

AI ASSISTED CODING

Name: P.Swaran Raj

Roll.No: 2303A52247

Batch: 37

Assignment-2

Task 1: Cleaning Sensor Data

Prompt used:

"#write a python code to separate all the negative numbers from the user given list of numbers and show new list which contains no negative number start the code from scrap and if the list is empty give a statement or text saying the data is clean if the no say invalid data found"

The screenshot shows the 2.2 Ai Assist application interface. On the left, there is a code editor window titled 'filter_non_negative_with_status.py' containing the following Python code:

```
def filter_non_negative_with_status(numbers_list):
    """
    Filters a list of numbers to return a new list containing only non-negative numbers.
    Provides status messages based on the input list's content.

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

    Returns:
        list: A new list containing only numbers >= 0, or an empty list if input was empty.
    """
    if not numbers_list:
        print("The provided list is empty. Data is clean.")
        return []
    non_negative_numbers = [num for num in numbers_list if num >= 0]
    if len(non_negative_numbers) == len(numbers_list):
        print("Data is clean. No negative numbers found.")
    else:
        print("Invalid data found. Negative numbers were removed.")
    return non_negative_numbers

# Example usage with various lists:
print("\n--- Example 1: List with mixed numbers ---")
list_a = [10, -3, 0, 7, -15, 22, -1, 5]
filtered_a = filter_non_negative_with_status(list_a)
print(f"Original list: {list_a}")
print(f"List containing no negative numbers: {filtered_a}")

print("\n--- Example 2: List with only negative numbers ---")
list_b = [-10, -20, 5, 0, -30]
filtered_b = filter_non_negative_with_status(list_b)
print(f"Original list: {list_b}")
print(f"List containing no negative numbers: {filtered_b}")
```

Below the code editor is a terminal window showing the output of the code execution.

On the right, there is a panel titled 'Gemini' with the following content:

- A prompt from the user asking for a Python code to filter negative numbers.
- A response from Gemini providing the code above.
- A summary of the results, stating the code executed successfully and provided clear status messages.
- A section titled 'Example 1 (Mixed Numbers)' with three sub-points: 'Compare the implemented functions for filtering negative numbers', 'Explain the logic in the filter_non_negative_with_status function', and 'Generate a new list containing only the negative numbers'.
- A 'What can I help you build?' input field.
- System status information at the bottom: 'Gemini 2.5 Flash', '10:20', and 'Python 3'.

The screenshot shows the 2.2 Ai Assist application interface. On the left, there is a code editor window with Python code demonstrating a function to filter non-negative numbers from a list. The code includes comments for four examples: an empty list, a list with only positive numbers, a list with mixed numbers, and a list with only negative numbers. The output shows the original lists and the filtered lists. On the right, there is a panel titled "Gemini" containing AI-generated feedback. The feedback includes a problem statement asking to separate negative numbers from a list, a solution snippet provided by Gemini, and a summary of the results. Below the summary, there are several callout boxes with suggestions like "Compare the implemented functions for filtering negative numbers" and "Explain the logic in the 'filter_non_negative_with_status' function". At the bottom right, there is a "Gemini 2.5 Flash" button.

```

print("\n--- Example 3: An empty list ---")
empty_list_example = []
filtered_empty = filter_non_negative_with_status(empty_list_example)
print("Original list: {empty_list_example}")
print("List containing no negative numbers: {filtered_empty}")

print("\n--- Example 4: List with only positive numbers ---")
list_c = [1, 2, 3, 4, 5]
filtered_c = filter_non_negative_with_status(list_c)
print("Original list: {list_c}")
print("List containing no negative numbers: {filtered_c}")

... -- Example 1: List with mixed numbers --
Invalid data found. Negative numbers were removed.
Original list: [10, -3, 0, 7, -15, 22, -1, 5]
List containing no negative numbers: [10, 0, 7, 22, 5]

... -- Example 2: List with only negative numbers --
Invalid data found. Negative numbers were removed.
Original list: [-10, -20, 5, 0, -30]
List containing no negative numbers: [5, 0]

... -- Example 3: An empty list --
The provided list is empty. Data is clean.
Original list: []
List containing no negative numbers: []

... -- Example 4: List with only positive numbers --
Data is clean. No negative numbers found.
Original list: [1, 2, 3, 4, 5]
List containing no negative numbers: [1, 2, 3, 4, 5]

```

Explanation:

Gemini generated a function that filters negative values using conditional checks.

The function also displays messages indicating whether the data is clean or invalid.

Output:

Before: [10, -3, 0, 7, -15, 22, -1, 5]

After: [10, 0, 7, 22, 5]

Observation:

Gemini accurately handled invalid sensor data.

The solution was simple and suitable for IoT data cleaning.

Task 2: String Character Analysis

Prompt used:

“Write a Python function to count the number of vowels, consonants, and digits in a string. Use a sample input string and display the counts as output.”

The screenshot shows the AI Assist interface with a code editor and a Gemini AI panel.

Code Editor:

```
[1]: def count_chars(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 (vowel_count, consonant_count, digit_count).
    """
    vowels = "aeiou"
    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in input_string:
        # Convert to lowercase for easier comparison
        char_lower = char.lower()

        if char_lower.isalpha(): # Check if it's an alphabet
            if char_lower in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
        elif char_lower.isdigit(): # Check if it's a digit
            digit_count += 1

    return vowel_count, consonant_count, digit_count

# Sample input string
sample_string = "Hello World 123! Python is Fun"

# Call the function
v, c, d = count_chars(sample_string)

# Display the counts
```

Gemini Panel:

Prompt: Write a Python function to count the number of vowels, consonants, and digits in a string. Use a sample input string and display the counts as output.

Response: Certainly! Here's a Python function that counts the number of vowels, consonants, and digits in a given string. I've included a sample string to demonstrate its functionality.

The code has executed successfully, and it correctly counted the vowels, consonants, and digits for the sample strings.

For the string "Hello World 123! Python is Fun":

- Number of Vowels: 6
- Number of Consonants: 15
- Number of Digits: 3

For the string "Colab rocks! 2024":

- Number of Vowels: 3
- Number of Consonants: 7
- Number of Digits: 4

Explain how the count_chars function works
Try the count_chars function with a new string like "AI in Google Colab 2024!"
Add functionality to count punctuation marks to the count_chars function

What can I help you build?

Gemini 2.5 Flash ▶

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

The screenshot shows the AI Assist interface with a code editor and a Gemini AI panel.

Code Editor:

```
[1]: # Call the function
v, c, d = count_chars(sample_string)

# Display the counts
print(f"Original String: '{sample_string}'")
print(f"Number of Vowels: {v}")
print(f"Number of Consonants: {c}")
print(f"Number of Digits: {d}")

# Another example
another_string = "colab rocks! 2024"
v2, c2, d2 = count_chars(another_string)
print(f"\nOriginal String: '{another_string}'")
print(f"Number of Vowels: {v2}")
print(f"Number of Consonants: {c2}")
print(f"Number of Digits: {d2}")

... Original String: 'Hello World 123! Python is Fun'
Number of Vowels: 6
Number of Consonants: 15
Number of Digits: 3

Original String: 'colab rocks! 2024'
Number of Vowels: 3
Number of Consonants: 7
Number of Digits: 4
```

Gemini Panel:

Prompt: Write a Python function to count the number of vowels, consonants, and digits in a string. Use a sample input string and display the counts as output.

Response: Certainly! Here's a Python function that counts the number of vowels, consonants, and digits in a given string. I've included a sample string to demonstrate its functionality.

The code has executed successfully, and it correctly counted the vowels, consonants, and digits for the sample strings.

For the string "Hello World 123! Python is Fun":

- Number of Vowels: 6
- Number of Consonants: 15
- Number of Digits: 3

For the string "Colab rocks! 2024":

- Number of Vowels: 3
- Number of Consonants: 7
- Number of Digits: 4

Explain how the count_chars function works
Try the count_chars function with a new string like "AI in Google Colab 2024!"
Add functionality to count punctuation marks to the count_chars function

What can I help you build?

Gemini 2.5 Flash ▶

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

Explanation:

Gemini generated a function that checks each character using loops and conditions. The function correctly separates vowels, consonants, and digits.

Output:

Input: Hello World 123! Python is Fun

Vowels: 6, Consonants: 15, Digits: 3

Observation:

Gemini produced readable and well-structured code. The logic was easy to understand and accurate.

Task 3: Palindrome Check – Tool Comparison

Prompt used:

"#write code for palindrome checking function and take dynamic input"

The screenshot shows a Colab notebook interface with two main sections: "task 3" and "Gemini".

task 3:

```
def is_palindrome(text):
    """
    Checks if a given string is a palindrome.
    A palindrome is a word, phrase, or sequence that reads the same backward as forward,
    ignoring spaces, punctuation, and capitalization.

    Args:
        text (str): The input string.

    Returns:
        bool: True if the string is a palindrome, False otherwise.
    """
    # Normalize the string: convert to lowercase and remove non-alphanumeric characters
    cleaned_text = ''.join(char.lower() for char in text if char.isalnum())

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

# Take dynamic input from the user
user_input_string = input("Enter a string to check if it's a palindrome: ")

# Call the function with the user's input and display the result
if is_palindrome(user_input_string):
    print(f"'{user_input_string}' IS a palindrome.")
else:
    print(f"'{user_input_string}' IS NOT a palindrome.")

... Enter a string to check if it's a palindrome: noooon
'noooon' IS a palindrome.
```

Gemini:

P write code for palindrome checking function and take dynamic input

Sure, here's the Python code for a palindrome checking function that takes dynamic input from you. You'll be prompted to enter a string when the code runs.

The palindrome checking function has executed successfully! As expected, for the input 'nooy', the output states that it IS NOT a palindrome.

What can I help you build?

Gemini 2.5 Flash

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

The screenshot shows a code editor with a Python file named 'ai assit 2.py'. The code defines a function 'is_palindrome' that checks if a string is a palindrome by removing spaces and converting it to lowercase, then comparing it to its reverse. It includes example usage and prints 's' as a palindrome or not. Below the editor is a terminal window showing the script's execution and output for the input 'sas'.

```
ai assit 2.py > ...
1  #write a python function to check whether a string is a palindrome
2  def is_palindrome(s):
3      # Remove spaces and convert to lowercase
4      s = s.replace(" ", "").lower()
5      # Check if the string is equal to its reverse
6      return s == s[::-1]
7  # Example usage
8  string = input("Enter a string: ")
9  if is_palindrome(string):
10     print(f'{string} is a palindrome.')
11 else:
12     print(f'{string} is not a palindrome.')

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding> & C:/Users/pulla/AppData/Local/Programs/Python/Python313/python.exe "ai assit 2.py"
● Enter a string: sas
"sas" is a palindrome.
○ PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding>
```

Explanation:

GitHub Copilot generated a short and efficient palindrome function using string reversal.

Gemini generated a more descriptive function with normalization and detailed comments.

Output:

Input: sas → Output: *Palindrome* (VS Code – Copilot)

Input: noooon → Output: *Palindrome* (Colab – Gemini)

Observation:

Copilot is faster and produces concise code suitable for quick development.

Gemini provides better explanations and readability, which is helpful for learning and understanding logic.

Task 4: Code Explanation Using AI

Prompt used: "#Explain this palindrome checking function line by line"

P explain this palindrome checking function line by line

✍ def is_palindrome(text):

◆ Certainly! Let's break down the `is_palindrome` function and the surrounding code in cell `50cac168` line by line:

def is_palindrome(text):

- This line defines a Python function named `is_palindrome` that accepts one argument, `text`, which is expected to be a string.

....

Checks if a given string is a palindrome.

[Test the 'count_chars' function with a new string](#)

[Explain the 'count_chars' function line by line](#)

[Refactor the 'is_palindrome' function for efficiency](#)

A palindrome is a word, phrase, or sequence that reads the same backward as forward, ignoring spaces, punctuation, and capitalization.

Args:

text (str): The input string.

Returns:

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

....

- This is a docstring, which provides a brief explanation of what the `is_palindrome` function does, its arguments (`Args`), and what it returns (`Returns`). It's good practice for documenting your code.

What it returns (`returns`). It's good practice for documenting code.

```
cleaned_text = ''.join(char.lower() for char in text)
```

- This is the core of the string normalization. Let's break it down further:

- `for char in text`: This iterates through each character in the input `text` string.
- `if char.isalnum()`: This condition checks if the current character `char` is alphanumeric (i.e., a letter or a number). Non-alphanumeric characters (like spaces, punctuation, etc.) are excluded.
- `char.lower()`: For each alphanumeric character, it

converts it to its lowercase equivalent. This ensures that the palindrome check is case-insensitive (e.g., 'Racecar' and 'racecar' are treated the same).

- `''.join(...)`: This joins all the processed characters together into a new string, `cleaned_text`.
- **In summary:** This line creates a new string `cleaned_text` containing only the lowercase alphanumeric characters from the original input `text`.

```
return cleaned_text == cleaned_text[::-1]
```

- This line performs the actual palindrome check and returns the result:
 - `cleaned_text[::-1]`: This is a Python slice operation

- string to check if it's a palindrome: " to the user in the console.
- It then waits for the user to type something and press Enter.
 - The text entered by the user is stored in the variable `user_input_string`.

```
# Call the function with the user's input and display the result
if is_palindrome(user_input_string):
```

- This line calls our `is_palindrome` function, passing the `user_input_string` as its argument.
- The `if` statement then checks the boolean value returned by `is_palindrome`.

```
    print(f'{user_input_string}' IS a palindrome.)
```

```
print(f'{user_input_string}' IS a palindrome.)
else:
```

- If `is_palindrome` returned `True` (meaning it's a palindrome), this line is executed, printing a message indicating that the input string is a palindrome.

```
    print(f'{user_input_string}' IS NOT a palindrome.)
```

- If `is_palindrome` returned `False` (meaning it's not a palindrome), this `else` block is executed, printing a message indicating that the input string is not a palindrome.

This structure effectively defines a reusable function and then demonstrates its usage with user interaction.

Explanation:

Gemini explained each line of the palindrome function in simple language. The explanation helped in understanding string normalization and comparison logic.

Output:

The function correctly identifies whether the input string is a palindrome.

Observation:

Gemini is very useful for understanding unfamiliar code. It helps students learn logic clearly through step-by-step explanations.

Conclusion:

AI tools like Gemini and Copilot improved coding speed, understanding, and code quality through generation and explanation.