ASSIGNMENT :2.2

NAME:AKSHITHA

HT NO:2303A51360

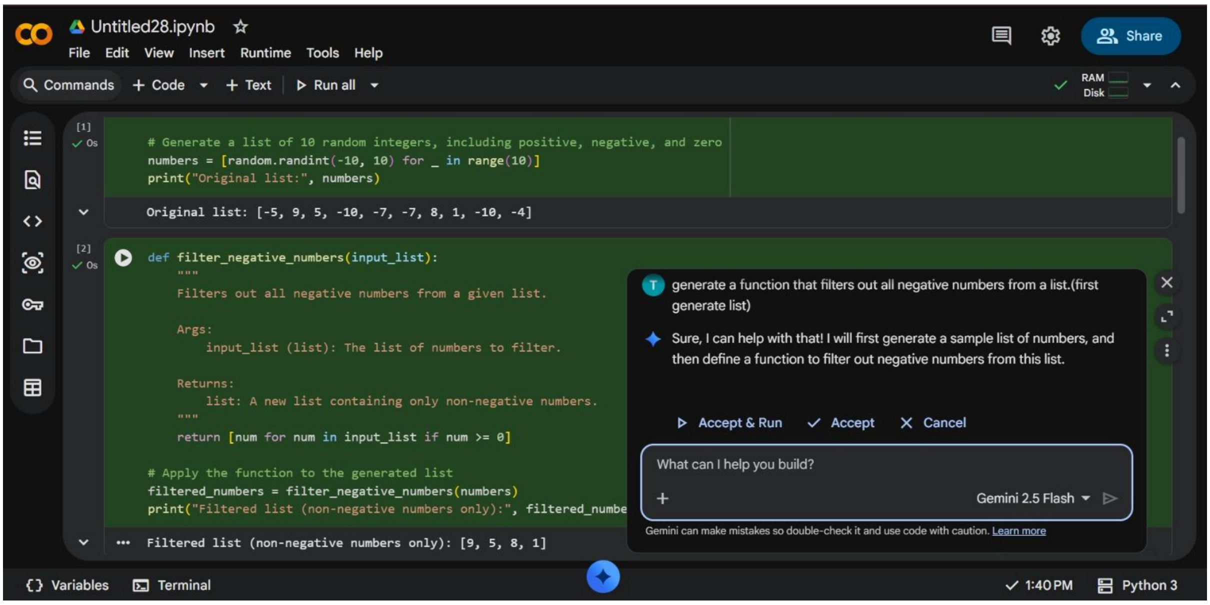
BATCH NO:29

TASK-1:

PROMPT:

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

CODE:



```
[1]
# 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]
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_numbe
```

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

generate a function that filters out all negative numbers from a list.(first generate list)

Sure, I can help with that! I will first generate a sample list of numbers, and then define a function to filter out negative numbers from this list.

Gemini can make mistakes so double-check it and use code with caution. Learn more

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).

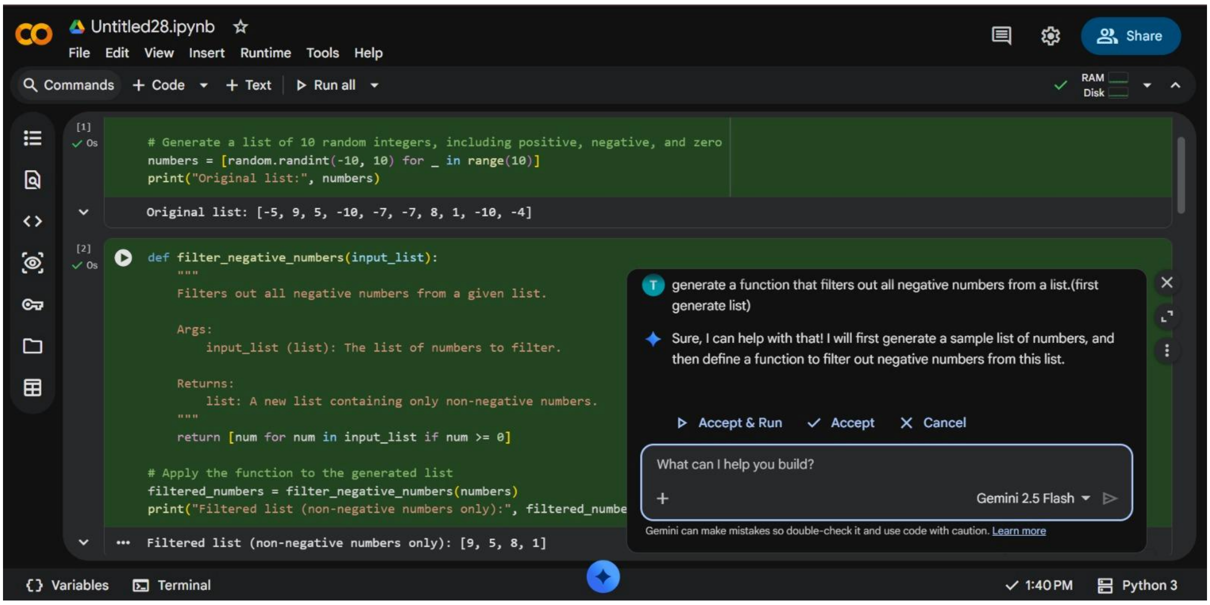
- The **order of elements is preserved**, meaning the relative sequence of numbers remains the same as in the original list.
- This confirms that the function works as intended and efficiently filters the data using **list comprehension**, which is both **concise and readable**.

TASK:2

PROMPT:

Use Gemini to generate a Python function that counts vowels, consonants,and digits in a string.

CODE:



```
# 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]
```

```
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_numbe
```

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

generate a function that filters out all negative numbers from a list.(first generate list)

Sure, I can help with that! I will first generate a sample list of numbers, and then define a function to filter out negative numbers from this list.

Gemini can make mistakes so double-check it and use code with caution. Learn more

OBSERVATION:

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

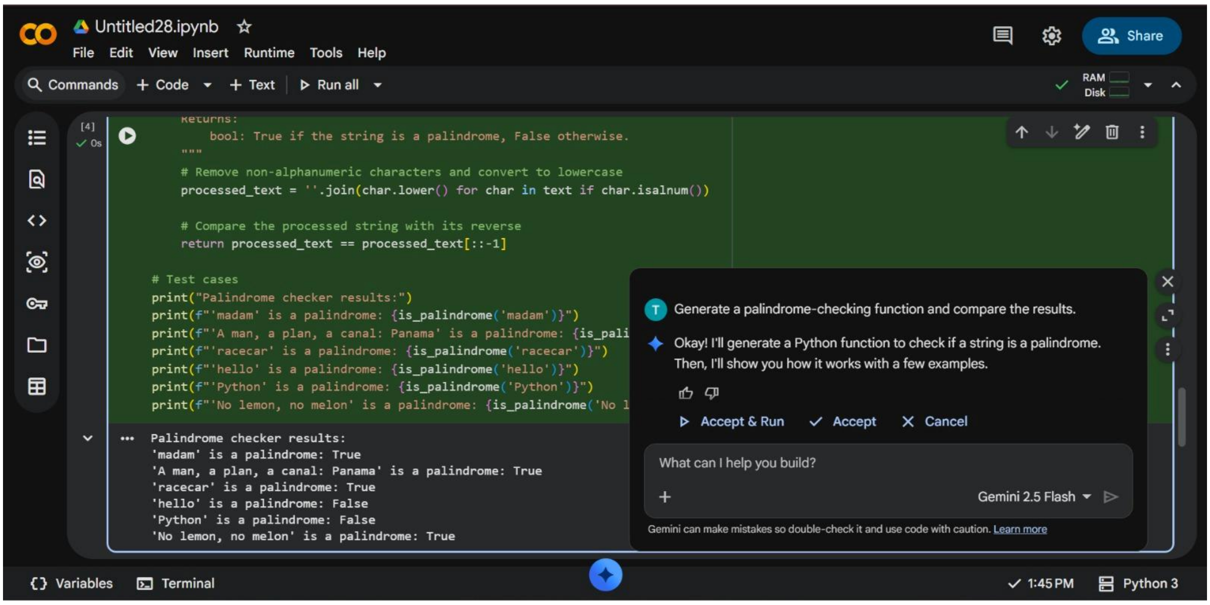
  - **Identifies vowels** by checking membership in the predefined vowel set (aeiouAEIOU).
  - **Counts consonants** by excluding vowels from alphabetic characters.
  - **Counts digits** using the isdigit() method.
- **Spaces and special characters** (like !) are **ignored**, as expected.
- For the given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



```
    Returns:
        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_pali
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 l
```

```
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
```

T 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.

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]
```

This line performs the actual palindrome check and returns the result:

processed_text[::-1]: This creates a reversed version of the processed_text string using Python's slicing notation. [::-1] means start from the end, go to the beginning, with a step of -1.

processed_text == processed_text[::-1]: It compares the processed_text with its reversed version. If they are identical, the expression evaluates to True, indicating it's a palindrome. Otherwise, it evaluates to False.

return: The boolean result (True or False) is returned by the function.