Assignment-2

Name: Alli Harika

Ht.no: 2303A510I7

Batch. No: 21

Task-1: Cleaning Sensor Data

Prompt used:

write a python code using functions to clean the sensor data ,valid data of positive numbers and zeros, invalid data of negative numbers. the list should be taken from the users. And also print list before cleaning and after cleaning, also the code should handle edge cases.

Code:

```python
def get_sensor_data():
    """
    Prompts the user to enter sensor data as a comma-separated list of numbers.
    Handles non-numeric input by asking the user to re-enter valid data.
    Returns a list of numbers.
    """
    while True:
        user_input = input("Please enter sensor data (comma-separated numbers, e.g., 10, -5, 0, 25.5): ")
        if not user_input.strip():
            print("Input cannot be empty. Please try again.")
            continue

        try:
            # Split the input string by commas, remove whitespace, and convert to float
            data_str_list = [item.strip() for item in user_input.split(',')]

            # Attempt to convert each item to a float. If any conversion fails, catch the ValueError.
            sensor_data = []
            for item in data_str_list:
                if item:
                    sensor_data.append(float(item))

            return sensor_data
        except ValueError:
            print("Invalid input. Please ensure all entries are numbers and separated by commas. Try again.")
        except Exception as e:
            print(f"An unexpected error occurred: {e}. Please try again.")

def clean_sensor_data(data_list):
    """
    Cleans the sensor data list by removing negative numbers.
    Valid data includes positive numbers and zeros.
    Returns a new list with only valid data.
    """
    if not data_list:
        print("\nEdge case: The provided data list is empty. No cleaning performed.")
        return []

    cleaned_data = [num for num in data_list if num >= 0]
    return cleaned_data

def display_data(message, data_list):
    """
    Prints a message followed by the given data list.
    """
    print(f"\n{message}: {data_list}")

# Main program execution
if __name__ == "__main__":
    # Get sensor data from the user
    raw_data = get_sensor_data()

    # Display the data before cleaning
    display_data("Original Sensor Data", raw_data)

    # Clean the sensor data
    cleaned_data = clean_sensor_data(raw_data)

    # Display the data after cleaning
    display_data("Cleaned Sensor Data (positive numbers and zeros only)", cleaned_data)
```

Output:

```
Please enter sensor data (comma-separated numbers, e.g., 10, -5, 0, 25.5): 25, -3,18,0,42,-7

Original Sensor Data: [25.0, -3.0, 18.0, 0.0, 42.0, -7.0]

Cleaned Sensor Data (positive numbers and zeros only): [25.0, 18.0, 0.0, 42.0]
```

JUSTIFICATION:

The program collects sensor readings from the user with proper input validation to ensure only numeric values are accepted. The cleaning function filters out all negative values, keeping only valid sensor data (zero and positive numbers). Displaying the data before and after cleaning clearly shows how invalid readings are removed, making the data reliable for further processing.

Task-2: String Character Analysis

Prompt used:
write a python code using functions that accepts string and counts vowels, consonants and digits in a string. the string is taken from the user , the code should handle edge cases also.

Code:

```python
def count_chars_in_string(input_string):
    """
    Counts the number of vowels, consonants, and digits in a given string.
    Ignores non-alphanumeric characters and is case-insensitive for letters.
    Returns a tuple (vowel_count, consonant_count, digit_count).
    """
    vowels = "aeiou"
    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    # Handle edge case: empty string
    if not input_string:
        print("\nEdge case: The input string is empty.")
        return 0, 0, 0

    for char in input_string:
        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
        # Other characters (spaces, symbols) are ignored
    return vowel_count, consonant_count, digit_count

def get_user_string():
    """
    Prompts the user to enter a string and returns it.
    """
    user_input = input("Please enter a string: ")
    return user_input

def display_counts(input_string, v_count, c_count, d_count):
    """
    Displays the original string and the counts of vowels, consonants, and digits.
    """
    print(f"\nAnalyzing string: '{input_string}'")
    print(f"Number of vowels: {v_count}")
    print(f"Number of consonants: {c_count}")
    print(f"Number of digits: {d_count}")
    print(f"Total alphanumeric characters: {v_count + c_count + d_count}")

# Main program execution
if __name__ == "__main__":
    # Get string from the user
    user_str = get_user_string()

    # Count characters
    vowels, consonants, digits = count_chars_in_string(user_str)

    # Display results
    display_counts(user_str, vowels, consonants, digits)
```

Output:

```
Please enter a string: hello123@#$world

Analyzing string: 'hello123@#$world'
Number of vowels: 3
Number of consonants: 7
Number of digits: 3
Total alphanumeric characters: 13
```

JUSTIFICATION:

The program converts each character to lowercase before checking it against the vowel list, making the logic case-insensitive. Because of this, both capital and small vowels (A, E, I, O, U and a, e, i, o, u) are counted correctly using a single vowel set. This approach simplifies the code and avoids writing separate conditions for uppercase and lowercase letters.

Task 3: Palindrome Check – Tool Comparison

Gemini code:

```
import re # Import regular expression module for string cleaning

def get_user_string_for_palindrome():
    """
    Prompts the user to enter a string and returns it.
    """
    user_input = input("Please enter a string to check if it's a palindrome: ")
    return user_input

def is_palindrome(input_string):
    """
    Checks if a given string is a palindrome.
    Ignores non-alphanumeric characters and is case-insensitive.
    Returns True if it's a palindrome, False otherwise.
    """
    if not input_string:
        print("\nEdge case: The input string is empty.")
        return False

    # Convert to lowercase and remove non-alphanumeric characters
    cleaned_string = re.sub(r'[^a-zA-Z0-9]', '', input_string).lower()

    # Handle edge case: string becomes empty after cleaning
    if not cleaned_string:
        print("\nEdge case: The input string contains no alphanumeric characters after cleaning.")
        return False

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

def display_palindrome_result(original_string, is_pal):
    """
    Displays whether the original string is a palindrome.
    """
    print(f"\nChecking string: '{original_string}'")
    if is_pal:
        print(f"'{original_string}' IS a palindrome.")
    else:
        print(f"'{original_string}' IS NOT a palindrome.")

# Main program execution
if __name__ == "__main__":
    # Get string from the user
    user_str_palindrome = get_user_string_for_palindrome()

    # Check if it's a palindrome
    is_pal_result = is_palindrome(user_str_palindrome)

    # Display results
    display_palindrome_result(user_str_palindrome, is_pal_result)
```

Output:

```
Please enter a string to check if it's a palindrome: madam

Checking string: 'madam'
'madam' IS a palindrome.
```

Copilot Code:

```python
# write a python code for checking whether a string is a palindrome or not, the string input should be taken from the user.
def is_palindrome(string):
    """
    Check whether a given string is a palindrome.

    Parameters:
    string (str): The string to check.

    Returns:
    bool: True if the string is a palindrome, False otherwise.
    """
    # Normalize the string by removing spaces and converting to lowercase
    normalized_string = ''.join(string.split()).lower()
    # Check if the normalized string is equal to its reverse
    return normalized_string == normalized_string[::-1]
# Get string input from user
user_input = input("Enter a string to check if it's a palindrome: ")
# Check and display whether the string is a palindrome
if is_palindrome(user_input):
    print(f'"{user_input}" is a palindrome.')
else:
    print(f'"{user_input}" is not a palindrome.')
```

Output:

```
PS D:\AI assist> python -u "d:\AI assist\code2.py"
Enter a string to check if it's a palindrome: madam
"madam" is a palindrome.
PS D:\AI assist>
```

Prompt used for both AI's:

write a python code for checking whether a string is a palindrome or not, the string input should be taken from the user.

- **Side-by-Side Code Comparison**

  Gemini-Generated Code

  - Uses **multiple functions** for input, processing, and output.
  - Cleans the string using **regular expressions**.
  - Handles **edge cases** like empty input and strings with only symbols.

- Provides **descriptive messages** during execution.

- **Copilot-Generated Code**
  - Uses a **single function** with concise logic.
  - Cleans the string by removing spaces and converting to lowercase.
  - Focuses only on the **core palindrome check**.
  - Minimal explanations and no edge-case handling.
- **Observations on Clarity and Structure**

Aspect	Gemini	Copilot
Code structure	Modular (multiple functions)	Simple (single function)
Readability	Very clear and descriptive	Compact but less detailed
Edge-case handling	Yes	No
Beginner suitability	High	Medium
Use of advanced concepts	Regular expressions	Basic string operations

My Experience on using both Gemini and Copilot:

Gemini provided clear, well-structured code with detailed explanations, which made the logic easy to understand. Copilot generated concise and efficient code but with minimal explanation. Overall, Gemini was better for learning, while Copilot was useful for quick coding.

Task-4: Code Explanation Using AI

Prompt used:

Explain palindrome checking code in python line by line using simple and clear language for beginners .

Code:

```
def is_palindrome(text):
    cleaned = ''.join(ch.lower() for ch in text if ch.isalnum())
    return cleaned == cleaned[::-1]
```

AI Explanation:

1. Defining the Function

`def is_palindrome(text):`

- **What it does:** This line creates the function and names it `is_palindrome`. It takes one input, which we call `text`.
- **The Goal:** We want this function to give us a "Yes" (True) or "No" (False) answer.

2. Cleaning the Text

`cleaned = ''.join(ch.lower() for ch in text if ch.isalnum())` This line is a "list comprehension." It processes the text to make sure the comparison is fair. It does three things:

- `ch.lower()`: It converts every letter to lowercase. This ensures that "Racecar" and "racecar" are seen as the same thing.
- `if ch.isalnum()`: This is a filter. It only keeps characters that are letters or numbers. It ignores spaces, periods, and commas.
- `''.join(...)`: It takes all those individual characters and "glues" them back together into one string named `cleaned`.

- `if ch.isalnum()`: This is a filter. It only keeps characters that are letters or numbers. It ignores spaces, periods, and commas.
- `''.join(...)`: It takes all those individual characters and "glues" them back together into one string named `cleaned`.

  Example: If you input `"A man, a plan!"`, this line turns it into `"amanaplan"`.

3. Reversing and Comparing

`return cleaned == cleaned[::-1]` This is where the magic happens.

- `cleaned[::-1]`: This is a Python "slice" trick. The `[::-1]` tells Python to start at the end of the string and walk backward to the beginning, effectively **reversing** the string.
- `==`: This checks if the `cleaned` string is identical to its reversed version.
- `return`: This sends the result back to you. If they match, it returns **True**; if not, it returns **False**.

How it works in practice:

Input	Cleaned Version	Reversed Version	Result
"Racecar"	"racecar"	"racecar"	**True**
"Hello"	"hello"	"olleh"	**False**
"12321"	"12321"	"12321"	**True**

Student comments on understanding:

From the AI explanation, I understood how the function first cleans the input string by converting all characters to lowercase and removing spaces and special symbols. This makes the palindrome check accurate and case-insensitive. I also understood how reversing the string using slicing helps compare it with the original cleaned string to decide whether it is a palindrome or not.