USE Case-1

Finding the winning strategy in a card game

Problem Description:

Imagine a card game where each player recieves a hand of cards with values. The objective is to find the best way to max the score for a player. Each player can either pick the first or last card from pile.

Assumption:

- Each player tries to maximize the score.
- cards are represented by integers.
- Two players alterate turns, and each player pick card from either the beginning or end.

Algorithm:

1. Define the Game: Represent the pile of cards as int
2. recursive strategy: A function will recursively determine the best score.
3. Score immediate to avoid recalculating them.
4. Base cases: When only one card is left, current player takes it.

. Program

```
def find_optimal_strategy(cards):
    n = len(cards)
    dp = [[0]* n for_in range(n)]
    for length in range(1, n+1):
        For i range in (n - length +1):
            j = i + length -1
```

Sample Output

array of cards : [3, 9, 1, 2]

1. First player can choose
   - Taking left most card (3), leaving cards [9, 1, 2]
   - Taking right most cards (2), leaving cards [3, 9, 1]

This pragm computes the best possible outcome

First player optimal score : 5.

```
if i == j:
    dp[i][j] = cards[i]
else:
    take_left = cards[i] - dp[i+1][j]
    take_right = cards[j] - dp[i][j-1]
    dp[i][j] = max(take_left, take_right)
return (dp[0][n-1] + sum(cards))
cards = [3, 9, 1, 2]
print("First player's optimal score:", Find_optimal
                                    _strategy(cards))
```

Explanation

- Dp: Each cell dp[i][j] represents difference in score between the two players played between cards from index i to j
- Two choices: For each move
  1. Pick the leftmost card card[i]
  2. Pick the rightmost card card[j],
- recursive relation: The value of each subproblem is maximizing the score difference between the players.

Result:

Thus the program for use case is executed successfully.