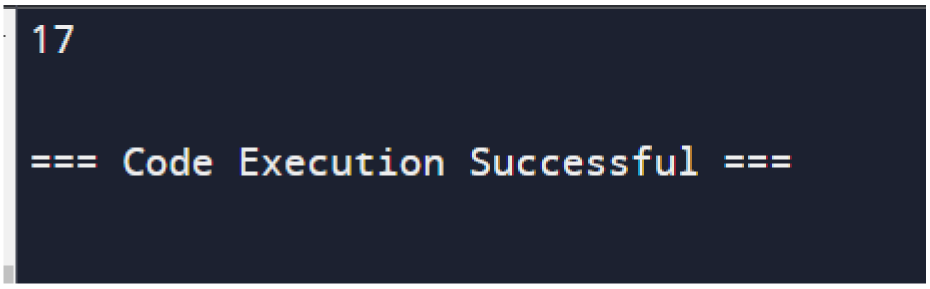
2. Knapsack Problem

PROGRAM:-

```
def knapsack(weights, values, capacity):
    n = len(values)

    # Create a 2D DP array where dp[i][w] represents the maximum value that can be obtained
    # with the first i items and a maximum weight capacity of w.
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]

    # Build the dp array
    for i in range(1, n + 1):
        for w in range(1, capacity + 1):
            if weights[i-1] <= w:
                dp[i][w] = max(dp[i-1][w], dp[i-1][w - weights[i-1]] + values[i-1])
            else:
                dp[i][w] = dp[i-1][w]

    # The maximum value will be in dp[n][capacity]
    return dp[n][capacity]

# Example usage:
weights = [1, 2, 3, 2]
values = [8, 4, 0, 5]
capacity = 5
print(knapsack(weights, values, capacity))  # Output: 13 (items with weights 1 and 2, both with
values 8 and 5)
```

OUTPUT:-



```
17


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

TIME COMPLEXITY:-O(n*m)