

AI ASSISTED CODING

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

T . ANANYA

2303A51128

BATCH – 03

13 – 01 – 2026

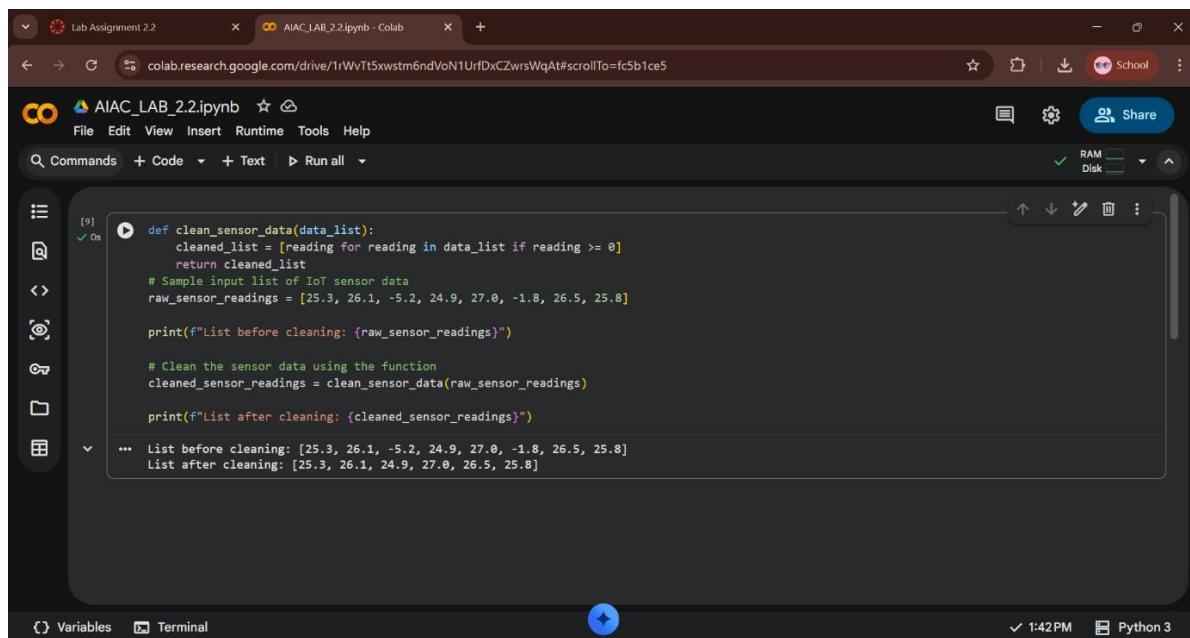
ASSIGNMENT – 2.2

Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI.

Task 1: Cleaning Sensor Data

Prompt: I am working with IoT sensor data where negative values are invalid. Generate a Python function that removes all negative numbers from a given list. Also print the list before cleaning and after cleaning. Provide a sample input and output.

Code:



The screenshot shows a Google Colab notebook titled "AIAC_LAB_2.2.ipynb". The code cell contains the following Python script:

```
[9]  def clean_sensor_data(data_list):
        cleaned_list = [reading for reading in data_list if reading >= 0]
        return cleaned_list
    # Sample input list of IoT sensor data
    raw_sensor_readings = [25.3, 26.1, -5.2, 24.9, 27.0, -1.8, 26.5, 25.8]

    print(f"List before cleaning: {raw_sensor_readings}")

    # Clean the sensor data using the function
    cleaned_sensor_readings = clean_sensor_data(raw_sensor_readings)

    print(f"List after cleaning: {cleaned_sensor_readings}")
...
List before cleaning: [25.3, 26.1, -5.2, 24.9, 27.0, -1.8, 26.5, 25.8]
List after cleaning: [25.3, 26.1, 24.9, 27.0, 26.5, 25.8]
```

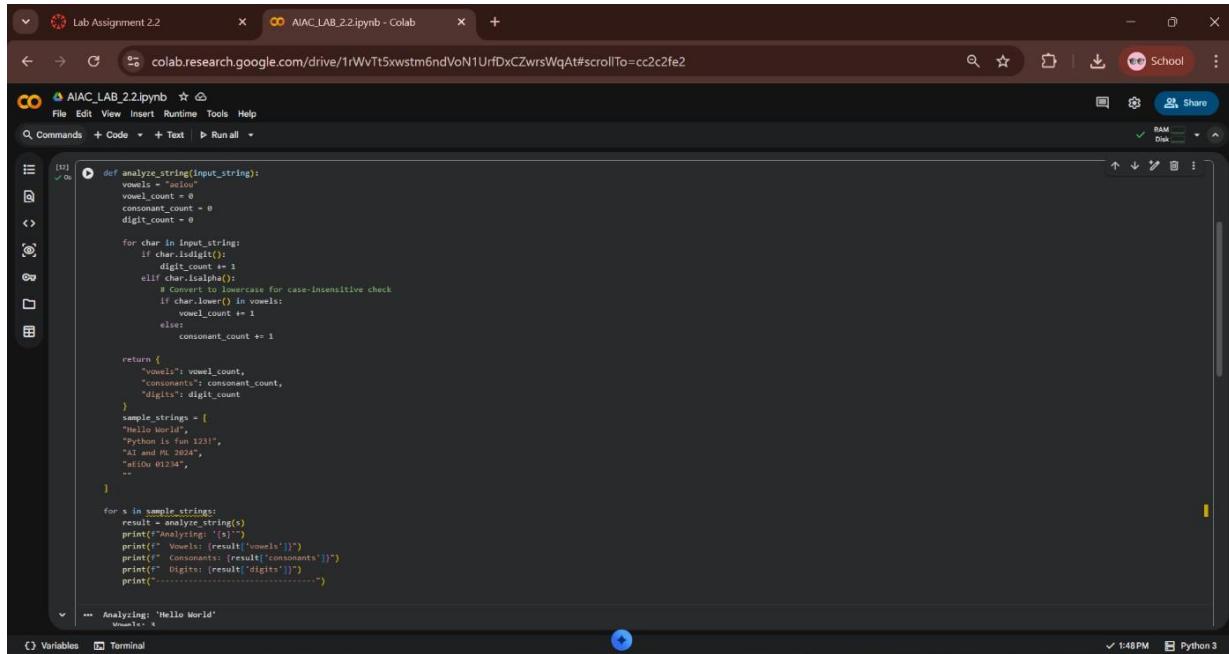
The output of the code is displayed below the code cell, showing the original list with negative values and the cleaned list without them.

Task 2: String Character Analysis.

Prompt: Create a Python function that analyzes a given string and counts the number of vowels, consonants, and digits present in it.

Show the function, explain briefly how it works, and provide sample inputs with corresponding outputs.

Code:



The screenshot shows a Google Colab notebook titled "AIAC_LAB_2.2.ipynb". The code defines a function `analyze_string` that takes an input string and returns a dictionary with counts of vowels, consonants, and digits. It uses a for loop to iterate through each character, checking if it's a digit, alphabetic, or vowel. The output shows the function being run on a sample string "Hello World" and printing the results.

```
def analyze_string(input_string):
    vowels = {"aeiou"}
    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in input_string:
        if char.isdigit():
            digit_count += 1
        elif char.isalpha():
            # Convert to lowercase for case-insensitive check
            if char.lower() in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

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

sample_strings = [
    "Hello World",
    "Python3.8.12345",
    "cat and mi 12345",
    "æilio. 01234",
    ""
]

for s in sample_strings:
    result = analyze_string(s)
    print(f"Analyzing: '{s}'")
    print(f" Vowels: {result['vowels']}")
    print(f" Consonants: {result['consonants']}")
    print(f" Digits: {result['digits']}")
    print("-----")
```

Working Function:

How the Function Works:

- Initialization:** It defines a set of `vowels` (lowercase) and initializes `vowel_count`, `consonant_count`, and `digit_count` to zero.
- Iteration:** It iterates through each character in the `input_string`.
- Digit Check:** For each character, it first checks if it's a digit using `char.isdigit()`. If true, `digit_count` is incremented.
- Alphabetic Check:** If the character is not a digit, it then checks if it's an alphabetic character using `char.isalpha()`.
- Vowel/Consonant Check:** If it's an alphabet, the character is converted to lowercase (`char.lower()`) to perform a case-insensitive check. If the lowercase character is in the `vowels` set, `vowel_count` is incremented; otherwise, `consonant_count` is incremented.
- Return Value:** Finally, it returns a dictionary containing the total counts for vowels, consonants, and digits.

Task 3: Palindrome Check.

Prompt (Gemini Colab): Write a Python function to check whether a given string is a palindrome. Keep the code simple and easy to understand.

Code:

```

▶ def is_palindrome(s):
    cleaned_s = ''.join(char.lower() for char in s if char.isalnum())
    return cleaned_s == cleaned_s[::-1]
sample_strings = [
    "madam",
    "A man, a plan, a canal: Panama",
    "racecar",
    "hello",
    "Was it a car or a cat I saw?",
    "No lemon, no melon",
    "12321",
    "Python"
]

for s in sample_strings:
    result = is_palindrome(s)
    print(f"'{s}' is a palindrome: {result}")

```

Prompt (Copilot): Generate a clean and optimized Python function that checks if a string is a palindrome and returns True or False.

Code:

```

Task_3.py > ...
1  #Generate a clean and optimized Python function that checks if a string is a palindrome and returns True
2  def is_palindrome(s):
3      cleaned_str = ''.join(char.lower() for char in s if char.isalnum())
4      return cleaned_str == cleaned_str[::-1]
5  # Example usage
6  user_input = input("Enter a string: ")
7  print("Is the string a palindrome?", is_palindrome(user_input))

```

Comparison: (Gemini vs GitHub Copilot)

Feature	Gemini	GitHub Copilot
Clarity	Very clear and beginner-friendly. Uses simple logic and explains steps well.	Clear logic but provides little explanation; assumes basic coding knowledge.
Structure	Step-by-step approach focusing on understanding before optimization.	Compact and efficient structure following professional coding standards.

Feature	Gemini	GitHub Copilot
Readability	Highly readable for students and beginners.	Readable for experienced programmers but slightly complex for beginners.
Explanation Quality	Provides detailed and easy-to-understand explanations.	Minimal explanation; focuses mainly on code generation.
Overall Strength	Best for learning and concept understanding.	Best for real-world coding and optimized solutions.

Task 4: Code Explanation Using AI.

Prompt: Explain the following Python function line by line in simple language so that a beginner can understand it clearly.(**prime check OR palindrome function**)

Line	Explanation
def is_palindrome(word):	This line creates a function named <code>is_palindrome</code> that takes one input called <code>word</code> .
word = word.lower()	Converts the word to lowercase so that capital and small letters are treated the same.
reversed_word = word[::-1]	Reverses the string using slicing. Example: "madam" becomes "madam", "hello" becomes "olleh".
if word == reversed_word:	Checks whether the original word and the reversed word are the same.
return True	If they are the same, the function returns True meaning it is a palindrome .
else:	Runs when the condition is false.

Line	Explanation
return False	Returns False meaning it is not a palindrome

This explanation helped me understand how strings can be reversed and compared in Python. The step-by-step logic made it easy to learn how palindrome checking works.

THANK YOU !!