

AI ASSISTANT CODING ASSIGNMENT 2

NAME : SAMPELLY SUHAS

ROLL NO : 2303A51453

BATCH NO : 21

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

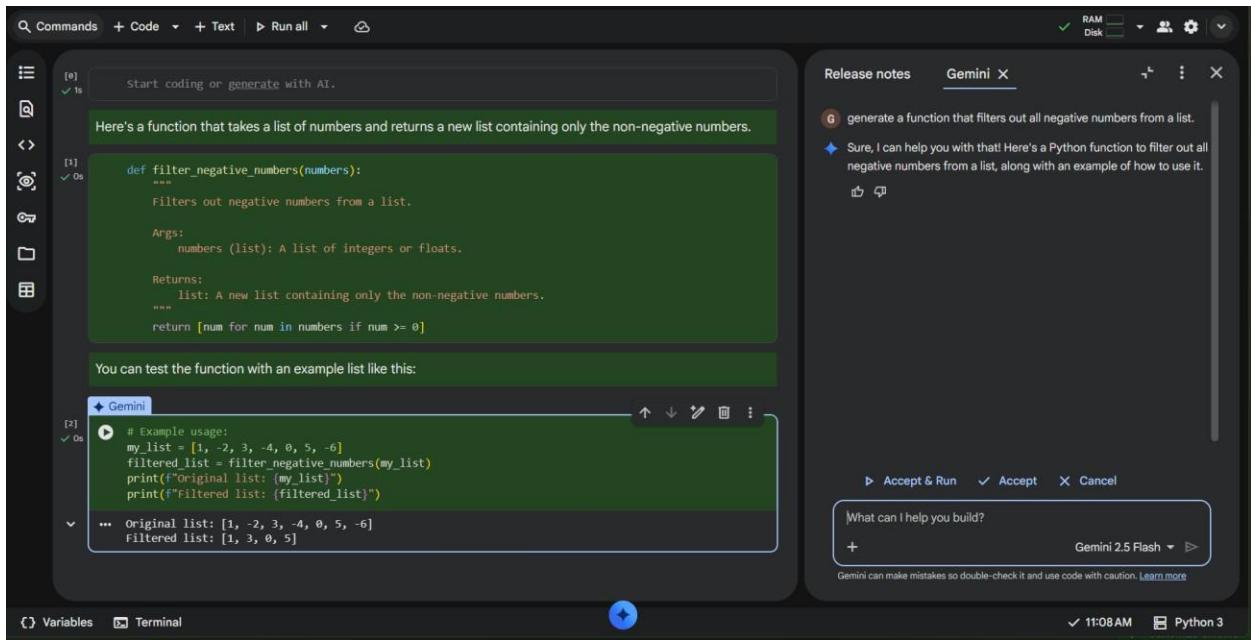
OUTPUT:

Here's a function that takes a list of numbers and returns a new list containing only the non-negative numbers.

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

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

    Returns:
        list: A new list containing only the non-negative numbers.
    """
    return [num for num in numbers if num >= 0]
```



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,

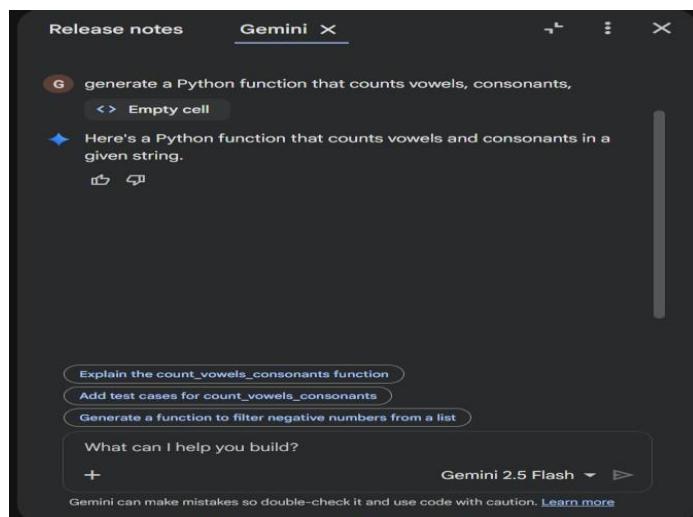
and digits in a string.

❖ Expected Output:

➤ Working function

➤ Sample inputs and outputs

OUTPUT:



```
▶ def count_vowels_consonants(text):
    """
    Counts the number of vowels and consonants in a given string.

    Args:
        text (str): The input string.

    Returns:
        dict: A dictionary containing the counts of 'vowels' and 'consonants'.
    """
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0

    for char in text:
        if char.isalpha(): # Check if the character is an alphabet
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

    return {"vowels": vowel_count, "consonants": consonant_count}
```

```
▶ # Example usage:
my_string = "Hello World"
counts = count_vowels_consonants(my_string)
print(f"Original string: '{my_string}'")
print(f"Vowels: {counts['vowels']}")
print(f"Consonants: {counts['consonants']}")

my_string_2 = "Python Programming"
counts_2 = count_vowels_consonants(my_string_2)
print(f"\nOriginal string: '{my_string_2}'")
print(f"Vowels: {counts_2['vowels']}")
print(f"Consonants: {counts_2['consonants']}")

... Original string: 'Hello World'
Vowels: 3
Consonants: 7

Original string: 'Python Programming'
Vowels: 4
Consonants: 13
```

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.

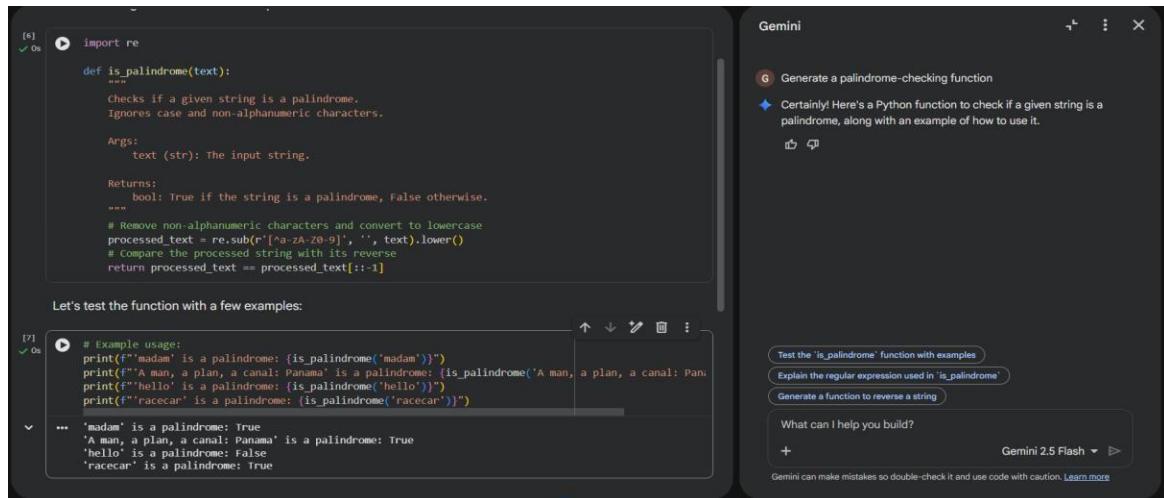
❖ Expected Output:

➤ Side-by-side code comparison

➤ Observations on clarity and structure

Output :

CODE GENERATED BY GEMINI:



The screenshot shows the Gemini interface with a generated Python function for checking palindromes. The code is as follows:

```
[6] 0s
import re

def is_palindrome(text):
    """
    Checks if a given string is a palindrome.
    Ignores case and non-alphanumeric characters.

    Args:
        text (str): The input string.

    Returns:
        bool: True if the string is a palindrome, False otherwise.
    """
    # Remove non-alphanumeric characters and convert to lowercase
    processed_text = re.sub(r'[^\w\-\_]', '', text).lower()
    # Compare the processed string with its reverse
    return processed_text == processed_text[::-1]

Let's test the function with a few examples:
```

Below the code, there are several examples of strings being checked for palindromicity:

```
[7] 0s
# Example usage:
print("madam" is a palindrome: {is_palindrome('madam')})
print("A man, a plan, a canal: Panama" is a palindrome: {is_palindrome('A man, a plan, a canal: Panama')})
print("hello" is a palindrome: {is_palindrome('hello')})
print("racecar" is a palindrome: {is_palindrome('racecar')})
```

... "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

The Gemini interface also includes a sidebar with prompts and a help section at the bottom.

CODE GENERATED BY COPILOT :

```
1 #Generate a palindrome check function
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 print(is_palindrome("A man a plan a canal Panama")) # Output: True
9 print(is_palindrome("Hello")) # Output: False
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS AZURE

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> & 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\new.py'
True
False
```

Observations on Clarity

Gemini Code:

- Very clear and well-documented.
- Includes a detailed docstring explaining the function's purpose, arguments, and return value.

Copilot Code:

- Simple and easy to follow, especially for beginners.
 - Uses a short variable name s, which is functional but less descriptive.
 - Output is minimal—just True or False—with context.
-

Observations on Structure

Gemini Code:

- Structured professionally with a reusable function and multiple test cases.
- Handles edge cases by removing non-alphanumeric characters and ignoring case.

Copilot Code:

- Structurally simple and compact.
- Focuses only on removing spaces and converting to lowercase.

Summary

- **Gemini's version** is more comprehensive, polished, and ready for broader use.
 - **Copilot's version** is concise and beginner-friendly, great for learning or quick testing.
-

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

OUTPUT :

The screenshot shows a code editor on the left and a Gemini AI interface on the right. The code editor displays a Python script named `is_palindrome.py` with the following content:

```
[6] ✓ 0s import re def is_palindrome(text):    """ Checks if a given string is a palindrome.    Ignores case and non-alphanumeric characters. Args:    text (str): The input string. Returns:    bool: True if the string is a palindrome, False otherwise.    """ # Remove non-alphanumeric characters and convert to lowercase processed_text = re.sub(r'[^\w\s]', '', text).lower() # Compare the processed string with its reverse return processed_text == processed_text[::-1]
```

Below the code, a comment says "Let's test the function with a few examples:" followed by some print statements. The Gemini interface on the right has a sidebar with a "Generate a palindrome-checking function" button. The main area shows a message from Gemini: "Certainly! Here's a Python function to check if a given string is a palindrome, along with an example of how to use it." There are also three buttons at the bottom: "Test the 'is_palindrome' function with examples", "Explain the regular expression used in 'is_palindrome'", and "Generate a function to reverse a string". At the bottom of the Gemini window, there is a message: "Gemini can make mistakes so double-check it and use code with caution. Learn more".

AI Explanation :

G Explain the code line by line

◆ Certainly! Let's break down the palindrome-checking code line by line.

Cell b252d9a5 : The `is_palindrome` function

```
import re
```

This line imports the `re` module, which provides regular expression operations. It's used here to remove non-alphanumeric characters from the input string.

```
def is_palindrome(text):
```

This defines a function named `is_palindrome` that takes one argument, `text`, which is expected to be a string.

```
    """
```

```
        Checks if a given string is a palindrome.
```

```
        Ignores case and non-alphanumeric characters.
```

Args:
`text (str):` The input string.

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

This is a docstring that explains what the `is_palindrome` function does, its arguments (`Args`), and what it returns (`Returns`). It specifies that the function checks for palindromes, ignoring case and non-alphanumeric characters.

```
# Remove non-alphanumeric characters and convert to lowercase
processed_text = re.sub(r'[^\w\-\_]', '', text).lower()
```

This is the core processing step:

- `re.sub(r'[^\w\-_]', '', text)` : This uses a regular expression to find all characters in the `text` string that are *not* alphanumeric (i.e., not a letter from 'a' to 'z' or 'A' to 'Z', or a digit from '0' to '9') and replaces them with an empty string, effectively removing them.
- `.lower()` : After removing non-alphanumeric characters, this converts the entire resulting string to lowercase. This ensures that the palindrome check is case-insensitive (e.g., 'Racecar' becomes 'racecar'). The result is stored in the `processed_text` variable.

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

This line checks if `processed_text` is equal to its reverse:

- `processed_text[::-1]` : This is a Python slice operation that creates a reversed copy of the `processed_text` string.
- `==` : This operator compares the `processed_text` with its reversed version. If they are identical, the string is a palindrome, and the expression evaluates to `True`; otherwise, it's `False`. This boolean result is then returned by the function.

Cell 01b3b2d9 : Example Usage

Cell 01b3b2d9 : Example Usage

```
# Example usage:
```

This is a comment indicating that the following lines demonstrate how to use the `is_palindrome` function.

```
print(f"'madam' is a palindrome: {is_palindrome('madam')}")
```

This line calls `is_palindrome` with the string 'madam'. The result (`True`) is then embedded into an f-string and printed to the console.

```
print(f"'A man, a plan, a canal: Panama' is a palindrome: {is_palindrome('A man, a plan, a canal: Panama')}")
```

This line calls `is_palindrome` with a classic palindrome phrase. The function will first process this string to 'amanaplanacanalpanama' (removing spaces, punctuation, and converting to lowercase) before checking if it's a palindrome. The result (`True`) is printed.

```
print(f"'hello' is a palindrome: {is_palindrome('hello')}")
```

This line calls `is_palindrome` with 'hello'. Since 'hello' is not the same as 'olleh', it will return `False`, which is then printed.

```
print(f"'racecar' is a palindrome: {is_palindrome('racecar')}")
```

This line calls `is_palindrome` with 'racecar'. Since 'racecar' is the same as its reverse, it will return `True`, which is then printed.

These example calls demonstrate how the function handles different inputs, including simple words, phrases with punctuation and mixed casing, and both palindromic and non-palindromic strings.

Clear Understanding:

- “*The explanation was very clear. I now understand how regular expressions are used to clean the string.*”
 - “*I get why .lower() is important – it makes the check case-insensitive.*”
 - “*The slicing trick [::-1] for reversing a string is neat, I hadn’t seen that before.*”
 - “*The examples helped me see how the function works with both simple words and complex phrases.*”
-