SUBJECT : MVC

TEAM MEMBERS : MUHAMMAD NOOR, SHAHOUD SHAHID, SAIF SHAHZAD

ROLL NO : I23-2520 / I23-2515 / I23-2634

TABLE OF CONTENTS :

OBJECTIVES :

The primary objective of this project is to design Python Software for an optimal layout of a weighing box that optimizes that amount of material used in packaging items into a single box. This involves developing an optimization algorithm where the goal is to determine the optimal quantities of items to pack into a box,considering weight constraint and optimizing total weight.It utilizes numerical techniques from numpy and sci.py libraries for optimization,along with matplotlib for visualization. The user provides input such as the number of items,weight capacity of the box, and cost per unit weight. The model calculates the optimized quantities of each item to optimize the total weight while ensuring the total weight does not exceed the box's capacity.It offers a streamline solution for businesses and individuals to reduce material waste and lower logistics expenses.

KEY GOALS :

Some of the important key goals ensured are noticed while making the program are discussed below :

1) **Maximize Space Utilization :** The tool aims to effectively allocate space within the weight capacity, ensuring that items are packed optimally to minimize wasted space.

2) **Minimize Total Cost :** By optimizing the item quantities and weights, an aim is set to reduce overall costs associated with packing , offering cost-effective solutions for businesses and individuals.

3) **User-Friendly Interaction :** An ease of use is prioritized, providing an intuitive interface for inputting data and visualizing optimization results,enhancing user experience and accessibility.

4) **Scalability and Flexibility:** The tool is designed to handle varying input sizes and requirements,ensuring scalability for large -scale problems while remaining flexible to accommodate different constraints and objectives.

5) **Simulation and Visualization :** Use Python to simulate the proposed model,visualize the optimal value and its impact on the weight items and demonstrate the overall effectiveness of the optimized layout.

These are the few steps methodology that will be followed in the project. Let's start with the step by step solution.

STEP-BY-STEP SOLUTION :

Understanding the project statement before implementing is very important. In this project, the Python software that will give the optimal layout of the weightbox is needed to be made. . However,designing the software will require a mathematical solution.Here are some assumptions regarding this solution :

Assumptions and Values:

In order to optimize the amount of material used , an optimized value is required at which the weight entered will be measured and optimal layout will be sketched while doing the mathematical calculations. Now next to optimizing the amount of material used, the cost will need to be calculated. For this, the possible arguments for this will be the cost of each unit weight and the weight capacity.The values for the amount of material to be found and optimized will be entered by the user himself and the number of items being placed inside the box will be determined . Weighing capacity of the box will too be determined .

Mathematical Solution :

Lets see how the assumptions and values entered by user contribute to the mathematical solution.Here is the detail mathematical solution :

1) Initial Guess :

The initial guess is crucial as it provides a starting point for optimization algorithm. Given the goal is to distribute the weight capacity equally among the items based on their weights, initial guess for quantity of item (q_i) can be computed as follows:

- Let W be the total weight capacity of box.
- Let w_i be the weight of item.
- Let (n) be the number of different items.

The formula for initial guess (q_0) assuming an equal weight distribution proportional to each item's weight is :

$$q_i^0 = \frac{W}{\sum_{i=1}^{n} w_i}$$

This formula divides the total weight capacity by sum of all item weights, scaling capacity ensuring an equal fractional use of capacity.

2) **Optimization Process:**

This optimization process involves finding the set of quantities (q_i) that minimizes objective function while adhering to constraints and bounds.

(i) **Objective Function:** The objective is to minimize the total weights

$$f(q) = \sum_{i=1}^{n} w_i q_i$$

(ii) **Constraints :** We need to ensure that total weight doesn't exceed box's capacity:

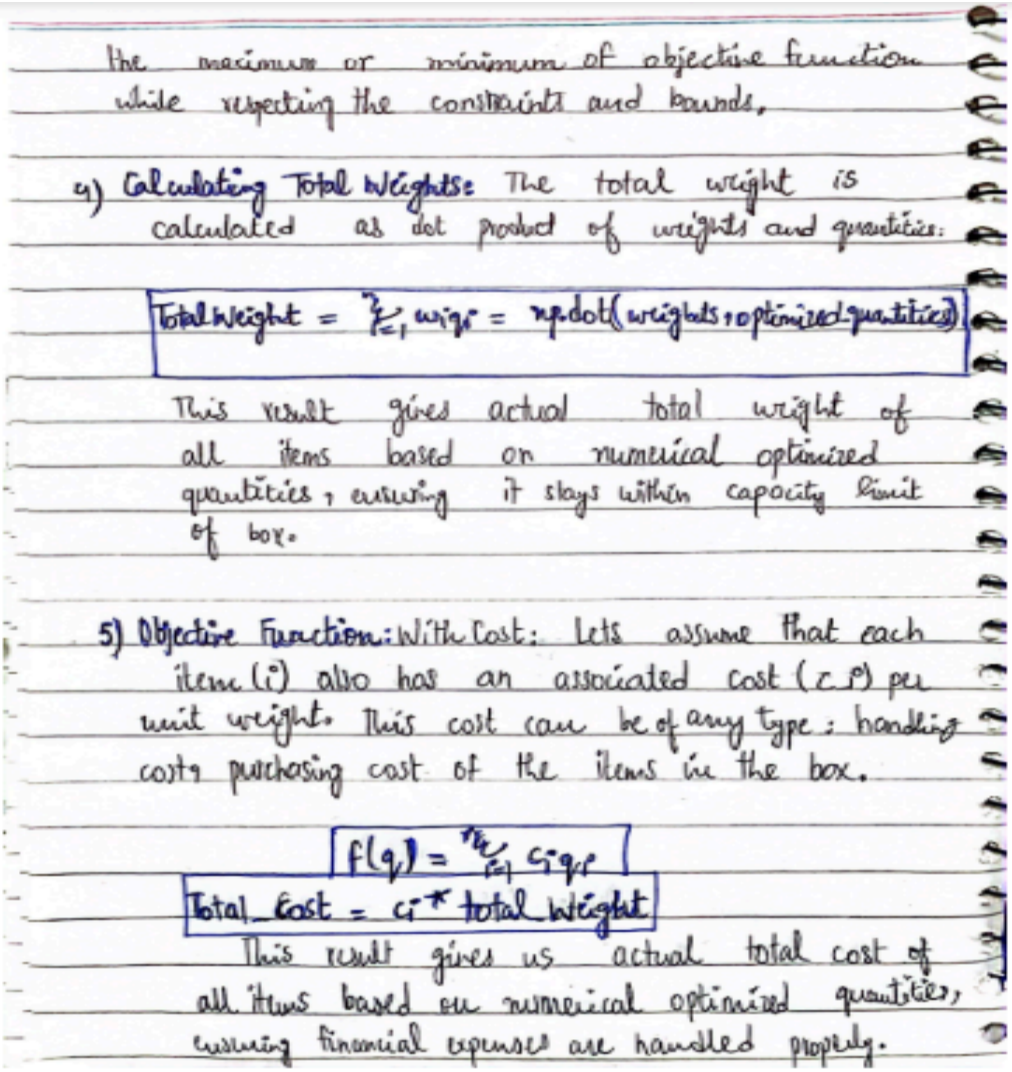
$$g(q) = \sum_{i=1}^{n} w_i q_i - W \leq 0$$

(iii) **Bounds :** All the quantities must be non-negative:

$q_i \geq 0$ for all i

3) **Extracting Optimized Quantities:** They refer to the optimal amount of each item to be included.

For this SLSQP, part of scipy.optimize module, is used. It adjusts quantities Iteratively to find



After this procedure, the solution from the optimized_quantities,total_weight and total_cost needed to pack the items is obtained . This solution uses optimization techniques to design a model by optimizing the material and displaying cost and optimized weight.

PYTHON SOLUTION :

Here is the primary goal of project which is designing of the model which optimizes material(weight) using Python :

+ Code + Text

```python
import numpy as np             #Import Numpy for numerical operations
from scipy.optimize import minimize    #Import minimize function from scipy.optimize
import matplotlib.pyplot as plt          #Import pyplot module from matplotlib for plotting
import sys                               #Import the sys module


def main():
    display_info()
    # Input number of items , weight capacity  and cost per unit weight of the box
    num_items = int(input('Enter the number of items: '))           #Prompting the user to enter items
    capacity = float(input('Enter the weight capacity of the box: ')) #Prompting the user to enter weight capacity
    cost_per_unitweight = float(input('Enter the cost per unit weight :')) #Prmpting the user to enter cost per unit weight

    weights = np.zeros(num_items)        # Initilize an array to store weights of each item

    # Input weights of each item
    print('Enter the weights  of each item:')
    for i in range(num_items):
        weights[i] = float(input(f'Weight of item {i+1}: ')) #Prompting the user to input weight of each item
    # Check if total weight exceeds capacity
    if np.sum(weights) > capacity:
        print('Error: Total weight exceeds capacity.')      #Print error message if total weight exceeds capacity
        sys.exit()   #Exit the Program
```

+ Code + Text

```python
    else:
        # Perform optimization
        optimized_quantities, total_weight = perform_optimization(weights , capacity )

        #Calculate total cost
        total_cost = cost_per_unitweight * total_weight

        # Display optimization results
        display_results(optimized_quantities, total_weight , weights  , capacity , total_cost , cost_per_unitweight )

#Function to display student info
def display_info () :
    print("Welcome to Our Software House")
    print("Developed By Team ")
    print("- Muhammad Noor {ID : I23-2520}")
    print("- Shahoud Shahid {ID : I23-2515}")
    print("- Saif Shahzad {ID : I23-2634}")

# Function to perform optimization
def perform_optimization(weights , capacity):
    # Objective function: minimizes the sum of quantities
    def objective(x):
       return -np.sum(weights * x)        # Return negative sum of product of weights and quantities
```

+ Code + Text

```
    #Minimize total weight used

    # Constraint function: Total weight shouldn't exceed box's capacity
    def weight_constraint(x):
        return  capacity -np.dot(weights, x)  #Return difference between weight capacity and dot product of weights and quantities

    #Non-negative constraint
    bounds = [(0,capacity) for _ in weights]  #Defining bounds for qunatities of each item(non-negative constraint)

    # Initial guess: equal quantities for each item
    x0 = np.full(len(weights), capacity / sum(weights))    #Initialize quantities for each item
    #Provides a more balanced start

    # Optimization using scipy's minimize function
    opt_result = minimize(objective , x0 , bounds = bounds , constraints={'type': 'ineq', 'fun': weight_constraint})

    # Extracting optimized quantities and total weight
    optimized_quantities = opt_result.x
    total_weight = np.dot(weights,optimized_quantities)

    return optimized_quantities, total_weight

# Function to display optimization results
def display_results(optimized_quantities, total_weight ,weights  , capacity  , total_cost , cost_per_unitweight):
```

+ Code + Text

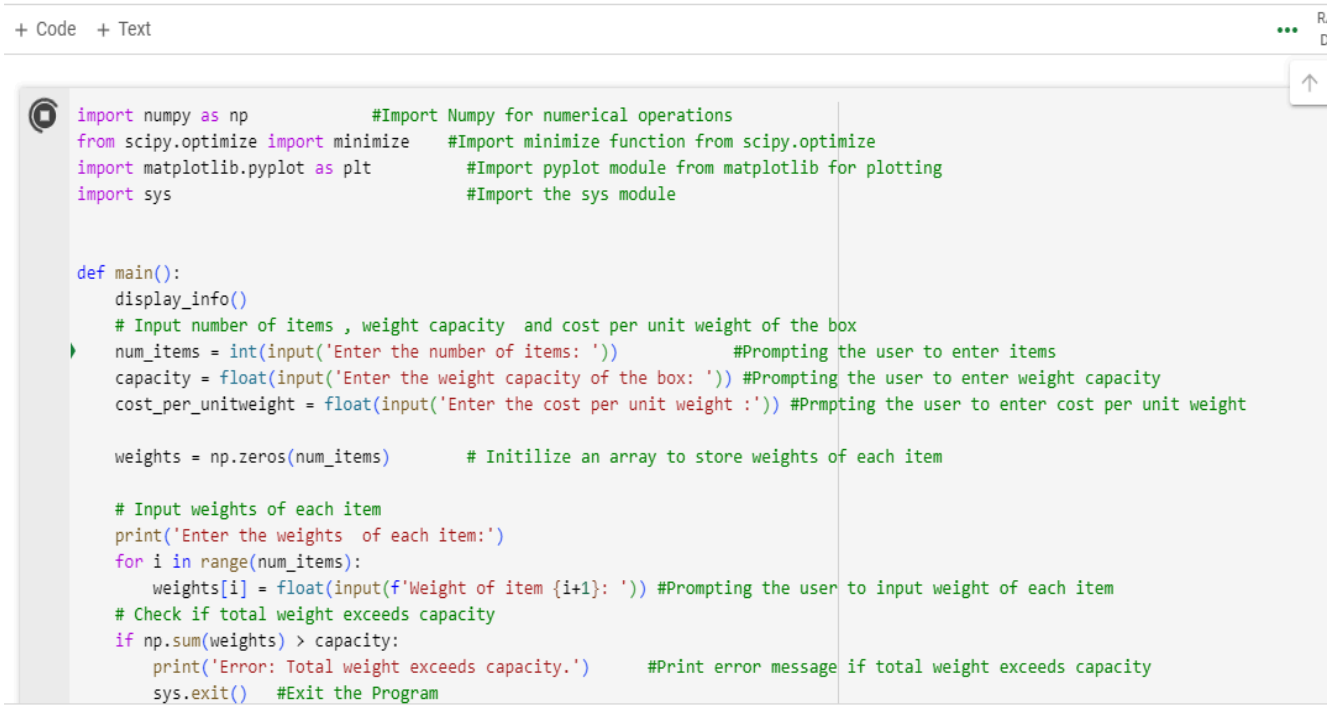
```
    print('Optimized quantities of each item:')
    for i, qty in enumerate(optimized_quantities):
        print(f'Item {i+1}: {qty:.3f}(Weight:{weights[i]*qty:.3f})')  #Printing optimized quantties and corresponding weights

    print(f'Total weight of the packaged items: {total_weight:.3f}')
    print(f'Total cost  of the packaged items: {total_cost:.3f}')

    if total_weight > capacity :
              print("Warning : Total weight exceeds the capacity !")   #Printing warning if total weight exceeds capacity
              sys.exit()                                               #Exit the Program1
    else :
              print ("Total weight is within the box's capacity.")     #Printing message if total weight is within capacity


    # Plotting the optimized quantities
    plt.bar(range(1, len(optimized_quantities) + 1), weights*optimized_quantities , color=['blue'])
    plt.xlabel('Item Number')
    plt.ylabel('Quantity')
    plt.title('Optimized Weights  of Each Item')
    plt.grid(True)
    plt.show()
# Calling main function to start the program
if __name__ == "__main__":
    main()    #Execute the main function if the script is executed directly
```

STEP-BY-STEP TRACKING FOR PYTHON CODE :

Here is the step by step example showing the demonstration of the above mentioned python code :

1) **Input Collection :** In this step , the user is prompted to enter essential parameters required for the optimization process. These parameters include the number of items , weight capacity and cost per unit weight and weights of each item. By collecting this input , the stage for defining and solving the optimization problem is set.

2) Variable Declaration : After gathering the user input, the variables needed to store this information are declared and set up the optimization problem. These variables include the number of items(“num_items ” , weight capacity of the box “capacity” , cost per unit weight (“cost_per_unitweight” , and an array to store the weights of each item (“weights”). These variables are initialized with the user-provided values in the subsequent steps.



```
+ Code  + Text

import numpy as np             #Import Numpy for numerical operations
from scipy.optimize import minimize    #Import minimize function from scipy.optimize
import matplotlib.pyplot as plt          #Import pyplot module from matplotlib for plotting
import sys                               #Import the sys module


def main():
    display_info()
    # Input number of items , weight capacity  and cost per unit weight of the box
    num_items = int(input('Enter the number of items: '))            #Prompting the user to enter items
    capacity = float(input('Enter the weight capacity of the box: ')) #Prompting the user to enter weight capacity
    cost_per_unitweight = float(input('Enter the cost per unit weight :')) #Prmpting the user to enter cost per unit weight

    weights = np.zeros(num_items)        # Initilize an array to store weights of each item

    # Input weights of each item
    print('Enter the weights  of each item:')
    for i in range(num_items):
        weights[i] = float(input(f'Weight of item {i+1}: ')) #Prompting the user to input weight of each item
    # Check if total weight exceeds capacity
    if np.sum(weights) > capacity:
        print('Error: Total weight exceeds capacity.')      #Print error message if total weight exceeds capacity
        sys.exit()   #Exit the Program
```

3) **Objective Function :** Once the variables are declared , we define the objective function that is aimed to minimize. This objective function calculates the negative sum of product of weights and quantities, representing the total weight of packaged items. Minimizing this function will lead to an optimal distribution of quantities to optimize total weight while meeting the constraints.

4) Optimization Process : In this step, the 'minimize ' function from the scipy.optimize module is utilized to perform the optimization . The objective function, initial guess for the quantities of the items,bounds for each quantity, and any constraints as arguments to the minimize function are passed . This optimization process aims to find the quantities of items that optimize the total weight while satisfying the specified constraints.

```
    #Minimize total weight used

    # Constraint function: Total weight shouldn't exceed box's capacity
    def weight_constraint(x):
        return  capacity -np.dot(weights, x)  #Return difference between weight capacity and dot product of weights and quantities

    #Non-negative constraint
    bounds = [(0,capacity) for _ in weights]  #Defining bounds for qunatities of each item(non-negative constraint)

    # Initial guess: equal quantities for each item
    x0 = np.full(len(weights), capacity / sum(weights))    #Initialize quantities for each item
    #Provides a more balanced start

    # Optimization using scipy's minimize function
    opt_result = minimize(objective , x0 , bounds = bounds , constraints={'type': 'ineq', 'fun': weight_constraint})

    # Extracting optimized quantities and total weight
    optimized_quantities = opt_result.x
    total_weight = np.dot(weights,optimized_quantities)

    return optimized_quantities, total_weight

# Function to display optimization results
def display_results(optimized_quantities, total_weight ,weights  , capacity  , total_cost , cost_per_unitweight):
```

5) Results : Finally the optimized quantities of each item, total weight of the packaged items, and total cost of packaged items are displayed . The condition is also checked if total weight exceeds the capacity and print a warning message if necessary. This step allows to examine the results of the optimization process and ensure that the solution meets the specified constraints.

```
+ Code  + Text
    print('Optimized quantities of each item:')
    for i, qty in enumerate(optimized_quantities):
        print(f'Item {i+1}: {qty:.3f}(Weight:{weights[i]*qty:.3f})')  #Printing optimized quantties and corresponding weights

    print(f'Total weight of the packaged items: {total_weight:.3f}')
    print(f'Total cost  of the packaged items: {total_cost:.3f}')

    if total_weight > capacity :
              print("Warning : Total weight exceeds the capacity !")   #Printing warning if total weight exceeds capacity
              sys.exit()                                               #Exit the Program1
    else :
              print ("Total weight is within the box's capacity.")     #Printing message if total weight is within capacity


    # Plotting the optimized quantities
    plt.bar(range(1, len(optimized_quantities) + 1), weights*optimized_quantities , color=['blue'])
    plt.xlabel('Item Number')
    plt.ylabel('Quantity')
    plt.title('Optimized Weights  of Each Item')
    plt.grid(True)
    plt.show()
# Calling main function to start the program
if __name__ == "__main__":
    main()    #Execute the main function if the script is executed directly
```

When comparing the by-hand solution to the Python solution , several similarities and differences emerge :

SIMILARITIES :

1) **Objective function :** Both approaches aim to optimize the total weight of the packaged items while meeting the specified constraints. The objective function used in the Python solution aligns with the mathematical formulation used in the by-hand solution .

2) **Optimization Process :** Both solutions utilize optimization techniques to find the optimal quantities of items. While the by-hand solution may involve manual iteration or calculation, the Python solution leverages the ‘minimize’ function from the sci.py optimize module to automate the optimization process.

3) **Constraints :** Both solutions consider constraints such as the weight capacity of the box. The constraints ensure that optimized solutions remain feasible within given constraints.

DIFFERENCES :

1) **Computational Efficiency :** The Python solution offers computational efficiency by automating the optimization process through libraries like ‘numpy’ and ‘scipy’ . This allows for faster optimization,iteration and exploration of different scenarios compared to the manual calculation in by-hand solution.

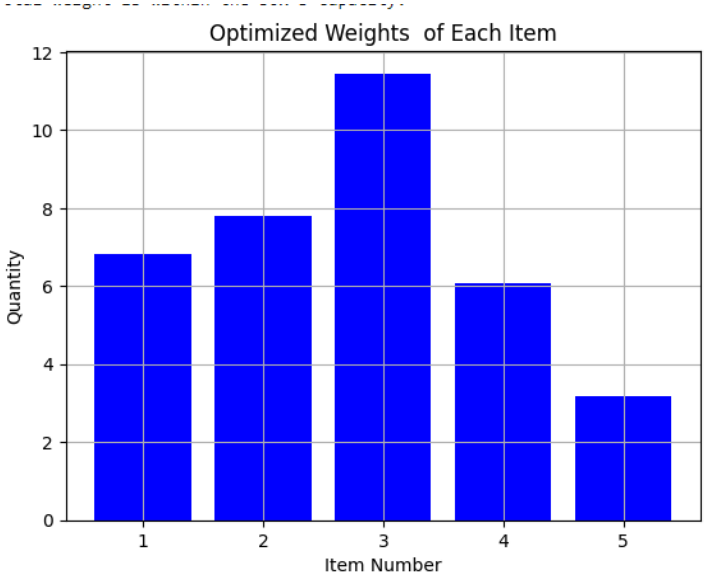
2) **Flexibility :** The Python solution provides greater flexibility in handling complex optimization problems with varying constraints and objectives.It allows for easy modification of parameters and constraints.

3) **Accuracy :** The Python solution can handle large datasets and more complex optimization problems without manual errors or limitations.

MODEL INTERPRETATION :

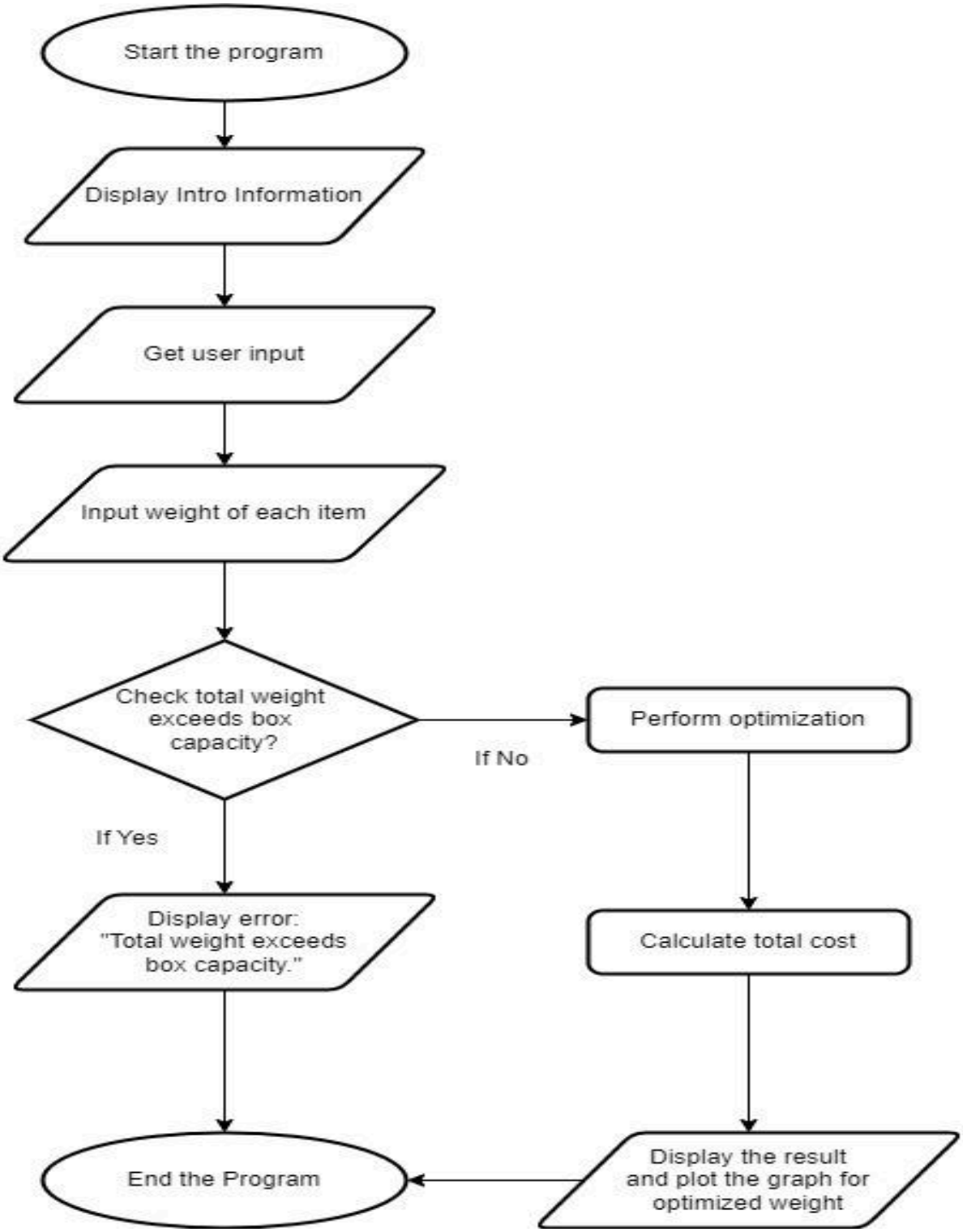
Here is the model interpretation of the above-mentioned points :

```
Welcome to Our Software House
Developed By Team
- Muhammad Noor {ID : I23-2520}
- Shahoud Shahid {ID : I23-2515}
- Saif Shahzad {ID : I23-2634}
Enter the number of items: 5
Enter the weight capacity of the box: 35.321
Enter the cost per unit weight :5.101
Enter the weights  of each item:
Weight of item 1: 3.123
Weight of item 2: 3.567
Weight of item 3: 5.245
Weight of item 4: 2.785
Weight of item 5: 1.456
Optimized quantities of each item:
Item 1: 2.184(Weight:6.819)
Item 2: 2.184(Weight:7.789)
Item 3: 2.184(Weight:11.453)
Item 4: 2.184(Weight:6.081)
Item 5: 2.184(Weight:3.179)
Total weight of the packaged items: 35.321
Total cost  of the packaged items: 180.172
Total weight is within the box's capacity.
```



FLOWCHART REPRESENTATION :

Here is the flowchart representation of the above-mentioned steps which are required to optimize the weight and find the cost :


Start the program
Display Intro Information
Get user input
Input weight of each item
Check total weight exceeds box capacity?
If No
If Yes
Perform optimization
Calculate total cost
Display the result and plot the graph for optimized weight
Display error: "Total weight exceeds box capacity."
End the Program

APPROACH UTILIZING LAGRANGE MULTIPLIER :

- Within the function ‘perform_optimization’,the optimization problem is approached using the Langrange multiplier method , which is inherent to the scipy.optimize.minimize function.

- Specifically, the Lagrange multiplier is employed to handle the inequality constraint, ensuring that the total weight of the selected items does not exceed the capacity of the box . This method effectively incorporates the constraint into the optimization process,enabling the algorithm to find the optimal solution while satisfying the given constraints.

- By utilizing the Lagrange Multiplier approach,the code effectively minimizes the objective function (total weight of selected items) while subject to the constraint (box’s weight capacity),ultimately producing the optimized quantities of each item for packing.

CONCLUSION :

This project focuses on optimizing item selection for packaging within a specified weight capacity .Leveraging the scipy.optimize.minimize function , the assignment systematically optimize total weight while adhering to weight constraints.Through meticulous analysis and implementation,the results offer valuable insights into efficient resource allocation making in logistical scenarios.

Results demonstrate the effectiveness of the solution methodology in effectively allocating resources while maintaining adherence to specific weight capacities. The utilization of the scipy.optimize.minimize function systematic optimization of item quantities, optimizing weight within weight constraints.

. Problem Focus : Addressing the challenge of efficient item selection within weight selection.

CONTRIBUTIONS :

The contribution and the effort being put into this by fellow team members is described below :

- **Muhammad Noor :** Led the Project by establishing the objectives,introducing key goals,and developing the Python Code. Provided detailed comments on the code, compared similarities and differences.and explored the utilization of functions.

- **Shahoud Shahid :** Contributed to the cost-effectiveness analysis,bringing insight into optimizing resource allocation strategies.His expertise enhanced the project's understanding of cost efficiency within logistical scenarios.

- **Saif Shahzad :** Played a crucial role in understanding the mechanism of flowcharts and deciphering the code's flow. His assistance ensured clarity in the project's visualization.

Differences Faced and Overcoming Them :

- **Coordination Challenges :** Coordinating between team members with varying schedules and responsibilities posed a challenge. To overcome this, regular meetings were scheduled to discuss progress, clarify doubts and align on the project's direction.

- **Code Integration Issues :** Integrating individual contributions into a cohesive project presented challenges,especially with version control and merging code. This was addressed by establishing clear naming conventions and documenting changes.

- **Communication Barriers :** Differences in communication in styles and preferences occasionally hindered collaboration . To mitigate this , open communication channels were fostered, and actively listening to each other's perspectives was ,and constructive feedback was encouraged to resolve any queries promptly.

—----------