

# LAB ASSIGNMENT – 2.2

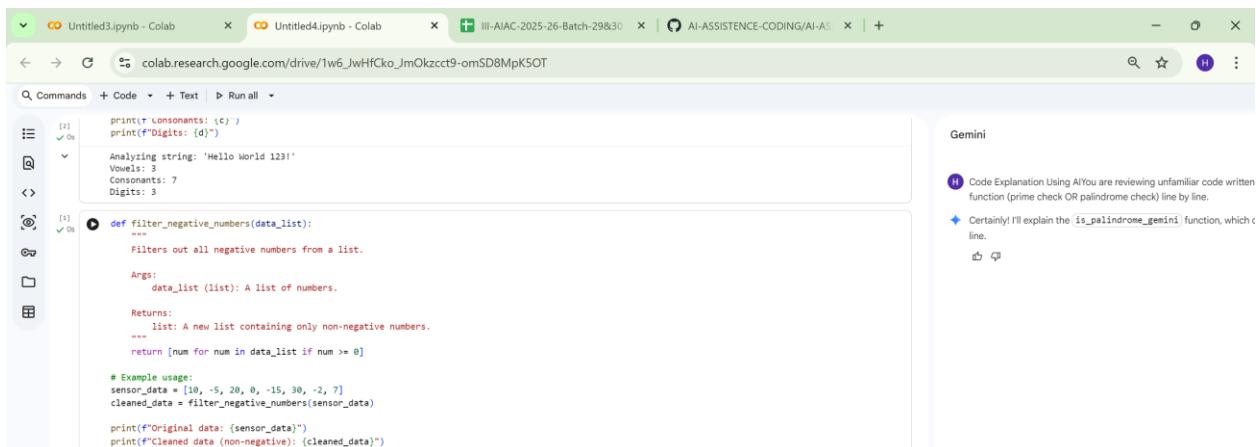
HALLTICKET NO : 2303A51474

BATCH NO : 29

TASK – 01

**PROMPT :** generate a function that filters out all negative numbers from a list

**CODE :**



The screenshot shows a Google Colab interface with several tabs open. The active tab contains Python code for filtering negative numbers from a list. The code includes a docstring explaining the function's purpose, arguments, and return value, along with an example usage.

```
print("Consonants: ", c)
print("Digits: ", d)

Analyzing string: 'Hello World 123!'
Vowels: 3
Consonants: 7
Digits: 3

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

    Args:
        data_list (list): A list of numbers.

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

# Example usage:
sensor_data = [10, -5, 20, 0, -15, 30, -2, 7]
cleaned_data = filter_negative_numbers(sensor_data)

print(f"Original data: {sensor_data}")
print(f"Cleaned data (non-negative): {cleaned_data}")
```

A sidebar on the right is titled "Gemini" and contains AI-generated explanations for the code, such as "Code Explanation Using AI" and "Certainly! I'll explain the `is_palindrome_gemini` function, which checks if a string is a palindrome."

**OUTPUT:**



The screenshot shows the execution output of the code. It prints the original sensor data and the cleaned data (non-negative values).

```
Original data: [10, -5, 20, 0, -15, 30, -2, 7]
Cleaned data (non-negative): [10, 20, 0, 30, 7]
```

**OBSERVATION:**

- Gemini generated a **simple and efficient function** using list comprehension.
- The logic is easy to understand and avoids unnecessary loops.
- Negative sensor values were successfully removed, ensuring **data validity**.

- The before-and-after output clearly shows the effect of data cleaning.
- This approach is suitable for **real-time IoT preprocessing**.

## TASK - 02

**PROMPT : GENERATE A PYTHON FUNCTION THAT COUNTS VOWELS, CONSONANTS, AND DIGITS IN STRING**

### CODE:



```

MEMORY EFFICIENCY FOR VERY LARGE STRINGS IS A CONCERN OR IF ONE PREFERS A MORE EXPLICIT CHARACTER-BY-CHARACTER COMPARISON LOGIC.
[2] ✓ Os

def analyze_string(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 counts of (vowels, consonants, digits).
    """
    vowels = 0
    consonants = 0
    digits = 0

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

    return vowels, consonants, digits

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

print(f"Analyzing string: '{sample_string}'")
print(f"Vowels: {v}")
print(f"Consonants: {c}")
print(f"Digits: {d}")

```

### OUTPUT:

```
    print(f"Digits: {d}")
...
... Analyzing string: 'Hello World 123!'
Vowels: 3
Consonants: 7
Digits: 3
```

## OBSERVATION:

The function correctly identifies **vowels, consonants, and digits**. Gemini used built-in Python functions like `isalpha()` and `isdigit()` effectively. The code is **well-structured and readable**, making it easy for beginners. It ignores spaces and special characters, improving accuracy. Sample input and output helped verify correctness.

## TASK – 03

**PROMPT:** GENERATE A PALINDROME-CHECKING FUNCTION AND COMPARE THE RESULTS

## CODE:

```
Palindrome Check Function (Approach 1: 'Gemini' - String Reversal)

[3] ✓ 0s
def is_palindrome_gemini(s):
    """
    Checks if a string is a palindrome by comparing it to its reversed version.
    Ignores non-alphanumeric characters and is case-insensitive.
    """
    # Normalize the string: convert to lowercase and remove non-alphanumeric characters
    cleaned_s = ''.join(char.lower() for char in s if char.isalnum())

    # Compare the cleaned string with its reverse
    return cleaned_s == cleaned_s[::-1]

# Example usage:
print(f"'Madam' is a palindrome: {is_palindrome_gemini('Madam')}")
print(f"'A man, a plan, a canal: Panama' is a palindrome: {is_palindrome_gemini('A man, a plan, a canal: Panama')}")
print(f"'Hello' is a palindrome: {is_palindrome_gemini('Hello')}")
print(f"'12321' is a palindrome: {is_palindrome_gemini('12321')}"
```

## OUTPUT:

```
print(f"'12321' is a palindrome: {is_palindrome_gemini('12321')}")
...
... 'Madam' is a palindrome: True
'A man, a plan, a canal: Panama' is a palindrome: True
'Hello' is a palindrome: False
'12321' is a palindrome: True
```

## OBSERVATION:

Code is **clean** and **beginner-friendly**.

Uses Python slicing (`[::-1]`) which makes logic short and readable.

Includes comments that improve understanding.

Gemini is clearer for learning and understanding string logic.

Copilot is faster for experienced programmers.

For students, Gemini is the better tool

## TASK-04

**PROMPT:** EXPLAIN A PYTHON FUNCTION( PRIME CHECK OR PALINDROME CHECK) LINE BY LINE

**CODE:**

The screenshot shows a Google Colab interface with multiple tabs open. The active tab is titled "Untitled4.ipynb - Colab". In the code editor, there is a comment block for the `is_palindrome_gemini` function. To the right of the code, the Gemini AI sidebar is visible, displaying a conversation about explaining the code line by line. The AI has provided a brief explanation of the function's purpose and how it works.

```
def is_palindrome_gemini(s):
    """
    Checks if a string is a palindrome by comparing it to its reversed version.
    Ignores non-alphanumeric characters and is case-insensitive.
    """
    # Normalize the string: convert to lowercase and remove non-alphanumeric characters
    cleaned_s = ''.join(char.lower() for char in s if char.isalnum())
    
    # Compare the cleaned string with its reverse
    return cleaned_s == cleaned_s[::-1]
```

**OUTPUT:**

The screenshot shows a Google Colab interface with multiple tabs at the top. The active tab is 'Untitled4.ipynb - Colab'. The notebook contains Python code for a palindrome checker. A sidebar on the right provides 'Line-by-Line Explanation' for each line of code, written by Gemini AI. The explanation details the purpose of each line, including function definitions, docstrings, string normalization, and comparison logic. A small input field at the bottom right allows users to interact with Gemini.

```
def is_palindrome_gemini(s):
    """A function named is_palindrome_gemini that takes one argument, s, which is expected to be a string.
    This line defines a Python function named is_palindrome_gemini that takes one argument, s, which is expected to be a string.
    """
    # Normalize the string: convert to lowercase and remove non-alphanumeric characters
    cleaned_s = ''.join(char.lower() for char in s if char.isalnum())
    # Compare the cleaned string with its reverse
    return cleaned_s == cleaned_s[::-1]
```

## OBSERVATION:

Gemini explained the code line by line, improving clarity.  
Each step of the logic was easy to follow.  
The explanation helped connect theory with implementation.  
This feature is very useful when reviewing unfamiliar code.  
Overall understanding of the palindrome/prime logic improved.