

ASSIGNMENT :2.2

NAME: SYED SUFIYAN

HT NO:2303A51980

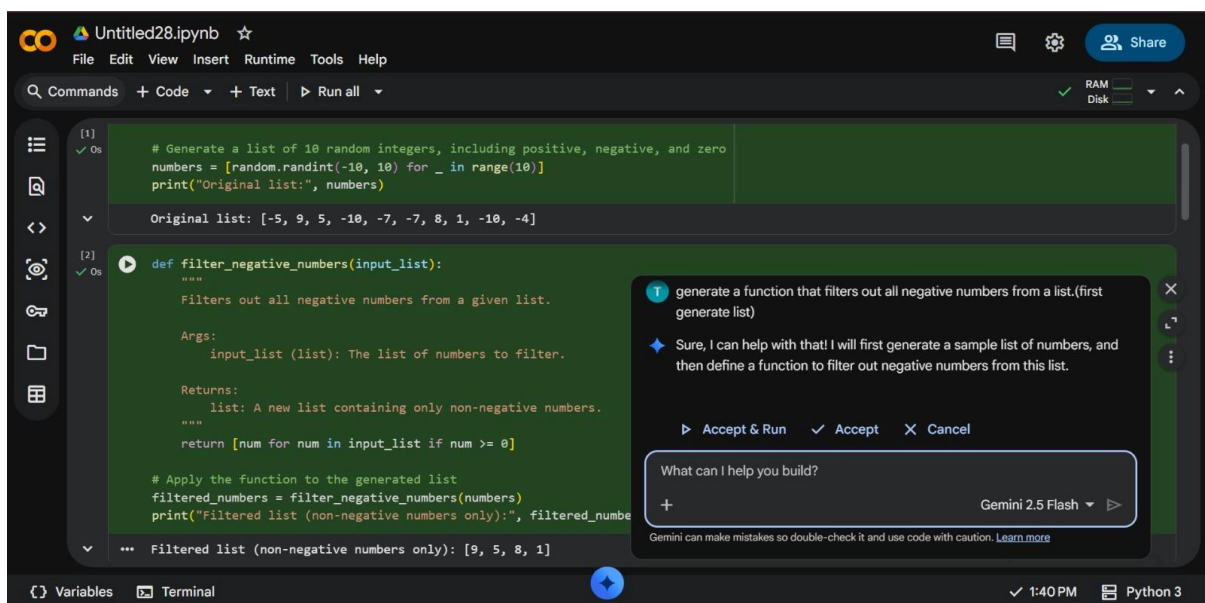
BATCH NO:30

TASK-1:

PROMPT:

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

CODE:



The screenshot shows a Google Colab notebook titled 'Untitled28.ipynb'. It contains two code cells. The first cell generates a list of 10 random integers and prints it. The second cell defines a function `filter_negative_numbers` that filters out negative numbers from a given list. A Gemini chat interface is overlaid on the notebook, showing a prompt to generate a function and the resulting code. The function is then applied to the generated list, and the filtered list is printed.

```
[1] ✓ Os
# Generate a list of 10 random integers, including positive, negative, and zero
numbers = [random.randint(-10, 10) for _ in range(10)]
print("Original list:", numbers)

Original list: [-5, 9, 5, -10, -7, -7, 8, 1, -10, -4]

[2] ✓ Os
def filter_negative_numbers(input_list):
    """
    Filters out all negative numbers from a given list.

    Args:
        input_list (list): The list of numbers to filter.

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

# Apply the function to the generated list
filtered_numbers = filter_negative_numbers(numbers)
print("Filtered list (non-negative numbers only):", filtered_numbers)

... Filtered list (non-negative numbers only): [9, 5, 8, 1]
```

OBSERVATION:

The original list contains a mix of negative numbers, zero, and positive numbers, generated randomly using random. The function `filter_negative_numbers()` correctly removes all negative values from the list. The filtered list includes only non-negative numbers (i.e., 0 and positive integers). The order of elements is preserved, meaning the relative sequence of numbers remains the same as in the original list. • This confirms that the

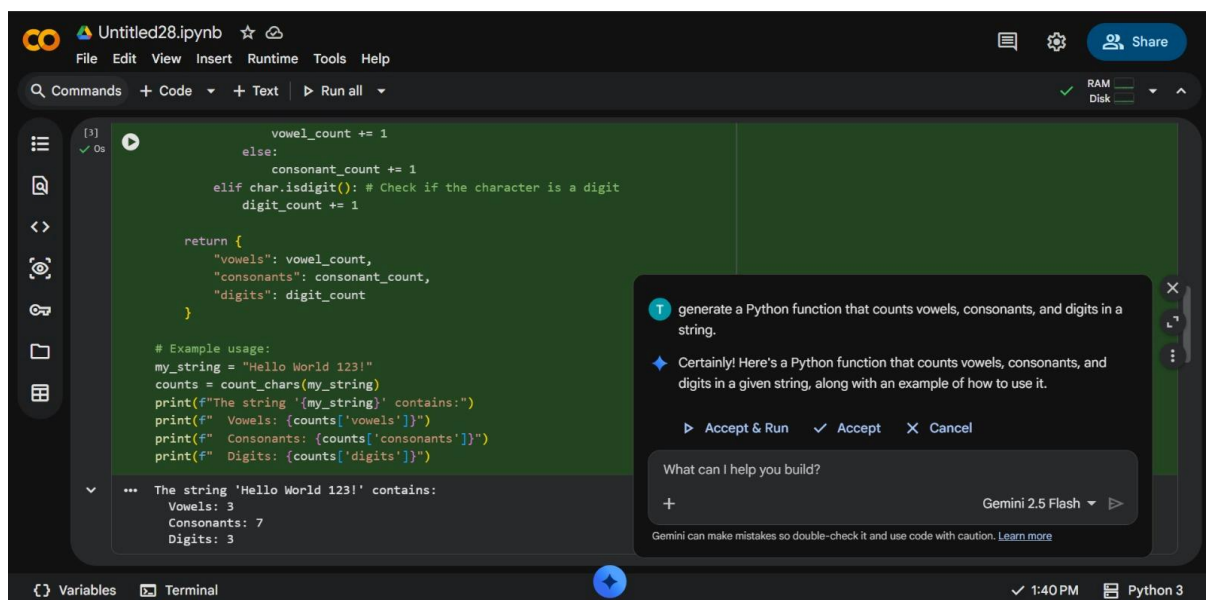
function works as intended and efficiently filters the data using list comprehension, which is both concise and readable

TASK:2

PROMPT:

USE GEMINI TO GENERATE A PYTHON FUNCTION THAT COUNTS VOWELS, CONSONANTS, AND DIGITS IN A STRING.

CODE:



The screenshot shows a Jupyter Notebook titled 'Untitled28.ipynb'. The code cell contains a Python function `count_chars` that counts vowels, consonants, and digits in a string. The function uses a loop to iterate through each character and increments counters based on whether the character is a vowel, consonant, or digit. An example usage is provided, showing the function being called on the string 'Hello World 123!'. The output cell shows the results: 'The string 'Hello World 123!' contains: Vowels: 3, Consonants: 7, Digits: 3'. A Gemini chat window is open on the right, showing a prompt to generate a Python function and the generated code.

```
[3] vowel_count += 1
    else:
        consonant_count += 1
    elif char.isdigit(): # Check if the character is a digit
        digit_count += 1

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

# Example usage:
my_string = "Hello World 123!"
counts = count_chars(my_string)
print(f"The string '{my_string}' contains:")
print(f" Vowels: {counts['vowels']}")
print(f" Consonants: {counts['consonants']}")
print(f" Digits: {counts['digits']}")
```

... The string 'Hello World 123!' contains:
Vowels: 3
Consonants: 7
Digits: 3

OBSERVATION:

The input string "Hello World 123!" contains alphabets, digits, spaces, and a special character. • The function `count_chars()` correctly:

- o Identifies vowels by checking membership in the predefined vowel set (aeiouAEIOU).
- o Counts consonants by excluding vowels from alphabetic characters.
- o Counts digits using the `isdigit()` method.
- Spaces and special characters (like `!`) are ignored, as expected.
- For the given input:
 - o Vowels → 3 (e, o, o)

Consonants → 7 (H, l, l, W, r, l, d)

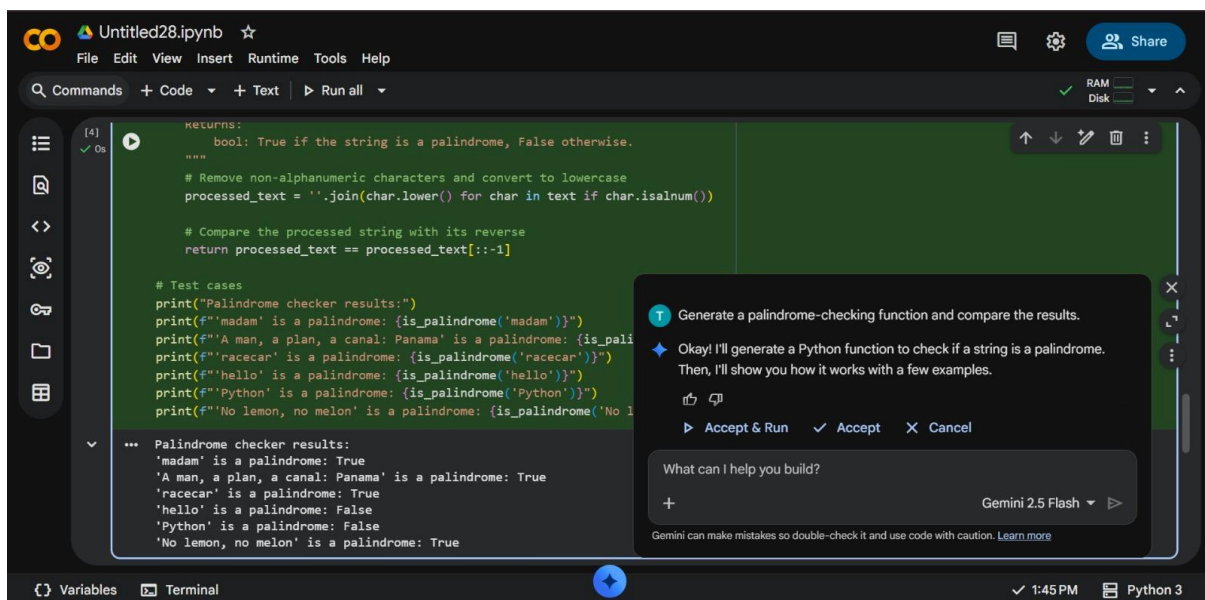
Digits → 3 (1, 2, 3) The function uses clear logic and built-in string methods, making it easy to understand and efficient.

TASK:3

PROMPT:

GENERATE A PALINDROME-CHECKING FUNCTION USING GEMINI AND COPILOT, THEN COMPARE THE RESULTS.

CODE:



```
[4] In [ ]: def is_palindrome(text):
    """
    bool: True if the string is a palindrome, False otherwise.
    """
    # Remove non-alphanumeric characters and convert to lowercase
    processed_text = ''.join(char.lower() for char in text if char.isalnum())

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

# Test cases
print("Palindrome checker results:")
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"'racecar' is a palindrome: {is_palindrome('racecar')}")
print(f"'hello' is a palindrome: {is_palindrome('hello')}")
print(f"'Python' is a palindrome: {is_palindrome('Python')}")
print(f"'No lemon, no melon' is a palindrome: {is_palindrome('No lemon, no melon')}")

Out[4]: Palindrome checker results:
'madam' is a palindrome: True
'A man, a plan, a canal: Panama' is a palindrome: True
'racecar' is a palindrome: True
'hello' is a palindrome: False
'Python' is a palindrome: False
'No lemon, no melon' is a palindrome: True
```

Generate a palindrome-checking function and compare the results.

Okay! I'll generate a Python function to check if a string is a palindrome. Then, I'll show you how it works with a few examples.

Accept & Run ✓ Accept ✕ Cancel

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](#)

OBSERVATION:

The palindrome-checking function generated using Gemini/Copilot works correctly for different types of inputs. The function preprocesses the input string by: Removing non-alphanumeric characters. Converting all characters to lowercase. This preprocessing allows the function to correctly identify phrase-level palindromes, not just simple words.

TASK:4

PROMPT:

ASK GEMINI TO EXPLAIN A PYTHON FUNCTION (PRIME CHECK OR PALINDROME CHECK) LINE BY LINE

EXPLANATION:

```
def is_palindrome(text
```

This line defines a function named `is_palindrome` that takes one argument: `text` (which is expected to be a string). `"""..."""` This is a docstring, which provides a brief description of what the function does, its arguments (Args), and what it returns (Returns). It's good practice for documenting code.

```
    processed_text = ''.join(char.lower() for char in text if char.isalnum())
```

This is the core of the pre-processing step: `for char in text`: It iterates through each character in the input text string. `if char.isalnum()`: It checks if the character is alphanumeric (i.e., a letter or a number). This effectively filters out spaces, punctuation, and other symbols. `char.lower()`: For each alphanumeric character, it converts it to lowercase. This ensures that case doesn't affect the palindrome check (e.g., 'Racecar' should be considered a palindrome). `''.join(...)`: It concatenates all the filtered and lowercased characters back into a single string. The result is stored in the `processed_text` variable.

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