

AI ASSISTANT CODING ASSIGNMENT - 2

NAME: JILLELA AKSHAYA

HT.NO: 2303A51629

BATCH: 22

LAB 2

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

Task 1: 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/after list

➢ Screenshot of Colab execution

CODE :

The screenshot shows a Google Colab notebook titled "Untitled3.ipynb". In the code editor, there is a function definition:

```
def filter_negative_numbers(numbers):
    """
    Filters out all negative numbers from a list.

    Args:
        numbers: A list of numbers (integers or floats).

    Returns:
        A new list containing only the non-negative numbers.
    """
    return [num for num in numbers if num >= 0]
```

A tooltip from "Gemini" provides an example usage:

```
# Example usage with your existing 'my_list'
filtered_list = filter_negative_numbers(my_list)
```

Below the code, a Gemini AI interface is shown, suggesting a response:

generate a python function that filters out all negative numbers from a list

Certainly! Here's a Python function to filter out negative numbers from a list, along with an example of how to use it with your existing `my_list` variable.

Accept and run Accept Cancel

What can I help you build?

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

The status bar at the bottom shows the date and time: 13:59, 14/03, 15-01-2026.

OUTPUT:

```
Args:
    numbers: A list of numbers (integers or floats).

    Returns:
        A new list containing only the non-negative numbers.
    """
    return [num for num in numbers if num >= 0]

# Example usage with your existing 'my_list'
filtered_list = filter_negative_numbers(my_list)

print(f"Original list: {my_list}")
print(f"Filtered list (non-negative numbers): {filtered_list}")

Original list: [-5, 10, -2, 8, 0, -1]
Filtered list (non-negative numbers): [10, 8, 0]
```

Task 2: String Character Analysis

❖ Scenario:

You are building a text-analysis feature.

❖ Task:

Use Gemini to generate a Python function that counts vowels, consonants, and digits in a string.

❖ Expected Output:

➤ Working function

➤ Sample inputs and outputs

CODE :

The screenshot shows a Google Colab notebook titled "Untitled3.ipynb". The code defines a function `count_characters` that takes a string and returns a tuple of vowel, consonant, and digit counts. A Gemini AI sidebar is open, asking for a Python function that counts vowels, consonants, and digits in a string. The AI generates the provided code. The AI sidebar also includes a "What can I help you build?" input field and a "Gemini 2.5 Flash" dropdown.

```
def count_characters(input_string):
    """
    Counts the number of vowels, consonants, and digits in a given string.

    Args:
        input_string (str): The string to analyze.

    Returns:
        tuple: A tuple containing (vowel_count, consonant_count, digit_count).
    """
    vowels = "aeiouAEIOU"
    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
```

OUTPUT:

The screenshot shows the execution of the `count_characters` function with the sample string "Hello World 123!". The AI sidebar provides example usage and prints the original string and the counts for vowels, consonants, and digits.

```
return vowel_count, consonant_count, digit_count

# Example usage:
sample_string = "Hello World 123!"
v, c, d = count_characters(sample_string)

print(f"Original string: {sample_string}")
print(f"Number of vowels: {v}")
print(f"Number of consonants: {c}")
print(f"Number of digits: {d}")
```

Task 3: Palindrome Check – Tool Comparison

❖ **Scenario:**

You must decide which AI tool is clearer for string logic.

Task:

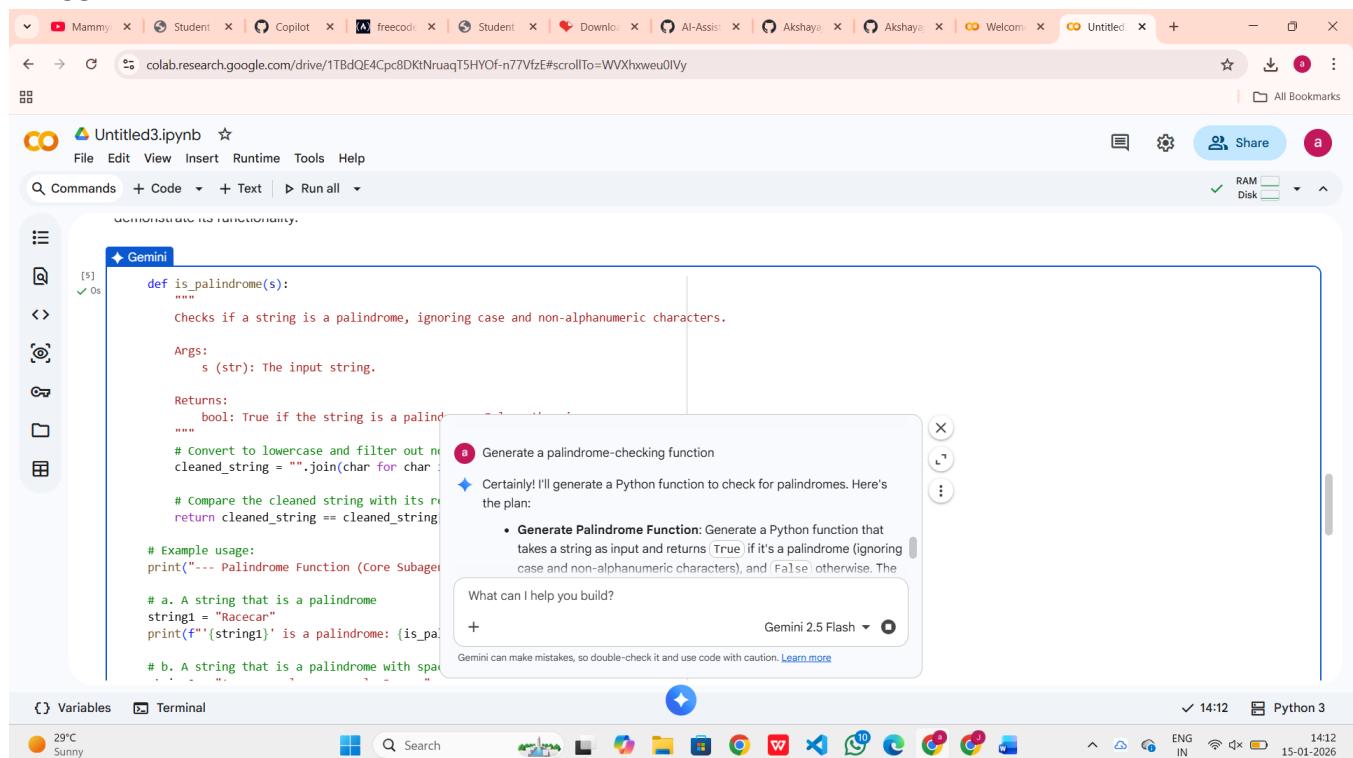
Generate a palindrome-checking function using Gemini and Copilot, then compare the results.

❖ Expected Output:

➤ Side-by-side code comparison

➤ Observations on clarity and structure

CODE:



The screenshot shows a Jupyter Notebook cell with the following code:

```
def is_palindrome(s):
    """
    Checks if a string is a palindrome, ignoring case and non-alphanumeric characters.

    Args:
        s (str): The input string.

    Returns:
        bool: True if the string is a palindrome.
    """
    # Convert to lowercase and filter out non-alphanumeric characters
    cleaned_string = ''.join(char for char in s if char.isalnum())
    
    # Compare the cleaned string with its reverse
    return cleaned_string == cleaned_string[::-1]
```

Below the code, there is a sidebar with the following text and options:

- Generate a palindrome-checking function
- Certainly! I'll generate a Python function to check for palindromes. Here's the plan:
 - Generate Palindrome Function: Generate a Python function that takes a string as input and returns `True` if it's a palindrome (ignoring case and non-alphanumeric characters), and `False` otherwise.

At the bottom of the sidebar, there is a "What can I help you build?" input field and a "Gemini 2.5 Flash" button.

OUTPUT:

```
+print(is_palindrome("A man, a plan, a canal: Panama")) # Output: True
+print(is_palindrome("race a car"))      # Output: False
+print(is_palindrome("hello"))          # Output: False
+
...
...  True
...  True
...  False
...  False
```

Task 4: Code Explanation Using AI

❖ Scenario:

You are reviewing unfamiliar code written by another developer.

❖ Task:

Ask Gemini to explain a Python function (prime check OR palindrome check) line by line.

❖ Expected Output:

➤ Code snippet

➤ AI explanation

➤ Student comments on understanding

➤ OUPUT

The screenshot shows the Gemini AI interface. At the top, there is a code snippet and its explanation:

1. `def is_palindrome(text):`
This line defines a Python function named `is_palindrome` that accepts one argument: `text` (which is expected to be a string).

2. `"""\n """`

Below the code, there are three buttons: `Accept & Run`, `Accept`, and `Cancel`.
A message bar at the bottom says: "What can I help you build?".
A note at the bottom states: "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)".
A blue star icon is located at the bottom center of the interface.

3. `processed_text = ''.join(char for char in text if char.isalnum()).lower()`

- This is the core of the preprocessing step:
 - `for char in text if char.isalnum()`: This part iterates through each character (`char`) in the input

Accept & Run Accept Cancel

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](#)

4. `return processed_text == processed_text[::-1]`

- This line performs the actual palindrome check and returns `True` or `False`.
- `processed_text[::-1]`: This creates a reversed version of the `processed_text` string. The `[::-1]`

Accept & Run Accept Cancel

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 working with **Gemini** in **Google Colab**, I noticed that the explanations were well-structured and easy to follow. It helped me understand the reasoning and flow behind Python programs by breaking them down step by step. This made Gemini particularly helpful for learning, debugging, and analyzing code logic.

On the other hand, **GitHub Copilot** focused more on speed and productivity. By generating code suggestions directly inside the editor, it allowed me to implement solutions quickly and continue coding without interruptions. Copilot felt more effective during active development, especially when working on longer programs.

In summary, Gemini supported deeper conceptual understanding, while GitHub Copilot enhanced coding speed. Using both tools together significantly improved my learning experience as well as overall coding efficiency.