

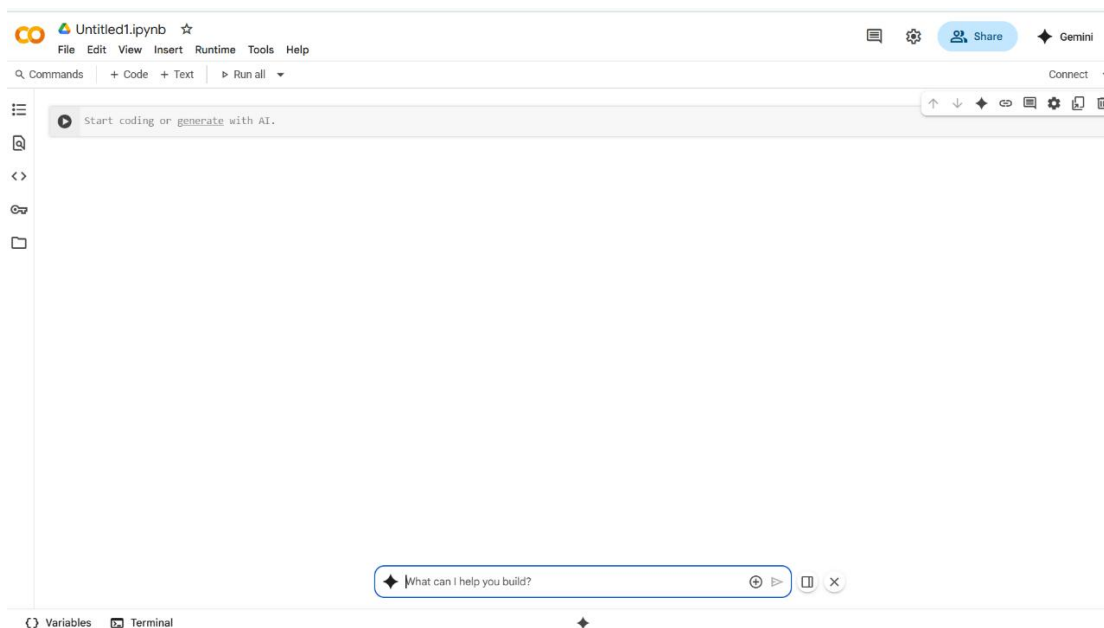
ASSIGNMENT

- 2

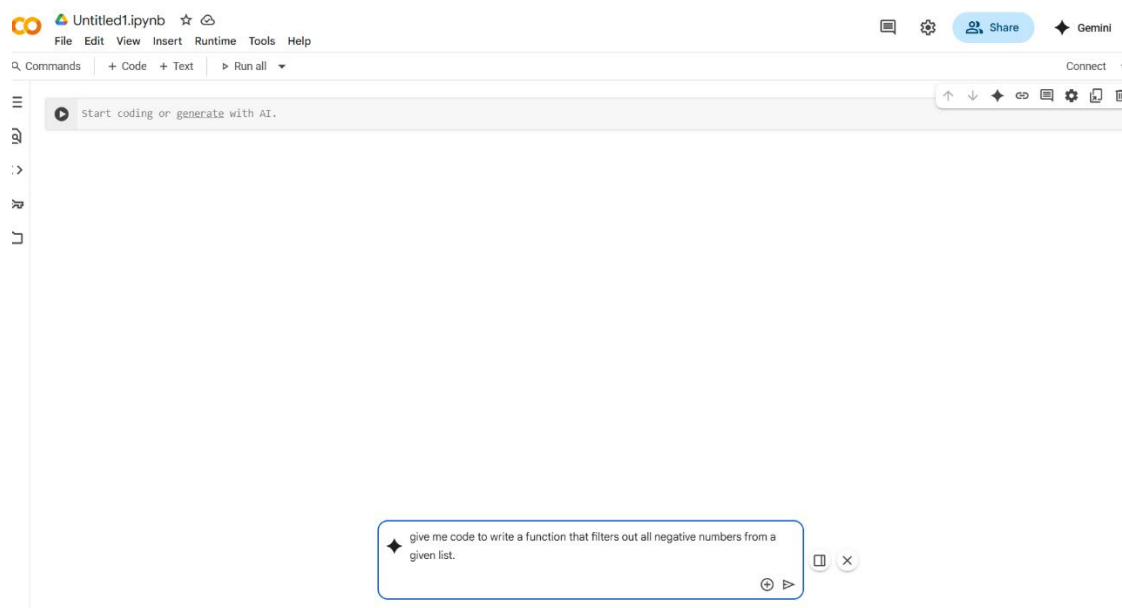
NAME : M.TANVI
COURSE : AI ASSISTED CODING
ROLL NO : 2403A510A4
DATE : 19 AUG 2025

TASK1:

OPEN GOOGLE COLAB:



INPUT:



The screenshot shows the Google Colab interface for a file named 'Untitled1.ipynb'. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a toolbar with icons for commands, code, text, and running. The main area contains a single code cell with the text 'Start coding or generate with AI.' and a prompt box at the bottom with the text 'give me code to write a function that filters out all negative numbers from a given list.'

CODE & OUTPUT:



The screenshot shows the Google Colab interface with the same file 'Untitled1.ipynb'. The code cell contains a Python function 'filter_negative_numbers' and an example usage. The output shows the original list and the filtered list.

```
def filter_negative_numbers(input_list):  
    """  
    Filters out all negative numbers from a given list.  
    Args:  
        input_list: A list of numbers.  
    Returns:  
        A new list containing only the non-negative numbers from the input list.  
    """  
    return [number for number in input_list if number >= 0]  
  
# Example usage:  
my_list = [1, -2, 3, -4, 5, 0, -6]  
filtered_list = filter_negative_numbers(my_list)  
print(f"Original list: {my_list}")  
print(f"Filtered list: {filtered_list}")
```

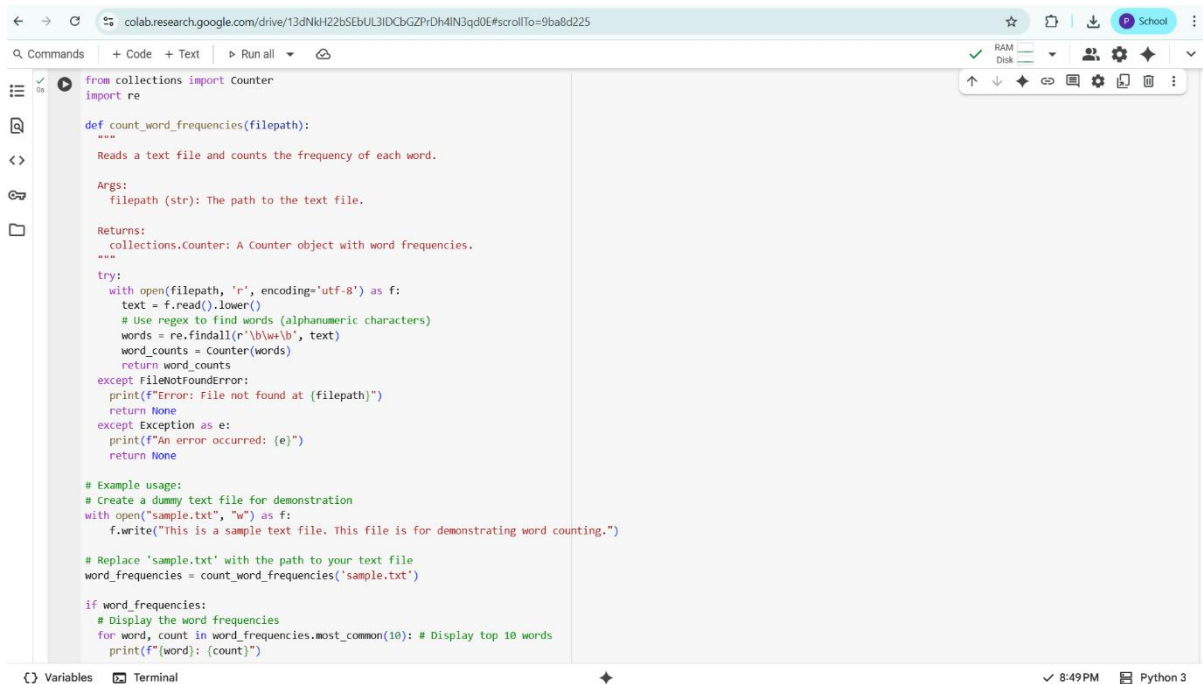
Original list: [1, -2, 3, -4, 5, 0, -6]
Filtered list: [1, 3, 5, 0]

TASK 2:

INPUT:

P give me code to generate code that reads a text file and counts the frequency of each word. Then explain the code

CODE :



```
from collections import Counter
import re

def count_word_frequencies(filepath):
    """
    Reads a text file and counts the frequency of each word.

    Args:
        filepath (str): The path to the text file.

    Returns:
        collections.Counter: A Counter object with word frequencies.
    """
    try:
        with open(filepath, 'r', encoding='utf-8') as f:
            text = f.read().lower()
            # Use regex to find words (alphanumeric characters)
            words = re.findall(r'\b\w+\b', text)
            word_counts = Counter(words)
            return word_counts
    except FileNotFoundError:
        print(f"Error: File not found at {filepath}")
        return None
    except Exception as e:
        print(f"An error occurred: {e}")
        return None

# Example usage:
# Create a dummy text file for demonstration
with open("sample.txt", "w") as f:
    f.write("This is a sample text file. This file is for demonstrating word counting.")

# Replace 'sample.txt' with the path to your text file
word_frequencies = count_word_frequencies('sample.txt')

if word_frequencies:
    # Display the word frequencies
    for word, count in word_frequencies.most_common(10): # Display top 10 words
        print(f"{word}: {count}")
```

OUTPUT:

```
1+ word_frequencies:
# Display the word frequencies
for word, count in word_frequencies.most_common(10): # Display top 10 words
    print(f"{word}: {count}")
```

```
this: 2
is: 2
file: 2
a: 1
sample: 1
text: 1
for: 1
demonstrating: 1
word: 1
counting: 1
```

EXPLANATION OF CODE BY GEMINI:

Gemini X

...

P give me code to generate code that reads a text file and counts the frequency of each word. Then explain the code

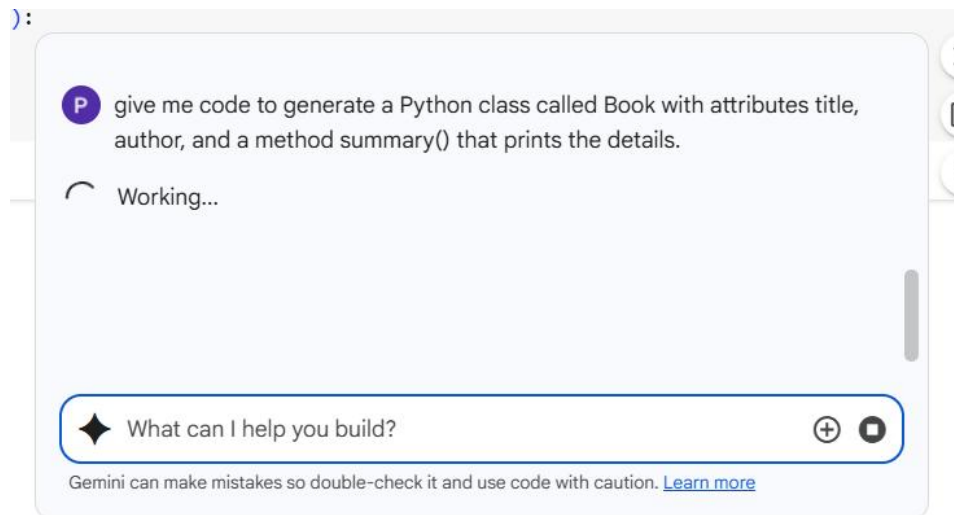
✦ Happy to help! I'll generate the code to read a text file and count word frequencies, and then explain it to you.

This code defines a function `count_word_frequencies` that takes a file path as input. It reads the file, converts the text to lowercase, and uses regular expressions to find all the words. It then uses the `Counter` class from the `collections` module to count the frequency of each word. Finally, it returns a `Counter` object containing the word frequencies. The example usage shows how to create a sample file, call the function, and print the top 10 most frequent words.

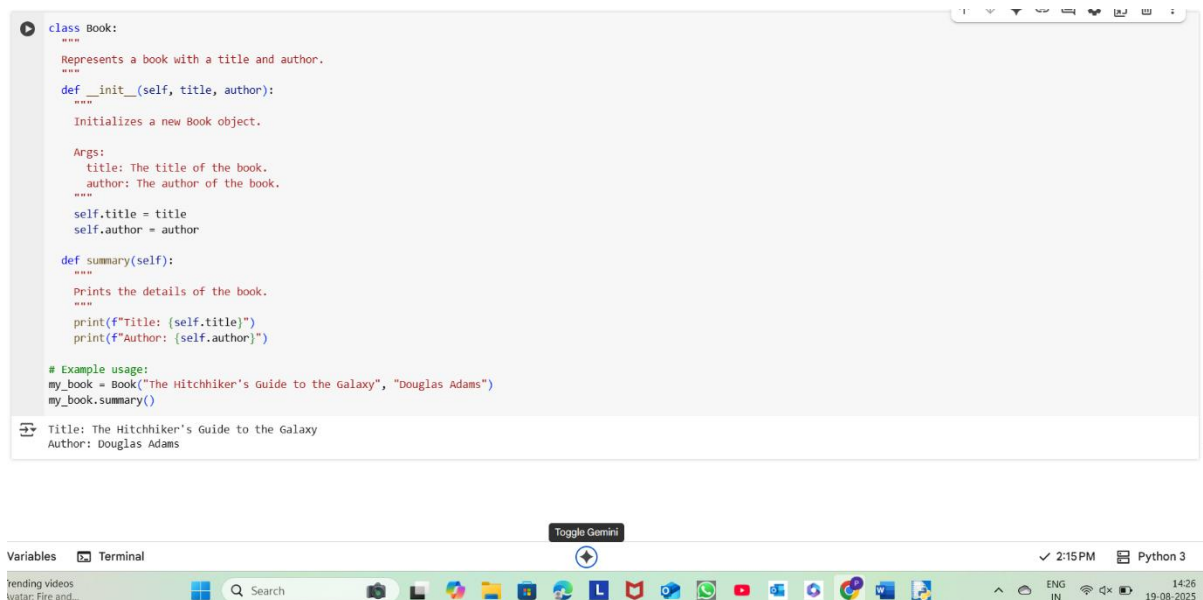
Would you like me to show you how to visualize the word frequencies?

TASK 3:

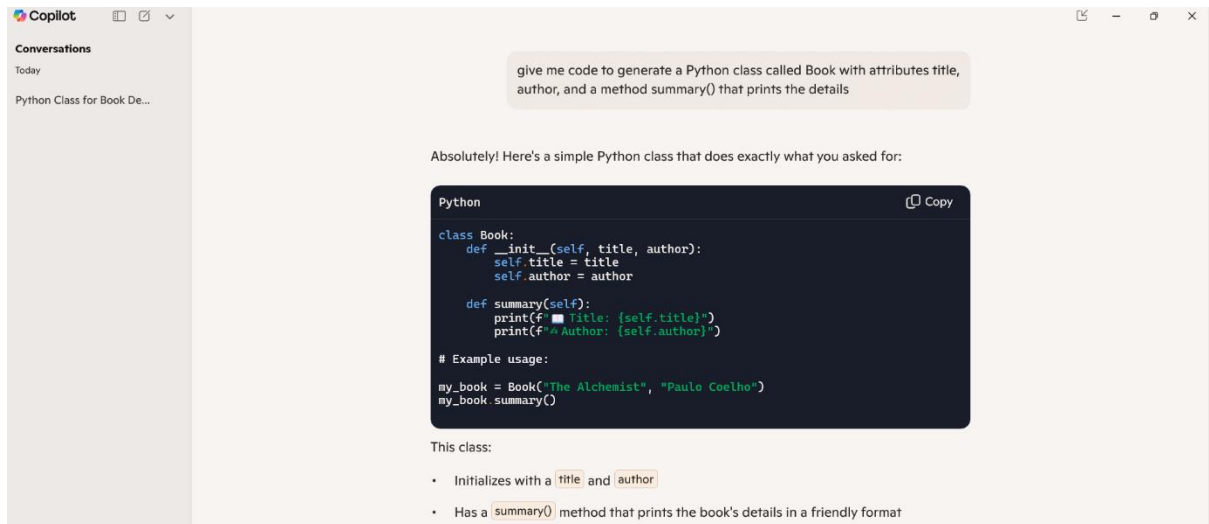
INPUT IN GEMINI:



CODE & OUTPUT IN GEMINI:



INPUT & CODE IN COPILOT:

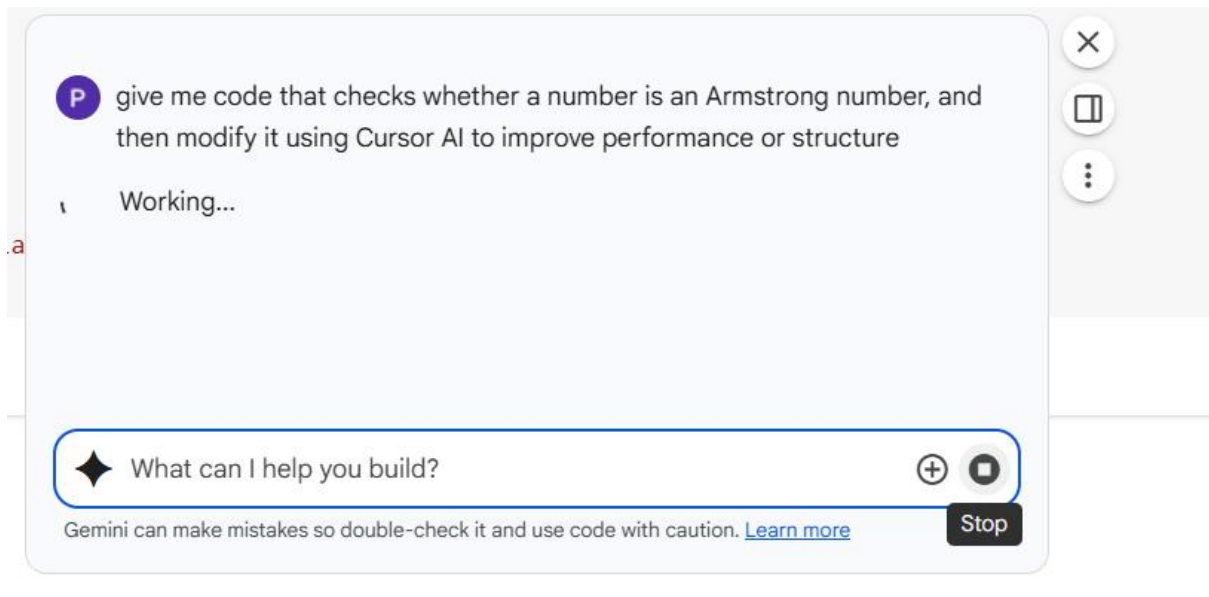


COMMENTING ON CODE CLARITY AND PERFORMANCE:

- ➔ IN GOOGLE COLAB WE CAN ABLE TO GENERATE THE CODE AND RUN THE CODE TO GET THE OUTPUT WITHOUT DOWNLOADING ANY PYTHON EXTENSION.
- ➔ IN COPILOT WE CAN ONLY ABLE TO GENERATE THE CODE WE CANNOT RUN AND GET THE OUTPUT LIKE GOOGLE COLAB.

TASK 4:

INPUT IN GEMINI:



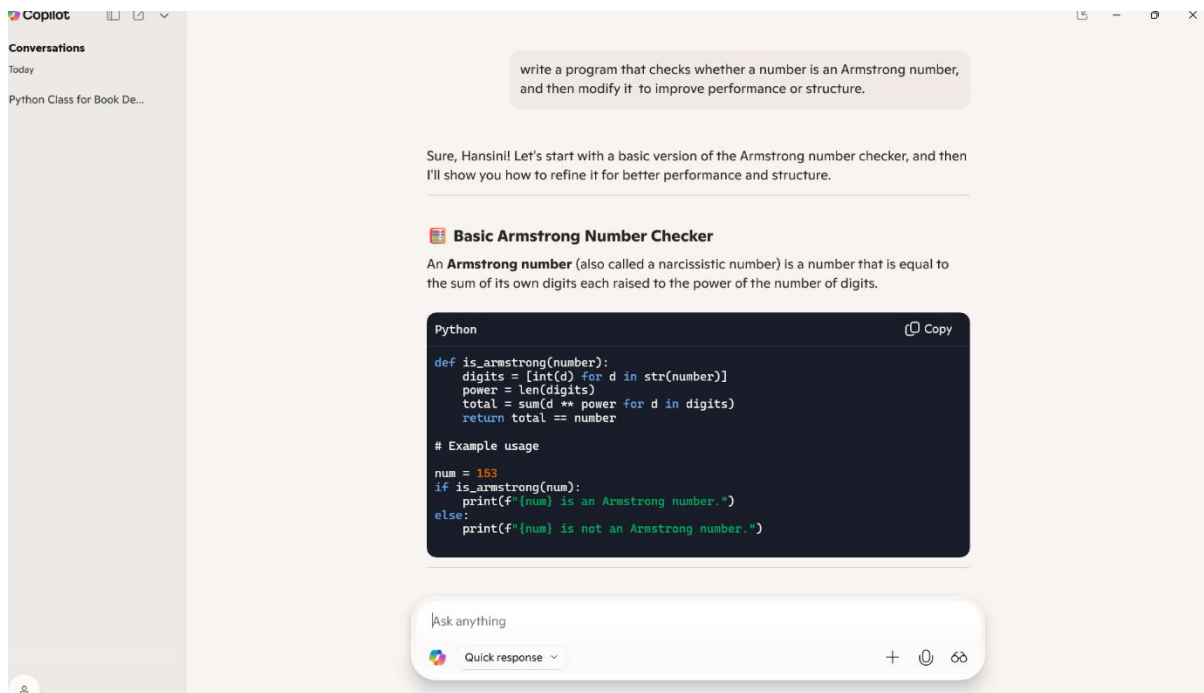
CODE & OUTPUT:

```
colab.research.google.com/drive/1Ooqcshlp7NRqDyv7V51n31mLId3P_7in#scrollTo=bee28099
```

```
def is_armstrong_number(number):  
    """  
    Checks if a number is an Armstrong number.  
  
    An Armstrong number is a number that is the sum of its own digits each  
    raised to the power of the number of digits.  
  
    Args:  
        number: An integer.  
  
    Returns:  
        True if the number is an Armstrong number, False otherwise.  
    """  
    # Convert the number to a string to get the number of digits  
    num_str = str(number)  
    num_digits = len(num_str)  
    sum_of_powers = 0  
    for digit in num_str:  
        sum_of_powers += int(digit) ** num_digits  
  
    return sum_of_powers == number  
  
# Example usage:  
num_to_check = 153  
if is_armstrong_number(num_to_check):  
    print(f"{num_to_check} is an Armstrong number")  
else:  
    print(f"{num_to_check} is not an Armstrong number")  
  
num_to_check = 123  
if is_armstrong_number(num_to_check):  
    print(f"{num_to_check} is an Armstrong number")  
else:  
    print(f"{num_to_check} is not an Armstrong number")
```

```
153 is an Armstrong number  
123 is not an Armstrong number
```

INPUT & CODE IN COPILOT:

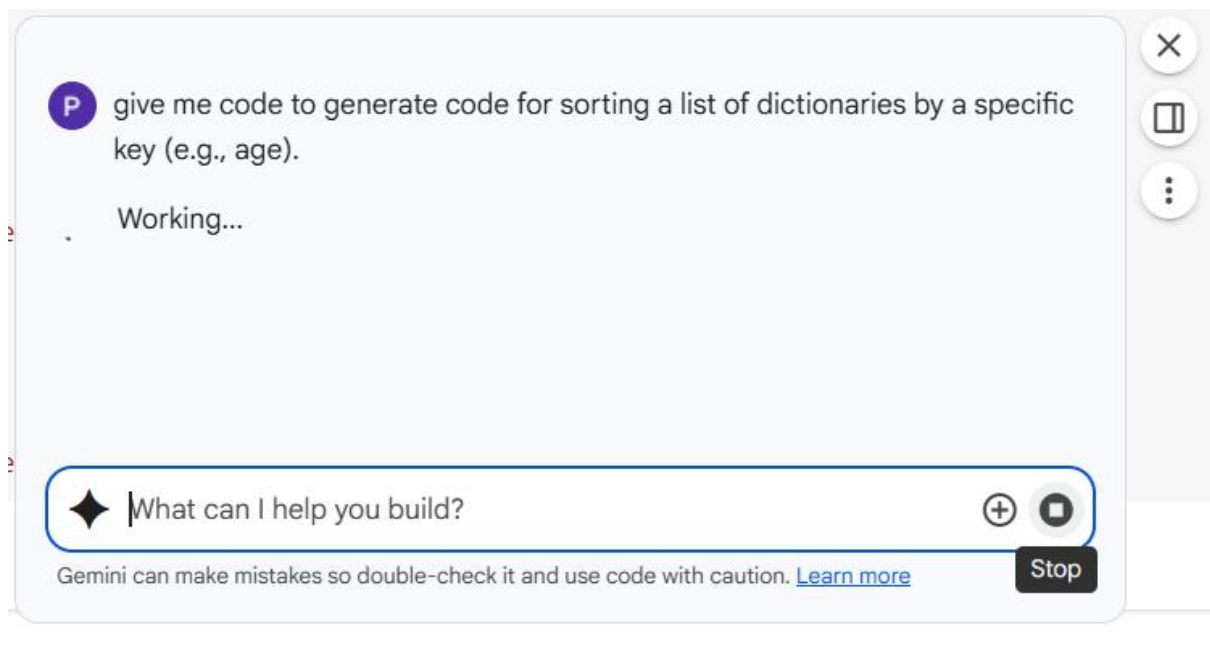


CHANGES IN COMPARISION OF GEMINI & COPILOT:

- ➔ IN GOOGLE COLAB WE CAN ABLE TO GENERATE THE CODE AND RUN THE CODE TO GET THE OUTPUT WITHOUT DOWNLOADING ANY PYTHON EXTENSION.
- ➔ IN COPILOT WE CAN ONLY ABLE TO GENERATE THE CODE WE CANNOT RUN AND GET THE OUTPUT LIKE GOOGLE COLAB.

TASK 5:

INPUT IN GEMINI:



CODE & OUTPUT:

```

def sort_list_of_dicts(list_of_dicts, sort_key):
    """
    Sorts a list of dictionaries by a specific key.

    Args:
        list_of_dicts: A list of dictionaries.
        sort_key: The key to sort by.

    Returns:
        A new list of dictionaries sorted by the specified key.
    """
    return sorted(list_of_dicts, key=lambda x: x[sort_key])

# Example usage:
people = [
    {'name': 'Alice', 'age': 30, 'city': 'New York'},
    {'name': 'Bob', 'age': 25, 'city': 'London'},
    {'name': 'Charlie', 'age': 35, 'city': 'Paris'},
    {'name': 'David', 'age': 25, 'city': 'Tokyo'}
]

sorted_by_age = sort_list_of_dicts(people, 'age')
print("Sorted by age:")
for person in sorted_by_age:
    print(person)

sorted_by_name = sort_list_of_dicts(people, 'name')
print("\nSorted by name:")
for person in sorted_by_name:
    print(person)

```

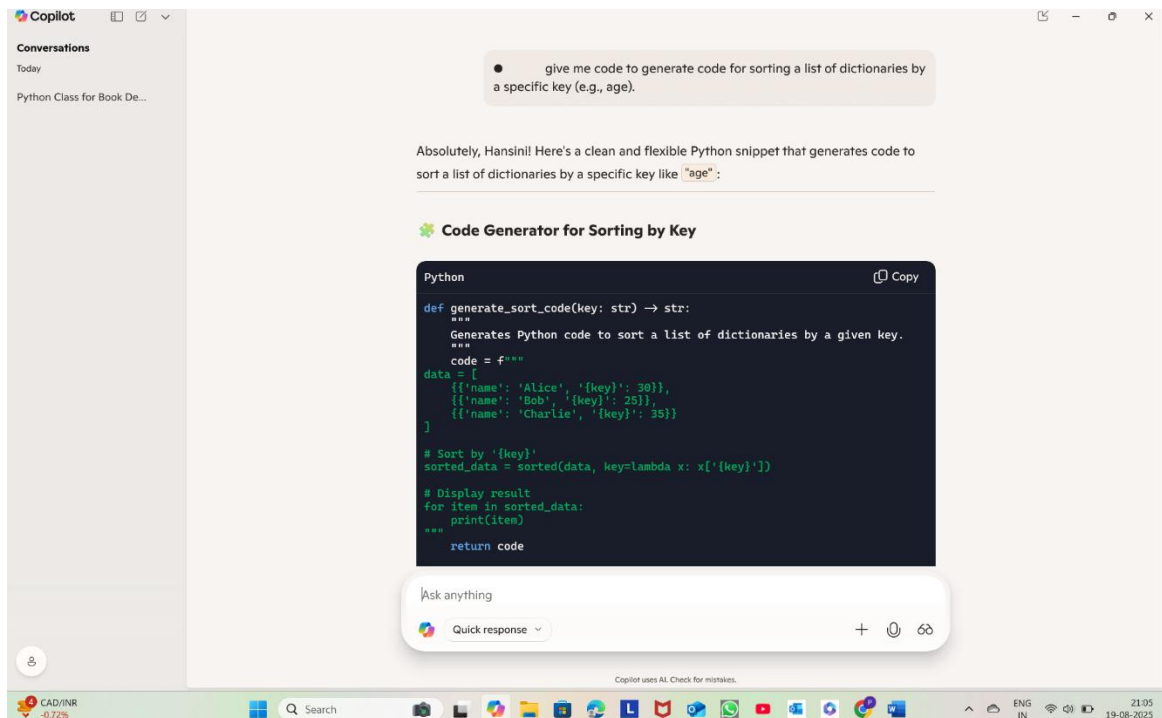
```

Sorted by age:
{'name': 'Bob', 'age': 25, 'city': 'London'}
{'name': 'David', 'age': 25, 'city': 'Tokyo'}
{'name': 'Alice', 'age': 30, 'city': 'New York'}
{'name': 'Charlie', 'age': 35, 'city': 'Paris'}

Sorted by name:
{'name': 'Alice', 'age': 30, 'city': 'New York'}
{'name': 'Bob', 'age': 25, 'city': 'London'}
{'name': 'Charlie', 'age': 35, 'city': 'Paris'}
{'name': 'David', 'age': 25, 'city': 'Tokyo'}

```

INPUT & CODE IN COPILOT:



CHANGES IN COMPARISION OF GEMINI & COPILOT:

- ➔ IN GOOGLE COLAB WE CAN ABLE TO GENERATE THE CODE AND RUN THE CODE TO GET THE OUTPUT WITHOUT DOWNLOADING ANY PYTHON EXTENSION.
- ➔ IN COPILOT WE CAN ONLY ABLE TO GENERATE THE CODE WE CANNOT RUN AND GET THE OUTPUT LIKE GOOGLE COLAB.