AIAC Assignment 2.3

Name: M Goutham

HT No : 2303A52010

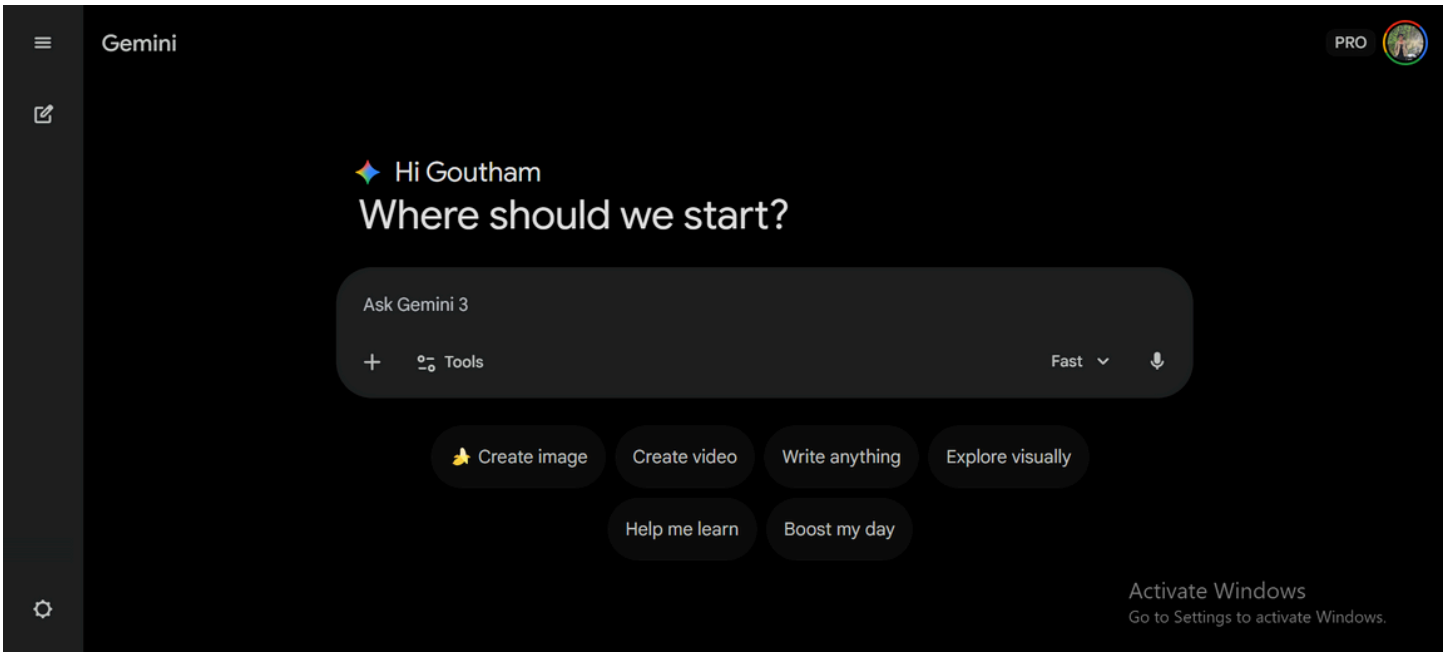
Batch 31

Question

Task 1:

Word Frequency from Text File

Scenario: You are analyzing log files for keyword frequency. Task: Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code

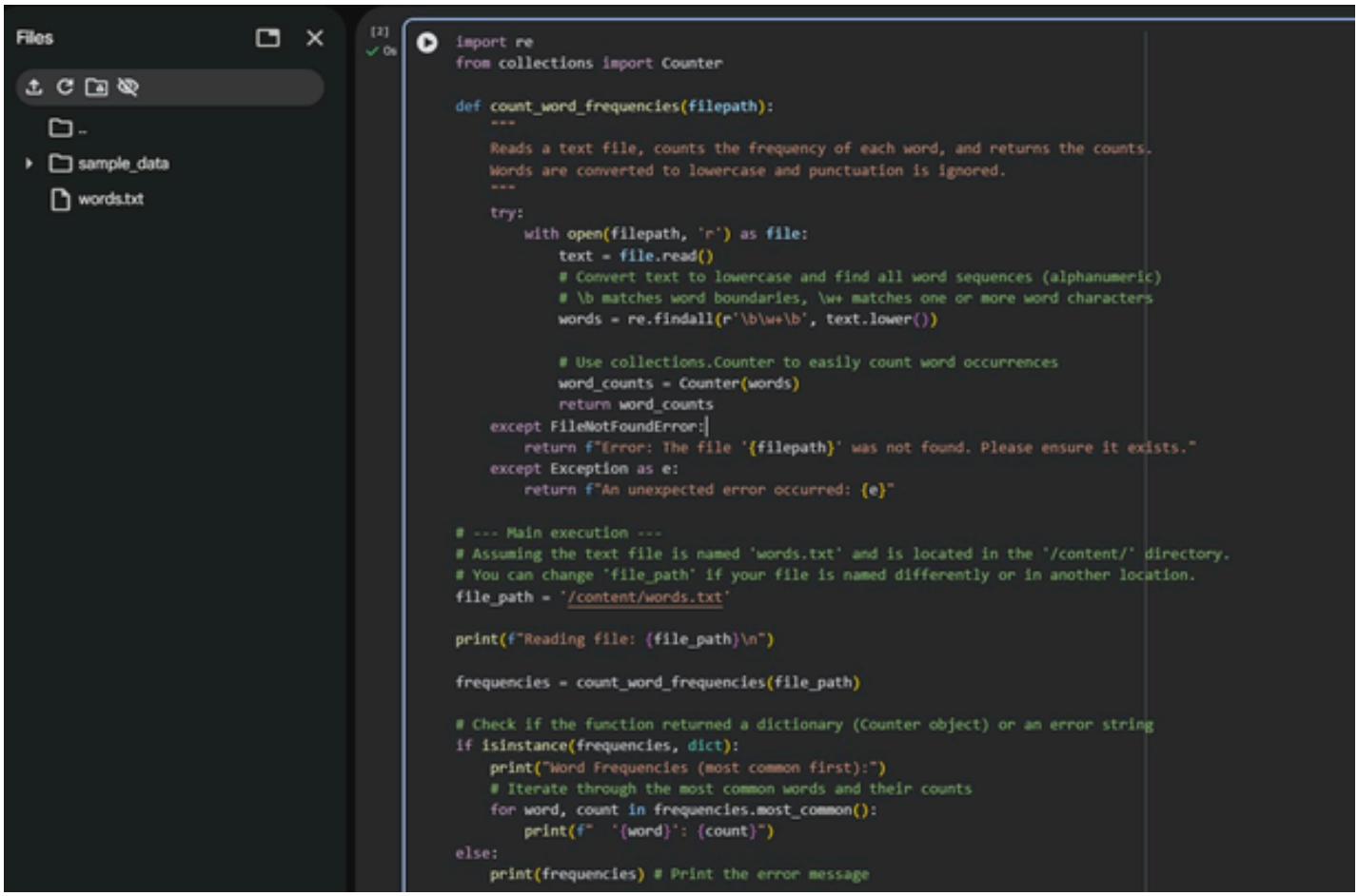


Prompt:

Create a Python program that creates a text file, writes sample text into it, then reads the file and displays its contents.

Add clear comments in the code.

code screenshot



Output:

```
Word Frequencies (most common first):
  'apple': 4
  'banana': 3
  'orange': 3
  'dog': 3
  'cat': 2
```

Explanation:

In this program, a text file is read and the content is split into a list of words.

A dictionary is used to store them, where each entry maps a word to its number of occurrences.

Repeated words have their count incremented. At the end, the program prints how many times each word is found in the file.

Question

Task 2:

File Operations Using Cursor AI

Scenario: You are automating basic file operations.

Task: Use Cursor AI to generate a program that:

Creates a text file

Writes sample text

Reads and displays the content

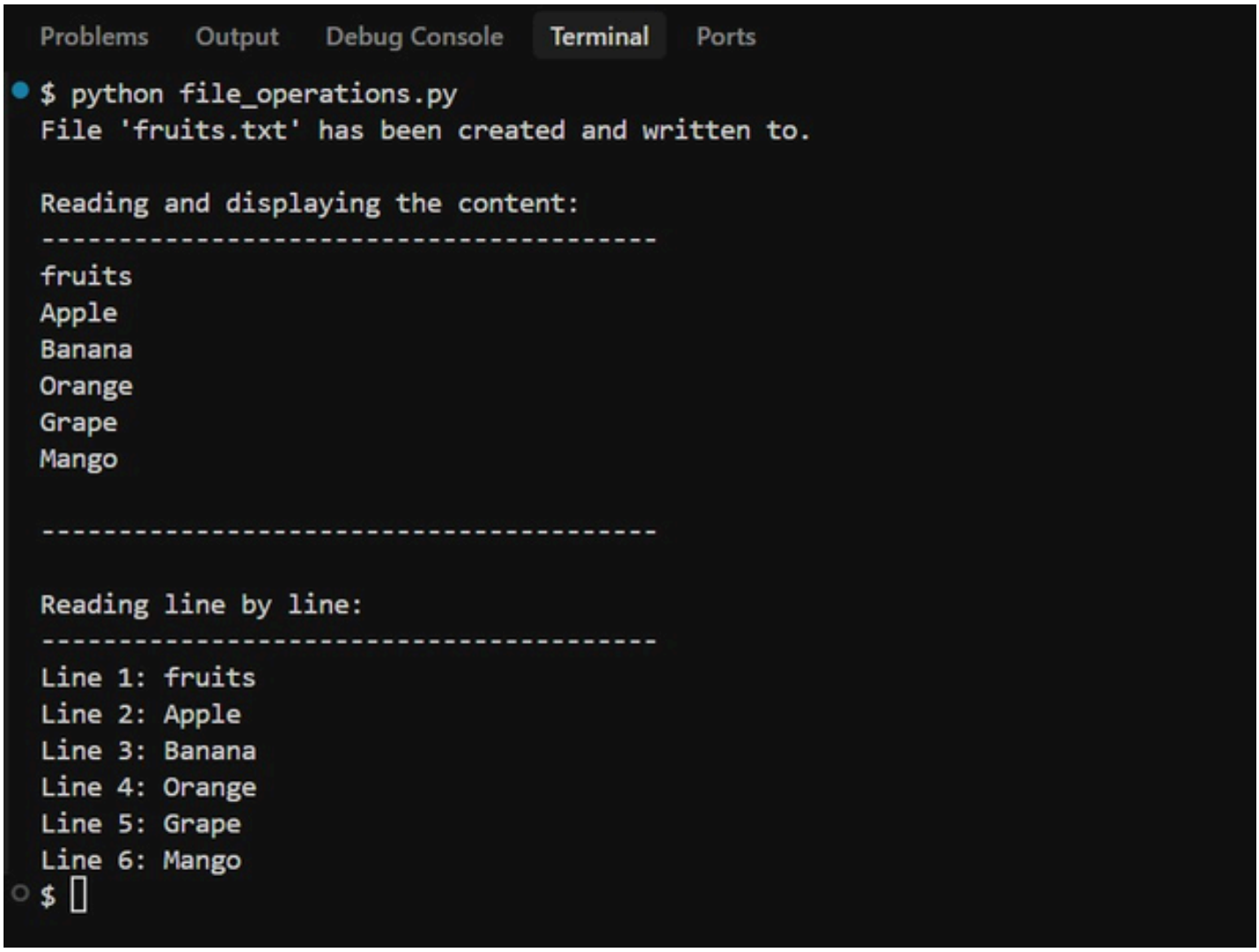
prompt:

generate a python program, which creates a text file, write sample txt eg.fruits in that, read and display that content.

Code screenshot:

```
# Create and write to a text file
filename = "fruits.txt"

# Write sample text to the file
with open(filename, 'w') as file:
    file.write("fruits\n")
    file.write("Apple\n")
    file.write("Banana\n")
    file.write("Orange\n")
    file.write("Grape\n")
    file.write("Mango\n")

print(f"File '{filename}' has been created and written to.\n")

# Read and display the content
print("Reading and displaying the content:")
print("-" * 40)
with open(filename, 'r') as file:
    content = file.read()
    print(content)
print("-" * 40)

# Alternative: Read line by line
print("\nReading line by line:")
print("-" * 40)
with open(filename, 'r') as file:
    for line_num, line in enumerate(file, 1):
        print(f"Line {line_num}: {line.strip()}")
```

Output:



Explanation:

In this program, Python's cursor is used to demonstrate fundamental file handling.

A text file is created and filled with sample text, then reopened in read mode so its data can be displayed.

Question

Task 3: CSV Data Analysis

Scenario: You are processing structured data from a CSV file. Task: Use Gemini in Colab to read a CSV file and calculate mean, min, and max.

Prompt:

Generate Python code to read a CSV file and calculate the mean, minimum, and maximum values of a numeric column.

Code :

```python
import pandas as pd
import io

# Uncomment the lines below and replace 'your_file.csv' with your CSV file path
try:
    df = pd.read_csv('scores.csv')
except FileNotFoundError:
    print("Error: CSV file not found. Please check the path.")
    exit()

# Specify the numeric column you want to analyze
column_name = 'score'

# Check if the column exists in the DataFrame
if column_name in df.columns:
    # Ensure the column is numeric (e.g., float or int)
    # pd.to_numeric will convert values to numeric, coercing errors to NaN
    numeric_column = pd.to_numeric(df[column_name], errors='coerce')

    # Drop rows where the numeric conversion resulted in NaN (non-numeric values)
    numeric_column = numeric_column.dropna()

    if not numeric_column.empty:
        # Calculate mean, minimum, and maximum
        mean_value = numeric_column.mean()
        min_value = numeric_column.min()
        max_value = numeric_column.max()

        print(f"Statistics for column '{column_name}':")
        print(f"  Mean: {mean_value:.2f}")
        print(f"  Minimum: {min_value:.2f}")
        print(f"  Maximum: {max_value:.2f}")
    else:
        print(f"Column '{column_name}' contains no valid numeric data after cleaning.")
else:
    print(f"Error: Column '{column_name}' not found in the CSV file. Available columns: {df.columns.tolist()}")
```

Output:

```
Statistics for column 'score':
  Mean: 82.00
  Minimum: 67.00
  Maximum: 92.00
```

Explanation:

This Python program opens a CSV file and collects numeric data from one of its columns.

It then computes statistics such as the average, minimum, and maximum values.

Analyzing CSV data is common in data science and various analytics tasks.

Question

Task 4: Sorting Lists Manual vs Built-in

Scenario: You are reviewing algorithm choices for efficiency.

Task: Use Gemini to generate:

Bubble sort

Python’s built-in sort() Compare both implementations.

code:

Manual Bubble Sort Code:

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

# --- Example Usage ---
my_list = [64, 34, 25, 12, 22, 11, 90]
print("Original list:", my_list)

sorted_list = bubble_sort(my_list)
print("Sorted list using Bubble Sort:", sorted_list)

my_list_2 = [5, 1, 4, 2, 8]
print("\nOriginal list 2:", my_list_2)

sorted_list_2 = bubble_sort(my_list_2)
print("Sorted list 2 using Bubble Sort:", sorted_list_2)
```

Sorting using sort() function:

output

Bubble sort:

```
Original list: [64, 34, 25, 12, 22, 11, 90]
Sorted list using Bubble Sort: [11, 12, 22, 25, 34, 64, 90]

Original list 2: [5, 1, 4, 2, 8]
Sorted list 2 using Bubble Sort: [1, 2, 4, 5, 8]
```

Sort Function:

```
Original list (list.sort()): [64, 34, 25, 12, 22, 11, 90]
Sorted list (list.sort()): [11, 12, 22, 25, 34, 64, 90]

Original list (descending): ['banana', 'apple', 'cherry', 'date']
Sorted list (descending): ['date', 'cherry', 'banana', 'apple']
```

Explanation

In bubble sort, elements next to each other are checked and swapped over and over until the list is sorted. This method is straightforward but slow when the input size grows. Python's built-in sorting method is faster, more compact, and highly optimized.