AI ASSISTANT CODING ASSIGNMENT 2

NAME : MD SHAHID

ROLL NO : 2303A51421

BATCH NO : 21

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

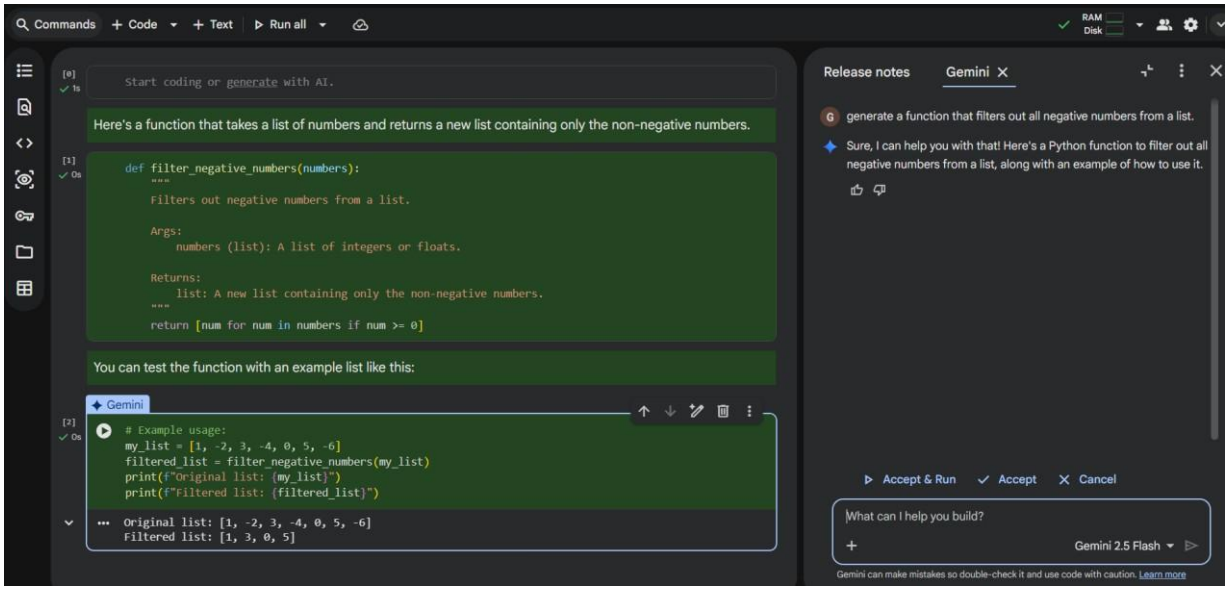
OUTPUT:

Here's a function that takes a list of numbers and returns a new list containing only the non-negative numbers.

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

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

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



Task 2: String Character Analysis

❖ **Scenario:**

You are building a text-analysis feature.

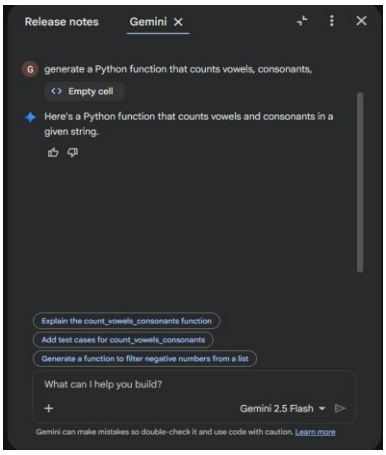
❖ **Task:**

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

❖ **Expected Output: ➢ Working function**

➢ **Sample inputs and outputs**

OUTPUT:



```python
def count_vowels_consonants(text):
    """
    Counts the number of vowels and consonants in a given string.

    Args:
        text (str): The input string.

    Returns:
        dict: A dictionary containing the counts of 'vowels' and 'consonants'.
    """
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0

    for char in text:
        if char.isalpha(): # Check if the character is an alphabet
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

    return {"vowels": vowel_count, "consonants": consonant_count}
```

```python
# Example usage:
my_string = "Hello World"
counts = count_vowels_consonants(my_string)
print(f"Original string: '{my_string}'")
print(f"Vowels: {counts['vowels']}")
print(f"Consonants: {counts['consonants']}")

my_string_2 = "Python Programming"
counts_2 = count_vowels_consonants(my_string_2)
print(f"\nOriginal string: '{my_string_2}'")
print(f"Vowels: {counts_2['vowels']}")
print(f"Consonants: {counts_2['consonants']}")
```

```
Original string: 'Hello World'
Vowels: 3
Consonants: 7

Original string: 'Python Programming'
Vowels: 4
Consonants: 13
```

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.

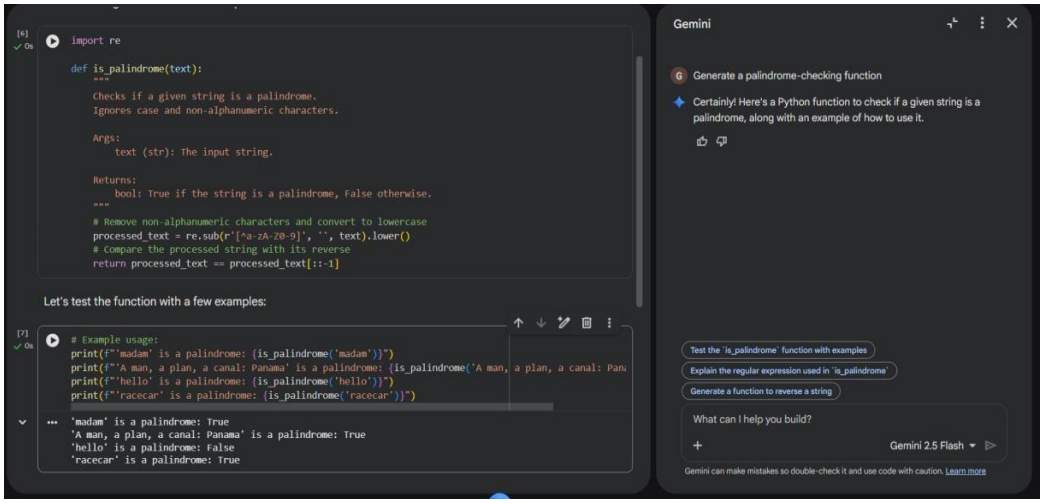
❖ Expected Output:

➢ Side-by-side code comparison

➢ Observations on clarity and structure

Output :

CODE GENERATED BY GEMINI:



CODE GENERATED BY COPILOT :

```python
#Generate a palindrome check function
def is_palindrome(s):
    # Remove spaces and convert to lowercase
    s = s.replace(" ", "").lower()
    # Check if the string is equal to its reverse
    return s == s[::-1]
# Example usage
print(is_palindrome("A man a plan a canal Panama"))  # Output: True
print(is_palindrome("Hello"))  # Output: False
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  AZURE

PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>  & 'c:\Users\23
3.13.exe' 'c:\Users\2303a\.vscode\extensions\ms-python.debugpy-2025.19.2026010901-win32-x6
'C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\new.py'
True
False
```

Observations on Clarity

Improved Version:

Gemini Code:

- **Clear, well-documented, and includes a proper docstring explaining the function's purpose, parameters, and return value.**
- **Professionally structured with a reusable function and multiple test cases.**
- **Handles edge cases thoroughly by stripping non-alphanumeric characters and normalizing case.**

Copilot Code:

- **Simple and easy to understand, especially for beginners.**
- **Uses a short variable name (s)—functional but not descriptive.**

- **Produces only a raw True/False output with no additional context.**
- **Structurally compact, focusing only on space removal and lowercase conversion.**

Summary:

- **Gemini's version is more complete, robust, and ready for real-world use.**
- **Copilot's version is minimal and beginner-friendly—fine for quick checks but not as comprehensive.**

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.

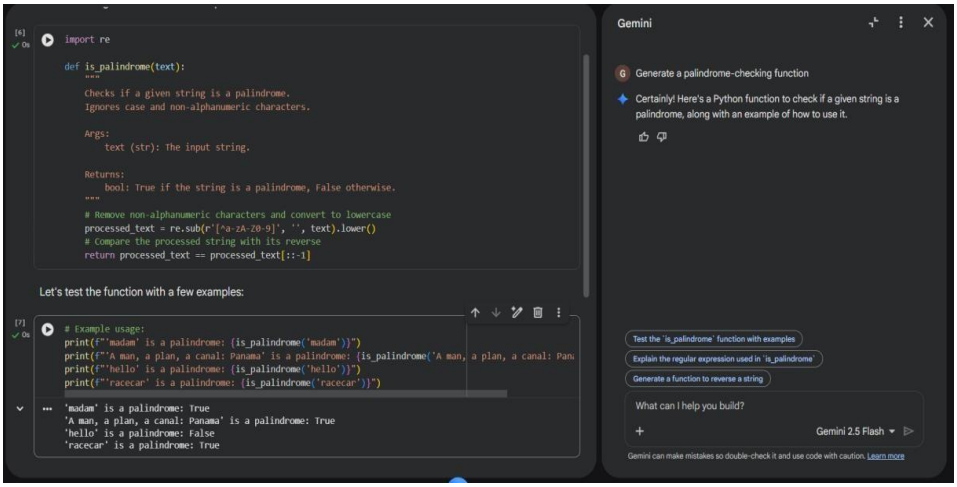
❖ Expected Output:

➢ Code snippet

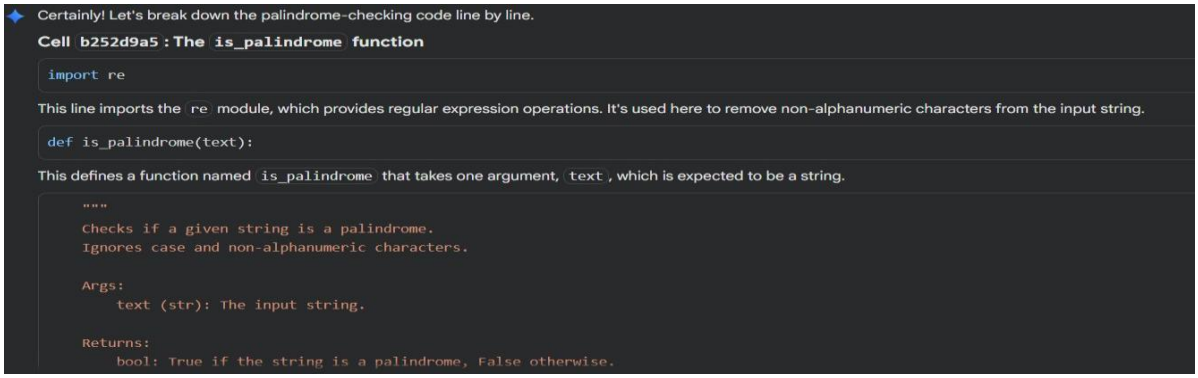
➢ AI explanation

➢ Student comments on understanding

OUTPUT :



AI Explanation :



Certainly! Let's break down the palindrome-checking code line by line.

Cell `b252d9a5` : The `is_palindrome` function

```
import re
```

This line imports the `re` module, which provides regular expression operations. It's used here to remove non-alphanumeric characters from the input string.

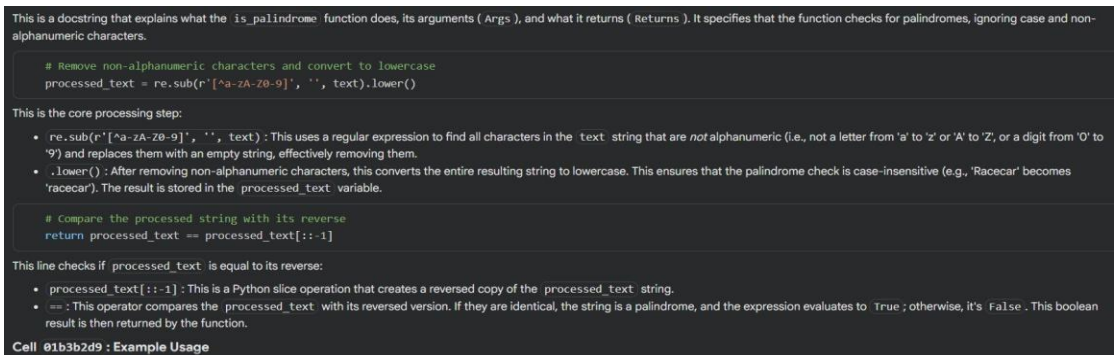
```
def is_palindrome(text):
```

This defines a function named `is_palindrome` that takes one argument, `text`, which is expected to be a string.

```
    """
    Checks if a given string is a palindrome.
    Ignores case and non-alphanumeric characters.

    Args:
        text (str): The input string.

    Returns:
        bool: True if the string is a palindrome, False otherwise.
```



This is a docstring that explains what the `is_palindrome` function does, its arguments (`Args`), and what it returns (`Returns`). It specifies that the function checks for palindromes, ignoring case and non-alphanumeric characters.

```
    # Remove non-alphanumeric characters and convert to lowercase
    processed_text = re.sub(r'[^a-zA-Z0-9]', '', text).lower()
```

This is the core processing step:

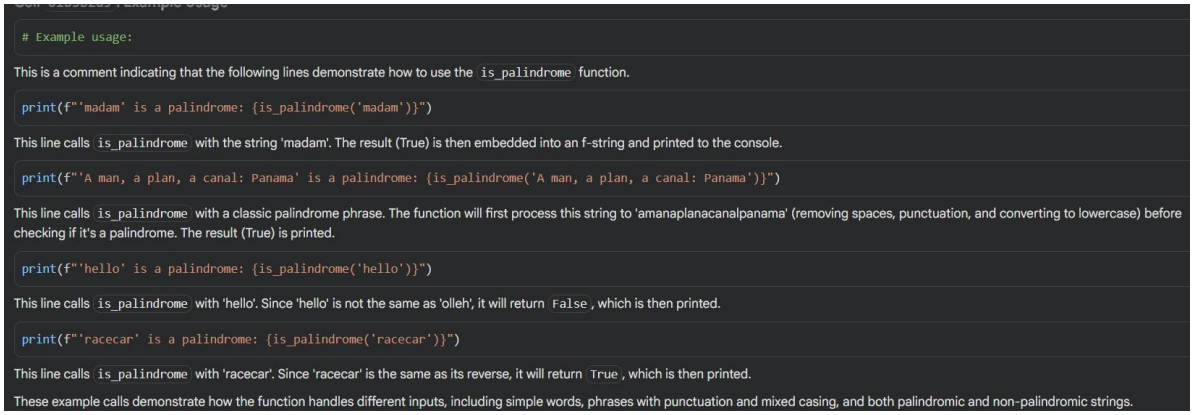
- `re.sub(r'[^a-zA-Z0-9]', '', text)` : This uses a regular expression to find all characters in the `text` string that are *not* alphanumeric (i.e., not a letter from 'a' to 'z' or 'A' to 'Z', or a digit from '0' to '9') and replaces them with an empty string, effectively removing them.
- `.lower()` : After removing non-alphanumeric characters, this converts the entire resulting string to lowercase. This ensures that the palindrome check is case-insensitive (e.g., 'Racecar' becomes 'racecar'). The result is stored in the `processed_text` variable.

```
    # Compare the processed string with its reverse
    return processed_text == processed_text[::-1]
```

This line checks if `processed_text` is equal to its reverse:

- `processed_text[::-1]` : This is a Python slice operation that creates a reversed copy of the `processed_text` string.
- `==` : This operator compares the `processed_text` with its reversed version. If they are identical, the string is a palindrome, and the expression evaluates to `True` ; otherwise, it's `False` . This boolean result is then returned by the function.

Cell `01b3b2d9` : Example Usage



```
# Example usage:
```

This is a comment indicating that the following lines demonstrate how to use the `is_palindrome` function.

```
print(f"'madam' is a palindrome: {is_palindrome('madam')}")
```

This line calls `is_palindrome` with the string 'madam'. The result (True) is then embedded into an f-string and printed to the console.

```
print(f"'A man, a plan, a canal: Panama' is a palindrome: {is_palindrome('A man, a plan, a canal: Panama')}")
```

This line calls `is_palindrome` with a classic palindrome phrase. The function will first process this string to 'amanaplanacanalpanama' (removing spaces, punctuation, and converting to lowercase) before checking if it's a palindrome. The result (True) is printed.

```
print(f"'hello' is a palindrome: {is_palindrome('hello')}")
```

This line calls `is_palindrome` with 'hello'. Since 'hello' is not the same as 'olleh', it will return `False` , which is then printed.

```
print(f"'racecar' is a palindrome: {is_palindrome('racecar')}")
```

This line calls `is_palindrome` with 'racecar'. Since 'racecar' is the same as its reverse, it will return `True` , which is then printed.

These example calls demonstrate how the function handles different inputs, including simple words, phrases with punctuation and mixed casing, and both palindromic and non-palindromic strings.

Clear Understanding:

- **“The explanation made the role of regular expressions clear—I finally understand how they clean the string.”**
- **“I see why .lower() matters now; it ensures the palindrome check is case-insensitive.”**
- **“The slicing trick [::-1] was new to me, and it’s an elegant way to reverse a string.”**
- **“The examples clarified everything, especially how the function handles both simple words and more complex phrases.”**