**def knapsack\_dp(weights, values, capacity):**

**"""**

**Function to solve the 0/1 Knapsack problem using dynamic programming.**

**:param weights: List of item weights**

**:param values: List of item values**

**:param capacity: Maximum weight capacity of the knapsack**

**:return: Maximum value that can be obtained**

**"""**

**# Number of items**

**n = len(values)**

**# Create a 2D DP table with (n+1) rows and (capacity+1) columns, initialized to 0**

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

**# Iterate through each item**

**for i in range(1, n + 1):**

**# Iterate through each weight from 1 to the capacity**

**for w in range(1, capacity + 1):**

**# If the current item's weight is less than or equal to the current capacity**

**if weights[i - 1] <= w:**

**# Take the maximum value between:**

**# 1. Including the current item (value of item + best value without current item's weight)**

**# 2. Excluding the current item (best value from the previous item)**

**dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w])**

**else:**

**# If the current item's weight exceeds the capacity, carry forward the previous value**

**dp[i][w] = dp[i - 1][w]**

**# Return the value at the bottom-right corner of the table (optimal solution)**

**return dp[n][capacity]**

**# Example usage**

**weights = [1, 2, 3, 5]**

**values = [1, 6, 10, 16]**

**capacity = 7**

**print(f"Maximum value using Dynamic Programming: {knapsack\_dp(weights, values, capacity)}")**