AI Assistant Coding

Assignment 1.2
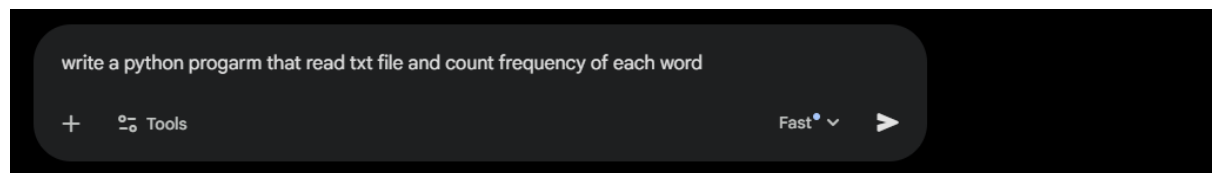
Name : V. Hansika   **HT. No :** 2303A52409 **Batch:** 32

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:

Code:

```python
import string
from collections import Counter
import os

def count_word_frequency(file_path):
    """Reads a file and returns a frequency count of each word."""
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            # 1. Read and lowercase
            text = file.read().lower()

            # 2. Remove punctuation (keeps words clean)
            text = text.translate(str.maketrans('', '', string.punctuation))

            # 3. Split into individual words
            words = text.split()

            # 4. Count occurrences using Counter
            return Counter(words)

    except FileNotFoundError:
        return "Error: The file was not found."
    except Exception as e:
        return f"An error occurred: {e}"

# --- EXECUTION ---
if __name__ == "__main__":
    target_file = 'logfile.txt'

    # Get the counts
    results = count_word_frequency(target_file)

    # Display results
    if isinstance(results, Counter):
        print(f"{'WORD':<15} | {'FREQUENCY'}")
        print("-" * 30)

        # .most_common() sorts them from highest to lowest automatically
        for word, count in results.most_common():
            print(f"{word:<15} | {count}")
    else:
        print(results)
```

Output:

```
PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/Activate.ps1
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/word.py
WORD            | FREQUENCY
----------------------------
python          | 4
is              | 3
and             | 2
an              | 1
amazing         | 1
programming     | 1
language        | 1
versatile       | 1
easy            | 1
to              | 1
learn           | 1
powerful        | 1
many            | 1
people          | 1
use             | 1
for             | 1
data            | 1
science         | 1
web             | 1
development     | 1
automation      | 1
learning        | 1
a               | 1
great           | 1
choice          | 1
(AIAC_env) PS D:\Course\AIAC>
```

Explanation:

This program reads a text file and splits the content into words. Each word is stored in a dictionary where the key is the word and the value is its count. If a word appears again, its count is increased. Finally, the program prints how many times each word appears in the file. This helps in analyzing text data and finding frequently used words.
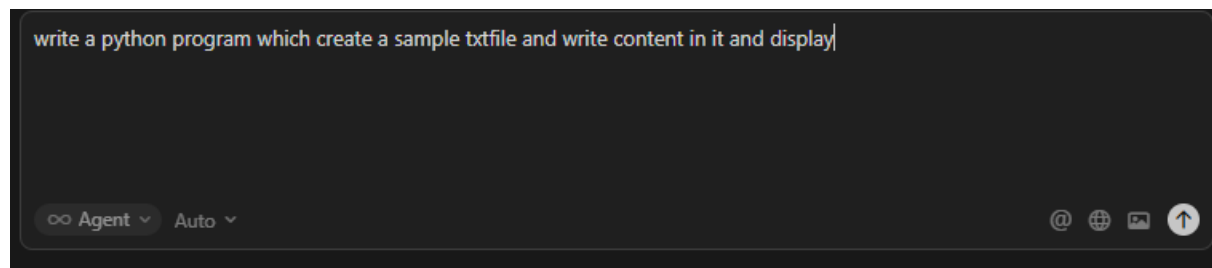
Task 2: File Operations Using Cursor AI
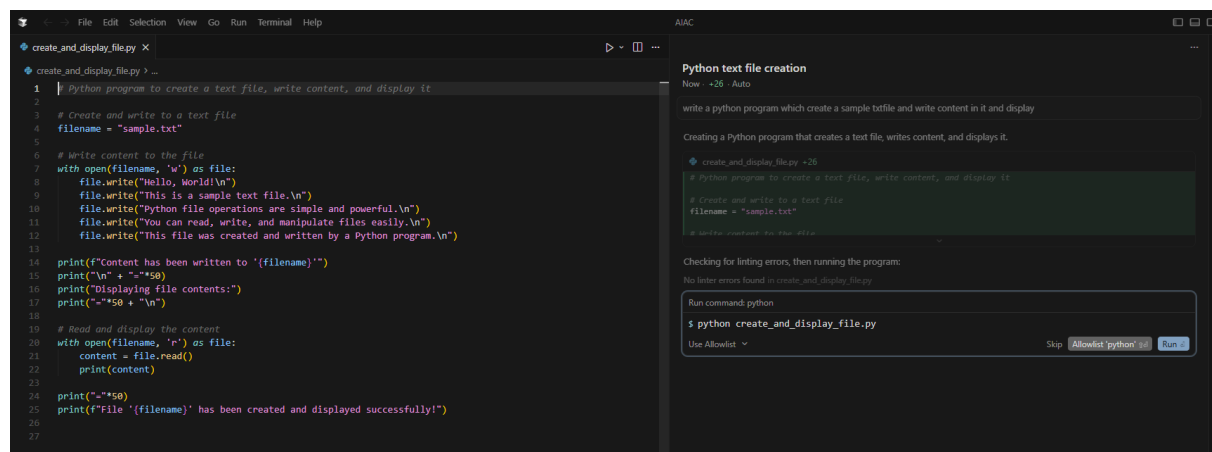
Scenario: You are automating basic file operations.

Task: Use Cursor AI to generate a program that:

➢ Creates a text file

➢ Writes sample text
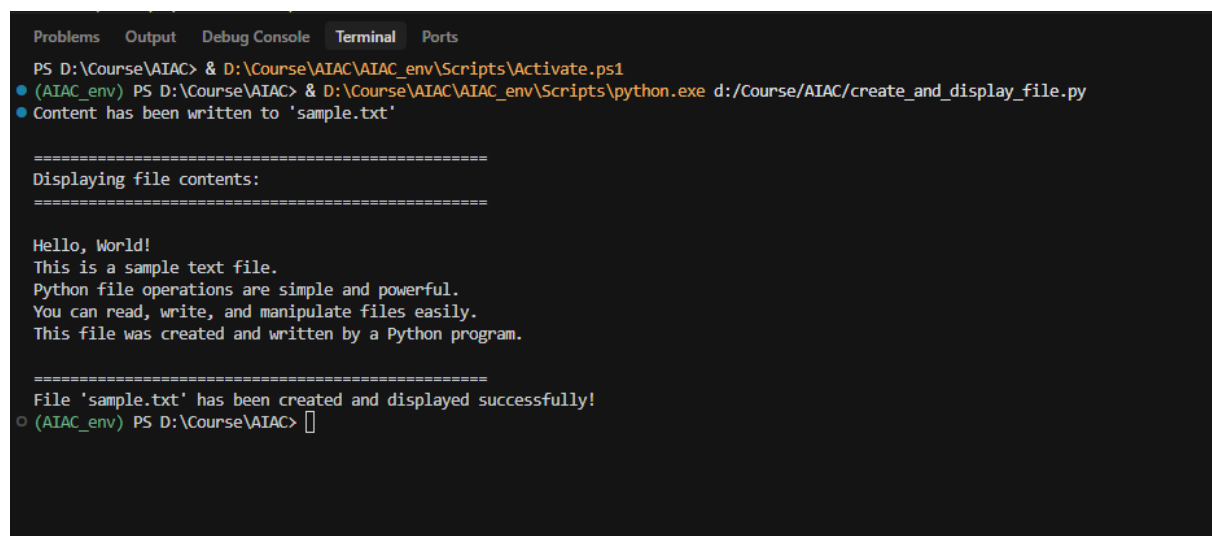
➢ Reads and displays the content

Prompt:

write a python program which create a sample txtfile and write content in it and display

∞ Agent ⌄   Auto ⌄   @ ⊕ ▭ ↑
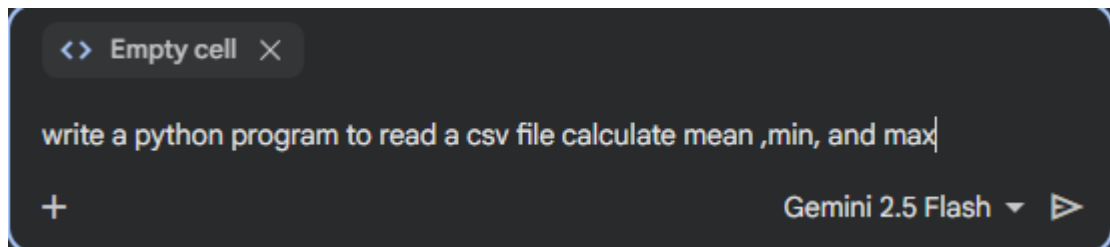
Code:

Output:

Explanation:

This program demonstrates basic file handling in Python using Cursor. First, a text file is created and sample text is written into it. Then, the same file is opened in read mode and its contents are displayed on the screen. It shows how Python can be used to create, write, and read files easily. Such operations are useful in automation and data storage tasks.

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:

```
<> Empty cell  ✕

write a python program to read a csv file calculate mean ,min, and max

+                                    Gemini 2.5 Flash ▼  ▷
```

Code:

```python
import pandas as pd

# Read the CSV file
df_sales = pd.read_csv("sales_data.csv")

# Calculate mean, min, and max of the 'Sales' column
mean_sales = df_sales["Sales"].mean()
min_sales = df_sales["Sales"].min()
max_sales = df_sales["Sales"].max()

# Print the results
print(f"Mean Sales: {mean_sales:.2f}")
print(f"Min Sales: {min_sales}")
print(f"Max Sales: {max_sales}")
```

Output:

```
Mean Sales: 154.29
Min Sales: 100
Max Sales: 200
```

Explanation:

This program reads data from a CSV file using Python. It extracts numerical values from a column and calculates the mean, minimum, and maximum. CSV analysis is used in data processing and analytics applications.
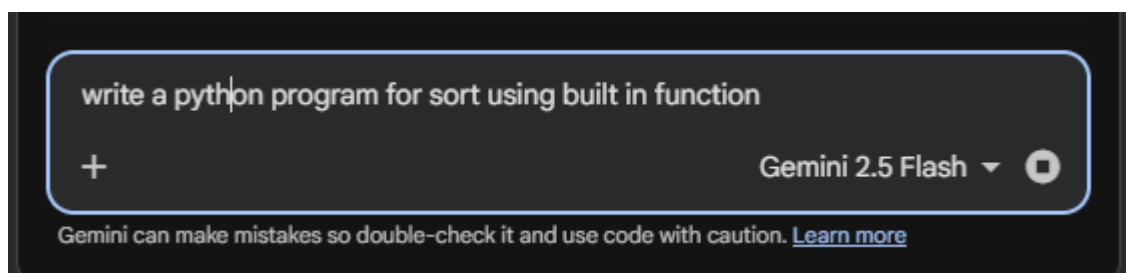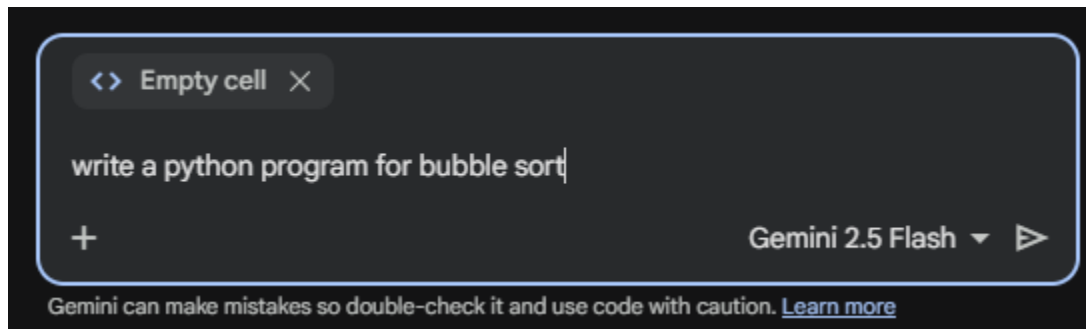

Task 4: Sorting Lists – Manual vs Built-in

Scenario: You are reviewing algorithm choices for efficiency.

Task: Use Gemini to generate:

➢ Bubble sort

➢ Python's built-in sort()

➢ Compare both implementations.

Prompt:

Code:

Bubble sort:

```python
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]
    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:", sorted_list)
```

Sort function:

```python
# Define a list to be sorted
my_list = [64, 34, 25, 12, 22, 11, 90]

print("Original list:", my_list)

# Use the built-in sorted() function to get a new sorted list
sorted_list_builtin = sorted(my_list)

print("Sorted list (using sorted()):", sorted_list_builtin)

# Alternatively, to sort the list in-place, you can use the .sort() method:
another_list = [5, 2, 8, 1, 9]
print("\nAnother list (original):", another_list)
another_list.sort()
print("Another list (sorted in-place with .sort()):", another_list)
```

Output:

Bubble sort

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

Sort function:

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

Another list (original): [5, 2, 8, 1, 9]
Another list (sorted in-place with .sort()): [1, 2, 5, 8, 9]
```

Explanation:

Bubble sort is a simple sorting algorithm that repeatedly compares and swaps adjacent elements. It is easy to understand but inefficient for large data sets. Python's built-in sort function is shorter, optimized and much faster. The built-in method should be preferred in real-world applications.