

# SR UNIVERSITY

## AI ASSIST CODING

### **LAB-2:** Exploring Additional AI Coding Tools – Gemini (Colab) and Cursor AI

**Name :**P. SUSMIJA

**Pin No:**2503A51L11

#### **Lab Objectives:**

- To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab.
- To understand and use Cursor AI for code generation, explanation, and refactoring.
- To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI.
- To perform code optimization and documentation using AI tools.

#### **Lab Outcomes (LOs):**

After completing this lab, students will be able to:

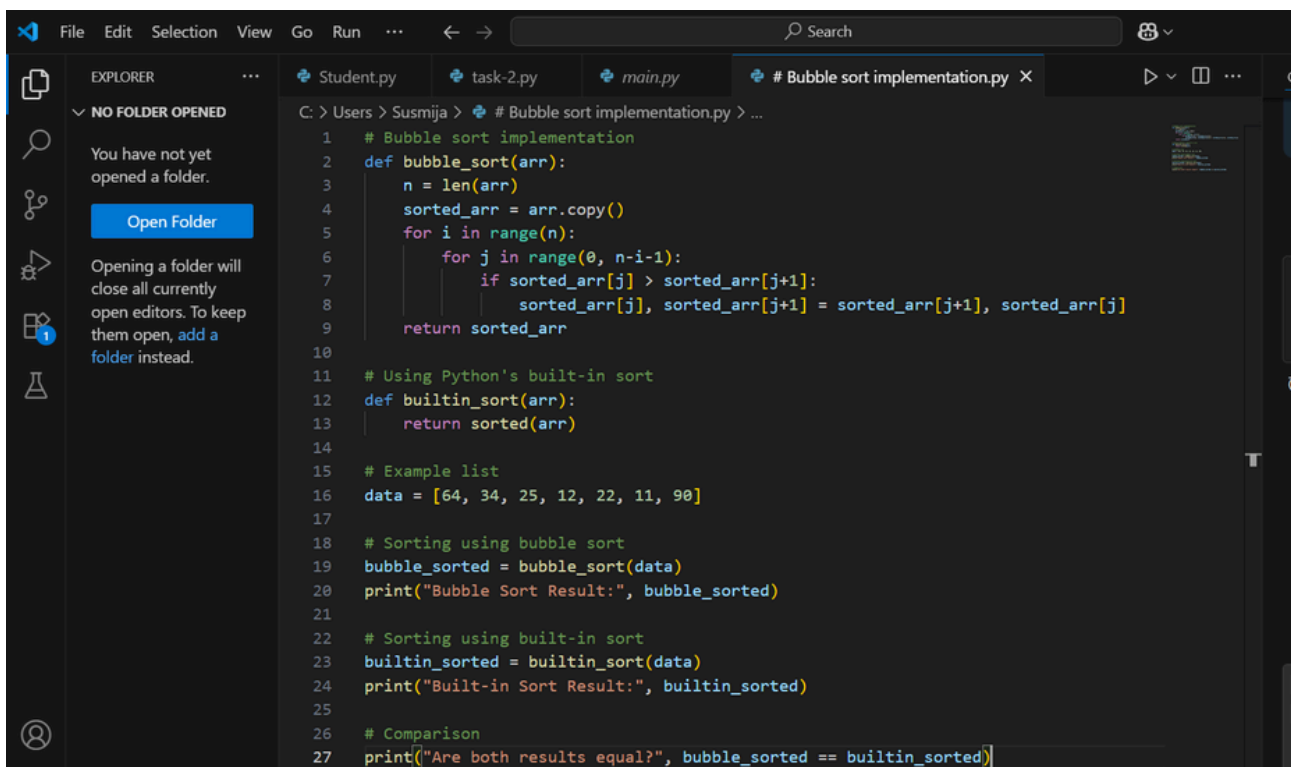
- Generate Python code using Google Gemini in Google Colab.
- Analyze the effectiveness of code explanations and suggestions by Gemini.
- Set up and use Cursor AI for AI-powered coding assistance.
- Evaluate and refactor code using Cursor AI features.
- Compare AI tool behavior and code quality across different platforms.

#### **TASK #1:**

##### **Prompt:**

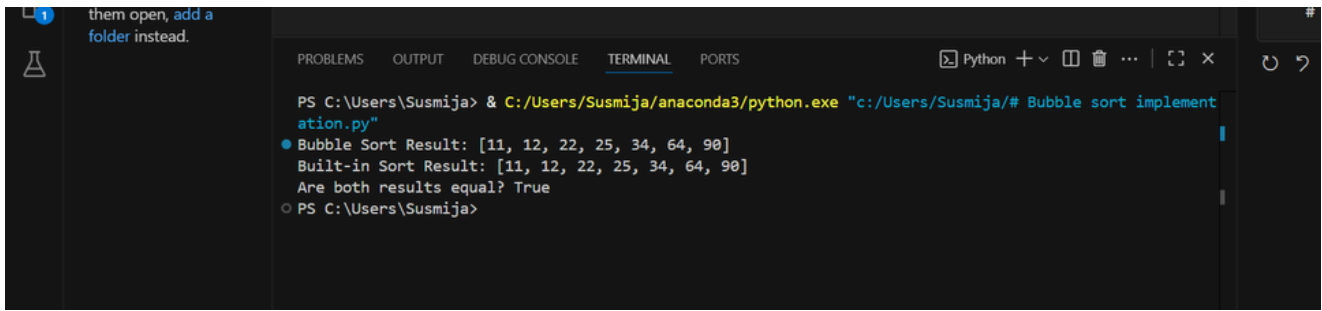
- Open Google Colab and use Google Gemini to generate Python code that performs sorting of a list using both the bubble sort algorithm and Python's built-in sort() function. Compare the two implementations.

#### **Code Generated:**

A screenshot of a code editor interface, likely Cursor AI, showing a Python file named 'Bubble sort implementation.py'. The code implements two sorting functions: 'bubble\_sort' and 'builtin\_sort'. The 'bubble\_sort' function uses a nested loop to compare and swap elements in an array. The 'builtin\_sort' function uses Python's built-in 'sorted' function. An example list 'data = [64, 34, 25, 12, 22, 11, 90]' is provided. The code then sorts the data using both methods and prints the results. A comparison is made at the end to check if both results are equal.

```
1 # Bubble sort implementation
2 def bubble_sort(arr):
3     n = len(arr)
4     sorted_arr = arr.copy()
5     for i in range(n):
6         for j in range(0, n-i-1):
7             if sorted_arr[j] > sorted_arr[j+1]:
8                 sorted_arr[j], sorted_arr[j+1] = sorted_arr[j+1], sorted_arr[j]
9     return sorted_arr
10
11 # Using Python's built-in sort
12 def builtin_sort(arr):
13     return sorted(arr)
14
15 # Example list
16 data = [64, 34, 25, 12, 22, 11, 90]
17
18 # Sorting using bubble sort
19 bubble_sorted = bubble_sort(data)
20 print("Bubble Sort Result:", bubble_sorted)
21
22 # Sorting using built-in sort
23 builtin_sorted = builtin_sort(data)
24 print("Built-in Sort Result:", builtin_sorted)
25
26 # Comparison
27 print("Are both results equal?", bubble_sorted == builtin_sorted)
```

#### **Output After executing Code:**



```
PS C:\Users\Susmija> & C:/Users/Susmija/anaconda3/python.exe "c:/Users/Susmija/# Bubble sort implementation.py"
● Bubble Sort Result: [11, 12, 22, 25, 34, 64, 90]
Built-in Sort Result: [11, 12, 22, 25, 34, 64, 90]
Are both results equal? True
○ PS C:\Users\Susmija>
```

## Your Observations:

### 1. Correct Implementation of Bubble Sort:

- The function `bubble_sort(arr)` correctly implements the bubble sort algorithm using nested loops and value swapping.

### 2. Data Integrity Preserved:

- The function uses `arr.copy()` to avoid modifying the original list, which is a good practice.

### 3. Clear Comparison with Built-in Sort:

- The code includes a custom sort function and compares its result with Python's built-in `sorted()` function — great for validating correctness.

### 4. Readable and Well-Structured:

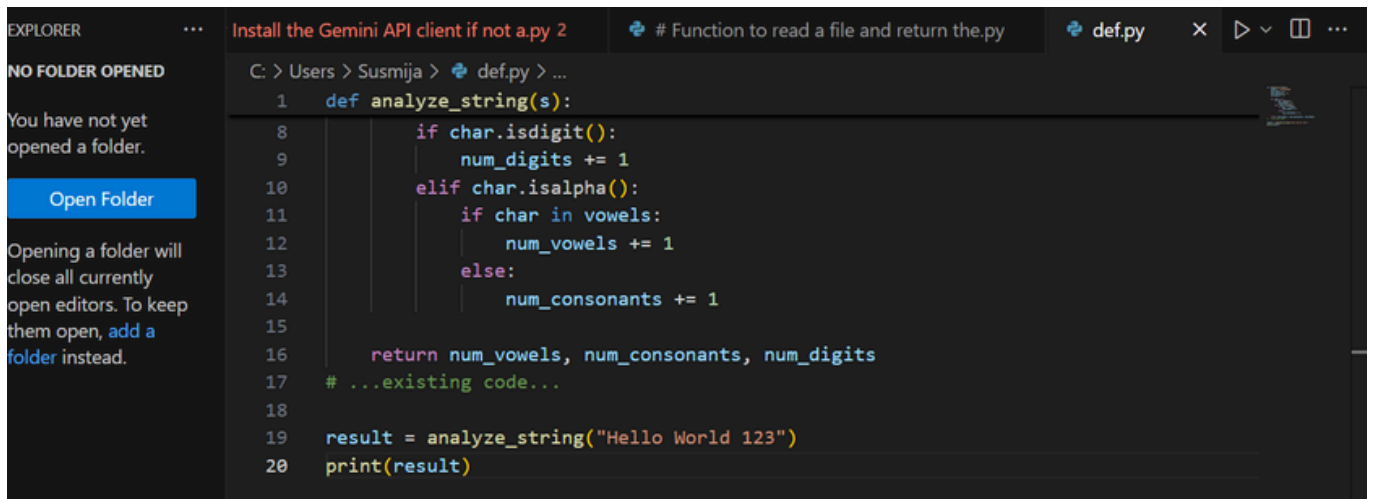
- The code is neatly organized into:
  - Custom sort
  - Built-in sort
  - Example list
  - Comparison

## TASK #2:

### Prompt:

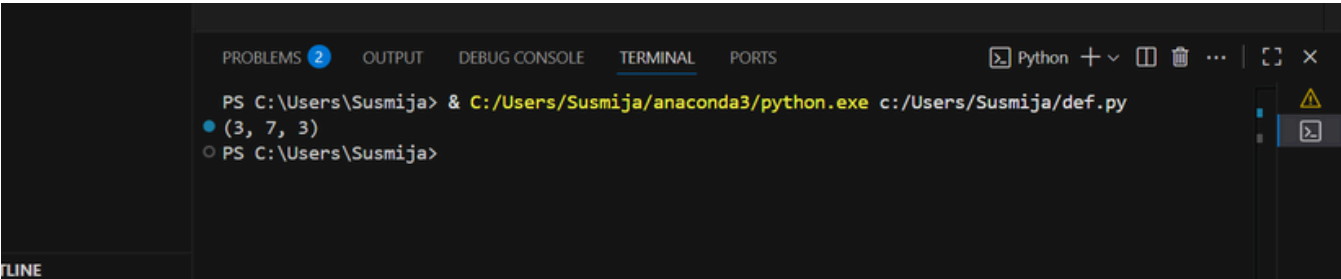
- In Colab, use Google Gemini to generate a Python function that takes a string and returns: The number of vowels, The number of consonants, The number of digits in the string

### Code Generated:



```
1 def analyze_string(s):
8     if char.isdigit():
9         num_digits += 1
10    elif char.isalpha():
11        if char in vowels:
12            num_vowels += 1
13        else:
14            num_consonants += 1
15
16    return num_vowels, num_consonants, num_digits
17 # ...existing code...
18
19 result = analyze_string("Hello World 123")
20 print(result)
```

Output After executing Code:



```
PS C:\Users\Susmija> & C:/Users/Susmija/anaconda3/python.exe c:/Users/Susmija/def.py
(3, 7, 3)
PS C:\Users\Susmija>
```

Your Observations:

Function Definition

1.def count\_lines(filename):

- A function count\_lines is defined that takes a filename (e.g., "example.txt") as input.

2. Try Block – Reading the File

- try:  
with open(filename, 'r') as file:  
return sum(1 for \_ in file)
- open(filename, 'r'): Tries to open the file in **read mode**.
- sum(1 for \_ in file): Counts each line using a generator expression.
  - It iterates through each line and adds 1 per line.
- If the file is found, it returns the line count.

3. Exception Handling – File Not Found

1. except FileNotFoundError:
2. If the file **doesn't exist**, this block is executed.

4. Created a Sample File

- with open(filename, 'w') as file:  
file.write("Hello\n")  
file.write("World\n")

```
file.write("This is a test file\n")
```

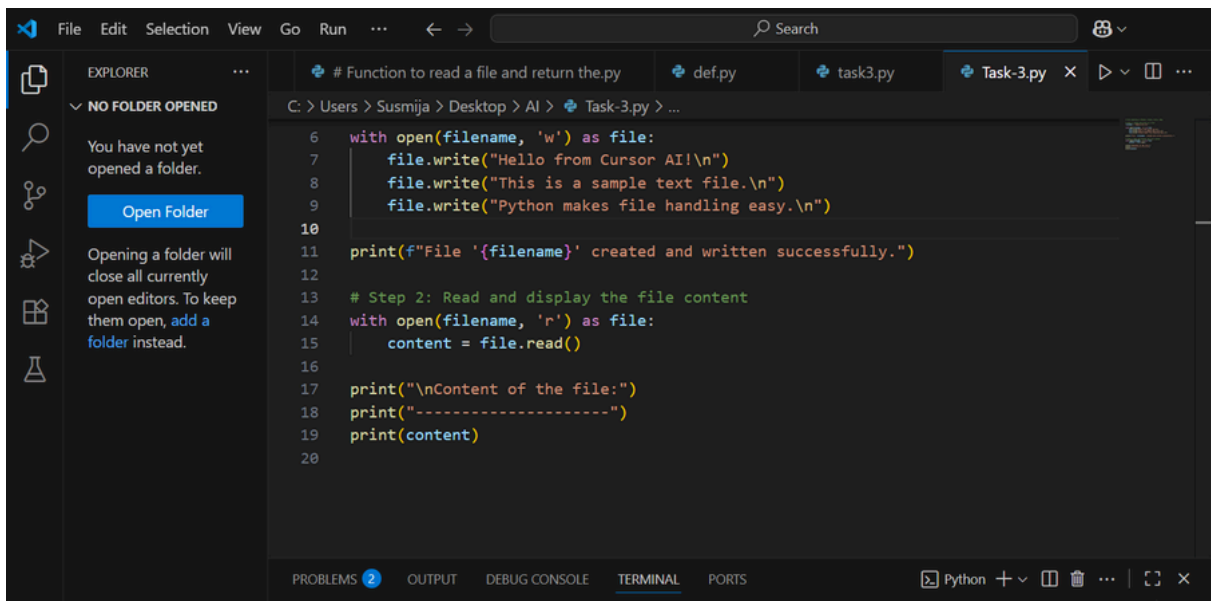
- The file is opened in **write mode** ('w') which **creates a new file**.

### TASK #3:

#### **Prompt:**

- Install and set up Cursor AI. Use it to generate a Python program that performs file handling:
  1. Create a text file
  2. Write sample text
  3. Read and display the content.

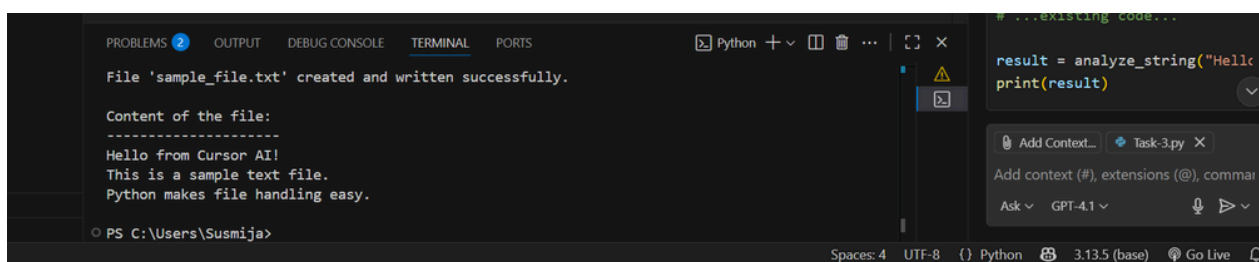
#### **Code Generated:**



The screenshot shows the Cursor AI IDE interface. The Explorer panel on the left indicates 'NO FOLDER OPENED'. The main editor displays a Python script named 'Task-3.py' with the following code:

```
6 with open(filename, 'w') as file:
7     file.write("Hello from Cursor AI!\n")
8     file.write("This is a sample text file.\n")
9     file.write("Python makes file handling easy.\n")
10
11 print(f"File '{filename}' created and written successfully.")
12
13 # Step 2: Read and display the file content
14 with open(filename, 'r') as file:
15     content = file.read()
16
17 print("\nContent of the file:")
18 print("-----")
19 print(content)
20
```

#### **Output After executing Code:**



The screenshot shows the terminal output of the Python script. The output is as follows:

```
File 'sample_file.txt' created and written successfully.

Content of the file:
-----
Hello from Cursor AI!
This is a sample text file.
Python makes file handling easy.
```

#### **Your Observations:**

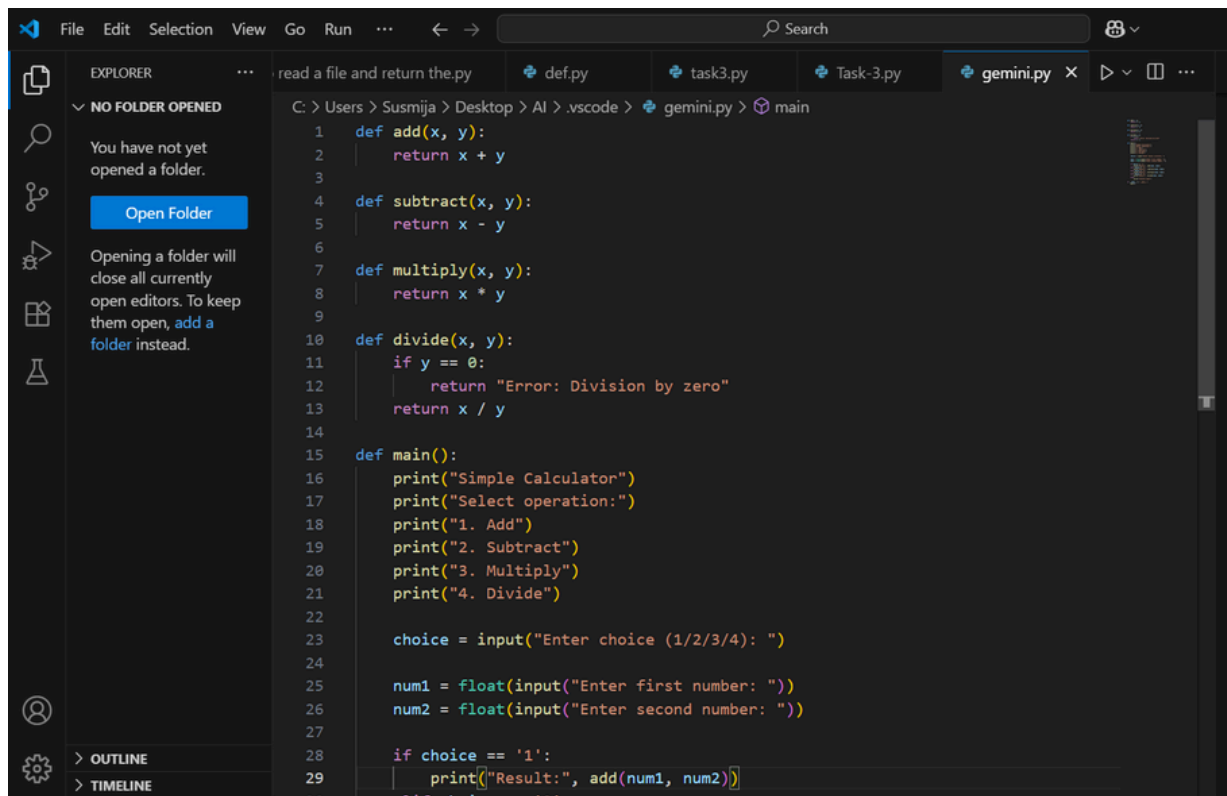
- 1.filename = "sample\_file.txt": Sets the name of the file.
- 2.with open(filename, 'w') as file::
  - Opens (or creates) the file in **write mode** ('w').
  - If the file already exists, it will be **overwritten**.
- 3.file.write(...): Writes 3 lines of text into the file, each ending with a newline (\n).
- 4.print(...): Confirms that the file was created and written successfully.

## TASK #4:

### Prompt:

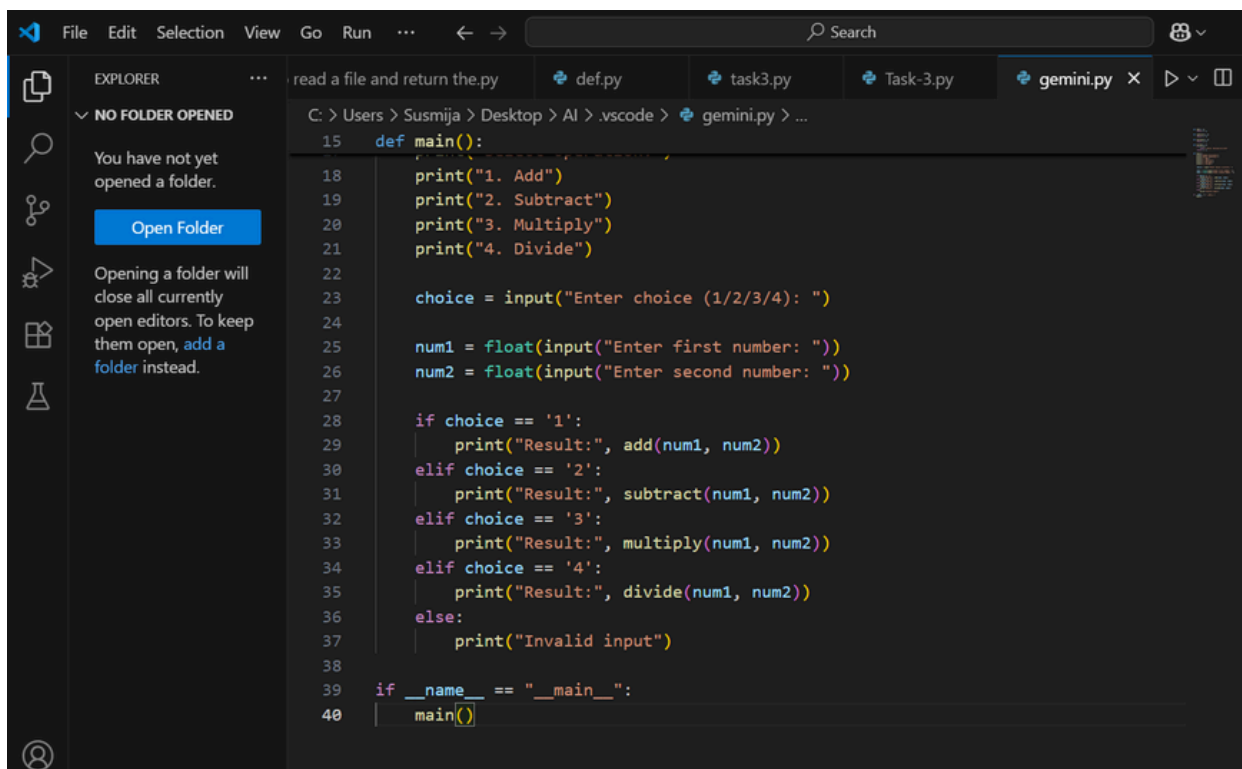
- Ask Google Gemini to generate a Python program that implements a simple calculator using functions (add, subtract, multiply, divide). Then, ask Gemini to explain how the code works.

### Code Generated:



The screenshot shows the Visual Studio Code editor with a file named `gemini.py` open. The code defines four functions: `add`, `subtract`, `multiply`, and `divide`. The `main` function prints a menu and takes user input for a choice and two numbers. It then calls the `add` function for choice '1'.

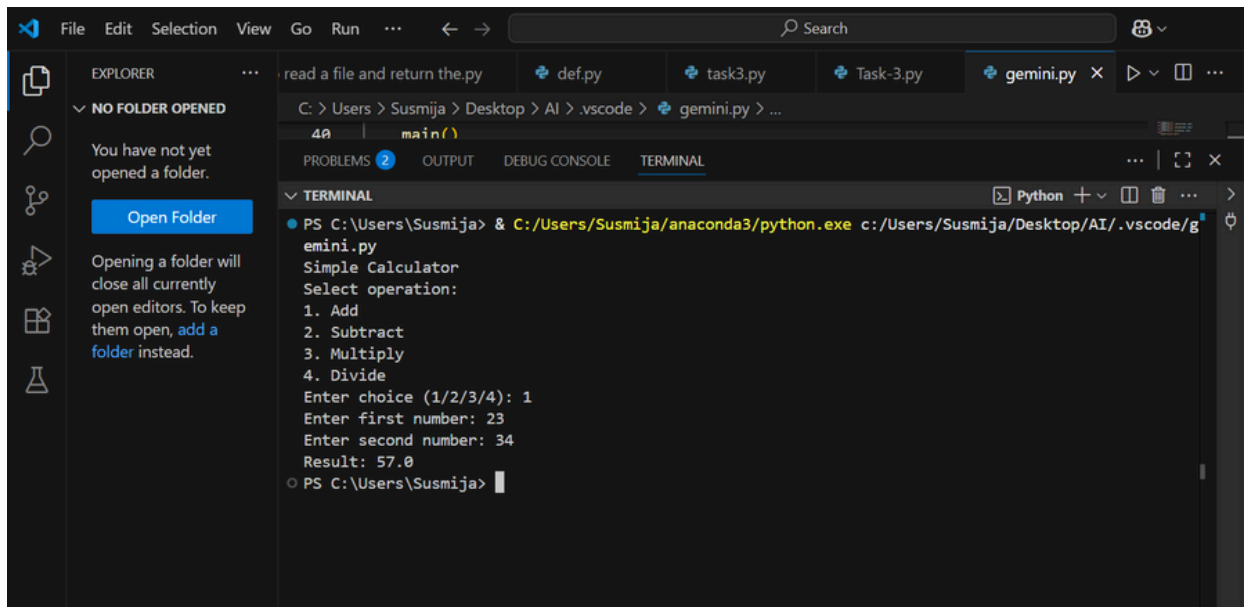
```
1 def add(x, y):
2     return x + y
3
4 def subtract(x, y):
5     return x - y
6
7 def multiply(x, y):
8     return x * y
9
10 def divide(x, y):
11     if y == 0:
12         return "Error: Division by zero"
13     return x / y
14
15 def main():
16     print("Simple Calculator")
17     print("Select operation:")
18     print("1. Add")
19     print("2. Subtract")
20     print("3. Multiply")
21     print("4. Divide")
22
23     choice = input("Enter choice (1/2/3/4): ")
24
25     num1 = float(input("Enter first number: "))
26     num2 = float(input("Enter second number: "))
27
28     if choice == '1':
29         print("Result:", add(num1, num2))
```



The screenshot shows the continuation of the `gemini.py` file. The `main` function continues with an `elif` chain for choices '2', '3', and '4', and an `else` clause for invalid input. The `if __name__ == "__main__":` block at the bottom calls the `main` function.

```
15 def main():
18     print("1. Add")
19     print("2. Subtract")
20     print("3. Multiply")
21     print("4. Divide")
22
23     choice = input("Enter choice (1/2/3/4): ")
24
25     num1 = float(input("Enter first number: "))
26     num2 = float(input("Enter second number: "))
27
28     if choice == '1':
29         print("Result:", add(num1, num2))
30     elif choice == '2':
31         print("Result:", subtract(num1, num2))
32     elif choice == '3':
33         print("Result:", multiply(num1, num2))
34     elif choice == '4':
35         print("Result:", divide(num1, num2))
36     else:
37         print("Invalid input")
38
39 if __name__ == "__main__":
40     main()
```

### Output After executing Code:



## Your Observations:

### 1. Function Definitions

- These functions perform basic arithmetic:

-->def add(x, y):

return x + y

-->def subtract(x, y):

return x - y

-->def multiply(x, y):

return x \* y

-->def divide(x, y):

if y == 0:

-->return "Error: Division by zero"

return x / y

- Each function takes two numbers x and y, and returns the result.
- The divide() function includes **error handling** for division by zero.

### 2. The main() Function

- This is where user interaction happens:

def main():

print("Simple Calculator")

print("Select operation:")

print("1. Add")

print("2. Subtract")

print("3. Multiply")

print("4. Divide")

- The program prints a menu of operations for the user.

**choice = input("Enter choice (1/2/3/4): ")**

- The user selects an operation (e.g., 1 for addition).

```
num1 = float(input("Enter first number: "))
```

```
num2 = float(input("Enter second number: "))
```

- The user inputs two numbers, which are converted to floats for accurate calculations.

### 3. Conditional Execution Based on Choice

```
if choice == '1':
```

```
    print("Result:", add(num1, num2))
```

```
elif choice == '2':
```

```
    print("Result:", subtract(num1, num2))
```

```
elif choice == '3':
```

```
    print("Result:", multiply(num1, num2))
```

```
elif choice == '4':
```

```
    print("Result:", divide(num1, num2))
```

```
else:
```

```
    print("Invalid input")
```

- Depending on the user's choice, the corresponding function is called.
- If the choice doesn't match 1–4, it prints "Invalid input."

### 4. Script Entry Point

```
if __name__ == "__main__":
```

```
    main()
```

- This ensures that the main() function runs **only when the script is executed directly**, not when imported.

### Terminal Output :

#### Simple Calculator

#### Select operation:

1. Add
2. Subtract
3. Multiply
4. Divide

Enter choice (1/2/3/4): 1

Enter first number: 23

Enter second number: 34

Result: 57.0

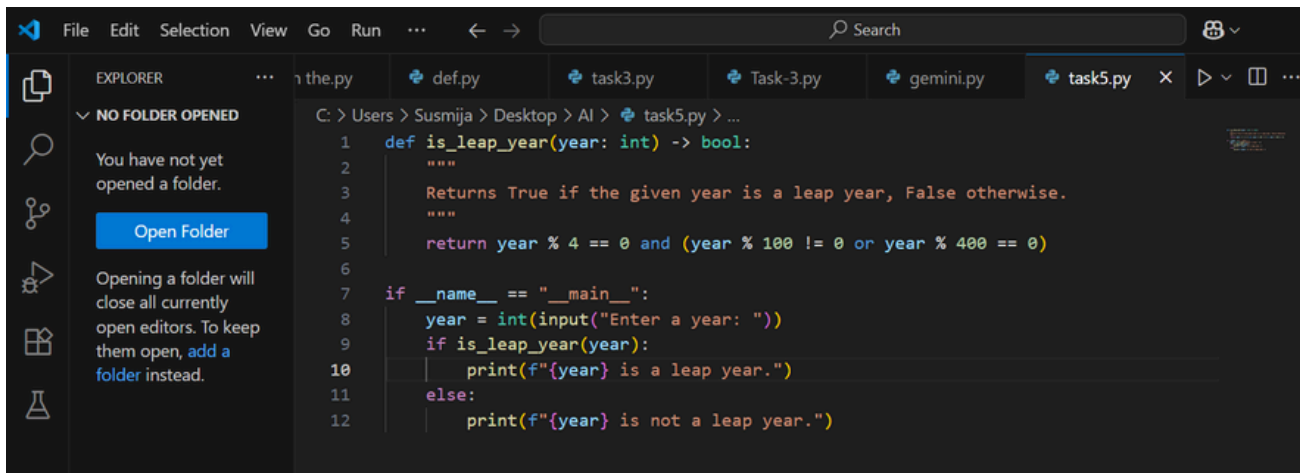
- The user selected **1 (Addition)**.
- Entered **23** and **34**.
- Got the correct result: 57.0.

### TASK #5:

## Prompt:

- Use Cursor AI to create a Python program that checks if a given year is a leap year or not. Try different prompt styles and see how Cursor modifies its code suggestions.

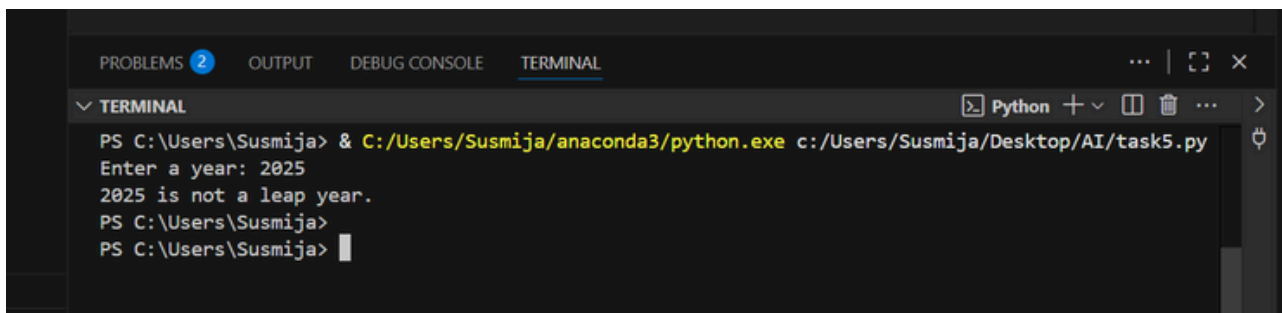
## Code Generated:



The screenshot shows a code editor with a dark theme. The Explorer panel on the left shows a file named 'task5.py'. The main editor area displays the following Python code:

```
1 def is_leap_year(year: int) -> bool:
2     """
3     Returns True if the given year is a leap year, False otherwise.
4     """
5     return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)
6
7 if __name__ == "__main__":
8     year = int(input("Enter a year: "))
9     if is_leap_year(year):
10        print(f"{year} is a leap year.")
11    else:
12        print(f"{year} is not a leap year.")
```

## Output After executing Code:



The screenshot shows a terminal window with the following output:

```
PS C:\Users\Susmija> & C:/Users/Susmija/anaconda3/python.exe c:/Users/Susmija/Desktop/AI/task5.py
Enter a year: 2025
2025 is not a leap year.
PS C:\Users\Susmija>
PS C:\Users\Susmija>
```

## Your Observations:

### 1. Function Definition

- `def is_leap_year(year: int) -> bool:`  
"""  
Returns True if the given year is a leap year, False otherwise.  
"""  
`return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)`
- Purpose: Checks leap year condition.
- Type hinting:
  - `year: int` → function expects an integer.
  - `-> bool` → function returns a boolean (True or False).
- Logic: Implements the leap year condition in one line.

### 2. Main Execution Block

```
if __name__ == "__main__":
    year = int(input("Enter a year: "))
    if is_leap_year(year):
        print(f"{year} is a leap year.")
```



else:

```
print(f"{year} is not a leap year.")
```

- `if __name__ == "__main__":`: Ensures this code only runs when the script is executed directly.
- `input(...)`: Takes user input and converts it to an integer.
- Conditional check: Calls `is_leap_year(year)` to check and print the appropriate message.

Output from Terminal:

**Enter a year:** 2025

2025 is not a leap year.

- Since 2025 is not divisible by 4, the output is correct.