AI-ASSISTANT CODING ASSIGNMENT-2

Name : Alli Harika

HT:NO : 2303A510i7

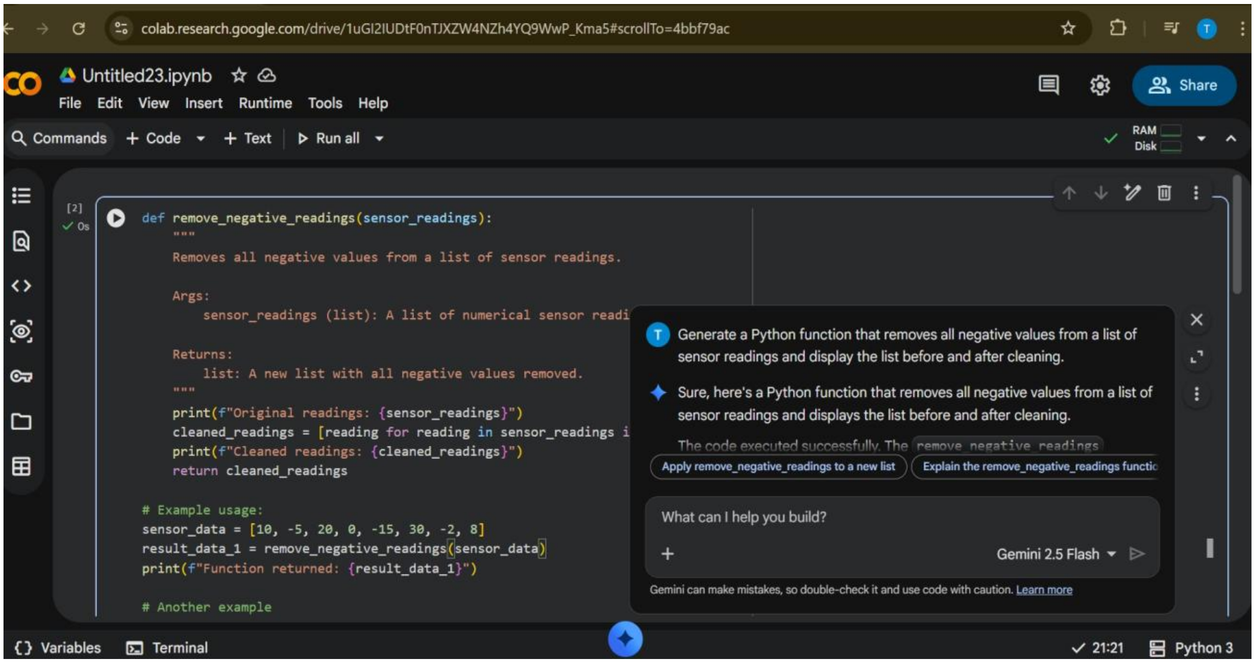
Batch: 21

LAB2:

Exploring Additional AI Coding Tools beyond Copilot –Gemini(Colab) and

Cursor AI Task1:Cleaning Sensor Data

- **Scenario:**
- **You are cleaning IoT sensor data where negative values are invalid.**
- **Task: Use Gemini in Colab to generate a function that filters out all negative numbers from a list.**
- **Expected Output:**
- ➢ **Before/afterlist**
- ➢ **Screenshot of Colab execution**
- ➢ **Input:**


colab.research.google.com/drive/1uGI2IUDtF0nTJXZW4NZh4YQ9WwP_Kma5#scrollTo=4bbf79ac
Untitled23.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands
+ Code
+ Text
Run all
RAM
Disk
[2]
0s
Apply remove_negative_readings to a new list
Explain the remove_negative_readings functic
What can I help you build?
Gemini 2.5 Flash
Variables
Terminal
21:21
Python 3

```
def remove_negative_readings(sensor_readings):
    """
    Removes all negative values from a list of sensor readings.

    Args:
        sensor_readings (list): A list of numerical sensor readi

    Returns:
        list: A new list with all negative values removed.
    """
    print(f"Original readings: {sensor_readings}")
    cleaned_readings = [reading for reading in sensor_readings i
    print(f"Cleaned readings: {cleaned_readings}")
    return cleaned_readings

# Example usage:
sensor_data = [10, -5, 20, 0, -15, 30, -2, 8]
result_data_1 = remove_negative_readings(sensor_data)
print(f"Function returned: {result_data_1}")

# Another example
```

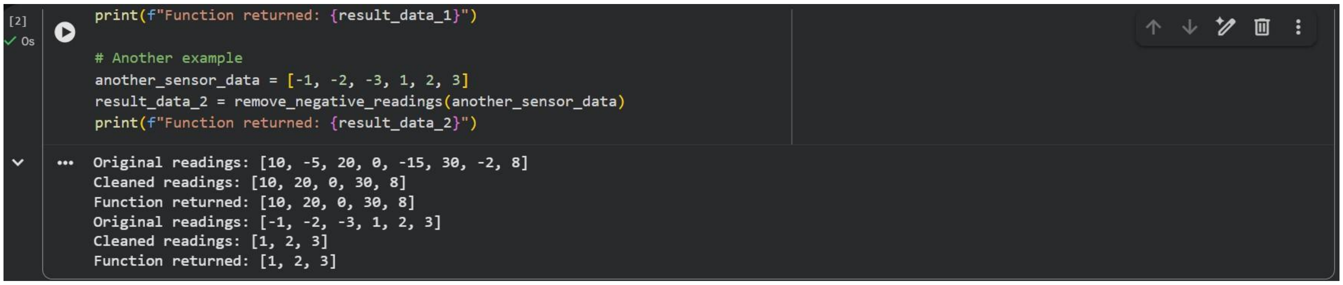
Generate a Python function that removes all negative values from a list of sensor readings and display the list before and after cleaning.

Sure, here's a Python function that removes all negative values from a list of sensor readings and displays the list before and after cleaning.

The code executed successfully. The `remove_negative_readings`

Gemini can make mistakes, so double-check it and use code with caution. Learn more

Output:



```
print(f"Function returned: {result_data_1}")

# Another example
another_sensor_data = [-1, -2, -3, 1, 2, 3]
result_data_2 = remove_negative_readings(another_sensor_data)
print(f"Function returned: {result_data_2}")
```

```
Original readings: [10, -5, 20, 0, -15, 30, -2, 8]
Cleaned readings: [10, 20, 0, 30, 8]
Function returned: [10, 20, 0, 30, 8]
Original readings: [-1, -2, -3, 1, 2, 3]
Cleaned readings: [1, 2, 3]
Function returned: [1, 2, 3]
```

Task2: String Character Analysis

❖ **Scenario:**

You are building a text-analysis feature.

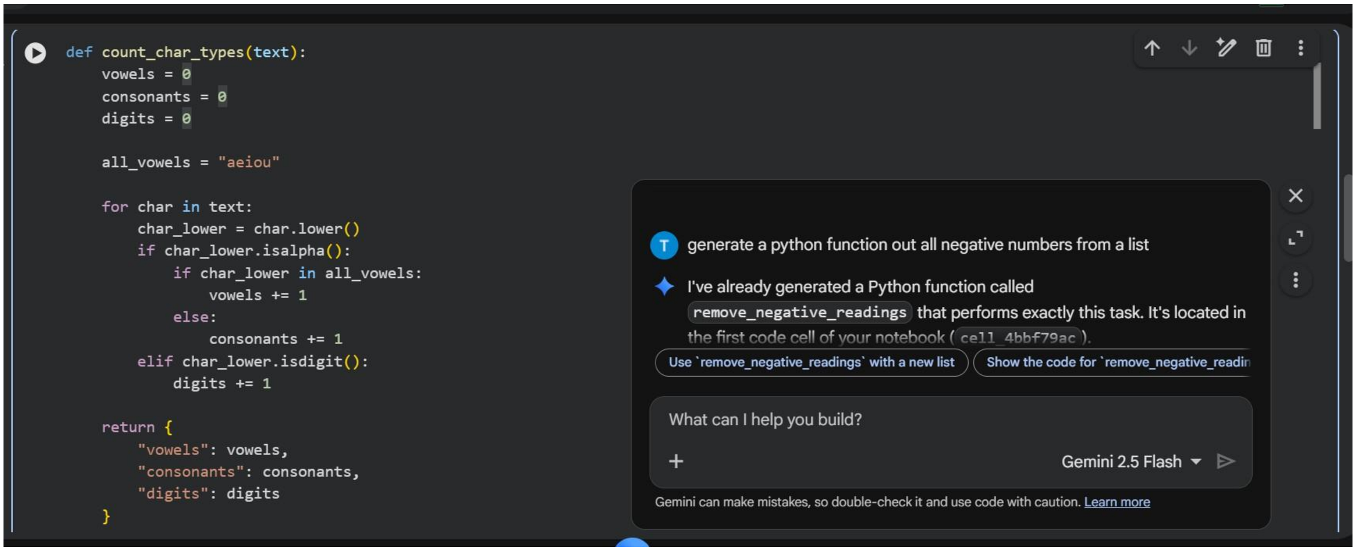
❖ **Task: Use Gemini to generate a Python functionthatcountsvowels,consonants,digits**

❖ **Expected Output:**

➢ **Working function**

➢ **Sample inputs and outputs**

➢ Input:



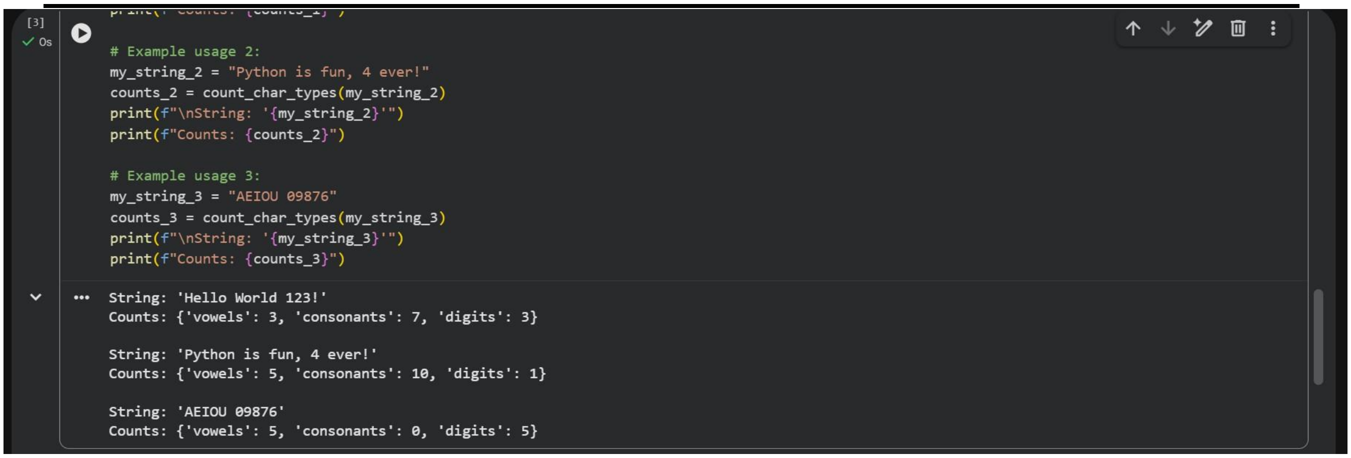
```
def count_char_types(text):
    vowels = 0
    consonants = 0
    digits = 0

    all_vowels = "aeiou"

    for char in text:
        char_lower = char.lower()
        if char_lower.isalpha():
            if char_lower in all_vowels:
                vowels += 1
            else:
                consonants += 1
        elif char_lower.isdigit():
            digits += 1

    return {
        "vowels": vowels,
        "consonants": consonants,
        "digits": digits
    }
```

generate a python function out all negative numbers from a list

I've already generated a Python function called `remove_negative_readings` that performs exactly this task. It's located in the first code cell of your notebook (`cell_4bbf79ac`).

Use `remove_negative_readings` with a new list
Show the code for `remove_negative_readin
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes, so double-check it and use code with caution. Learn more

Output:



```
# Example usage 2:
my_string_2 = "Python is fun, 4 ever!"
counts_2 = count_char_types(my_string_2)
print(f"\nString: '{my_string_2}'")
print(f"Counts: {counts_2}")

# Example usage 3:
my_string_3 = "AEIOU 09876"
counts_3 = count_char_types(my_string_3)
print(f"\nString: '{my_string_3}'")
print(f"Counts: {counts_3}")
```

```
String: 'Hello World 123!'
Counts: {'vowels': 3, 'consonants': 7, 'digits': 3}

String: 'Python is fun, 4 ever!'
Counts: {'vowels': 5, 'consonants': 10, 'digits': 1}

String: 'AEIOU 09876'
Counts: {'vowels': 5, 'consonants': 0, 'digits': 5}
```

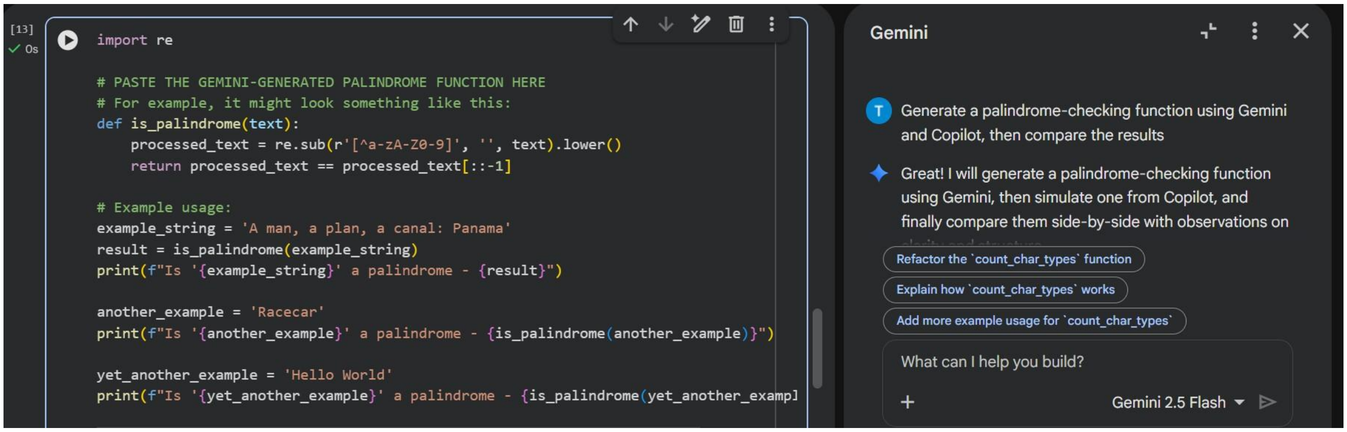
❖ **Task:**

Generate a palindrome-checking function using Gemini and Copilot, then compare the results.

❖ **Expected Output:**

➢ **Side-by-side code comparison**

➢ **Observations on clarity and structure**



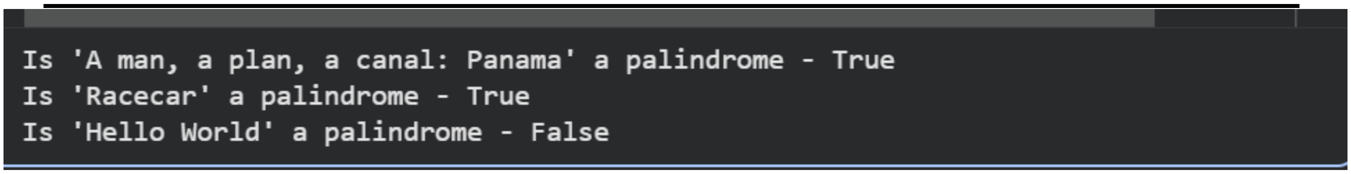
```python
import re

# PASTE THE GEMINI-GENERATED PALINDROME FUNCTION HERE
# For example, it might look something like this:
def is_palindrome(text):
    processed_text = re.sub(r'[^a-zA-Z0-9]', '', text).lower()
    return processed_text == processed_text[::-1]

# Example usage:
example_string = 'A man, a plan, a canal: Panama'
result = is_palindrome(example_string)
print(f"Is '{example_string}' a palindrome - {result}")

another_example = 'Racecar'
print(f"Is '{another_example}' a palindrome - {is_palindrome(another_example)}")

yet_another_example = 'Hello World'
print(f"Is '{yet_another_example}' a palindrome - {is_palindrome(yet_another_exampl
```

T Generate a palindrome-checking function using Gemini and Copilot, then compare the results

Great! I will generate a palindrome-checking function using Gemini, then simulate one from Copilot, and finally compare them side-by-side with observations on

Refactor the `count_char_types` function
Explain how `count_char_types` works
Add more example usage for `count_char_types`
What can I help you build?
+
Gemini 2.5 Flash

Output:



```
Is 'A man, a plan, a canal: Panama' a palindrome - True
Is 'Racecar' a palindrome - True
Is 'Hello World' a palindrome - False
```

Task4:Code Explanation Using AI

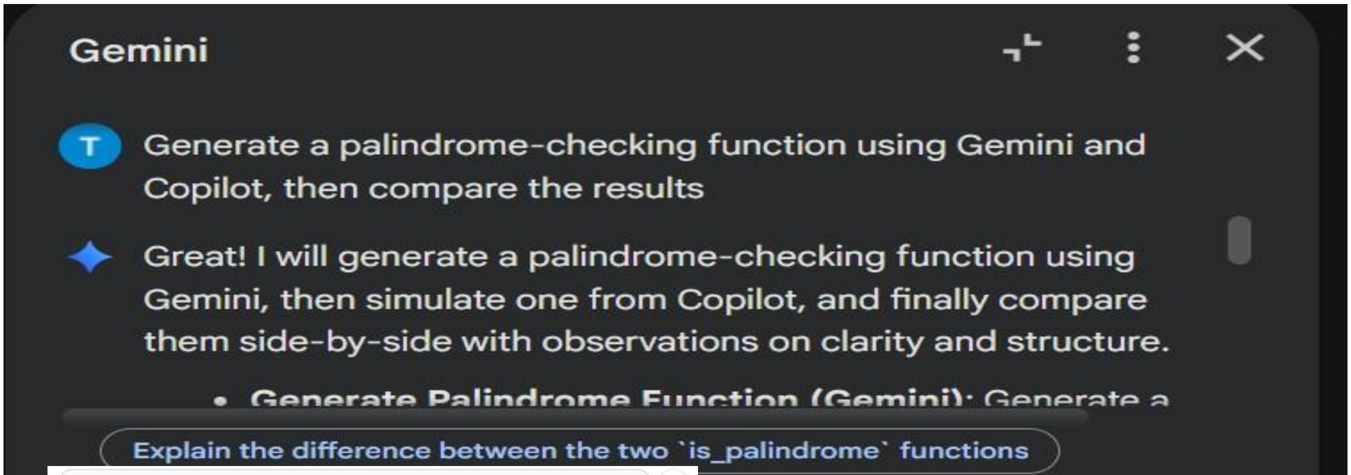
- ❖ **Scenario:**

You are reviewing un familiar code written by another developer.

- ❖ **Task: Ask Gemini to explaina Python function(prime check OR palindrome check) line by line.**
- ❖ **Expected Output:**
- ➢ **Code snippet**
- ➢ **AI explanation**

Student comments on understanding

OUTPUT:

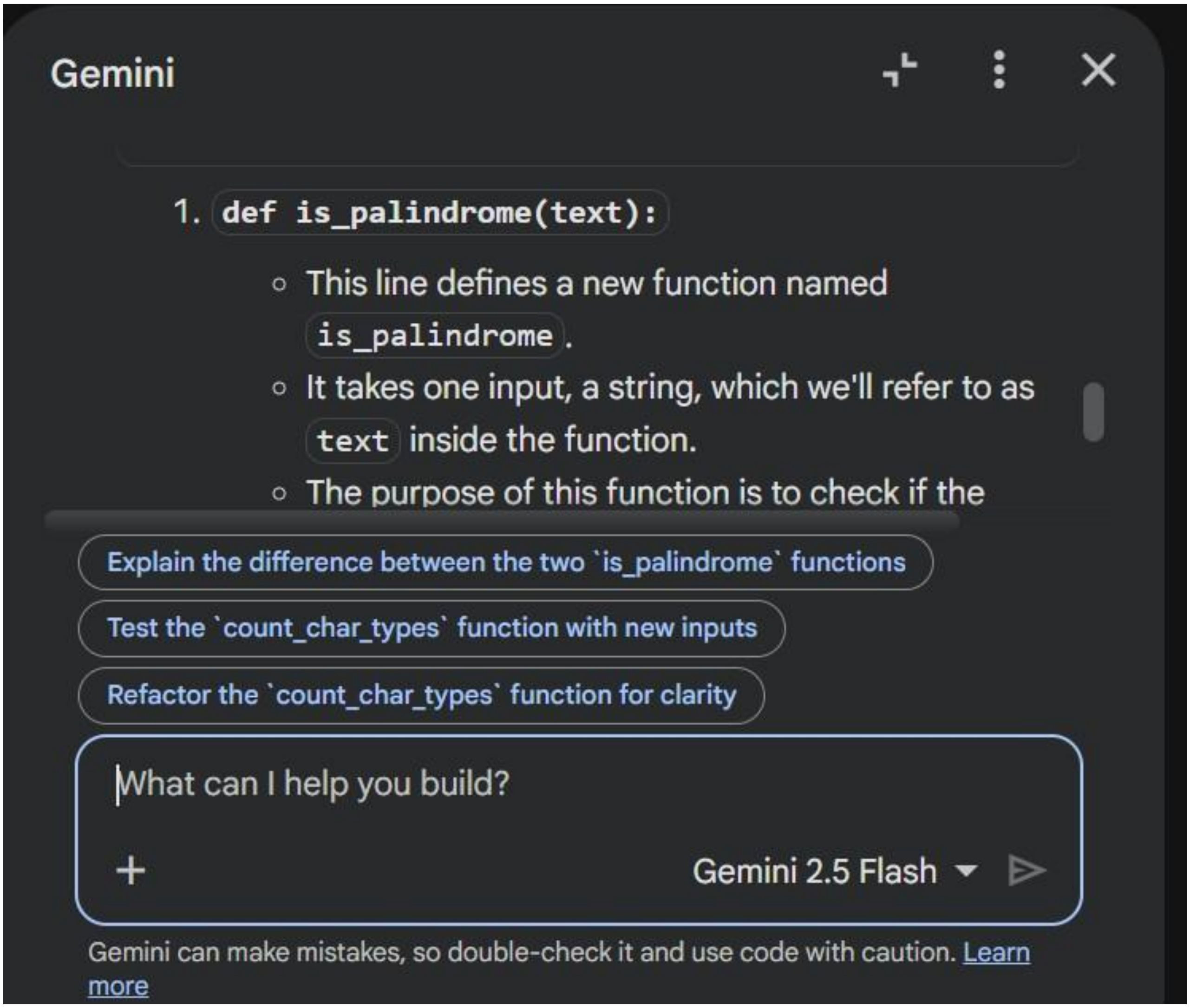


T Generate a palindrome-checking function using Gemini and Copilot, then compare the results

Great! I will generate a palindrome-checking function using Gemini, then simulate one from Copilot, and finally compare them side-by-side with observations on clarity and structure.

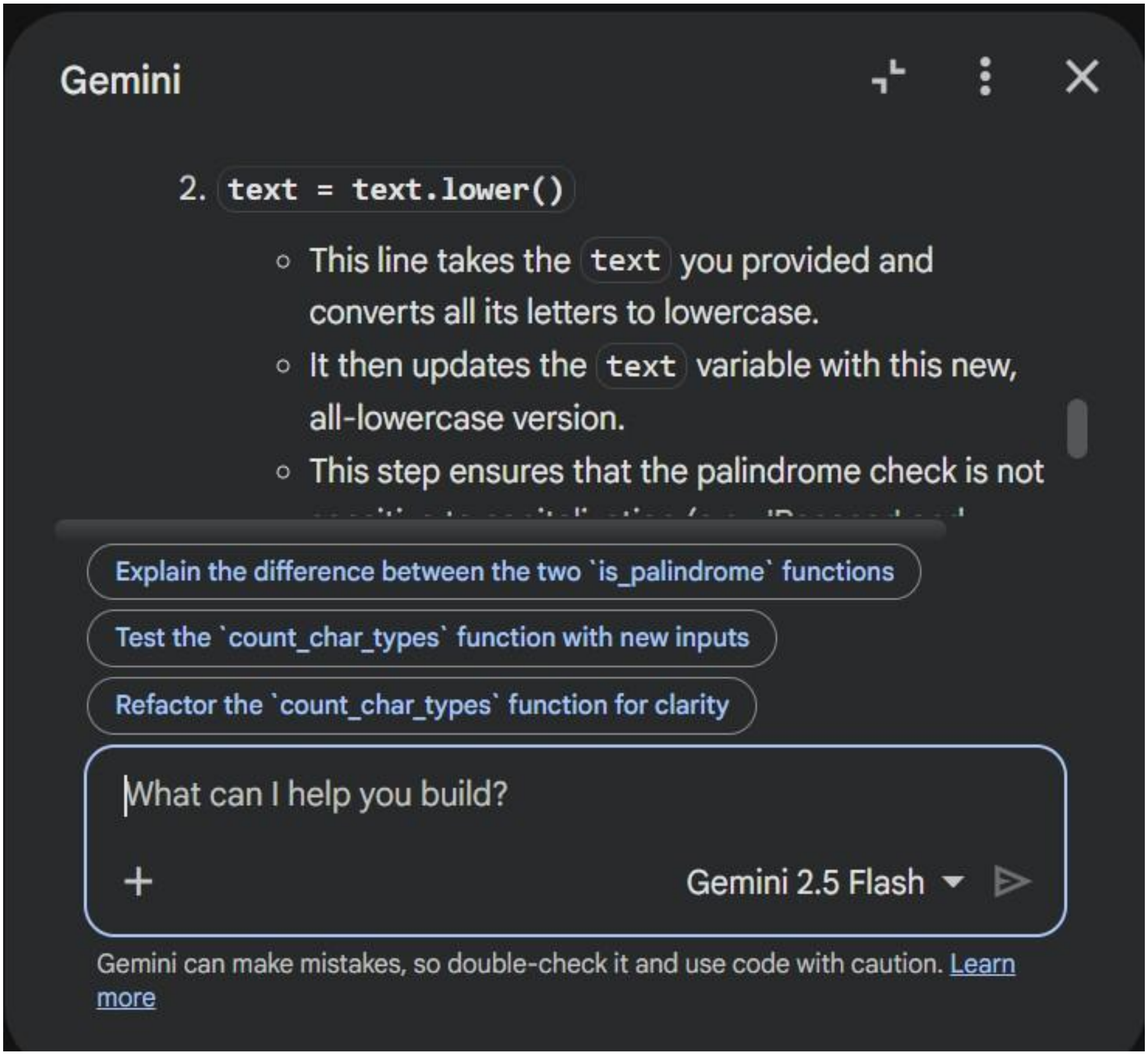
- **Generate Palindrome Function (Gemini):** Generate a

Explain the difference between the two `is_palindrome` functions


Gemini
Explain the difference between the two `is_palindrome` functions
Test the `count_char_types` function with new inputs
Refactor the `count_char_types` function for clarity
What can I help you build?
+
Gemini 2.5 Flash

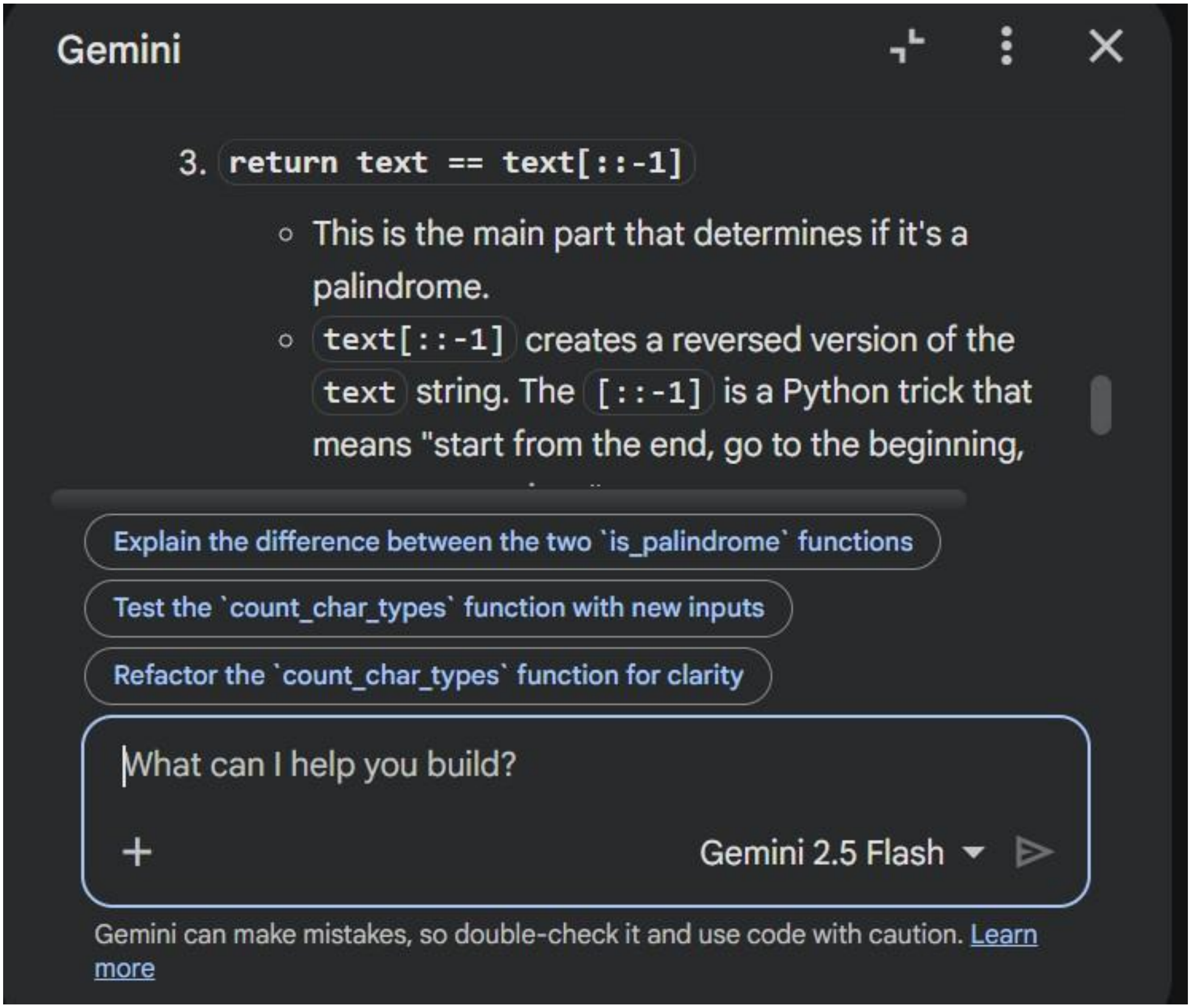
1. `def is_palindrome(text):`
    - This line defines a new function named `is_palindrome`.
    - It takes one input, a string, which we'll refer to as `text` inside the function.
    - The purpose of this function is to check if the

Gemini can make mistakes, so double-check it and use code with caution. Learn more


Gemini
Explain the difference between the two `is_palindrome` functions
Test the `count_char_types` function with new inputs
Refactor the `count_char_types` function for clarity
What can I help you build?
Gemini 2.5 Flash

2. `text = text.lower()`
    - This line takes the `text` you provided and converts all its letters to lowercase.
    - It then updates the `text` variable with this new, all-lowercase version.
    - This step ensures that the palindrome check is not

Gemini can make mistakes, so double-check it and use code with caution. Learn more



3. `return text == text[::-1]`
   - This is the main part that determines if it's a palindrome.
   - `text[::-1]` creates a reversed version of the `text` string. The `[::-1]` is a Python trick that means "start from the end, go to the beginning,

Explain the difference between the two `is_palindrome` functions
Test the `count_char_types` function with new inputs
Refactor the `count_char_types` function for clarity
What can I help you build?
Gemini 2.5 Flash

Gemini can make mistakes, so double-check it and use code with caution. Learn more

My own experience using both Gemini and GitHub Copilot:

While using Gemini in Google Colab, I found its explanations very clear and helpful in understanding the logic behind Python programs step by step. Gemini was useful for learning and analyzing code conceptually. GitHub Copilot, on the other hand, was faster in generating code directly inside the editor. It helped complete coding tasks quickly and was suitable for continuous coding. Overall, using both tools together improved my understanding and coding efficiency.