

AI ASSISTED CODING

LAB ASSIGNMENT – 4.3

NAVYA MASADI

2403A53020

24BTCAICYB01-2-1

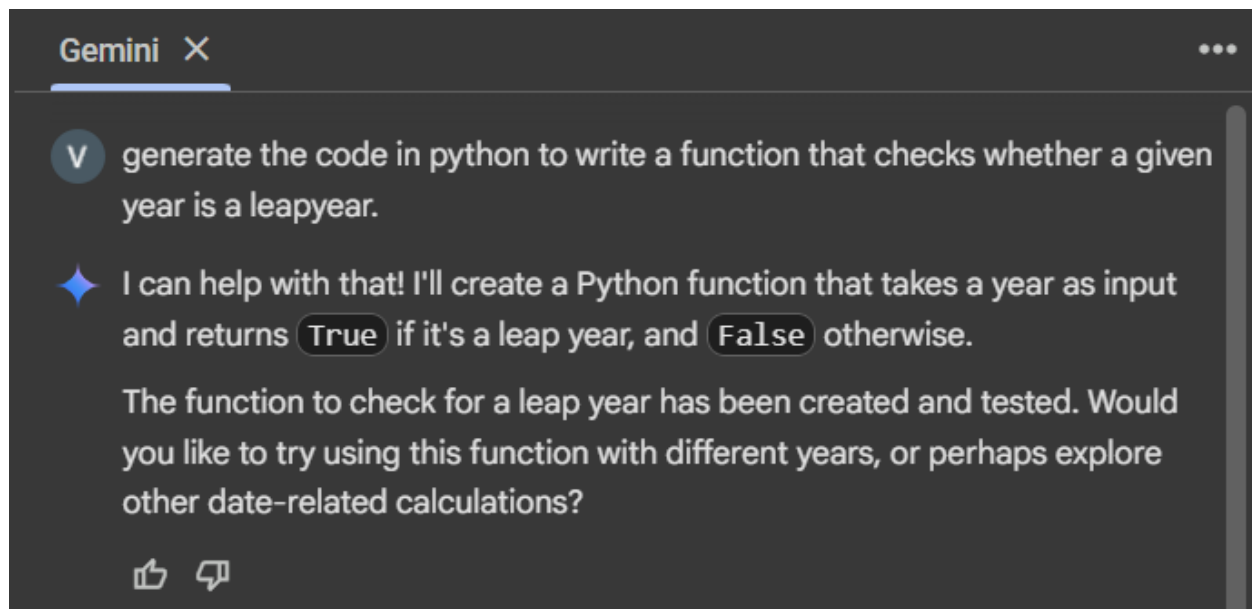
Task Description#1

- Zero-shot: Prompt AI to write a function that checks whether a given year is a leapyear.

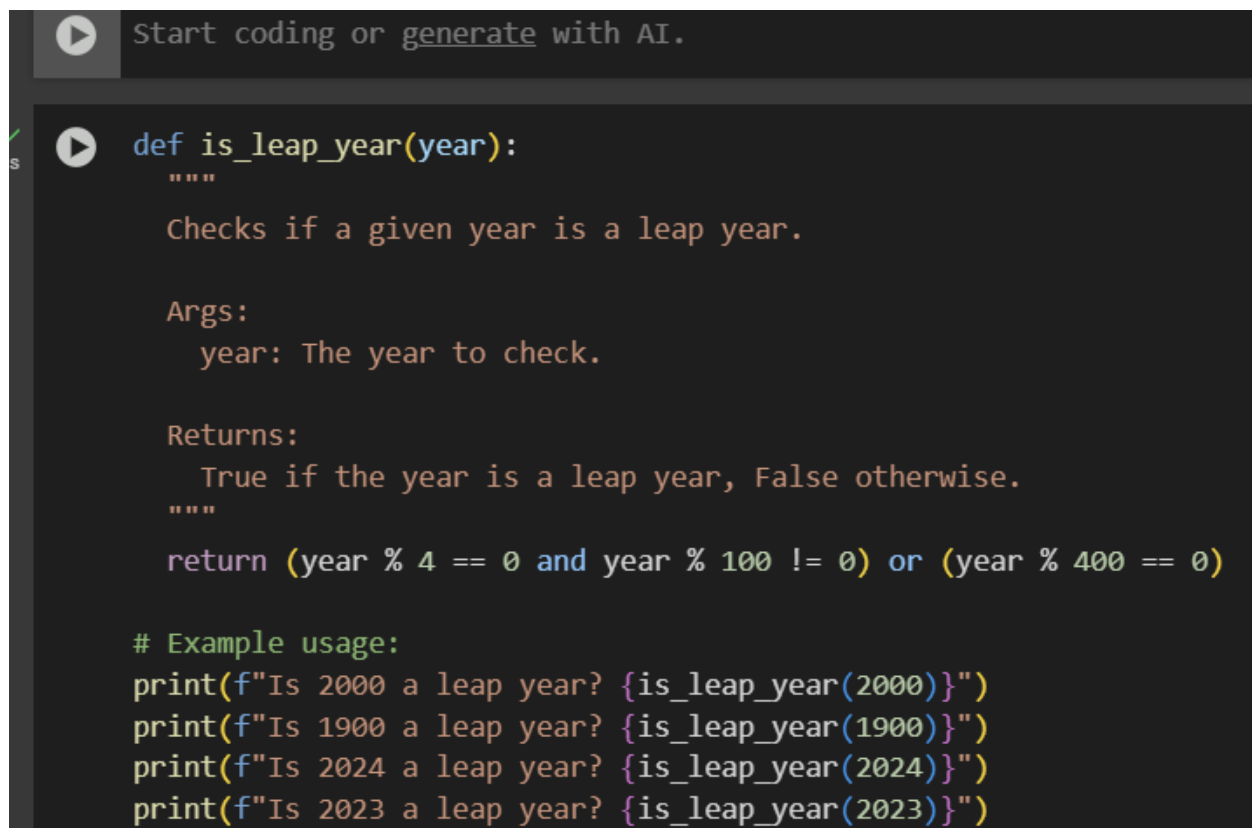
Expected Output#1

- AI-generated function with no examples provided

PROMPT:



CODE:



OUTPUT:

```
⇒ Is 2000 a leap year? True  
Is 1900 a leap year? False  
Is 2024 a leap year? True  
Is 2023 a leap year? False
```

Task Description#2

- One-shot: Give one input-output example to guide AI in writing a function that converts centimeters to inches.

Expected Output#2

- Function with correct conversion logic.

PROMPT:

Gemini X

V Generate input-output example to guide AI in writing a function that converts centimeters to inches.

✦ Sure, I can provide an input-output example for a function that converts centimeters to inches. Here's an example:

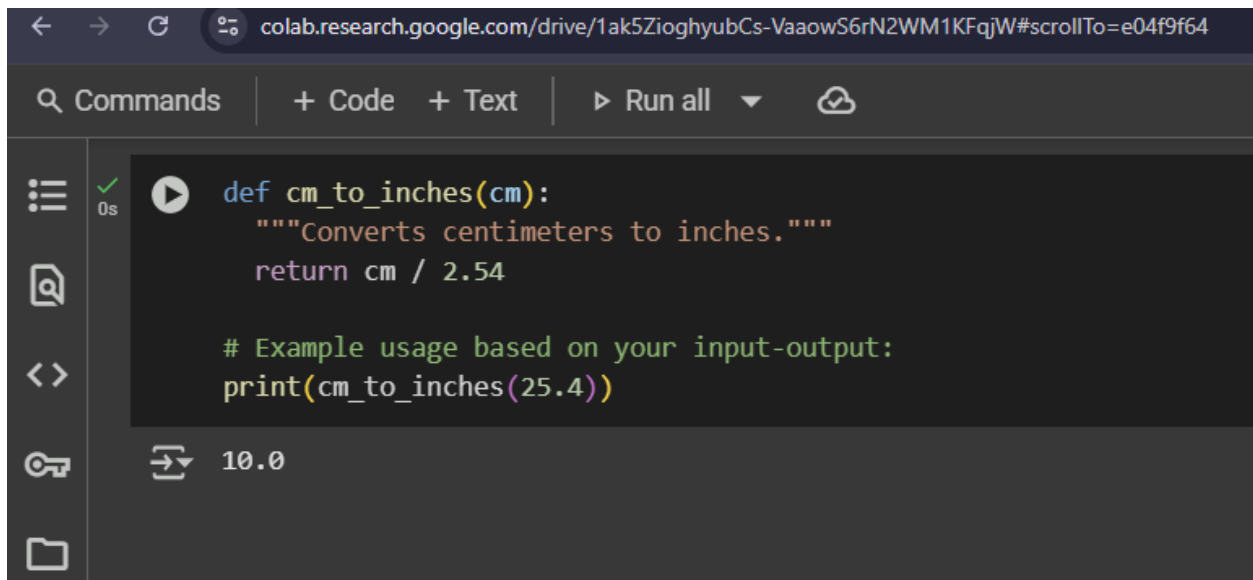
Input: `cm_to_inches(25.4)`

Expected Output: `10.0`

This example shows that if the input is 25.4 centimeters, the expected output is 10.0 inches (since 1 inch equals 2.54 centimeters).

Let me know if you'd like to see another example or want me to generate the code for this conversion!

CODE AND OUTPUT:



```
def cm_to_inches(cm):  
    """Converts centimeters to inches."""  
    return cm / 2.54  
  
# Example usage based on your input-output:  
print(cm_to_inches(25.4))
```

10.0

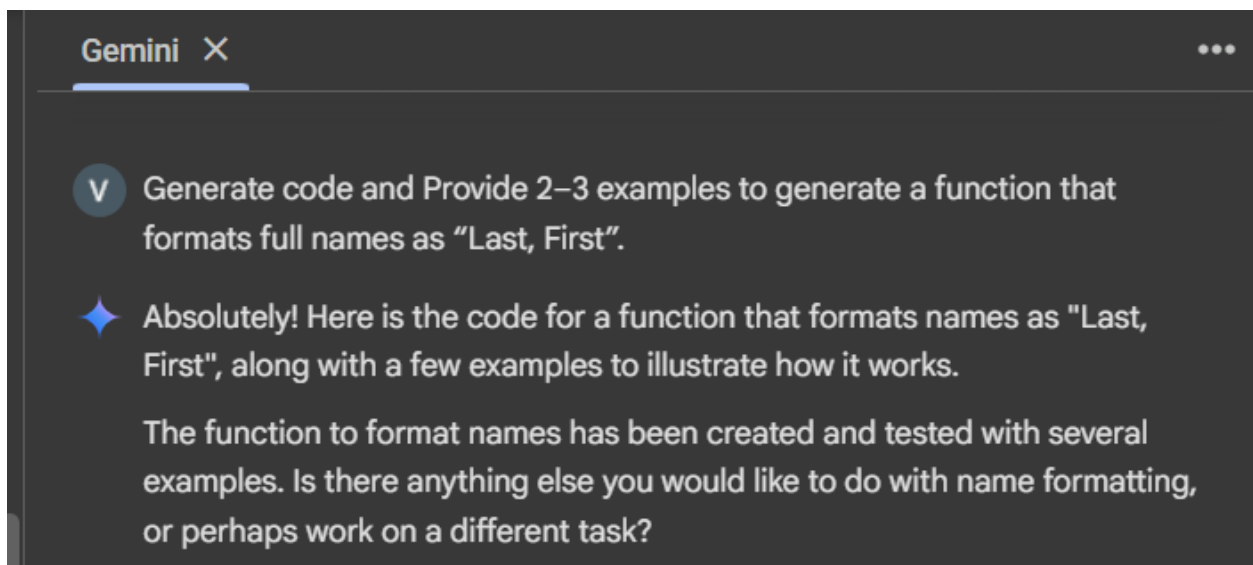
Task Description#3

- Few-shot: Provide 2–3 examples to generate a function that formats full names as “Last, First”.

Expected Output#3

- Well-structured function respecting the examples.

PROMPT:



CODE AND OUTPUT:

The screenshot shows a Google Colab notebook interface. The top bar displays the URL: `colab.research.google.com/drive/1ak5ZioghycCs-VaaowS6rN2WM1KFqjW#scrollTo=73b50ddb`. Below the toolbar, the code editor contains a Python function definition for `format_name`. The function's docstring describes its purpose: "Formats a full name string as 'Last, First'." It lists the argument `full_name` as a string and the return value as a formatted string or the original string if formatting fails. The function code splits the input string, checks if there are at least two parts, and formats them accordingly. Below the code, a text block provides three examples of the function's usage. At the bottom, a code cell shows example usage with `print` statements, and the output cell displays the results: `'Doe, John'`, `'Smith, Mary Jane'`, and `'Alice'`.

```
def format_name(full_name):  
    """  
    Formats a full name string as "Last, First".  
  
    Args:  
        full_name: A string containing the full name (e.g., "John Doe").  
  
    Returns:  
        A string formatted as "Last, First" (e.g., "Doe, John").  
        Returns the original string if formatting is not possible (e.g., single name).  
    """  
    names = full_name.split()  
    if len(names) >= 2:  
        return f"{names[-1]}, {' '.join(names[:-1])}"  
    else:  
        return full_name
```

Here are a few examples of how the `format_name` function works:

1. **Example 1: Standard two-part name** Input: "John Doe" Output: "Doe, John" This shows a typical case where the last name is separated by a space from the first name.
2. **Example 2: Name with a middle name** Input: "Mary Jane Smith" Output: "Smith, Mary Jane" This demonstrates how the function handles names with more than two parts, treating the last part as the last name and the rest as the first name(s).
3. **Example 3: Single name** Input: "Alice" Output: "Alice" This shows how the function handles cases where the input is just a single name, returning the original input as formatting is not applicable in the "Last, First" style.

```
[4] # Example usage:  
print(f'{format_name("John Doe")}')  
print(f'{format_name("Mary Jane Smith")}')  
print(f'{format_name("Alice")}')
```

```
'Doe, John'  
'Smith, Mary Jane'  
'Alice'
```

Task Description#4

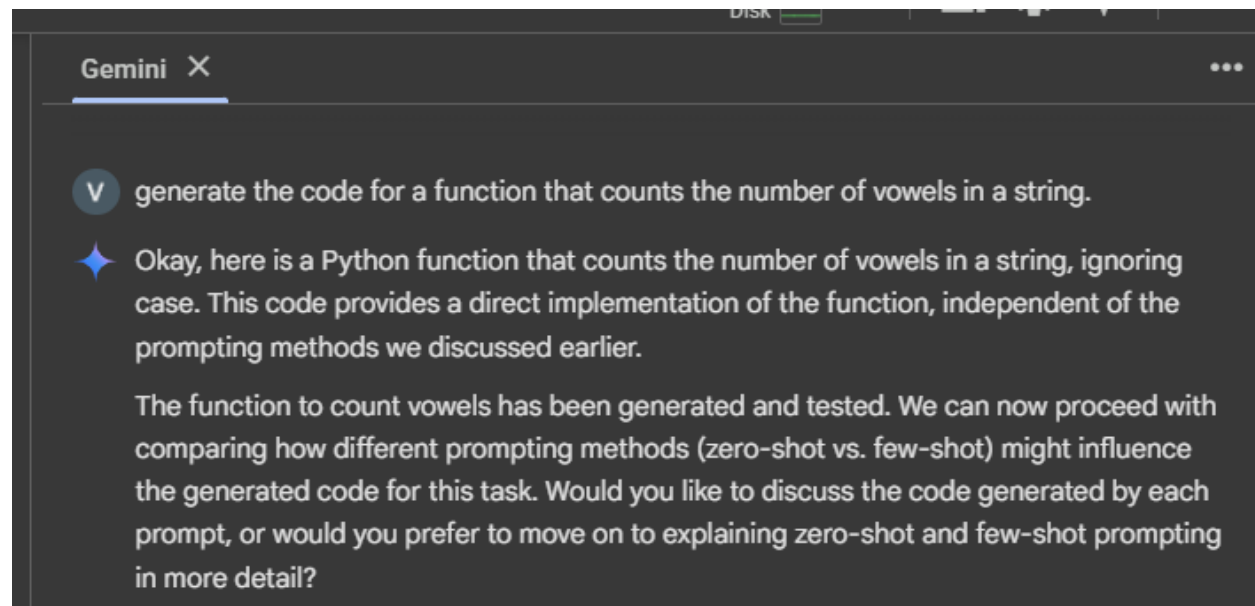
- Compare zero-shot and few-shot prompts for writing a function that counts the number of vowels in a string

.

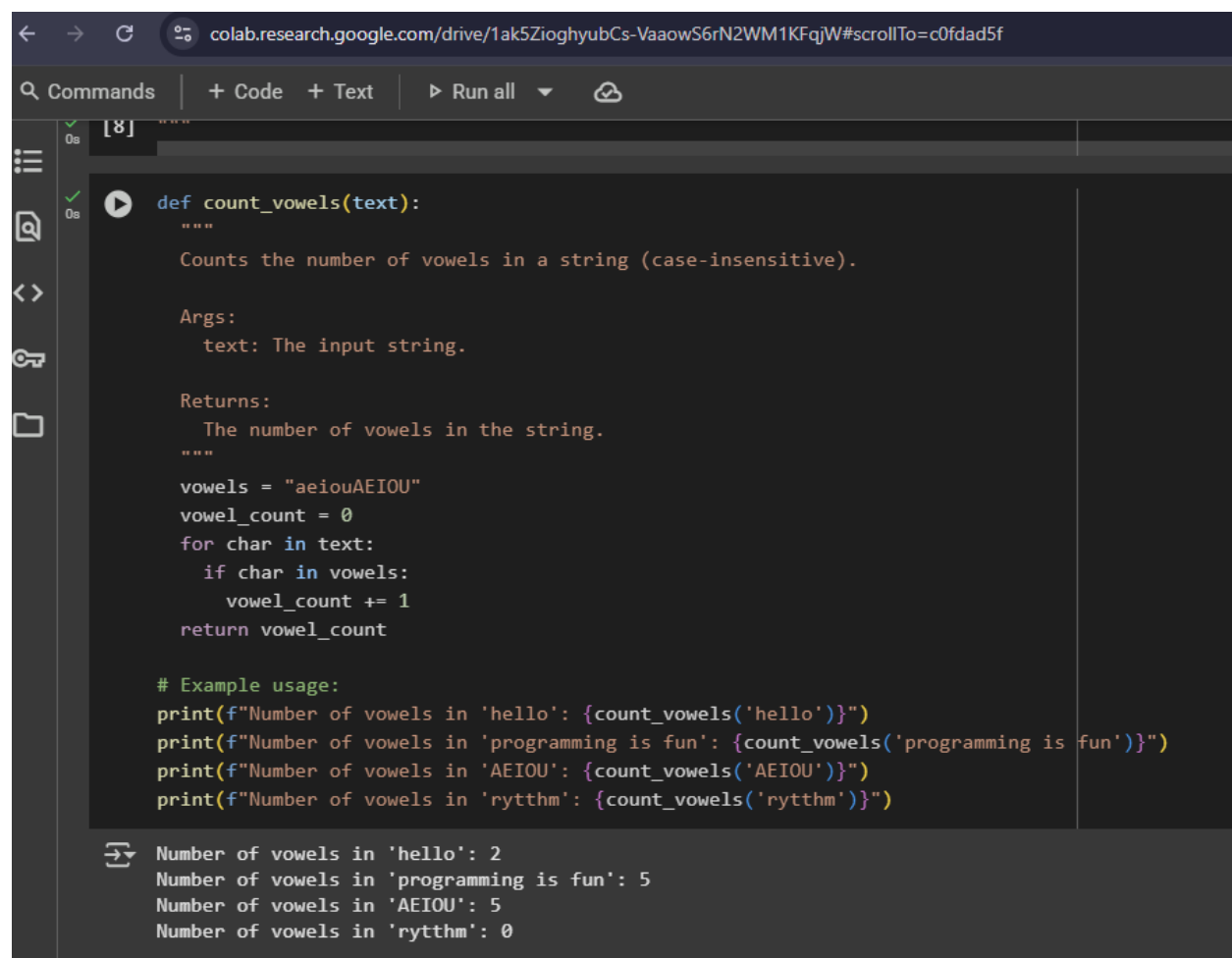
Expected Output#4

- Functional output and comparative reflection.

PROMPT:



CODE AND OUTPUT:



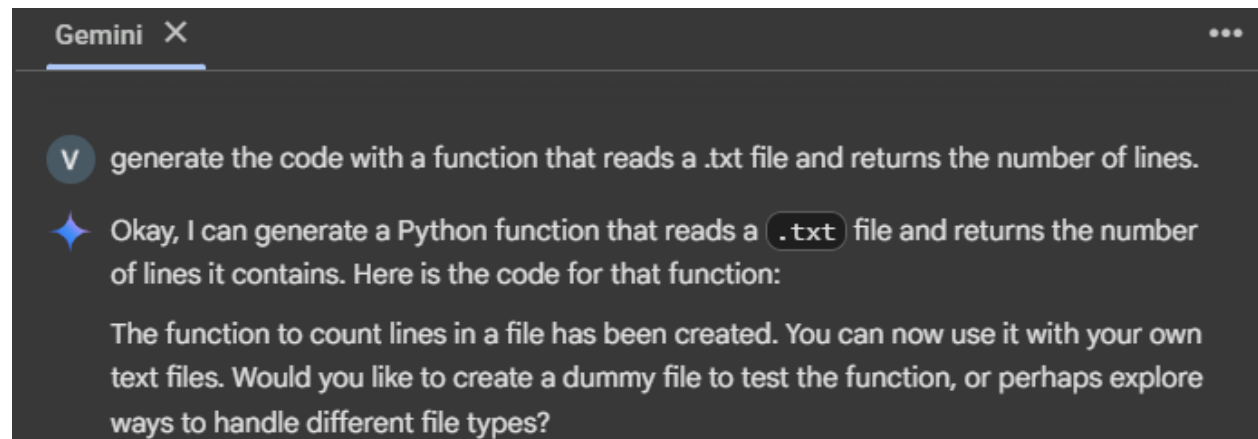
Task Description#5

- Use few-shot prompting to generate a function that reads a .txt file and returns the number of lines.

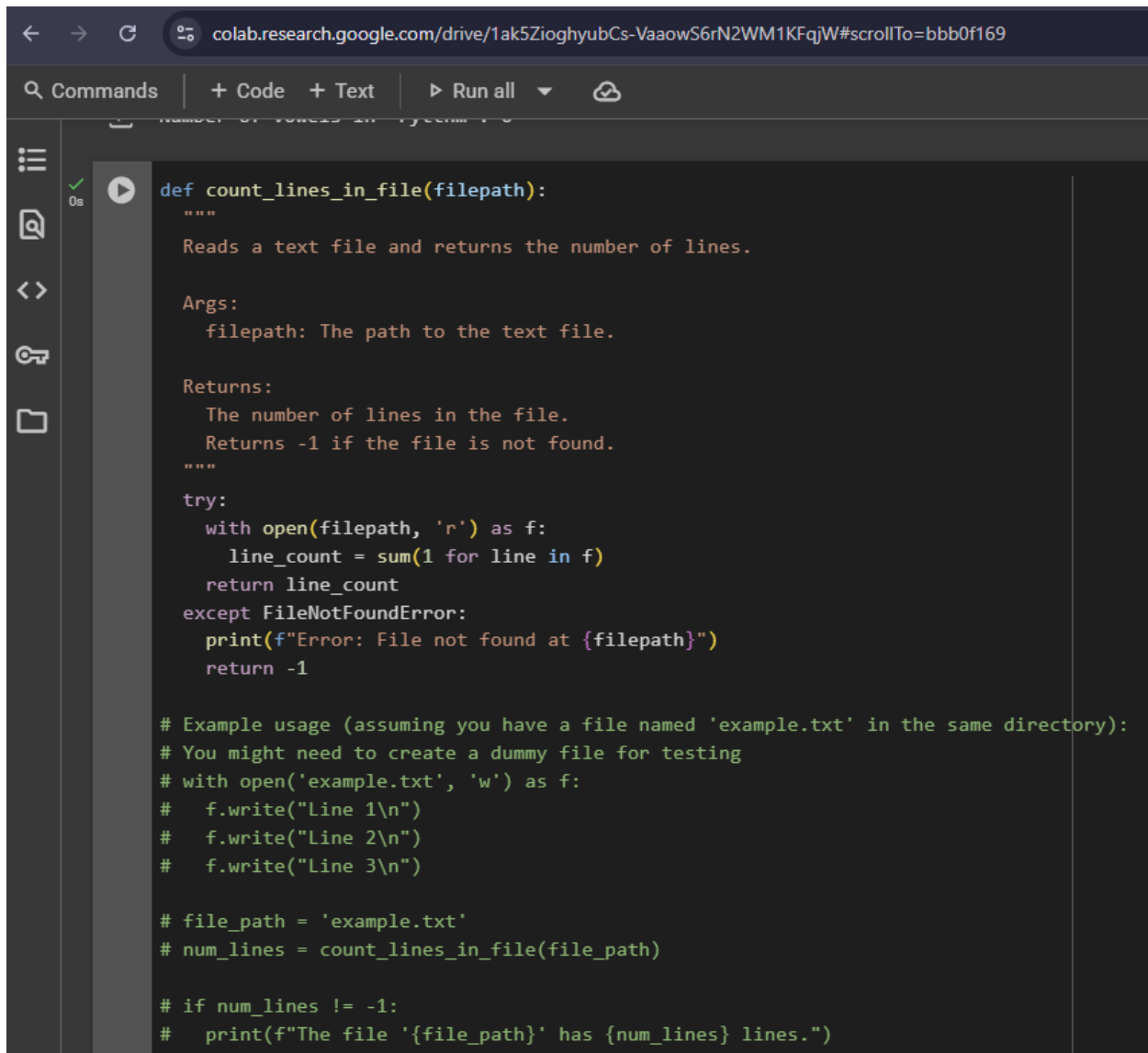
Expected Output#5

- Working file-processing function with AI-guided logic.

PROMPT:



CODE:



The image shows a Google Colab notebook interface. The top bar includes navigation icons and a URL: `colab.research.google.com/drive/1ak5ZioghyubCs-VaaowS6rN2WM1KFqjW#scrollTo=bbb0f169`. Below the bar is a toolbar with 'Commands', '+ Code', '+ Text', 'Run all', and a cloud icon. On the left is a sidebar with icons for file explorer, search, and other tools. The main area contains a code cell with the following Python code:

```
def count_lines_in_file(filepath):  
    """  
    Reads a text file and returns the number of lines.  
  
    Args:  
        filepath: The path to the text file.  
  
    Returns:  
        The number of lines in the file.  
        Returns -1 if the file is not found.  
    """  
    try:  
        with open(filepath, 'r') as f:  
            line_count = sum(1 for line in f)  
            return line_count  
    except FileNotFoundError:  
        print(f"Error: File not found at {filepath}")  
        return -1  
  
# Example usage (assuming you have a file named 'example.txt' in the same directory):  
# You might need to create a dummy file for testing  
# with open('example.txt', 'w') as f:  
#     f.write("Line 1\n")  
#     f.write("Line 2\n")  
#     f.write("Line 3\n")  
  
# file_path = 'example.txt'  
# num_lines = count_lines_in_file(file_path)  
  
# if num_lines != -1:  
#     print(f"The file '{file_path}' has {num_lines} lines.")
```