to reach sum 7: 6
Test Case 2: Number of ways to reach sum 10: 6

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

main.py

Share Run Output Clea

```python
def assembly_line_scheduling(n, a1, a2, t1, t2, e1, e2, x1, x2):
    f1 = [0] * n
    f2 = [0] * n
    f1[0] = e1 + a1[0]
    f2[0] = e2 + a2[0]

    for i in range(1, n):
        f1[i] = min(f1[i - 1] + a1[i], f2[i - 1] + t2[i - 1] + a1[i]
            )
        f2[i] = min(f2[i - 1] + a2[i], f1[i - 1] + t1[i - 1] + a2[i]
            )

    return min(f1[n - 1] + x1, f2[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

min_time = assembly_line_scheduling(n, a1, a2, t1, t2, e1, e2, x1,
    x2)
print(min_time)
```

```
27

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

main.py

Share Run

Output

```python
import numpy as np

def min_production_time(station_times, transfer_times, dependencies):
    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] +
            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])

station_times = [[5, 9, 3], [6, 8, 4], [7, 6, 5]]
```

```
Minimum Production Time: 17

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

main.py

Share Run

Output

```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 = sum(matrix[perm[i]][perm[i + 1]] for i in
            range(n - 1))
        current_distance += matrix[perm[-1]][perm[0]]
        min_distance = min(min_distance, current_distance)
    return min_distance

test_cases = [
    [[0, 10, 15, 20], [10, 0, 35, 25], [15, 35, 0, 30], [20, 25, 30,
        0]],
    [[0, 10, 10, 10], [10, 0, 10, 10], [10, 10, 0, 10], [10, 10, 10,
        0]],
    [[0, 1, 2, 3], [1, 0, 4, 5], [2, 4, 0, 6], [3, 5, 6, 0]]
]

for i, matrix in enumerate(test_cases):
    result = calculate_min_path_distance(matrix)
    print(f"Output for Test Case {i + 1}: {result}")
```

```
Output for Test Case 1: 80
Output for Test Case 2: 40
Output for Test Case 3: 14

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

main.py

Share Run

```python
from itertools import permutations

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 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))

cities = ['A', 'B', 'C', 'D', 'E']

shortest_route = min(permutations(cities), key=lambda route:
    total_distance(route + (route[0],)))
shortest_distance = total_distance(shortest_route + (shortest_route[0]
    ,))

print(f'Shortest route: {" -> ".join(shortest_route)} ->
    {shortest_route[0]}, Total distance: {shortest_distance}')
```

Output Clear

```
Shortest route: A -> B -> D -> E -> C -> A, Total distance: 85

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

main.py

Share Run

```python
def longest_palindrome(s: str) -> str:
    def expand_around_center(left: int, right: int) -> str:
        while left >= 0 and right < len(s) and s[left] == s[right]:
            left -= 1
            right += 1
        return s[left + 1:right]

    longest = ""
    for i in range(len(s)):
        odd_palindrome = expand_around_center(i, i)
        even_palindrome = expand_around_center(i, i + 1)
        longest = max(longest, odd_palindrome, even_palindrome, key
            =len)

    return longest
print(longest_palindrome("babad"))
```

Output

```
bab

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

main.py

Share Run

```python
def length_of_longest_substring(s: str) -> int:
    char_index_map = {}
    left = max_length = 0

    for right in range(len(s)):
        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

print(length_of_longest_substring("abcabcbb"))
print(length_of_longest_substring("bbbbb"))
print(length_of_longest_substring("pwwkew"))

```

Output

```
3
1
3

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

main.py

Share Run

```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)]

# Example usage
print(wordBreak("leetcode", ["leet", "code"]))
print(wordBreak("applepenapple", ["apple", "pen"]))
print(wordBreak("catsandog", ["cats", "dog", "sand", "and", "cat"]))
```

Output

```
True
True
False

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

main.py

```python
def word_break_recursive(s, word_dict):
    if not s:
        return True
    for word in word_dict:
        if s.startswith(word):
            if word_break_recursive(s[len(word):], word_dict):
                return True
    return False

dictionary = {"i", "like", "sam", "sung", "samsung", "mobile", "ice", "cream", "icecream", "man", "go", "mango"}
input_string1 = "ilike"
input_string2 = "ilikesamsung"

print("Input:", input_string1, "Output:", "Yes" if word_break_recursive(input_string1, dictionary) else "No")
print("Input:", input_string2, "Output:", "Yes" if word_break_recursive(input_string2, dictionary) else "No")

```

Output

```
Input: ilike Output: Yes
Input: ilikesamsung Output: Yes

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

main.py

```python
def fullJustify(words, maxWidth):
    res, current_line, num_of_letters = [], [], 0

    for word in words:
        if num_of_letters + len(word) + len(current_line) > maxWidth:
            for i in range(maxWidth - num_of_letters):
                current_line[i % (len(current_line) - 1 or 1)] += ' '
            res.append(''.join(current_line))
            current_line, num_of_letters = [], 0
        current_line.append(word)
        num_of_letters += len(word)

    res.append(' '.join(current_line).ljust(maxWidth))
    return res

words1 = ["This", "is", "an", "example", "of", "text", "justification."]
maxWidth1 = 16
print(fullJustify(words1, maxWidth1))

```

Output

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

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

main.py

```python
class WordFilter:
    def __init__(self, words):
        self.words = words

    def f(self, pref, suff):
        for i in range(len(self.words) - 1, -1, -1):
            if self.words[i].startswith(pref) and self.words[i].endswith(suff):
                return i
        return -1

wordFilter = WordFilter(["apple"])
print(wordFilter.f("a", "e"))
```

Output

```
0

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