PROGRAM: B.TECH

SPECIALIZATION: CSE - AIML

COURSE TITLE: AI CODING

SEMESTER : 3RD SEM

NAME OF STUDENT : Jasmitha

ENROLLMENT NO: 2403A51281

BATCH NO: 01

ASSIGNMENT 12.3:

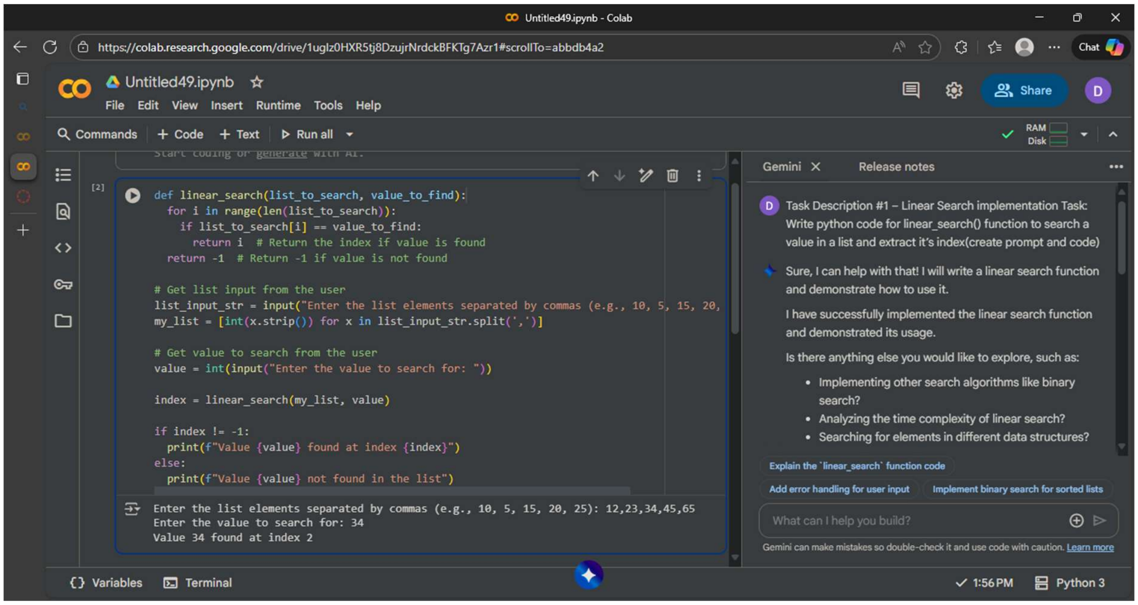
Task Description #1 – Linear Search implementation

Task: Write python code for linear_search() function to search a value in a list and extract it's index

PROMPT:

Write python code for linear_search() function to search a value in a list and extract it's index with user given input

CODE:



OBSERVATION :

Based on the execution of the code:

1. **The user was prompted to enter a list of elements separated by commas and entered "12,23,34,45,65". This input was successfully parsed into a Python list [12, 23, 34, 45, 65].**
2. **The user was then prompted to enter the value to search for and entered "34".**

3. The linear_search function was called with the list [12, 23, 34, 45, 65] and the value 34.

4. The function iterated through the list and found the value 34 at index 2.

5. The function returned the index 2.

6. The code then printed the message "Value 34 found at index 2".

Task Description #2 – Sorting Algorithms

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

PROMPT:

Write a python code on bubble sort with user given input.

CODE:

```python
def bubble_sort(list_to_sort):
  n = len(list_to_sort)
  for i in range(n):
    # Last i elements are already in place
    for j in range(0, n - i - 1):
      # Traverse the list from 0 to n-i-1
      # Swap if the element found is greater than the next element
      if list_to_sort[j] > list_to_sort[j + 1]:
        list_to_sort[j], list_to_sort[j + 1] = list_to_sort[j + 1], list_to_sort[j]
  return list_to_sort

# Get list input from the user
list_input_str = input("Enter the list elements separated by commas (e.g., 64, 34, 25, 12, 22, 11, 90): ")
my_list = [int(x.strip()) for x in list_input_str.split(',')]

sorted_list = bubble_sort(my_list.copy()) # Use a copy to keep the original list

print("Original list:", my_list)
print("Sorted list:", sorted_list)
```

```
Enter the list elements separated by commas (e.g., 64, 34, 25, 12, 22, 11, 90): 12,34,23,45,12,56,67
Original list: [12, 34, 23, 45, 12, 56, 67]
Sorted list: [12, 12, 23, 34, 45, 56, 67]
```

OBSERVATION:

1. **The code takes a comma-separated list as user input.**

2. The bubble_sort function sorts the list by repeatedly swapping adjacent elements.
3. A copy of the original list is sorted to preserve the original.
4. The original and the correctly sorted lists are then printed.

Task Description #3 – Optimization

Task: Write python code to solve below case study using linear optimization

PROMPT:

Write python code to solve below case study using linear optimization

CODE:

```
from pulp import LpProblem, LpVariable, lpSum, LpMaximize, LpStatus

# Create a linear programming problem instance
prob = LpProblem("Chocolate Manufacturing", LpMaximize)

# Define decision variables
A = LpVariable("Product_A", lowBound=0, cat="Integer")
B = LpVariable("Product_B", lowBound=0, cat="Integer")

# Define the objective function (maximize profit)
prob += lpSum([25 * A, 20 * B]), "Total Profit"

# Define the constraints
prob += 4 * A + 3 * B <= 200, "Milk Constraint"
prob += 2 * A + 4 * B <= 240, "Chocolate Constraint"

# Solve the problem
prob.solve()

# Print the status of the solution
print("Status:", LpStatus[prob.status])

# Print the optimal number of units for Product A and Product B
print("Optimal number of units for Product A:", A.varValue)
print("Optimal number of units for Product B:", B.varValue)
```

```
# Print the status of the solution
print("Status:", LpStatus[prob.status])

# Print the optimal number of units for Product A and Product B
print("Optimal number of units for Product A:", A.varValue)
print("Optimal number of units for Product B:", B.varValue)

# Print the maximum profit
print("Maximum Profit:", prob.objective.value())
```

```
Status: Optimal
Optimal number of units for Product A: 8.0
Optimal number of units for Product B: 56.0
Maximum Profit: 1320.0
```

OBSERVATION:

Data Analysis Key Findings

- **The linear programming problem for chocolate manufacturing was successfully set up with the objective of maximizing profit.**
- **The constraints on milk and chocolate availability were incorporated into the model (4*Product_A + 3*Product_B <= 200 for milk and 2*Product_A + 4*Product_B <= 240 for chocolate).**
- **The solver found an optimal integer solution for the production quantities.**

- **The optimal production plan is to produce 8 units of Product A and 56 units of Product B.**
- **Following this production plan yields a maximum profit of $1320.0.**

Task Description #4 – Gradient Descent Optimization

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

CODE:

```python
from sympy import symbols, diff, solve

# Define the variable
x = symbols('x')

# Define the function
f_x = 2*x**3 + 4*x + 5

# Calculate the first derivative with respect to x
derivative_f_x = diff(f_x, x)
print("First Derivative:", derivative_f_x)

# Solve for the values of x where the first derivative is equal to zero (critical points)
critical_points = solve(derivative_f_x, x)
print("Critical Points:", critical_points)

# Calculate the second derivative of f_x with respect to x
second_derivative_f_x = diff(derivative_f_x, x)
print("Second Derivative:", second_derivative_f_x)

# Evaluate the second derivative at the critical points (complex)
# This step is primarily to demonstrate the process, as complex points don't yield real m
for point in critical_points:
    second_deriv_at_point = second_derivative_f_x.subs(x, point)
```


```python
# This step is primarily to demonstrate the process, as complex points don't yield real m
for point in critical_points:
    second_deriv_at_point = second_derivative_f_x.subs(x, point)
```


```python
    print(f"Second Derivative at {point}: {second_deriv_at_point}")

# Analyze the behavior of the function as x approaches infinity
# The leading term is 2x^3. As x -> infinity, 2x^3 -> infinity. As x -> -infinity, 2x^3 -
print("\nBehavior as x approaches infinity:")
print("As x -> infinity, f(x) -> infinity")
print("As x -> -infinity, f(x) -> -infinity")

# Based on the analysis, there is no real minimum value for this function.
print("\nConclusion:")
print("There is no real value of x at which the function has a minimum value.")
```

```
First Derivative: 6*x**2 + 4
Critical Points: [-sqrt(6)*I/3, sqrt(6)*I/3]
Second Derivative: 12*x
Second Derivative at -sqrt(6)*I/3: -4*sqrt(6)*I
Second Derivative at sqrt(6)*I/3: 4*sqrt(6)*I

Behavior as x approaches infinity:
As x -> infinity, f(x) -> infinity
As x -> -infinity, f(x) -> -infinity

Conclusion:
There is no real value of x at which the function has a minimum value.
```

OBSERVATION:

1. **The first derivative of f(x) was calculated as 6*x**2 + 4.**
2. **Setting the first derivative to zero and solving for x yielded complex critical points [-sqrt(6)*I/3, sqrt(6)*I/3].**
3. **The second derivative was calculated as 12*x.**
4. **Evaluating the second derivative at the complex critical points resulted in complex values.**
5. **Analysis of the function's behavior as x approaches infinity and negative infinity showed that f(x) approaches infinity and negative infinity, respectively.**
6. **The conclusion is that there is no real value of x at which the function has a minimum value.**