

AI ASSISTANT CODING

ASSIGNMENT-2.2

Name: N. Paul Benjamin

Ht.no: 2303A51116

Batch: 03

Task 1: Cleaning Sensor Data

❖ Scenario:

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

❖ Task:

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

❖ Expected Output:

➤ Before/after list

➤ Screenshot of Collab execution

Prompt: generate a function that filters out all negative

numbers from a list

Code:

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

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

    Returns:
        list: 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, -4, 0, 5, -6]
filtered_list = filter_negative_numbers(my_list)
print(f"Original list: {my_list}")
print(f"Filtered list (non-negative numbers): {filtered_list}")

... Original list: [1, -2, 3, -4, 0, 5, -6]
    Filtered list (non-negative numbers): [1, 3, 0, 5]
```

Output:

```
... Original list: [1, -2, 3, -4, 0, 5, -6]
    Filtered list (non-negative numbers): [1, 3, 0, 5]
```

Explanation:

This Python code defines a function `filter_negative_numbers` that takes a list of numbers as input. It then iterates through the input list and returns a new list containing only the numbers that are greater than or equal to zero, effectively filtering out any negative values.

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.

Code:

```
def count_chars(input_string):
    """
    Counts 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 = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0
    digit_count = 0

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

    return vowel_count, consonant_count, digit_count

# Example usage:
my_string = "Hello World 123!"
v, c, d = count_chars(my_string)
print(f"Original string: '{my_string}'")
print(f"Number of vowels: {v}")
print(f"Number of consonants: {c}")
print(f"Number of digits: {d}")
```

Output:

```
... Original string: 'Hello World 123!'
Number of vowels: 3
Number of consonants: 7
Number of digits: 3
```

Explanation:

This code defines a function `count_chars` that takes a string as input. It initializes counters for vowels, consonants, and digits. It then iterates through each character in the string, checking if it's an alphabet character (and then if it's a vowel or a consonant) or a digit, incrementing the appropriate counter. Finally, it returns the total counts of vowels, consonants, and digits.

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 :

Gemini prompt: Generate a palindrome-checking function

Code :

```
▶ 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'[^\w\s]', '', text).lower()

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

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

Output :

```
... '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
```

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 prompt: Generate a palindrome-checking function

Code :

```
# 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>
```

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.

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'[^a-zA-Z0-9]', ' ', 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	"A man a plan a canal Panama"	"A man, a plan, a canal: Panama", "No lemon, no melon", 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**

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.

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.

