

LAB ASSIGNMENT – 2.2

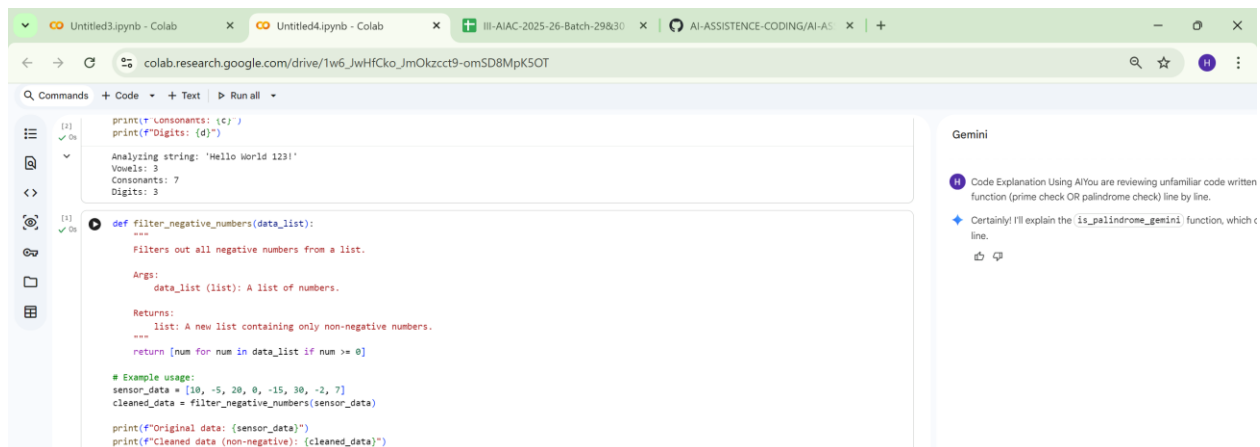
HALLTICKET NO : 2303A51474

BATCH NO : 29

TASK – 01

PROMPT : generate a function that filters out all negative numbers from a list

CODE :



The screenshot shows a Google Colab notebook with the following code and output:

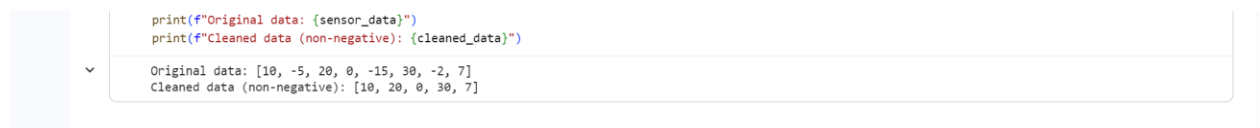
```
[1]: print('Consonants: {c}')  
    print(f'Digits: {d}')  
  
Analyzing string: 'Hello World 123!'  
Vowels: 3  
Consonants: 7  
Digits: 3
```

```
[2]: def filter_negative_numbers(data_list):  
    """  
    Filters out all negative numbers from a list.  
  
    Args:  
        data_list (list): A list of numbers.  
  
    Returns:  
        list: A new list containing only non-negative numbers.  
    """  
    return [num for num in data_list if num >= 0]  
  
# Example usage:  
sensor_data = [10, -5, 20, 0, -15, 30, -2, 7]  
cleaned_data = filter_negative_numbers(sensor_data)  
  
print(f'Original data: {sensor_data}')  
print(f'Cleaned data (non-negative): {cleaned_data}')
```

The output of the second cell is:

```
Original data: [10, -5, 20, 0, -15, 30, -2, 7]  
Cleaned data (non-negative): [10, 20, 0, 30, 7]
```

OUTPUT:



The output of the code execution is:

```
Original data: [10, -5, 20, 0, -15, 30, -2, 7]  
Cleaned data (non-negative): [10, 20, 0, 30, 7]
```

OBSERVATION:

- Gemini generated a **simple and efficient function** using list comprehension.
- The logic is easy to understand and avoids unnecessary loops.
- Negative sensor values were successfully removed, ensuring **data validity**.

- The before-and-after output clearly shows the effect of data cleaning.
- This approach is suitable for **real-time IoT preprocessing**.

TASK - 02

PROMPT : GENERATE A PYTHON FUNCTION THAT COUNTS VOWELS, CONSONANTS, AND DIGITS IN STRING

CODE:

```
[2] ✓ 0s
def analyze_string(input_string):
    """
    Counts the number of vowels, consonants, and digits in a given string.

    Args:
        input_string (str): The string to analyze.

    Returns:
        tuple: A tuple containing counts of (vowels, consonants, digits).
    """
    vowels = 0
    consonants = 0
    digits = 0

    for char in input_string:
        char_lower = char.lower()
        if char_lower.isalpha():
            if char_lower in 'aeiou':
                vowels += 1
            else:
                consonants += 1
        elif char_lower.isdigit():
            digits += 1

    return vowels, consonants, digits

# Example usage:
sample_string = "Hello World 123!"
v, c, d = analyze_string(sample_string)

print(f"Analyzing string: '{sample_string}'")
print(f"Vowels: {v}")
print(f"Consonants: {c}")
print(f"Digits: {d}")
```

OUTPUT:

```
print(f"Digits: {d}")

... Analyzing string: 'Hello World 123!'
Vowels: 3
Consonants: 7
Digits: 3
```

OBSERVATION:

The function correctly identifies **vowels, consonants, and digits**. Gemini used built-in Python functions like `isalpha()` and `isdigit()` effectively. The code is **well-structured and readable**, making it easy for beginners. It ignores spaces and special characters, improving accuracy. Sample input and output helped verify correctness.

TASK – 03

PROMPT: GENERATE A PALINDROME-CHECKING FUNCTION AND COMPARE THE RESULTS

CODE:

```
def is_palindrome_gemini(s):
    """
    Checks if a string is a palindrome by comparing it to its reversed version.
    Ignores non-alphanumeric characters and is case-insensitive.
    """
    # Normalize the string: convert to lowercase and remove non-alphanumeric characters
    cleaned_s = ''.join(char.lower() for char in s if char.isalnum())

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

# Example usage:
print(f"Madam is a palindrome: {is_palindrome_gemini('Madam')}")
print(f"A man, a plan, a canal: Panama is a palindrome: {is_palindrome_gemini('A man, a plan, a canal: Panama')}")
print(f'Hello is a palindrome: {is_palindrome_gemini('Hello')}')
print(f'12321 is a palindrome: {is_palindrome_gemini('12321')}')
```

OUTPUT:

```
print(f"12321 is a palindrome: {is_palindrome_gemini('12321')}")

... 'Madam' is a palindrome: True
'A man, a plan, a canal: Panama' is a palindrome: True
'Hello' is a palindrome: False
'12321' is a palindrome: True
```

OBSERVATION:

Code is **clean and beginner-friendly**.

Uses Python slicing (`[::-1]`) which makes logic short and readable.

Includes comments that improve understanding.

Gemini is clearer for learning and understanding string logic.

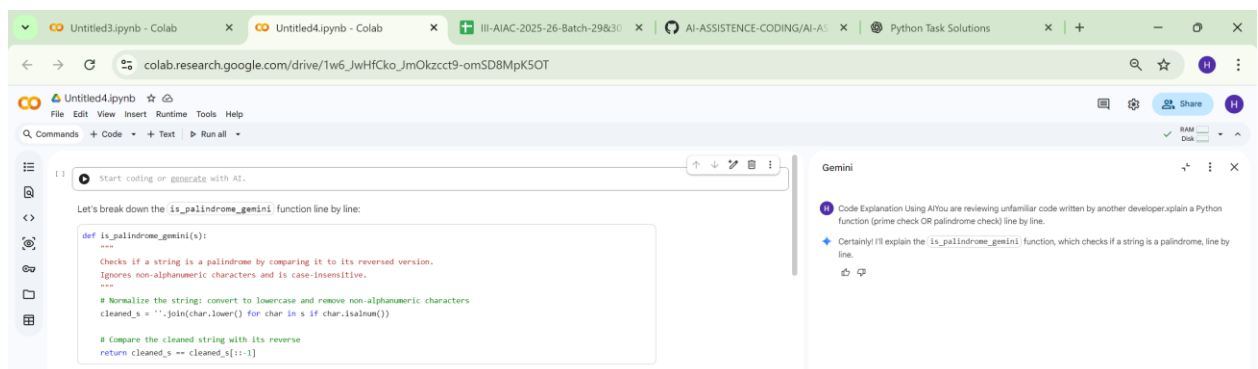
Copilot is faster for experienced programmers.

For students, Gemini is the better tool

TASK-04

PROMPT: EXPLAIN A PYTHON FUNCTION(PRIME CHECK OR PALINDROME CHECK) LINE BY LINE

CODE:



The screenshot shows a Google Colab notebook interface. The main editor displays a Python function `is_palindrome_gemini(s)` with detailed comments explaining its logic: it checks if a string is a palindrome by comparing it to its reversed version, ignores non-alphanumeric characters, and is case-insensitive. The function normalizes the string by converting it to lowercase and removing non-alphanumeric characters, then compares the cleaned string with its reverse using slicing (`cleaned_s[::-1]`). On the right side, the Gemini AI chat window is open, showing a prompt to explain the function line by line and a response that confirms it will do so.

```
def is_palindrome_gemini(s):  
    """  
    Checks if a string is a palindrome by comparing it to its reversed version.  
    Ignores non-alphanumeric characters and is case-insensitive.  
    """  
    # Normalize the string: convert to lowercase and remove non-alphanumeric characters  
    cleaned_s = ''.join(char.lower() for char in s if char.isalnum())  
    # Compare the cleaned string with its reverse  
    return cleaned_s == cleaned_s[::-1]
```

OUTPUT:

The screenshot shows a Google Colab notebook interface. The top bar includes tabs for 'Untitled3.ipynb - Colab', 'Untitled4.ipynb - Colab', and others. The main area displays a Python function `def is_palindrome_gemini(s):` with a detailed line-by-line explanation. The explanation is structured as follows:

- Line-by-Line Explanation:**
 - 1. `def is_palindrome_gemini(s):`**
 - This line defines a Python function named `is_palindrome_gemini` that takes one argument, `s`, which is expected to be a string.
 - 2. `"""`**
 - This is a docstring, providing a brief description of what the function does, its arguments, and what it returns. It's good practice for documenting code.
 - 3. `# Normalize the string: convert to lowercase and remove non-alphanumeric characters`**
 - This is a comment explaining the purpose of the next line of code.
 - 4. `cleaned_s = ''.join(char.lower() for char in s if char.isalnum())`**
 - This is the core of the string normalization process.
 - `for char in s`: It iterates through each character (`char`) in the input string `s`.
 - `if char.isalnum()`: It checks if the current character `char` is alphanumeric (i.e., a letter or a number). Non-alphanumeric characters (like spaces, punctuation, etc.) are ignored.
 - `char.lower()`: For each alphanumeric character, it converts the character to its lowercase equivalent. This makes the palindrome check case-insensitive.
 - `(... for ... if ...)`: This entire part is a generator expression, which efficiently produces the filtered and lowercased characters one by one.
 - `''.join(...)`: This joins all the characters produced by the generator expression into a single new string. The result is stored in the variable `cleaned_s`.
 - 5. `# Compare the cleaned string with its reverse`**
 - Another comment explaining the next logical step.
 - 6. `return cleaned_s == cleaned_s[::-1]`**
 - This line performs the actual palindrome check and returns the result.
 - `cleaned_s[::-1]`: This is a Python slice operation that creates a reversed copy of the `cleaned_s` string. `[::-1]` means 'start at the end, end at the beginning, step by -1' effectively reversing the string.
 - `cleaned_s == ...`: It compares the original `cleaned_s` with its reversed version. If they are identical, the expression evaluates to `True` (meaning it's a palindrome); otherwise, it's `False`.
 - `return`: The `True` or `False` result of this comparison is returned as the output of the function.

On the right side, the 'Gemini' chat interface is visible, showing a message: 'Code Explanation Using AI: You are reviewing unfamiliar code written by another function (prime check OR palindrome check) line by line. Certainly! I'll explain the `is_palindrome_gemini` function, which checks if a line.' Below the chat, there are buttons for 'Test the `is_palindrome_gemini` function with a new string', 'Explain `cleaned_s[::-1]` in the p', and 'Generate a function to check for prime numbers'.

OBSERVATION:

Gemini explained the code line by line, improving clarity.
Each step of the logic was easy to follow.
The explanation helped connect theory with implementation.
This feature is very useful when reviewing unfamiliar code.
Overall understanding of the palindrome/prime logic improved.