

School of Computer Science and Artificial Intelligence**Lab Assignment # 2.2**

Program : B. Tech (CSE)
Specialization :-
Course Title : AI Assisted coding
Course Code : 23CS002PC304
Semester : II
Academic Session : 2025-2026
Name of Student : I.Sathwik
Enrollment No. : 2403A51L03
Batch No. : 51
Date : 13-01-2026

Submission Starts here**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.

```
def clean_sensor_data(sensor_values):  
    cleaned_data = []  
    for value in sensor_values:  
        if value >= 0:  
            cleaned_data.append(value)  
    return cleaned_data
```

```
# Original sensor data (with invalid negative values)  
sensor_data = [25, -3, 18, -7, 0, 42, -1]  
  
print("Before Cleaning:", sensor_data)  
  
# Clean the data  
cleaned_data = clean_sensor_data(sensor_data)  
  
print("After Cleaning:", cleaned_data)  
  
Before Cleaning: [25, -3, 18, -7, 0, 42, -1]  
After Cleaning: [25, 18, 0, 42]
```

Line-by-Line Explanation

- 1** `def clean_sensor_data(sensor_values):`
 - Defines a function named **clean_sensor_data**
 - `sensor_values` is the input list containing sensor readings (may include negative values)
- 2** `cleaned_data = []`
 - Creates an **empty list**
 - This list will store only **valid (non-negative)** sensor values
- 3** `for value in sensor_values:`

- Loops through **each value** in the input sensor list
- Processes one sensor reading at a time

4) if value >= 0:

- Checks whether the sensor value is **greater than or equal to zero**
- This condition ensures that **negative values are ignored**

5) cleaned_data.append (value)

- Adds the valid sensor value to the cleaned_data list

6) return cleaned_data

- Returns the final list containing **only valid sensor readings**

► Example Execution Explanation

```
sensor_data = [25, -3, 18, -7, 0, 42, -1]
```

- Input list contains both valid and invalid sensor values

```
cleaned_data = clean_sensor_data(sensor_data)
```

- Function removes all negative values

✓ Output:

```
[25, 18, 0, 42]
```

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.

```
def analyze_string(text):
    vowels = 0
    consonants = 0
    digits = 0

    for ch in text:
        if ch.isdigit():
            digits += 1
        elif ch.isalpha():
            if ch.lower() in "aeiou":
                vowels += 1
            else:
                consonants += 1

    return vowels, consonants, digits

text = "Hello123"
result = analyze_string(text)

print("String:", text)
print("Vowels:", result[0])
print("Consonants:", result[1])
print("Digits:", result[2])

... String: Hello123
Vowels: 2
Consonants: 3
Digits: 3
```

Explanation:

- 1 def analyze_string(text):
 - Defines a function named analyze_string
 - text is the input string that will be analyzed
- 2 vowels = 0, consonants = 0, digits = 0
 - Initializes three counters
 - These variables store the count of vowels, consonants, and digits
- 3 for ch in text:
 - Loops through each character in the input string
 - Processes one character at a time
- 4 if ch.isdigit():
 - Checks if the character is a number (0–9)
 - If true, the digit counter is increased
- 5 digits += 1:
 - Increments the digit count by 1
- 6 elif ch.isalpha():
 - Checks if the character is an alphabet
 - Ignores spaces and special characters
- 7 if ch.lower() in "aeiou":
 - Converts the character to lowercase
 - Checks if it is a vowel (a, e, i, o, u)
- 8 vowels += 1
 - Increments the vowel count if the condition is true
- 9 else:
 - If the alphabet is **not a vowel**, it must be a consonant
- consonants += 1**
 - Increments the consonant count
- 10 return vowels, consonants, digits
 - Returns all three counts as a tuple

#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.

```
def is_palindrome_copilot(text):
    text = text.lower()
    left = 0
    right = len(text) - 1

    while left < right:
        if text[left] != text[right]:
            return False
        left += 1
        right -= 1

    return True

word = "Madam"

print(is_palindrome_copilot(word))
print(is_palindrome_copilot(word))

*** True
*** True
```

Explanation:

- 1) `def is_palindrome_gemini(text):`
 - Defines a function to check whether a string is a palindrome
 - `text` is the input string
- 2) `text = text.lower()`
 - Converts all characters to lowercase
 - This avoids case mismatch (e.g., Madam vs madam)
- 3) `text[::-1]`
 - Reverses the string using Python slicing
 - `[::-1]` means read the string from end to start
- 4) `return text == text[::-1]`
 - Compares the original string with its reversed version
 - Returns True if both are the same, otherwise False
- `def is_palindrome_copilot(text):`
 - Defines a function to check if a string is a palindrome
- 2) `text = text.lower()`
 - Converts the string to lowercase for case-insensitive comparison
- 3) `left = 0`
 - Points to the **first character** of the string
- 4) `right = len(text) - 1`
 - Points to the **last character** of the string
- 5) `while left < right:`
 - Loop runs until both pointers meet in the middle
- 6) `if text[left] != text[right]:`
 - Compares characters from both ends
 - If they are not equal, the string is **not a palindrome**
- 7) `return False`
 - Immediately stops and returns False if mismatch is found
- 8) `left += 1`
 - Moves the left pointer forward
- 9) `right -= 1`
 - Moves the right pointer backward
- `return True`
 - If all characters match, the string is a palindrome

#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.

```
def is_palindrome(text):
    text = text.lower()
    left = 0
    right = len(text) - 1

    while left < right:
        if text[left] != text[right]:
            return False
        left += 1
        right -= 1

    return True
text=input()
print(text)
print(is_palindrome(text))

... nani
    nani
    False
```

Explanation:**1) def is_palindrome(text):**

- Defines a function named is_palindrome
- Takes a string text as input

2) text = text.lower()

- Converts all characters in the string to lowercase
- Ensures case-insensitive comparison (e.g., Madam = madam)

3) left = 0

- Initializes a pointer to the **first character** of the string

4) right = len(text) - 1

- Initializes a pointer to the **last character** of the string

5) while left < right:

- Starts a loop that runs until both pointers meet in the middle

6) if text[left] != text[right]:

- Compares characters at the left and right pointers
- If they are not equal, the string is **not a palindrome**

7) return False

- Immediately exits the function if a mismatch is found

8) left += 1

- Moves the left pointer one step forward

`right -= 1`

- Moves the right pointer one step backward

return True

- If all character pairs match, the string is a palindrome