

SCHOOL OF COMPUTER SCIENCE and ARTIFICIAL INTELLIGENCE

**Program :**B.tech(CSE)

**Specialization :**AIML

**Course Title :**AI Assisted Coding

**Course Code :**24CS002PC215

**Semester :**3rd semester

**Academic Session :**2025-2026

**Name of Student :** Kaveti Manohar

**Enrollment No. :**2403A52079

**Batch No. :**02

**Date :22**/10/2025

#LAB ASSIGNMENT-12.3:

* TASK DESCRIPTION-1:

Task: Write python code for linear\_search() function to search a value in a list and extract it’s index..

* PROMPT:

Write a Python program that defines a function linear\_search() which takes a list and a value to search for, and returns the index of the value if it is found, otherwise returns -1. The program should also ask the user to enter the list elements and the value to search, and then display the result..

* CODE With OUTPUT: A screenshot of a computer

  AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

EXPLANATION:

This code defines a function linear\_search that iterates through a list to find a specific value. It returns the index of the value if found, and -1 otherwise. The program then prompts the user to enter a list of numbers and a value to search for, handles potential errors in the input, and finally calls the linear\_search function to find the value and prints the result.

* TASK DESCRIPTION-2:

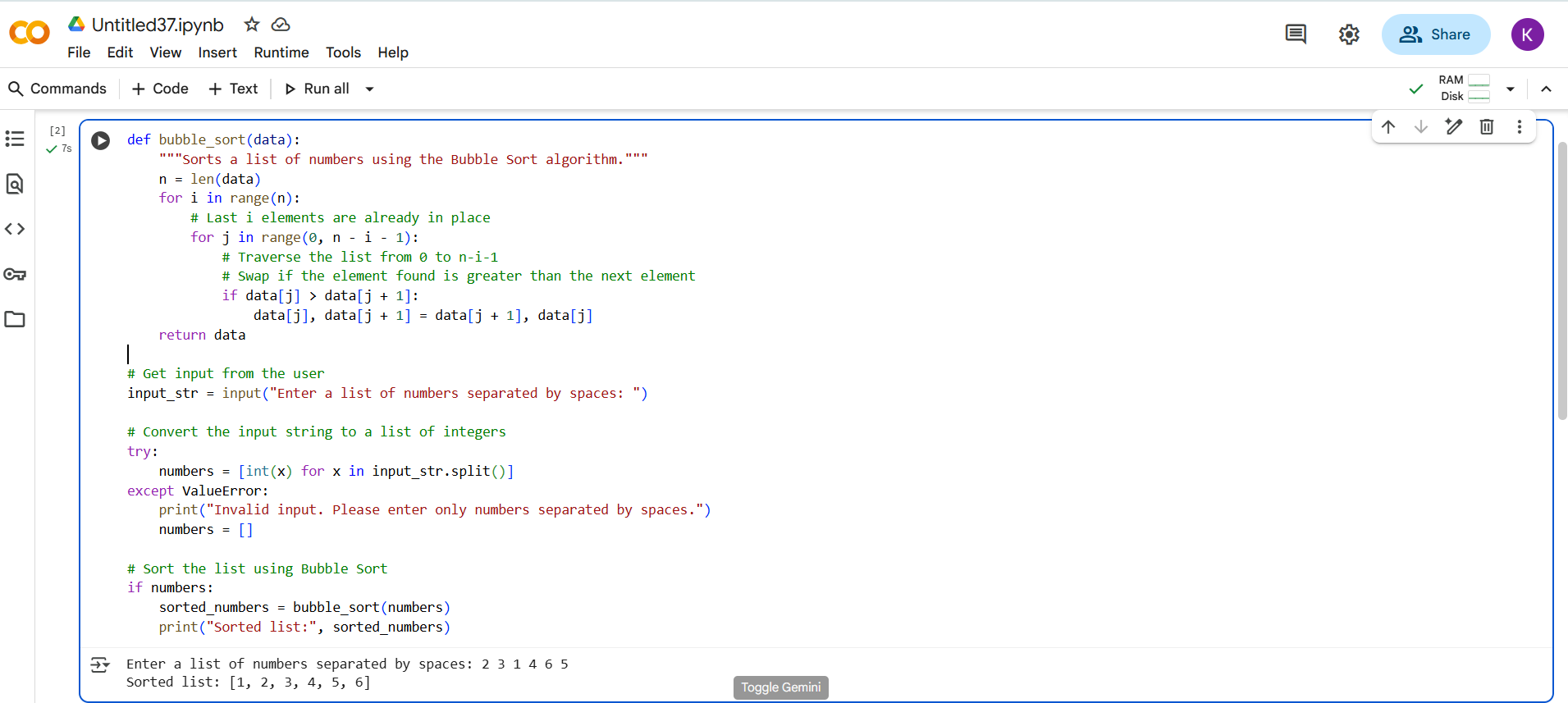
Task: Ask AI to implement Bubble Sort and check sorted output

* PROMPT:

Write a Python program that implements the Bubble Sort algorithm to sort a list of numbers entered by the user. After sorting, display the sorted list as the output.

* QUESTION:
* A screenshot of a computer

  AI-generated content may be incorrect.
* CODE With OUTPUT:

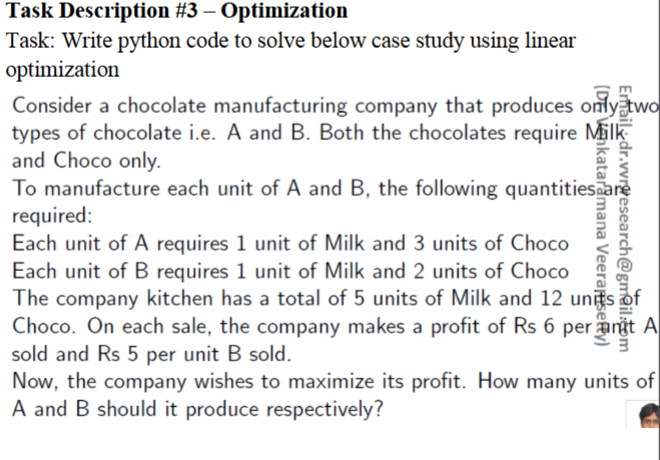


EXPLANATION:

The code implements the **Bubble Sort** algorithm to sort a list of numbers. Here's a breakdown:

1. **bubble\_sort(data) function:**
   * This function takes a list of numbers (data) as input.
   * n = len(data) gets the number of elements in the list.
   * The outer for loop (for i in range(n):) iterates through the list. In each iteration, the largest unsorted element "bubbles up" to its correct position at the end of the unsorted portion.
   * The inner for loop (for j in range(0, n - i - 1):) traverses the unsorted portion of the list.
   * if data[j] > data[j + 1]: compares adjacent elements. If the element on the left is greater than the element on the right, they are swapped (data[j], data[j + 1] = data[j + 1], data[j]).
   * The function returns the sorted list.
2. **Getting user input:**
   * input\_str = input("Enter a list of numbers separated by spaces: ") prompts the user to enter numbers separated by spaces.
   * The try...except block attempts to convert the input string into a list of integers:
     + numbers = [int(x) for x in input\_str.split()] splits the input string by spaces and converts each part to an integer.
     + If a ValueError occurs (meaning the input wasn't valid numbers), an error message is printed, and numbers is set to an empty list.
3. **Sorting and displaying the result:**
   * if numbers: checks if the numbers list is not empty (i.e., the input was valid).
   * sorted\_numbers = bubble\_sort(numbers) calls the bubble\_sort function to sort the list.
   * print("Sorted list:", sorted\_numbers) displays the sorted list to the user.

* TASK DESCRIPTION-3:



* PROMPT:

Write a Python program using linear optimization to maximize the profit of a chocolate manufacturing company that produces two types of chocolates, A and B.

Each unit of A requires 1 unit of Milk and 3 units of Choco.

Each unit of B requires 1 unit of Milk and 2 units of Choco.

The company has a total of 5 units of Milk and 12 units of Choco.

Profit for A is Rs 6 per unit, and profit for B is Rs 5 per unit.

Find how many units of A and B should be produced to maximize profit.

QUESTION:

A screenshot of a computer

AI-generated content may be incorrect.

* CODE With OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

EXPLANANTION:

1. **Import libraries:**
   * import numpy as np: Imports the NumPy library, commonly used for numerical operations. Although not strictly necessary for this specific linprog usage, it's often imported alongside SciPy.
   * from scipy.optimize import linprog: Imports the linprog function specifically from the SciPy optimize module. This function is used to solve linear programming problems.
2. **Define the problem parameters:**
   * c = [-6, -5]: This is the coefficient vector for the objective function. Since linprog minimizes by default, and we want to maximize profit, we negate the profit values (Rs 6 for A and Rs 5 for B).
   * A\_ub = [[1, 1], [3, 2]]: This is the matrix of coefficients for the inequality constraints. The first row [1, 1] represents the milk constraint (1 unit of milk for A, 1 unit for B). The second row [3, 2] represents the chocolate constraint (3 units of chocolate for A, 2 units for B).
   * b\_ub = [5, 12]: This is the right-hand side vector for the inequality constraints. 5 is the total available units of milk, and 12 is the total available units of chocolate.
   * x0\_bounds = [(0, None), (0, None)]: These are the bounds for the decision variables (the number of units of A and B). (0, None) means that the number of units for both chocolate A and chocolate B must be greater than or equal to 0 (you can't produce a negative number of chocolates), and there is no upper limit specified by this bound itself (though the resource constraints will implicitly create limits).
3. **Solve the linear programming problem:**
   * result = linprog(c, A\_ub=A\_ub, b\_ub=b\_ub, bounds=x0\_bounds): This is the core of the code. It calls the linprog function with the defined parameters to find the optimal solution that minimizes -6A - 5B (which is equivalent to maximizing 6A + 5B) subject to the given constraints.
4. **Access and print the results:**
   * optimal\_a = result.x[0]: The result object contains the solution. result.x is an array holding the values of the decision variables at the optimum. result.x[0] is the optimal number of units of chocolate A.
   * optimal\_b = result.x[1]: result.x[1] is the optimal number of units of chocolate B.
   * max\_profit = -result.fun: result.fun is the value of the objective function at the optimum. Since we minimized the negative of the profit, we negate result.fun to get the maximum profit.
   * The print statement formats and displays the optimal number of units for A and B, and the calculated maximum profit. The .0f and .2f format specifiers are used to display the numbers with no decimal places for units and two decimal places for profit, respectively.

* TASK DISCRIPTION-4:

Task: Write python code to find value of x at which the function  
f(x)=2X3+4x+5 will be minimum.

* PROMPT:

Write python code to find value of x at which the function  
f(x)=2X3+4x+5 will be minimum.

QUESTION:

A screenshot of a chat

AI-generated content may be incorrect.

* CODE With OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

EXPLANATION:

1. **from scipy.optimize import minimize**: This line imports the minimize function from the scipy.optimize module. This function is a general-purpose minimizer for scalar functions of one or more variables.
2. **def objective\_function(x):**: This defines the function that we want to find the minimum of. It takes a single argument x and returns the value of $2x^3 + 4x + 5$$2x^3 + 4x + 5$.
3. **result = minimize(objective\_function, 0)**: This is where the optimization happens.
   * minimize: The function being called to perform the minimization.
   * objective\_function: The function we want to minimize.
   * 0: This is the initial guess for the value of x where the minimum might occur. The optimization algorithm starts its search from this point.
4. **min\_x = result.x[0]**: After the minimize function runs, it returns a result object. result.x is an array containing the optimized value(s) of the variable(s). Since our function only has one variable (x), result.x[0] gives us the value of x that the optimizer found.
5. **min\_value = result.fun**: result.fun contains the value of the objective function (objective\_function) at the optimized value of x found by the minimize function.
6. **print(...) statements**: These lines print the results in a user-friendly format, showing the value of x where the minimum was found and the corresponding function value. The .4f formatting ensures that the numbers are displayed with four decimal places.
7. **print("\nFull optimization result:") print(result)**: This part prints the entire result object. This object contains more detailed information about the optimization process, such as whether the optimization was successful, the number of iterations, and other diagnostic messages. In this specific case, the output shows success: False and a message about precision loss, indicating that the minimizer did not find a true minimum, which is expected for this particular cubic function as it doesn't have a global minimum.

---------------Thank You------------