

Lab 2: Exploring Additional AI Coding Tools

Gemini (Colab) and Cursor AI

Course Code: 24CSBTB41 Course Title: Assistant Coding Assignment No: 2.4

Roll Number: 2503A51L36

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

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: Compare the two bubble sort algorithm and Python's built-in sort() function

Prompt: 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

Code: (Bubble Sort Algorithm)

```
def bubble_sort(data_list):
```

```
    """
```

```
    Sorts a list using the bubble sort algorithm.
```

Args:

data_list: The list to be sorted.

Returns:

The sorted list.

"""

```
n = len(data_list)

for i in range(n - 1):
    for j in range(n - i - 1):
        if data_list[j] > data_list[j + 1]:
            data_list[j], data_list[j + 1] = data_list[j + 1], data_list[j]

return data_list
```

Code: `(Built-in Sort())`

```
import random
```

```
# Generate test data (this will be used later for comparison)
```

```
test_data = [random.randint(0, 1000) for _ in range(1000)]
```

```
# 1. Create a new list that is a copy of the test data list.
```

```
copied_list = test_data[:]
```

```
# 2. Use the built-in sort() method directly on this copied list.
```

```
copied_list.sort()
```

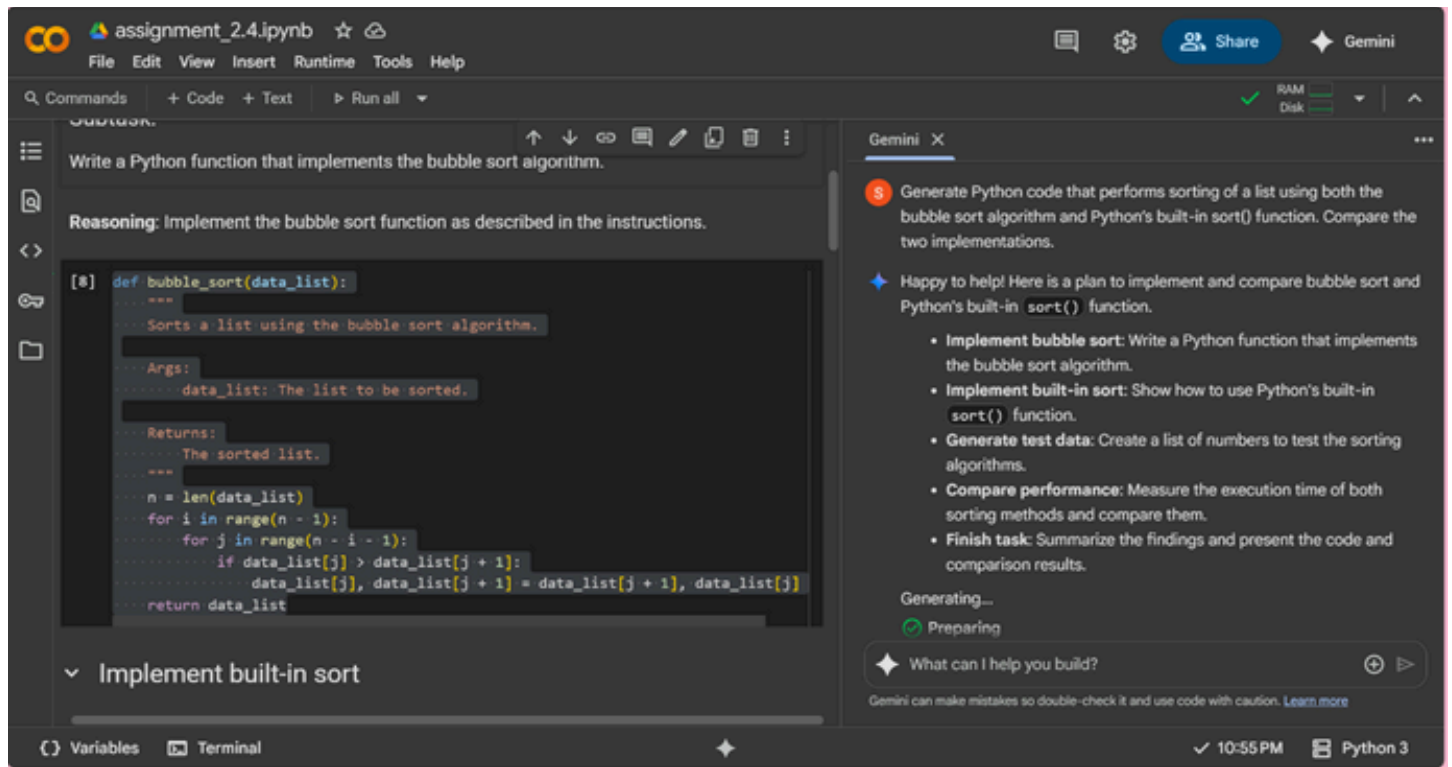
```
# Display the first few elements of the sorted list to confirm
```

```
print("First 10 elements of the sorted list:")
```

```
print(copied_list[:10])
```

OBSERVATION:

The built-in `sort()` method was significantly faster than the bubble sort algorithm for sorting the generated list



Expected Output:

Two sorting implementations: Bubble sort (manual logic) and Built-in sort()

Task 2: The number of vowels, The number of consonants, The number of digits in the string

Prompt: 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:

```
def count_chars(input_string):
```

```
    """
```

```
    Counts the number of vowels, consonants, and digits in a string.
```

```
    Args:
```

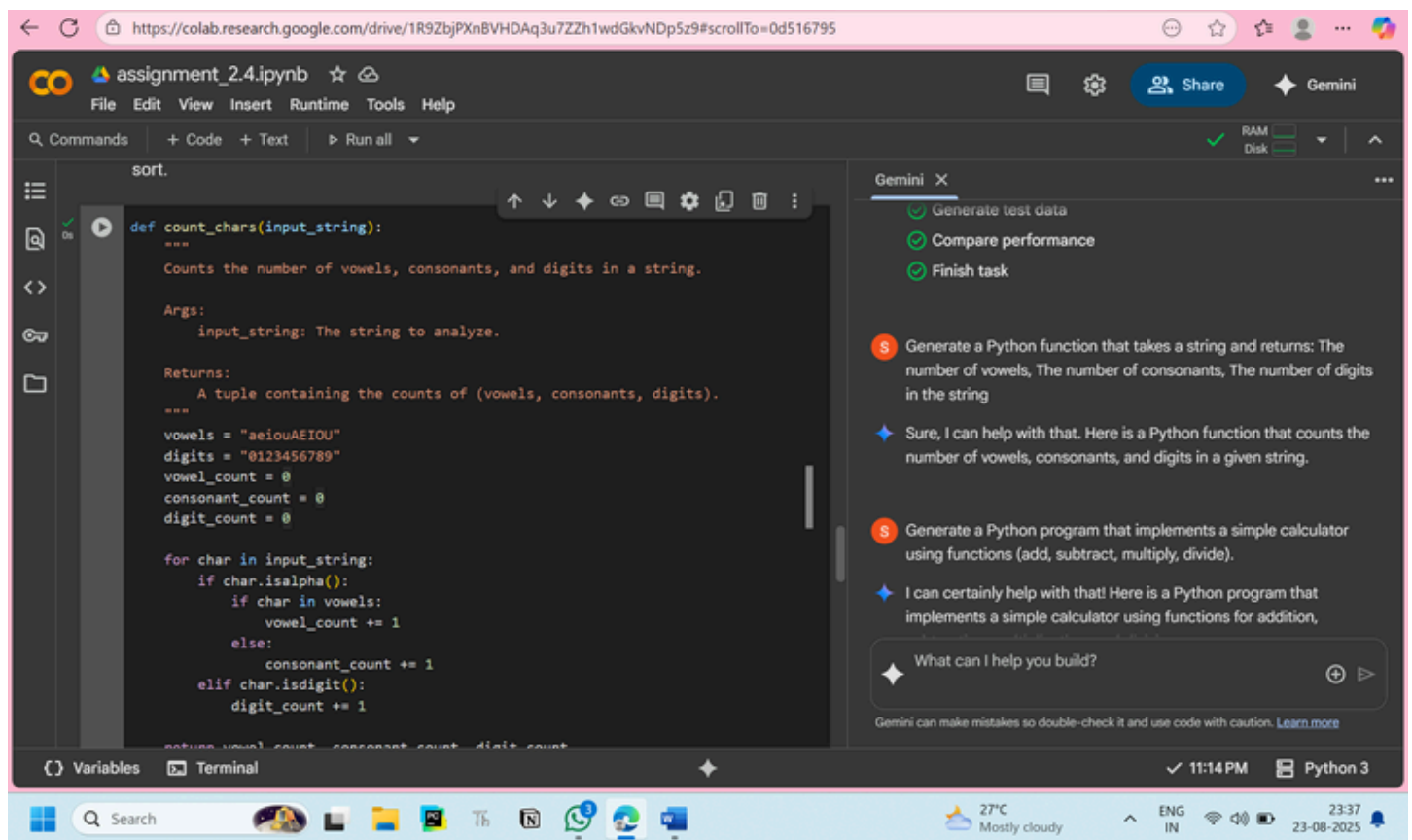
```
        input_string: The string to analyze.
```

```
    Returns:
```

A tuple containing the counts of (vowels, consonants, digits).

```
"""  
  
vowels = "aeiouAEIOU"  
  
digits = "0123456789"  
  
vowel_count = 0  
  
consonant_count = 0  
  
digit_count = 0  
  
  
for char in input_string:  
  
    if char.isalpha():  
  
        if char in vowels:  
  
            vowel_count += 1  
  
        else:  
  
            consonant_count += 1  
  
    elif char.isdigit():  
  
        digit_count += 1  
  
  
    return vowel_count, consonant_count, digit_count  
  
  
# Example usage  
  
test_string = "Hello World 123!"  
  
vowels, consonants, digits = count_chars(test_string)  
  
  
print(f"String: '{test_string}'")  
  
print(f"Number of vowels: {vowels}")  
  
print(f"Number of consonants: {consonants}")  
  
print(f"Number of digits: {digits}")
```

OBSERVATION: AI completed function that Iterates through characters of a string and Counts vowels, consonants, and digits



Expected Output:

String: 'Hello World 123!'

Number of vowels: 3

Number of consonants: 7

Number of digits: 3

Task 3: Install and set up Cursor AI

Prompt: Generate a Python program that performs file handling:

- Create a text file
- Write sample text
- Read and display the content

Code:

```
def create_and_write_file():
```

```
"""Create a text file and write sample text to it"""
```

```
try:
```

```
    with open("sample_file.txt", "w") as file:
```

```
        file.write("Hello! This is a sample text file.\n")
```

```
        file.write("This file was created using Python file handling.\n")
```

```
        file.write("It demonstrates how to:\n")
```

```
        file.write("- Create a new file\n")
```

```
        file.write("- Write text content\n")
```

```
        file.write("- Read and display the content\n")
```

```
print("✓ File 'sample_file.txt' created successfully!")
```

```
print("✓ Sample text written to the file.")
```

```
except Exception as e:
```

```
    print(f"Error creating/writing file: {e}")
```

```
def read_and_display_file():
```

```
    """Read the text file and display its content"""
```

```
    try:
```

```
        with open("sample_file.txt", "r") as file:
```

```
            content = file.read()
```

```
            print("\n" + "="*50)
```

```
            print("FILE CONTENT:")
```

```
            print("="*50)
```

```
            print(content)
```

```
            print("="*50)
```

```
except FileNotFoundError:
```

```
    print("Error: File 'sample_file.txt' not found!")
```

```
except Exception as e:
```

```
    print(f"Error reading file: {e}")
```

```
def main():
```

```
    """Main function to demonstrate file handling operations"""
```

```
print("PYTHON FILE HANDLING DEMONSTRATION")
```

```
print("="*40)
```

```
# Step 1: Create file and write text
```

```
print("\nStep 1: Creating file and writing text...")
```

```
create_and_write_file()
```

```
# Step 2: Read and display content
```

```
print("\nStep 2: Reading and displaying file content...")
```

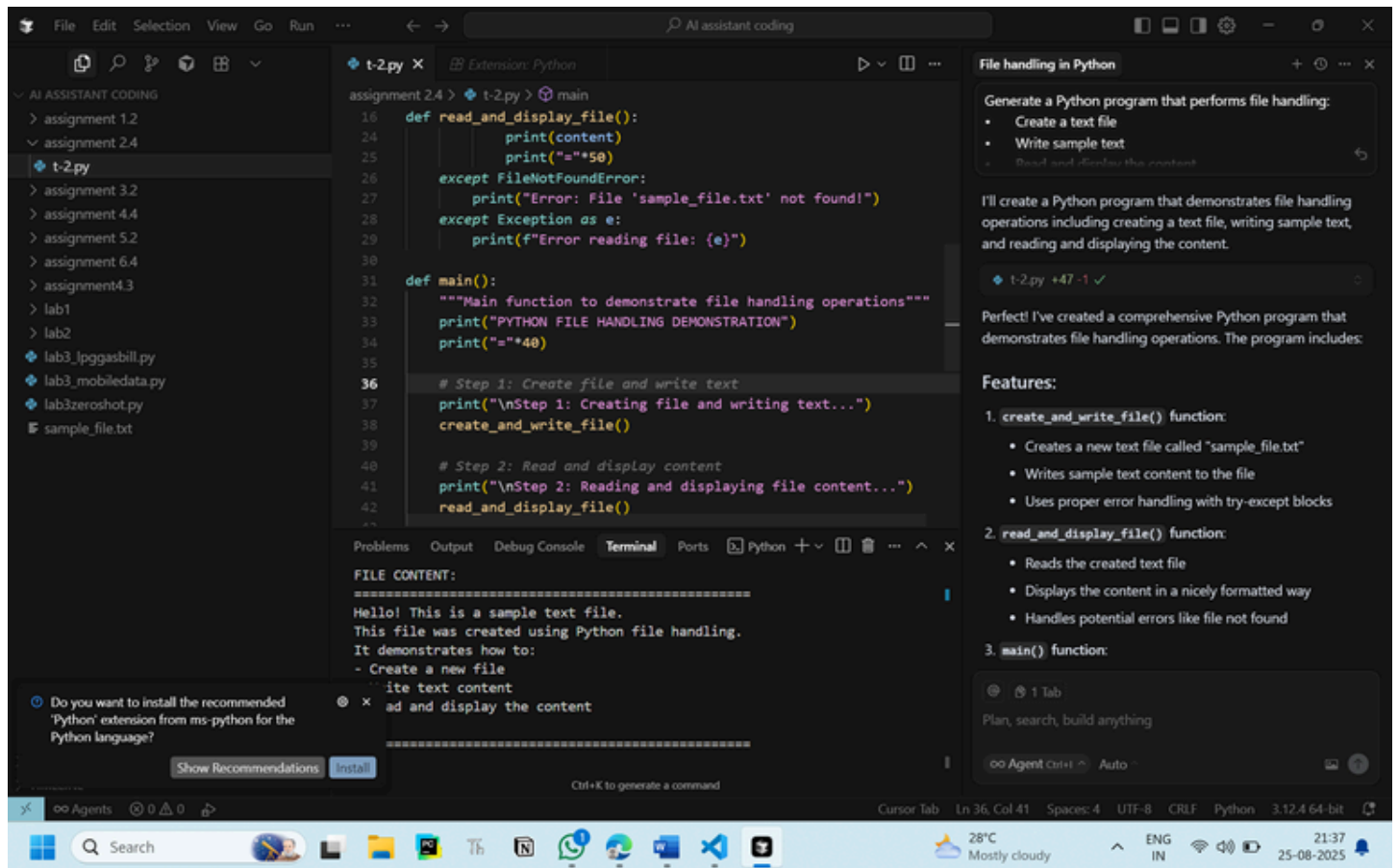
```
read_and_display_file()
```

```
print("\n✓ File handling demonstration completed!")
```

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

```
    main()
```

OBSERVATION: cursor AI generated the program with good efficiency and great logic and easy to understand



Expected Output:

PYTHON FILE HANDLING DEMONSTRATION

=====

Step 1: Creating file and writing text...

✓ File 'sample_file.txt' created successfully!

✓ Sample text written to the file.

Step 2: Reading and displaying file content...

=====

FILE CONTENT:

=====

Hello! This is a sample text file.

This file was created using Python file handling.

It demonstrates how to:

- Create a new file
- Write text content
- Read and display the conten

=====

Task 4: A Simple Calculator

Prompt: Generate a Python program that implements a simple calculator using functions (add, subtract, multiply, divide) using google collab

Code:

```
def add(x, y):

    """Adds two numbers."""

    return x + y


def subtract(x, y):

    """Subtracts the second number from the first."""

    return x - y


def multiply(x, y):

    """Multiplies two numbers."""

    return x * y


def divide(x, y):

    """Divides the first number by the second. Handles division by zero."""

    if y == 0:

        return "Error: Division by zero"

    return x / y


# Example usage

num1 = 10

num2 = 5
```

```
print(f"{num1} + {num2} = {add(num1, num2)}")  
  
print(f"{num1} - {num2} = {subtract(num1, num2)}")  
  
print(f"{num1} * {num2} = {multiply(num1, num2)}")  
  
print(f"{num1} / {num2} = {divide(num1, num2)}")
```

```
num3 = 10
```

```
num4 = 0
```

```
print(f"{num3} / {num4} = {divide(num3, num4)}")
```

OBSERVATION: Complete calculator code with user input and operation selection.

Line-by-line explanation or markdown-style explanation provided by Gemini.

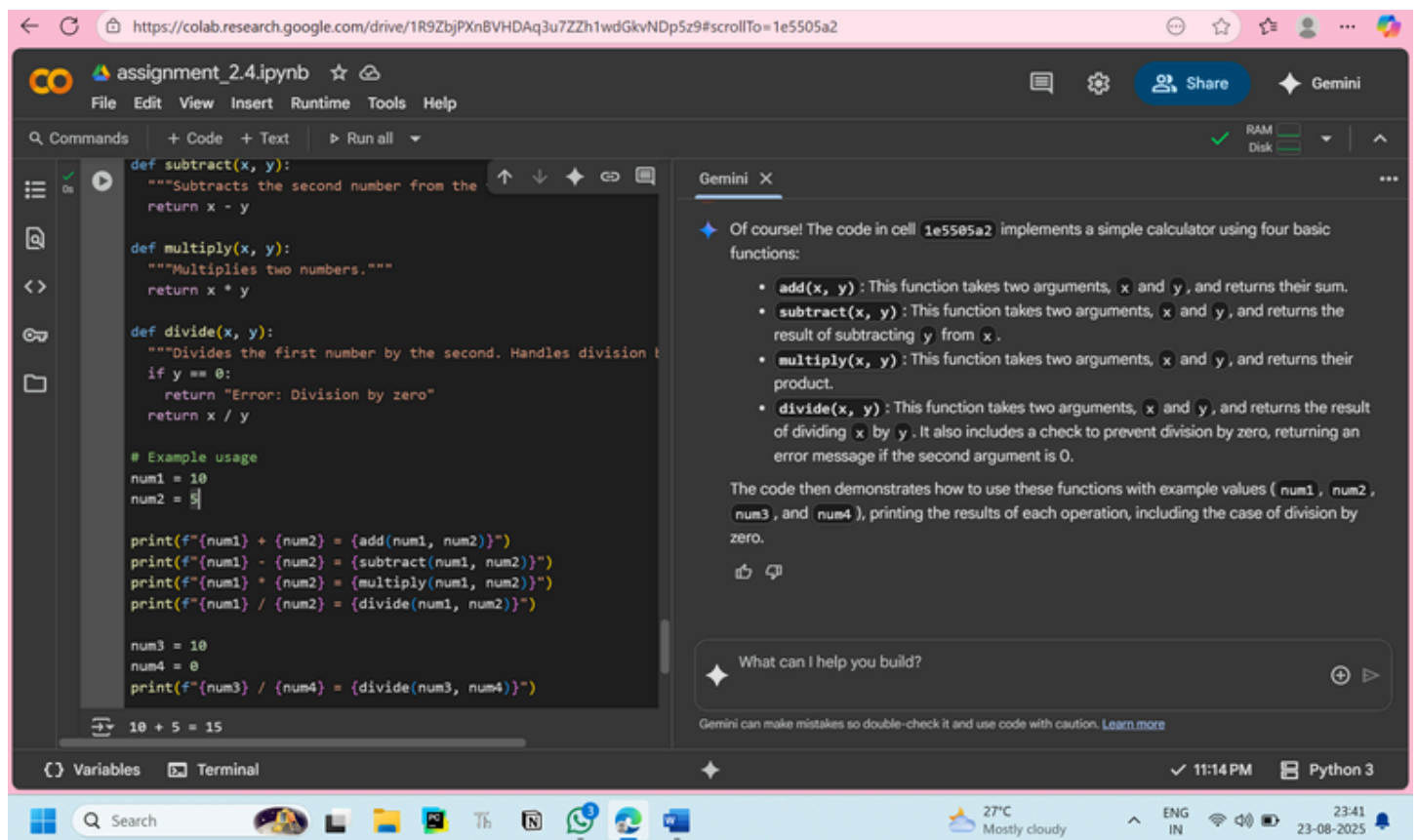
The screenshot displays a Google Colab notebook titled "assignment_2.4.ipynb". The notebook contains a Python program for a simple calculator. The code defines three functions: `subtract`, `multiply`, and `divide`. It also includes example usage with `num1 = 10` and `num2 = 5`, and another example with `num3 = 10` and `num4 = 0`. The output of the first example is shown in the terminal: `10 + 5 = 15` and `10 - 5 = 5`.

On the right side of the notebook, the Gemini AI chat interface is visible. It shows three messages:

- A blue diamond icon followed by the text: "Sure, I can help with that. Here is a Python function that counts the number of vowels, consonants, and digits in a given string."
- An orange square icon followed by the text: "Generate a Python program that implements a simple calculator using functions (add, subtract, multiply, divide)."
- A blue diamond icon followed by the text: "I can certainly help with that! Here is a Python program that implements a simple calculator using functions for addition, subtraction, multiplication, and division."

At the bottom of the chat interface, there is a text input field with the placeholder "What can I help you build?" and a "Send" button. Below the input field, a note states: "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)".

The bottom of the screenshot shows the Windows taskbar with the search bar, task view button, and several open applications. The system tray on the right indicates the temperature is 27°C, mostly cloudy, and the time is 11:14 PM on 23-08-2025.



Expected Output:

$10 + 5 = 15$

$10 - 5 = 5$

$10 * 5 = 50$

$10 / 5 = 2.0$

$10 / 0 = \text{Error: Division by zero}$

Task 5: Leap Year

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:

```
def is_leap_year(year):
```

```
    """
```

Check if a given year is a leap year.

Leap year rules:

- Year must be divisible by 4

- If year is divisible by 100, it must also be divisible by 400

Args:

year (int): The year to check

Returns:

bool: True if leap year, False otherwise

"""

if year % 4 != 0:

return False

elif year % 100 != 0:

return True

else:

return year % 400 == 0

def get_user_input():

"""Get a valid year input from the user"""

while True:

try:

year = int(input("Enter a year to check (e.g., 2024): "))

if year > 0:

return year

else:

print("Please enter a positive year number.")

except ValueError:

print("Invalid input! Please enter a valid year number.")

def main():

"""Main function to run the leap year checker"""

print("=" * 50)

print(" LEAP YEAR CHECKER")

```
print("=" * 50)
```

```
while True:
```

```
    # Get year from user
```

```
    year = get_user_input()
```

```
    # Check if it's a leap year
```

```
    if is_leap_year(year):
```

```
        print(f"\n✓ {year} is a LEAP YEAR!")
```

```
        print(f" February has 29 days in {year}")
```

```
    else:
```

```
        print(f"\n✗ {year} is NOT a leap year.")
```

```
        print(f" February has 28 days in {year}")
```

```
    # Ask if user wants to check another year
```

```
    print("\n" + "-" * 30)
```

```
    another = input("Check another year? (y/n): ").lower().strip()
```

```
    if another not in ['y', 'yes']:
```

```
        break
```

```
print("\nThank you for using the Leap Year Checker!")
```

```
print("=" * 50)
```

```
# Example usage and testing
```

```
def test_examples():
```

```
    """Test the leap year function with some examples"""
```

```
    print("Testing with example years:")
```

```
    test_years = [2000, 2020, 2024, 1900, 2023, 2100]
```

```
    for year in test_years:
```

```
        result = "LEAP YEAR" if is_leap_year(year) else "NOT a leap year"
```

```
print(f" {year}: {result}")
```

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

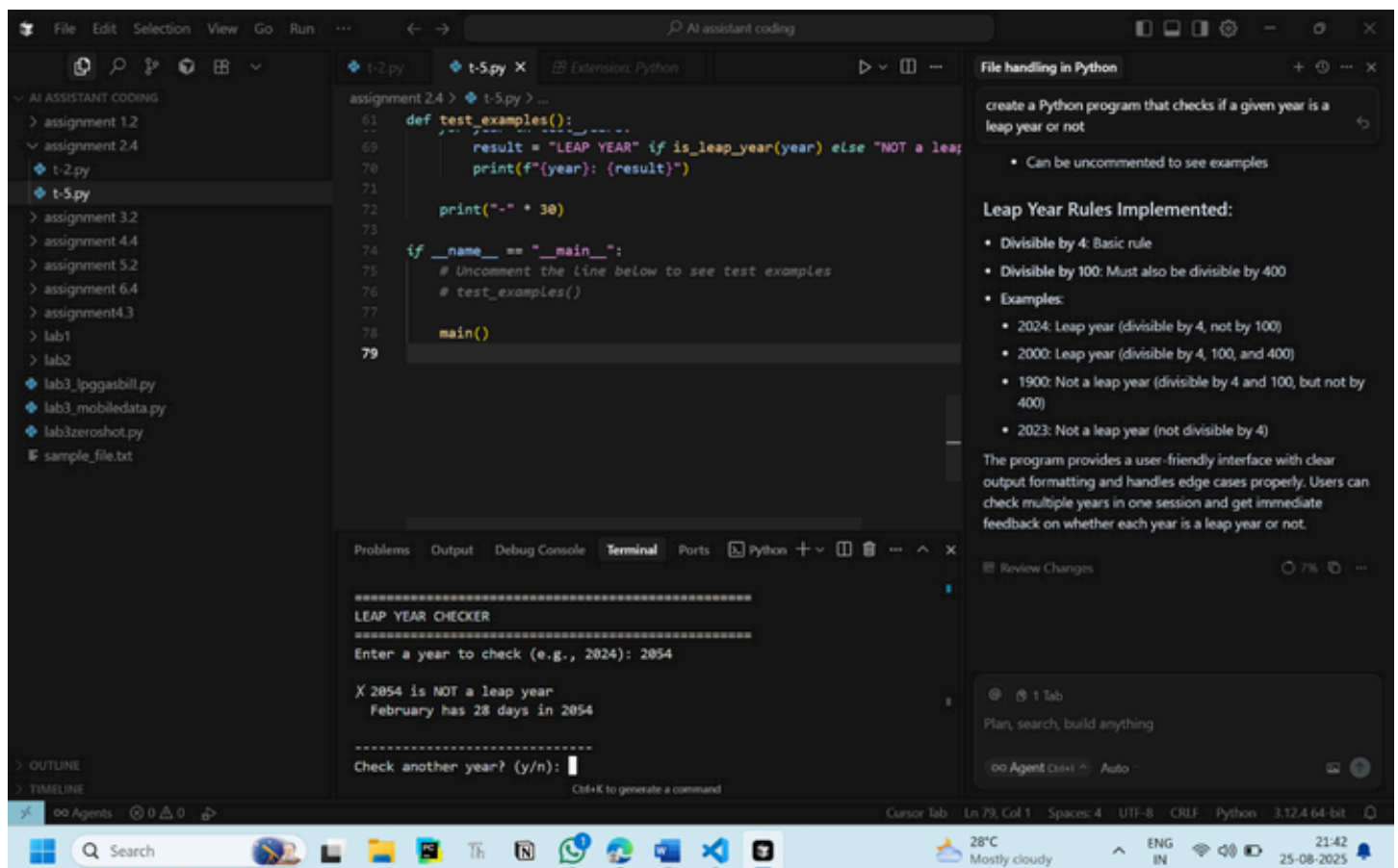
```
    # Uncomment the line below to run tests instead of the interactive program
```

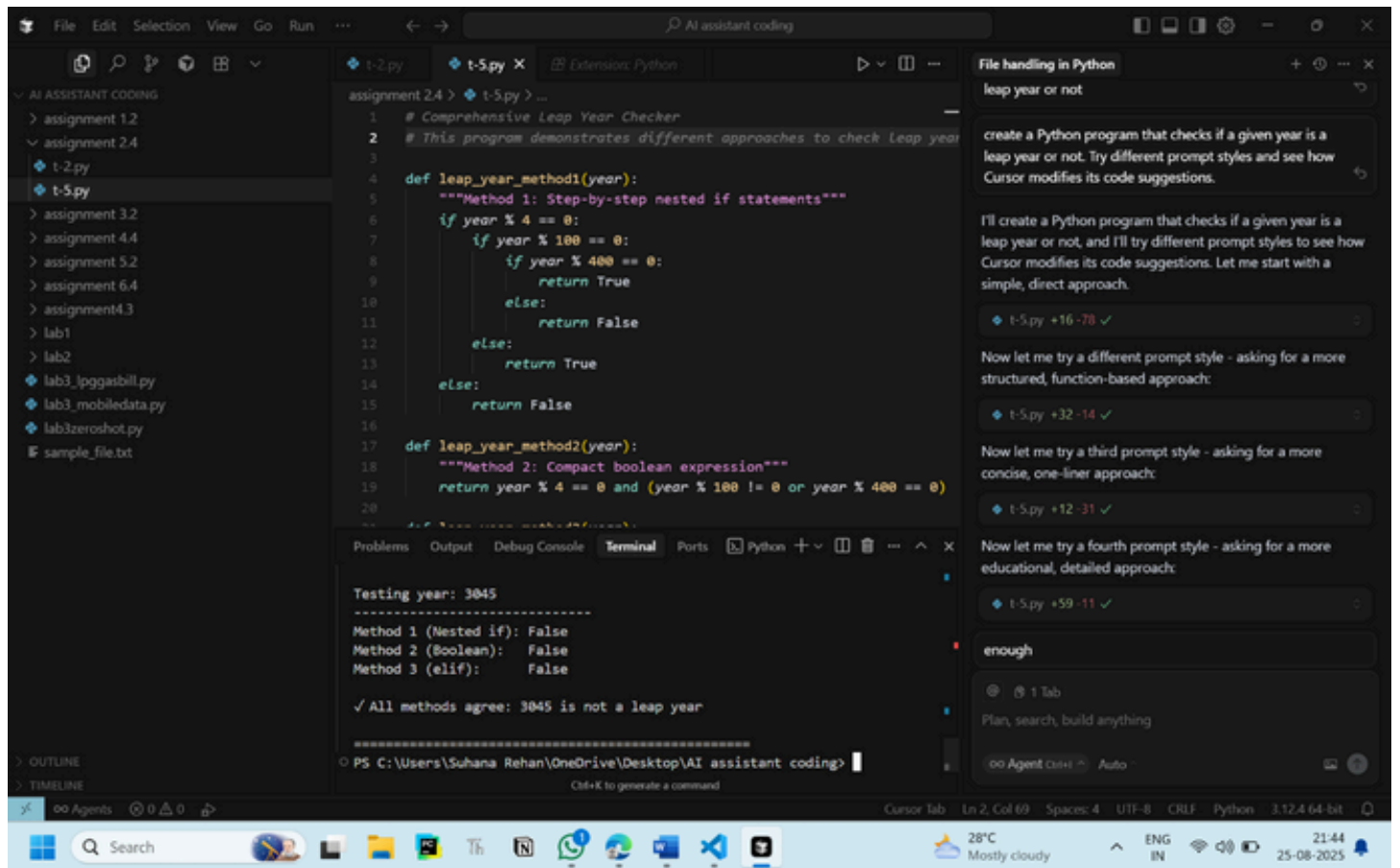
```
    # test_examples()
```

```
    # Run the main interactive program
```

```
    main()
```

OBSERVATION: Cursor adapts complexity based on whether you ask for simple, structured, or comprehensive approaches





Expected Output:

=====

LEAP YEAR CHECKER

=====

Enter a year to check (e.g., 2024): 4576

✓ 4576 is a LEAP YEAR!

February has 29 days in 4576

=====

Check another year? (y/n):

