

2303A51203
Batch:04
Assignment :2.2

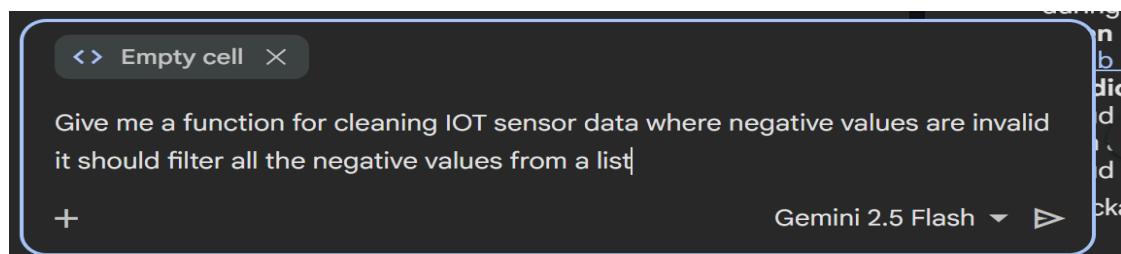
Task 1: Cleaning Sensor Data

- ❖ 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.

- ❖ Expected Output:
- Before/after list
- Screenshot of Colab execution

Prompt : Give me a function for cleaning IOT sensor data where negative values are invalid it should filter all the negative values from a list



Code :

```
def clean_iot_sensor_data(data_list):  
    """
```

Cleans IoT sensor data by filtering out negative (invalid) values.

Args:

 data_list (list): A list of sensor readings, which may include negative values.

Returns:

 list: A new list containing only the valid (non-negative) sensor readings.

 """

```
  cleaned_data = [value for value in data_list if value >= 0]  
  return cleaned_data
```

```
sensor_readings = [23.5, -1.2, 25.1, 0.0, -0.5, 22.9]  
cleaned_readings = clean_iot_sensor_data(sensor_readings)  
print(f"Original readings: {sensor_readings}")
```

```
print(f"Cleaned readings: {cleaned_readings}")
```

Output:

Original readings: [23.5, -1.2, 25.1, 0.0, -0.5, 22.9]

Cleaned readings: [23.5, 25.1, 0.0, 22.9]

Task02: 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 function which analyzes text and counts vowels, consonants, and digits in a string which is provided as an input to the function

Code :

```
def analyze_text(text_string):
```

```
    """
```

Analyzes a given string and counts the number of vowels, consonants, and digits.

Args:

text_string (str): The input string to be analyzed.

Returns:

dict: A dictionary containing the counts of 'vowels', 'consonants', and 'digits'.

```
    """
```

```
vowels = "aeiou"
```

```
vowel_count = 0
```

```
consonant_count = 0
```

```
digit_count = 0
```

```
for char in text_string:
```

```
    char_lower = char.lower()
```

```
    if char_lower.isalpha():
```

```
        if char_lower in vowels:
```

```
            vowel_count += 1
```

```
        else:
```

```
            consonant_count += 1
```

```

        elif char_lower.isdigit():
            digit_count += 1

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

```

Input and Output :

Text: 'Hello World 123!'

Vowels: 3

Consonants: 7

Digits: 3

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 Collab : Generate a palindrome checking function where a string is passed to it to check it is either palindrome or not.

import re

def is_palindrome(text):

"""

Checks if a given string is a palindrome.

Ignores non-alphanumeric characters and is case-insensitive.

Args:

text (str): The input string to check.

Returns:

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

"""

Remove non-alphanumeric characters and convert to lowercase
processed_text = re.sub(r'[^a-zA-Z0-9]', " ", 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"'racecar' is a palindrome: {is_palindrome('racecar')}")"
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"'Was it a car or a cat I saw?' is a palindrome: {is_palindrome('Was it a car or a cat I saw?')}")"

```

Input and Output :

'Madam' is a palindrome: True
'racecar' is a palindrome: True
'A man, a plan, a canal: Panama' is a palindrome: True
'hello' is a palindrome: False
'Was it a car or a cat I saw?' is a palindrome: True

Prompt github copilot : Generate a palindrome checking function where a string is passed to it to check it is either palindrome or not.

Code:

```

def is_palindrome(s):
    """
    Check if a string is a palindrome.

    Args:
        s (str): The string to check.

    Returns:
        bool: True if the string is a palindrome, False otherwise.
    """
    return s == s[::-1]

if __name__ == "__main__":
    user_input = input("Enter a string to check if it's a palindrome: ")
    if is_palindrome(user_input):
        print(f"{user_input} is a palindrome.")
    else:
        print(f"{user_input} is not a palindrome.")

```

Output:

Enter a string to check if it's a palindrome: madam

'madam' is a palindrome.

Enter a string to check if it's a palindrome: adithya

'adithya' is not a palindrome.

Observation on clarity and structure : Here in colab gemini gave a function where only alpha numeric characters are present and others are removed and then it is converted to lower() using function and then used a slicing function to reverse it.

In vscode it is only processing a given string and not removing any non alphanumeric characters it will return true or false based on that.

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

```
def is_palindrome(s):
```

```
    """
```

Check if a string is a palindrome.

Args:

s (str): The string to check.

Returns:

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

```
    """
```

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

```
if __name__ == "__main__":
```

```
    user_input = input("Enter a string to check if it's a palindrome: ")
```

```
    if is_palindrome(user_input):
```

```
        print(f"{user_input} is a palindrome.")
```

```
    else:
```

```
        print(f"{user_input} is not a palindrome.")
```

Review :

Certainly! Let's break down this Python code line by line:

- `def is_palindrome(s)::`
 - This line defines a function named `is_palindrome` that takes one argument, `s` (which is expected to be a string).
- `""" Check if a string is a palindrome. ... Returns: bool: True if the string is a palindrome, False otherwise.""";`
 - This is a docstring, which provides a concise summary of what the function does, its arguments, and what it returns. It's good practice for code documentation.
- `return s == s[::-1];`
 - This is the core logic of the `is_palindrome` function.
 - `s[::-1]` creates a reversed version of the input string `s`. The `[::-1]` slice notation means 'start at the beginning, end at the end, step backwards by 1'.
 - `s == s[::-1]` compares the original string `s` with its reversed version. If they are identical, the expression evaluates to `True`; otherwise, it's `False`.
 - The function then returns this boolean result.
- `if __name__ == "__main__":`
 - This is a standard Python construct. Code inside this if block will only run when the script is executed directly (not when it's imported as a module into another script).
 - This is typically where you put your main program logic or test cases.
- `user_input = input("Enter a string to check if it's a palindrome: ")`
 - This line prompts the user to enter a string using the `input()` function.
 - The text inside the parentheses is displayed to the user.
 - Whatever the user types (before pressing Enter) is stored in the `user_input` variable as a string.
- `if is_palindrome(user_input)::`
 - This calls the `is_palindrome` function with the `user_input` obtained from the user.
 - It then checks if the return value of `is_palindrome(user_input)` is `True`.
- `print(f"\'{user_input}' is a palindrome.")`
 - If the if condition is `True` (meaning the string is a palindrome), this line prints a message indicating that the entered string is a palindrome.
 - It uses an f-string to embed the `user_input` directly into the message.
- `else::`
 - If the if condition is `False` (meaning the string is not a palindrome), the code inside this `else` block is executed.
- `print(f"\'{user_input}' is not a palindrome.")`
 - This line prints a message indicating that the entered string is not a palindrome.

My understanding : gave a function where only alpha numeric characters are present and others are removed and then it is converted to lower() using function and then used a slicing function to reverse it.

\

