

# ASSIGNMENT-02

Name: Thalla Sreeja

Roll No: 2303A51220

Batch No: 04

## Task 1: Cleaning Sensor Data

**Scenario:**

❖ You are cleaning IoT sensor data where negative values are invalid.

❖ Task:

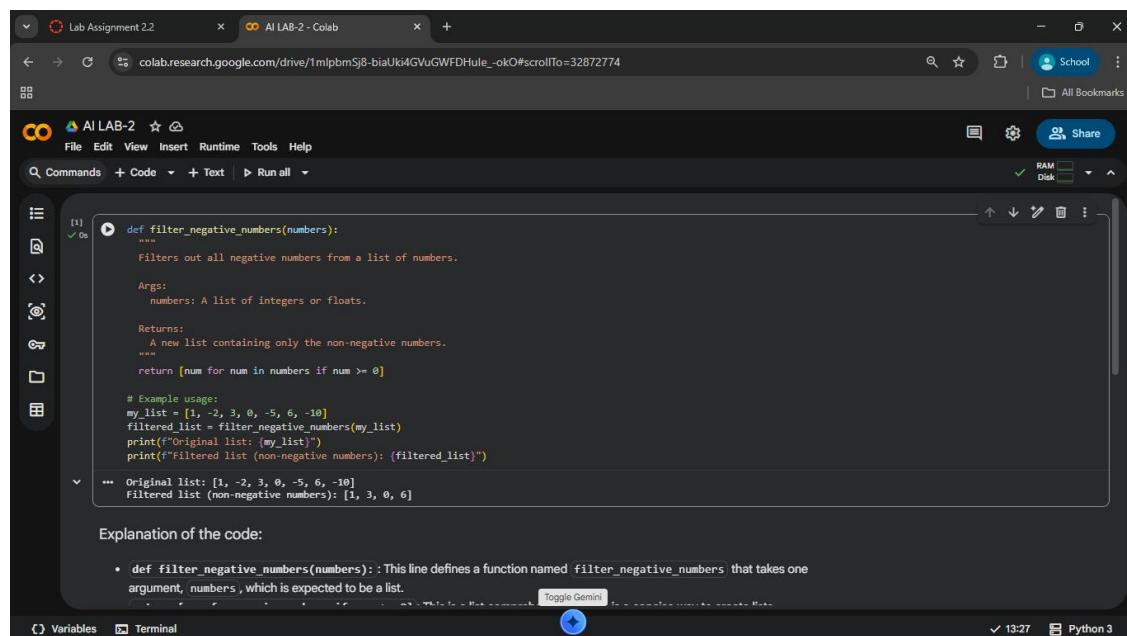
Use Gemini in Colab to generate a function that filters out all negative numbers from a list.

❖ Expected Output:

➤ Before/after list

➤ Screenshot of Colab execution

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



The screenshot shows a Google Colab notebook titled "AI LAB-2". The code cell contains the following Python function:

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

    Args:
        numbers: A list of integers or floats.

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

# Example usage:
my_list = [1, -2, 3, 0, -5, 6, -10]
filtered_list = filter_negative_numbers(my_list)
print("Original list: ", my_list)
print("Filtered list (non-negative numbers): ", filtered_list)
```

The output of the code is displayed below the cell:

```
Original list: [1, -2, 3, 0, -5, 6, -10]
Filtered list (non-negative numbers): [1, 3, 0, 6]
```

An explanation of the code is provided at the bottom of the cell:

- `def filter_negative_numbers(numbers):`: This line defines a function named `filter_negative_numbers` that takes one argument, `numbers`, which is expected to be a list.

## OUTPUT:

```
... Original list: [1, -2, 3, 0, -5, 6, -10]
Filtered list (non-negative numbers): [1, 3, 0, 6]
```

## Code Explanation:

The **filter\_negative\_numbers** function takes a list of numbers as input. It uses a list comprehension to iterate through the input list, selecting only the numbers that are greater than or equal to zero. Finally, it returns a new list containing these non-negative numbers.

## 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

Prompt: generate a Python function that counts vowels, consonants, and digits in a string

```

  s: The input string.
  Returns:
    A dictionary containing the counts of 'vowels', 'consonants', and 'digits'.
    """
    vowels = "aeiou"
    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in s:
        char_lower = char.lower()
        if 'a' <= char_lower <= 'z': # Check if it's an alphabet
            if char_lower in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
            elif '0' <= char <= '9': # Check if it's a digit
                digit_count += 1

    return {
        "vowels": vowel_count,
        "consonants": consonant_count,
        "digits": digit_count
    }

# Example usage:
my_string = "Hello World! 123"
counts_1 = count_chars(my_string)
print("Original string: '{my_string}'")
print("Counts: {counts_1}")

another_string = "Python Programming is fun 2024"
counts_2 = count_chars(another_string)
print("Original string: '{another_string}'")
print("Counts: {counts_2}")

```

The screenshot shows a Google Colab interface with a dark theme. The code cell contains a Python function named `count\_chars` that takes a string `s` as input and returns a dictionary with counts for vowels, consonants, and digits. The function uses a lowercase conversion loop and a dictionary comprehension to count each category. It includes example usages with two strings: "Hello World! 123" and "Python Programming is fun 2024". The output cell shows the results of these examples.

## OUTPUT:

```

  PRINT(counts_1) →
  --- Original string: 'Hello World! 123'
  Counts: {'vowels': 3, 'consonants': 7, 'digits': 3}

  PRINT(counts_2) →
  --- Original string: 'Python Programming is fun 2024'
  Counts: {'vowels': 6, 'consonants': 16, 'digits': 4}

```

## Code Explaination:

The **count\_chars** function iterates through a given string, converting each character to lowercase. It categorizes characters as vowels, consonants, or digits based on predefined sets and ranges. The function then tallies the counts for each category and returns them in a dictionary.

## 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

**Prompt:**

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

**Gemini**

The screenshot shows a Google Colab interface with two tabs: "Lab Assignment 2.2" and "AI LAB-2 - Colab". The "AI LAB-2 - Colab" tab is active, displaying a code cell with the following content:

```
Original string: 'Python Programming is fun 2024'
Counts: {'vowels': 6, 'consonants': 16, 'digits': 4}

import re
def is_palindrome(text):
    """
    Checks if a given string is a palindrome. Case-insensitive and ignores non-alphanumeric characters.

    Args:
        text (str): The string to check.

    Returns:
        bool: True if the string is a palindrome, False otherwise.
    """
    # convert to lowercase and remove non-alphanumeric characters
    processed_text = re.sub(r"[^a-zA-Z0-9]", "", text).lower()

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

# Example usage:
print("'madam' is a palindrome: (is_palindrome('madam'))")
print("A man, a plan, a canal: Panama' is a palindrome: (is_palindrome('A man, a plan, a canal: Panama'))")
print("'hello' is a palindrome: (is_palindrome('hello'))")
print("'Racecar' is a palindrome: (is_palindrome('Racecar'))")
print("No lemon, no melon' is a palindrome: (is_palindrome('No lemon, no melon'))")

... 'madam' is a palindrome: True
'A man, a plan, a canal: Panama' is a palindrome: True
'hello' is a palindrome: False
'Racecar' is a palindrome: True
'No lemon, no melon' is a palindrome: True
```

Below the code cell, there is an explanation of the `count_chars` function, a "Toggle Gemini" button, and a "Variables" and "Terminal" tab. The status bar at the bottom right shows "13:52" and "Python 3".

**OUTPUT**

The terminal output shows the execution of the `is_palindrome` function with various strings:

```
print("No lemon, no melon' is a palindrome: (is_palindrome('No lemon, no melon'))")
...
... 'madam' is a palindrome: True
'A man, a plan, a canal: Panama' is a palindrome: True
'hello' is a palindrome: False
'Racecar' is a palindrome: True
'No lemon, no melon' is a palindrome: True
```

## Code Explanation

This code defines a Python function `is_palindrome` that determines if a given string is a palindrome. It first cleans the input string by converting it to lowercase and removing all non-alphanumeric characters using regular expressions. Then, it checks if the cleaned string is equal to its reversed counterpart. If they are the same, the function returns True, indicating it's a palindrome; otherwise, it returns False.

## Copilot

```
❸ palindrome.py > ...
1  #Generate a palindrome-checking function
2  def is_palindrome(s):
3      # Remove spaces and convert to lowercase for uniformity
4      cleaned_s = ''.join(s.split()).lower()
5      # Check if the cleaned string is equal to its reverse
6      return cleaned_s == cleaned_s[::-1]
7  # Example usage
8  test_string = "A man a plan a canal Panama"
9  print(f'{test_string}' is a palindrome: {is_palindrome(test_string)})
```

## OUTPUT:

```
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting> & C:/Users/Apple/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/Apple/OneDrive/Desktop/Ai Assisting/palindrome.py"
"A man a plan a canal Panama" is a palindrome: True
PS C:\Users\Apple\OneDrive\Desktop\Ai Assisting>
```

## Code Explanation

This function checks if a string is a palindrome by first removing spaces and converting all characters to lowercase for consistency. It then compares the cleaned string with its reversed version (`cleaned_s[::-1]`). If both are equal, the function returns True, meaning the string is a palindrome. For example, "A man a plan a canal Panama" evaluates to True because it reads the same backward and forward.

## Key Differences Between the Two Codes

Aspect	First Code	Second Code
Library Used	No external library	Uses <code>re</code> (regular expressions)
Cleaning Method	Removes only spaces ( <code>''.join(s.split())</code> )	Removes all non-alphanumeric characters ( <code>re.sub(r'[^\w\W]', '', text)</code> )
Case Handling	Converts to lowercase	Converts to lowercase
Scope of Cleaning	Only ignores spaces	Ignores spaces, punctuation, and symbols (e.g., commas, colons, etc.)
Flexibility	Works for simple phrases without punctuation	Works for complex phrases with punctuation and mixed characters
Examples Tested	<code>"A man a plan a canal Panama"</code>	<code>"A man, a plan, a canal: Panama"</code> , <code>"No lemon, no melon"</code> , etc.

## 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

**Gemini Explanation:** This code defines a Python function `is_palindrome` that determines if a given string is a palindrome. It first cleans the input string by converting it to lowercase and removing all non-alphanumeric characters using regular expressions. Then, it checks if the cleaned string is equal to its reversed counterpart. If they are the same, the function returns True, indicating it's a palindrome; otherwise, it returns False

**Copilot Explanation:**

This function checks if a string is a palindrome by first removing spaces and converting all characters to lowercase for consistency. It then compares the cleaned string with its reversed version (`cleaned_s[::-1]`). If both are equal, the function returns `True`, meaning the string is a palindrome. For example, "A man a plan a canal Panama" evaluates to `True` because it reads the same backward and forward