AI ASSISTED CODING

LAB ASSIGNMENT – 2

Program : B-TECH
Specialization : AIML
Name of Student : AKULA MANOJ
Enrollment-No : 2403A52031
Batch No : 02
Date : 19-08-2025

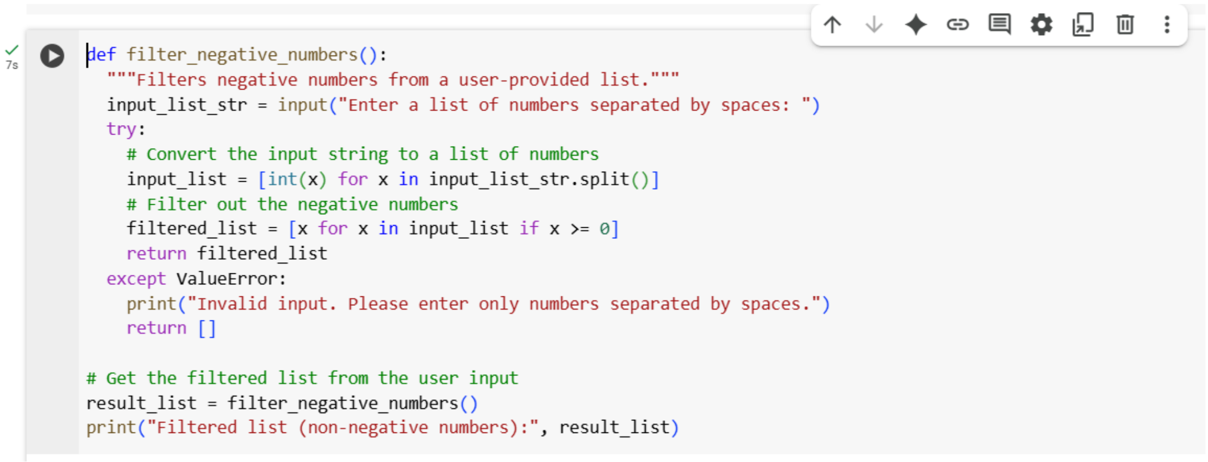
TASK DESCRIPTION 1:

USE GEMINI IN COLAB TO WRITE A FUNCTION THAT FILTERS OUT ALL NEGATIVE NUMBERS FROM A GIVEN LIST.

PROMPT:-

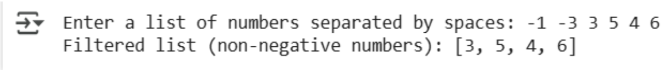
Write a function that filters out all the negative numbers from a given list and it should take user input.

GENERATED CODE:



```python
def filter_negative_numbers():
  """Filters negative numbers from a user-provided list."""
  input_list_str = input("Enter a list of numbers separated by spaces: ")
  try:
    # Convert the input string to a list of numbers
    input_list = [int(x) for x in input_list_str.split()]
    # Filter out the negative numbers
    filtered_list = [x for x in input_list if x >= 0]
    return filtered_list
  except ValueError:
    print("Invalid input. Please enter only numbers separated by spaces.")
    return []

# Get the filtered list from the user input
result_list = filter_negative_numbers()
print("Filtered list (non-negative numbers):", result_list)
```

OUTPUT:



```
Enter a list of numbers separated by spaces: -1 -3 3 5 4 6
Filtered list (non-negative numbers): [3, 5, 4, 6]
```

OBSERVATION:

While using Gemini AI in Google Colab, I explored how efficient code can be generated through appropriate prompts. By experimenting with different instructions, I learned that the quality of the prompt directly influences the quality of the generated code.

Conclusion:

Through this activity, I gained valuable insights into how AI can be effectively leveraged for coding tasks, while also reinforcing my understanding of systematic code development.

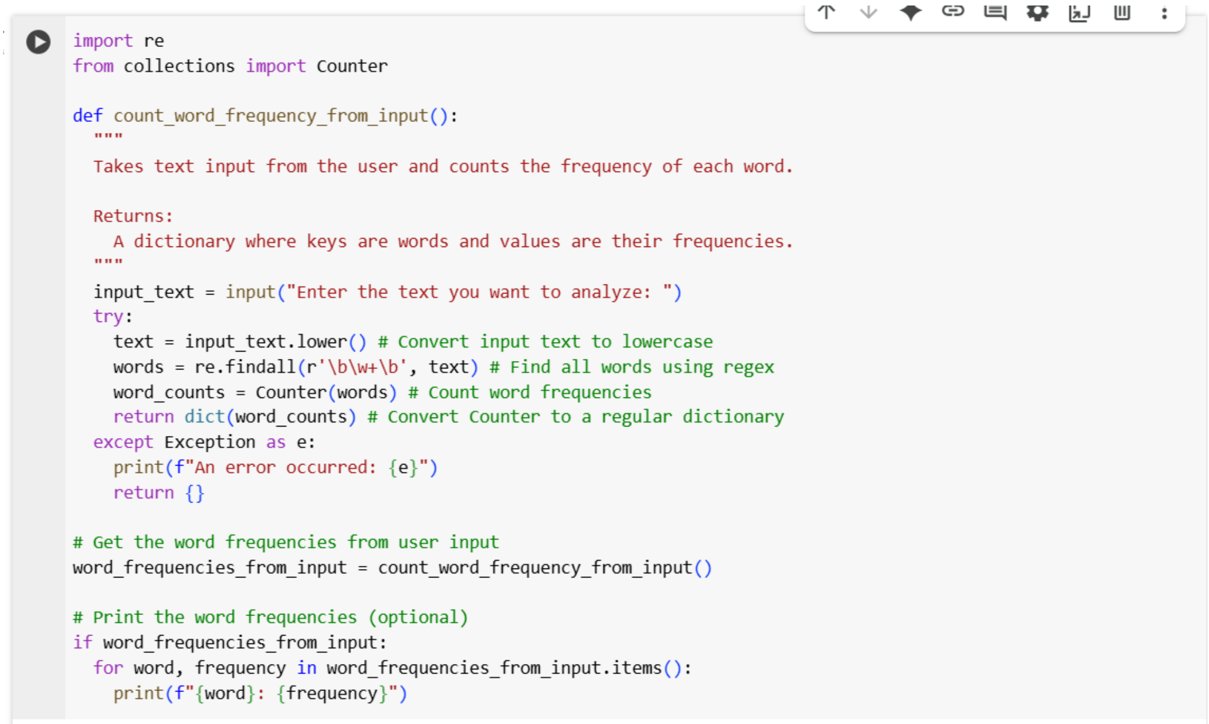
TASK DESCRIPTION 2:

Ask Gemini to generate code that reads a text file and counts the frequency of each word. Then ask it to explain the code.

PROMPT:

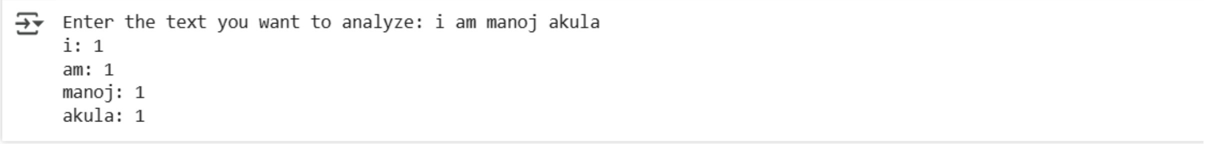
write a code that reads a text file and counts the frequency of each word and it should take input from me manually without saving the file, and explain the code step by step.

GENERATED CODE:



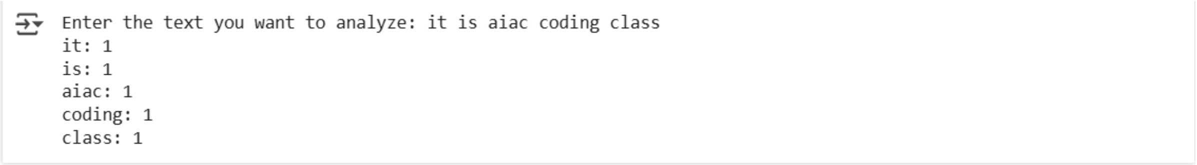
```
import re
from collections import Counter

def count_word_frequency_from_input():
  """
  Takes text input from the user and counts the frequency of each word.

  Returns:
    A dictionary where keys are words and values are their frequencies.
  """
  input_text = input("Enter the text you want to analyze: ")
  try:
    text = input_text.lower() # Convert input text to lowercase
    words = re.findall(r'\b\w+\b', text) # Find all words using regex
    word_counts = Counter(words) # Count word frequencies
    return dict(word_counts) # Convert Counter to a regular dictionary
  except Exception as e:
    print(f"An error occurred: {e}")
    return {}

# Get the word frequencies from user input
word_frequencies_from_input = count_word_frequency_from_input()

# Print the word frequencies (optional)
if word_frequencies_from_input:
  for word, frequency in word_frequencies_from_input.items():
    print(f"{word}: {frequency}")
```

OUTPUT-1 :-



```
Enter the text you want to analyze: i am manoj akula
i: 1
am: 1
manoj: 1
akula: 1
```

OUTPUT-2 :-



```
Enter the text you want to analyze: it is aiac coding class
it: 1
is: 1
aiac: 1
coding: 1
class: 1
```

GEMINI’S EXPLANATION ON THE CODE STEP BY STEP:

Here is the code that will take text input from you and count the frequency of each word:

Great! The code executed successfully with your input.

Now, let's go through the code step by step:

1. Import Libraries:
   - import re: This line imports the re module for working with regular expressions, which we use to find words.
   - from collections import Counter: This line imports the Counter class, which is a convenient tool for counting the occurrences of items.
2. Define the Function count_word_frequency_from_input:
   - def count_word_frequency_from_input():: This defines a function that will handle the process of taking input and counting word frequencies. It doesn't require any arguments because it gets the input directly from the user.
   - """Docstring""": This explains what the function does and what it returns.
3. Get Input from the User:
   - input_text = input("Enter the text you want to analyze: "): This is the key change from the previous code.
     The input() function displays the message "Enter the text you want to analyze: " to the user and waits for them to type something and press Enter. The text the user types is stored in the input_text variable.

4. Error Handling (try...except block):
    - try:: This block contains the code that might potentially cause an error.
    - text = input_text.lower(): This converts the user's input text to lowercase. This ensures that words like "The" and "the" are treated as the same word when counting.
    - words = re.findall(r'\b\w+\b', text): This uses the re.findall() function with the regular expression r'\b\w+\b' to find all whole words in the input text.
        - \b: Matches word boundaries.
        - \w+: Matches one or more word characters (letters, numbers, and underscore).
        - This effectively extracts a list of words from the input string.
    - word_counts = Counter(words): This creates a Counter object named word_counts and initializes it with the list of words. The Counter automatically counts how many times each unique word appears in the list.
    - return dict(word_counts): This converts the Counter object into a regular Python dictionary and returns it. The dictionary contains words as keys and their counts as values.
    - except Exception as e:: This catches any other unexpected errors that might occur during the process, prints an error message, and returns an empty dictionary.
5. Call the Function and Store Results:
    - word_frequencies_from_input = count_word_frequency_from_input(): This line calls the

function we just defined. It prompts the user for input, processes it, and stores the resulting dictionary of word frequencies in the word_frequencies_from_input variable.

6. Print the Word Frequencies (Optional):
    - if word_frequencies_from_input:: This checks if the word_frequencies_from_input dictionary is not empty (meaning the process was successful).
    - for word, frequency in word_frequencies_from_input.items():: This loop iterates through each word and its corresponding frequency in the dictionary.
    - print(f"{word}: {frequency}"): This prints each word and its frequency in a formatted string.

This version of the code is more interactive as it takes the text directly from you when you run the cell!

OBSERVATION:
While using Gemini AI in Google Colab, I found the way of ai writing and approaching the prompt and taking my commads it worked perfectly by giving code.I explored how efficient code can be generated through appropriate prompts. By experimenting with different instructions, I learned that the quality of the prompt directly influences the quality of the generated code.

Conclusion:
Through this activity, I gained valuable insights into how AI can be effectively leveraged for coding tasks, while also reinforcing my understanding of systematic code development.

TASK DESCRIPTION 3:

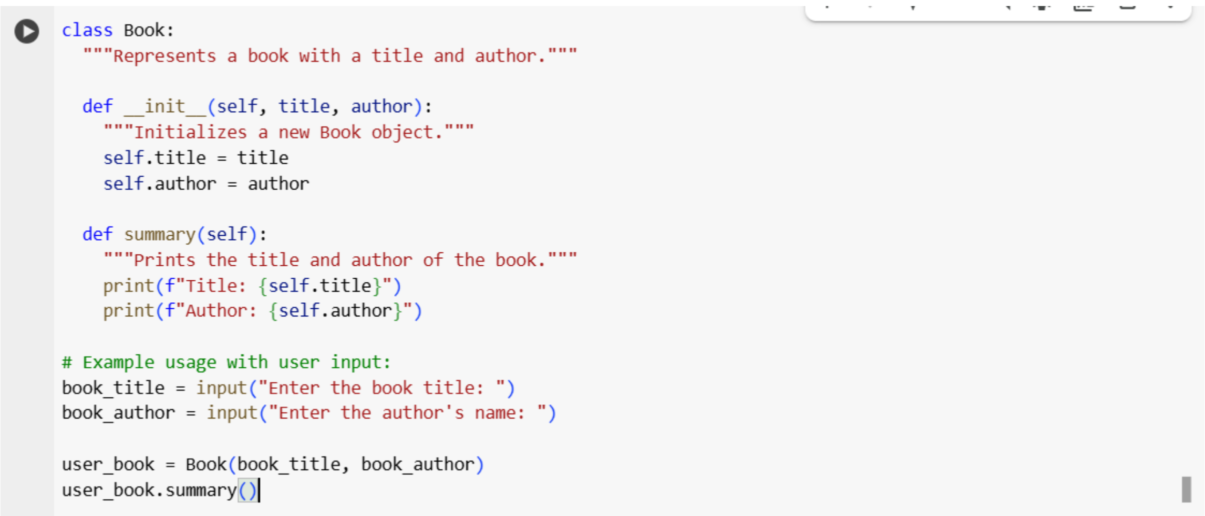
Use Cursor AI to generate a Python class called Book with attributes title, author, and a method summary() that prints the details.

NOTE:- Instead of cursor AI, I am using GEMINI GOOGLE COLLAB.

PROMPT:-

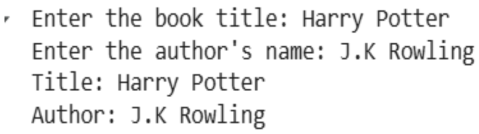
Generate a Python class called Book with attributes title, author, and a method summary() that prints the details. Write a code in Python and it should take user input and make sure to give method summary() that prints details.

CODE GENERATED BY GEMINI AI:-



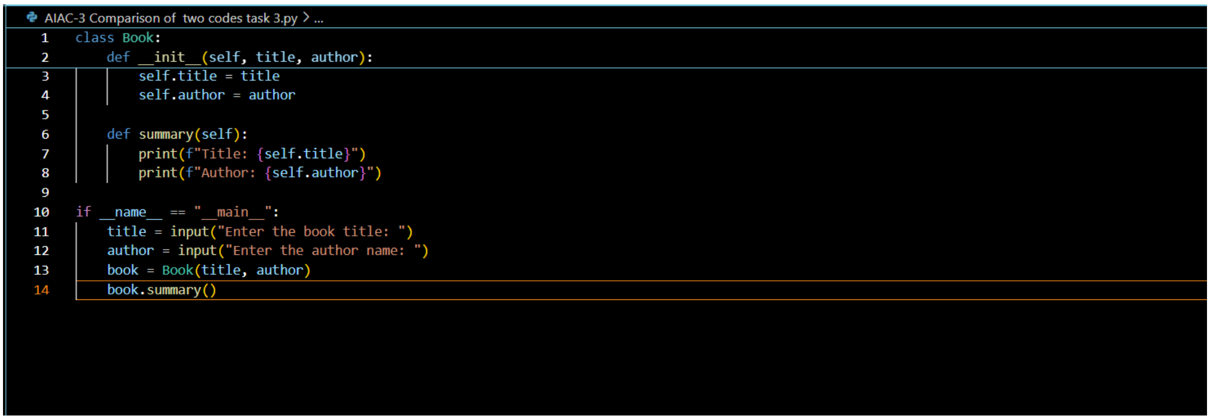
```
class Book:
  """Represents a book with a title and author."""

  def __init__(self, title, author):
    """Initializes a new Book object."""
    self.title = title
    self.author = author

  def summary(self):
    """Prints the title and author of the book."""
    print(f"Title: {self.title}")
    print(f"Author: {self.author}")

# Example usage with user input:
book_title = input("Enter the book title: ")
book_author = input("Enter the author's name: ")

user_book = Book(book_title, book_author)
user_book.summary()
```

OUTPUT:-



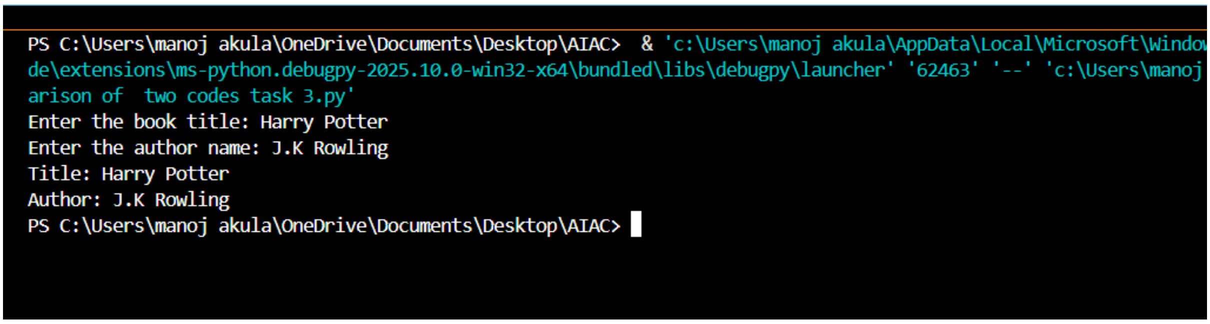
```
Enter the book title: Harry Potter
Enter the author's name: J.K Rowling
Title: Harry Potter
Author: J.K Rowling
```

CODE GENERATED BY GIT HUB COPILOT:-



```python
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def summary(self):
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")

if __name__ == "__main__":
    title = input("Enter the book title: ")
    author = input("Enter the author name: ")
    book = Book(title, author)
    book.summary()
```

OUTPUT:-



```
PS C:\Users\manoj akula\OneDrive\Documents\Desktop\AIAC>  & 'c:\Users\manoj akula\AppData\Local\Microsoft\Windo
de\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '62463' '--' 'c:\Users\manoj
arison of  two codes task 3.py'
Enter the book title: Harry Potter
Enter the author name: J.K Rowling
Title: Harry Potter
Author: J.K Rowling
PS C:\Users\manoj akula\OneDrive\Documents\Desktop\AIAC> 
```

COMPARISON: I observed that the git GitHub Copilot provides faster and more user friendly .

CONCLUSION:

Through this activity. i found that the experience of both copilot and gemini google collab. I gained valuable insights into how AI can be effectively leveraged for coding tasks, while also reinforcing my understanding of systematic code development.

TASK DESCRIPTION 4:

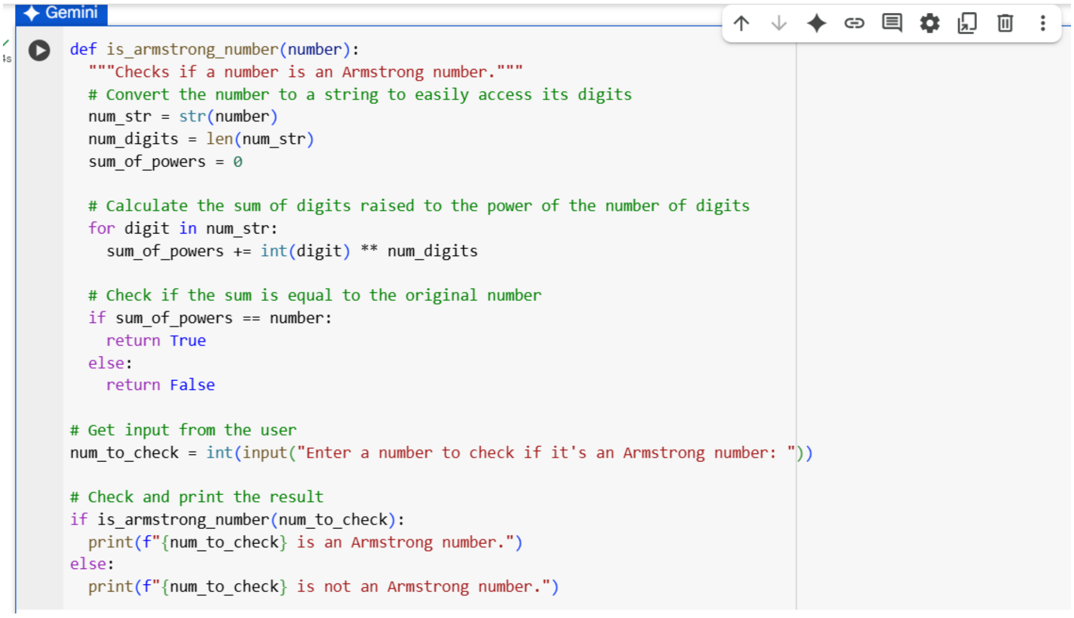
Ask Gemini to write a program that checks whether a number is an Armstrong number, and then modify it using Cursor AI to improve performance or structure.

NOTE:- Instead of cursor AI, I am using GEMINI GOOGLE COLLAB.

PROMPT:

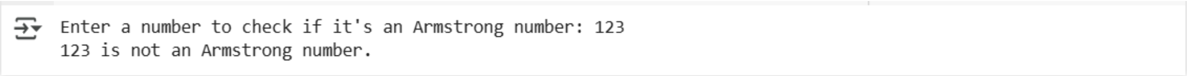
write a program that checks whether a number is an Armstrong number in simple way.

CODE GENERATED BY GEMINI AI:-



```
def is_armstrong_number(number):
  """Checks if a number is an Armstrong number."""
  # Convert the number to a string to easily access its digits
  num_str = str(number)
  num_digits = len(num_str)
  sum_of_powers = 0

  # Calculate the sum of digits raised to the power of the number of digits
  for digit in num_str:
    sum_of_powers += int(digit) ** num_digits

  # Check if the sum is equal to the original number
  if sum_of_powers == number:
    return True
  else:
    return False

# Get input from the user
num_to_check = int(input("Enter a number to check if it's an Armstrong number: "))

# Check and print the result
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.")
```

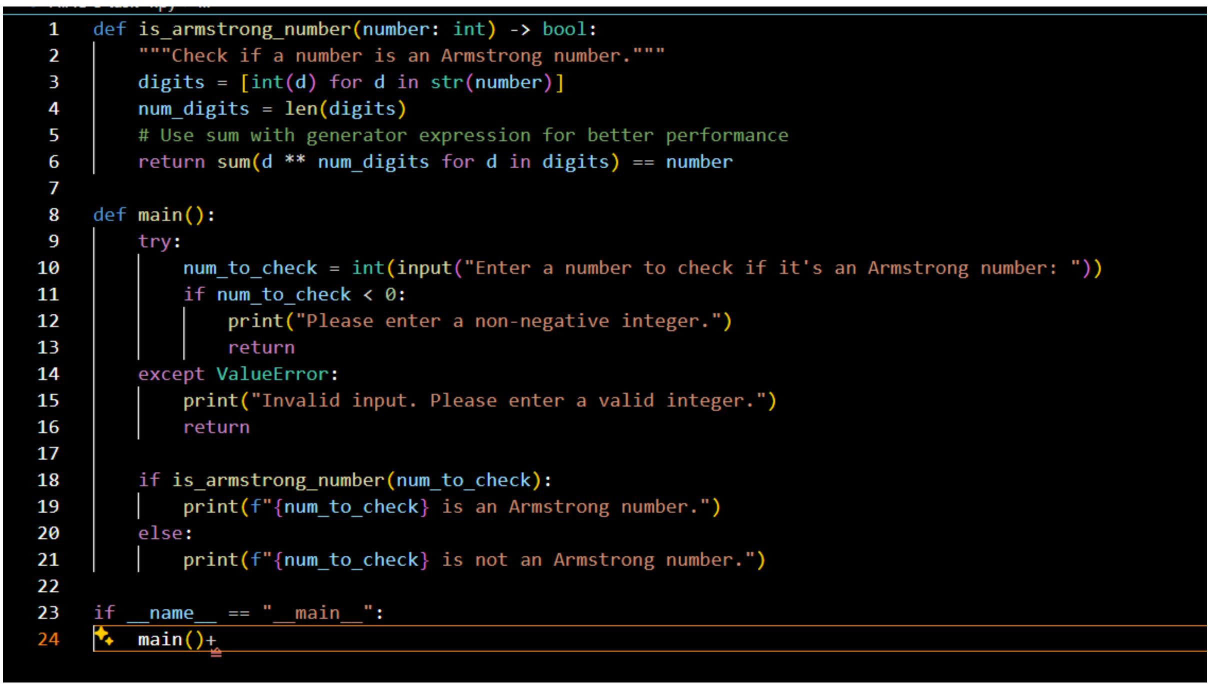
OUTPUT :-



```
Enter a number to check if it's an Armstrong number: 123
123 is not an Armstrong number.
```

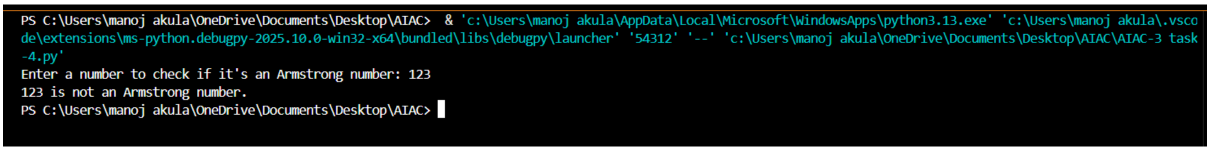
CODE GENERATED BY COPILOT-

Pasting code generated by gemini-ai and giving prompt to modifiy and improve performance and structure.



```python
def is_armstrong_number(number: int) -> bool:
    """Check if a number is an Armstrong number."""
    digits = [int(d) for d in str(number)]
    num_digits = len(digits)
    # Use sum with generator expression for better performance
    return sum(d ** num_digits for d in digits) == number

def main():
    try:
        num_to_check = int(input("Enter a number to check if it's an Armstrong number: "))
        if num_to_check < 0:
            print("Please enter a non-negative integer.")
            return
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
        return

    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.")

if __name__ == "__main__":
    main()
```

OUTPUT:-



```
PS C:\Users\manoj akula\OneDrive\Documents\Desktop\AIAC>  & 'c:\Users\manoj akula\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\manoj akula\.vsco
de\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '54312' '--' 'c:\Users\manoj akula\OneDrive\Documents\Desktop\AIAC\AIAC-3 task
-4.py'
Enter a number to check if it's an Armstrong number: 123
123 is not an Armstrong number.
PS C:\Users\manoj akula\OneDrive\Documents\Desktop\AIAC>
```

SUMMARY & CONCLUSION:-

observed that while GitHub Copilot provided an initial code structure, Gemini AI in Google Colab generated more refined and efficient code, which I preferred over the modified version from Copilot.

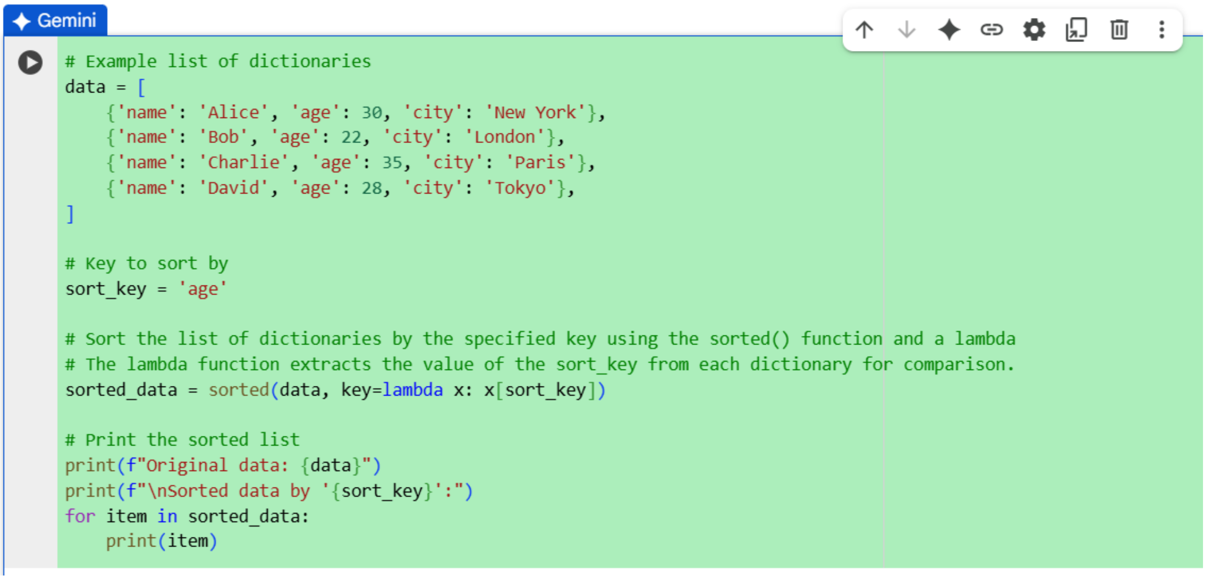
TASK DESCRIPTION 5:

Use both Gemini and Cursor AI to generate code for sorting a list of dictionaries by a specific key (e.g., age).

NOTE:- USING GITHUB COPILOT INSTEAD OF CURSOR AI.

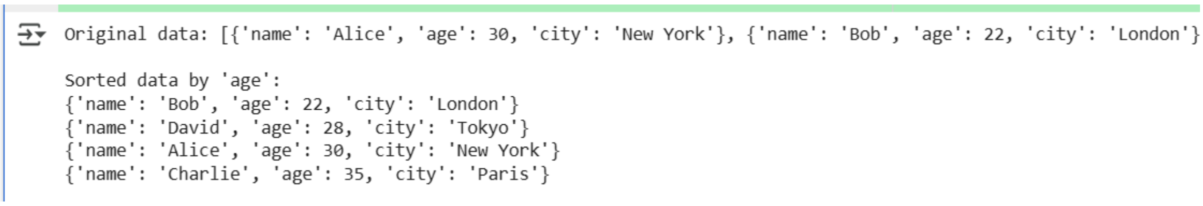
PROMPT:- generate a code for sorting a list of dictionaries by a specific key in an efficient way (e.g., age).

CODE GENERATED BY GEMINI:-



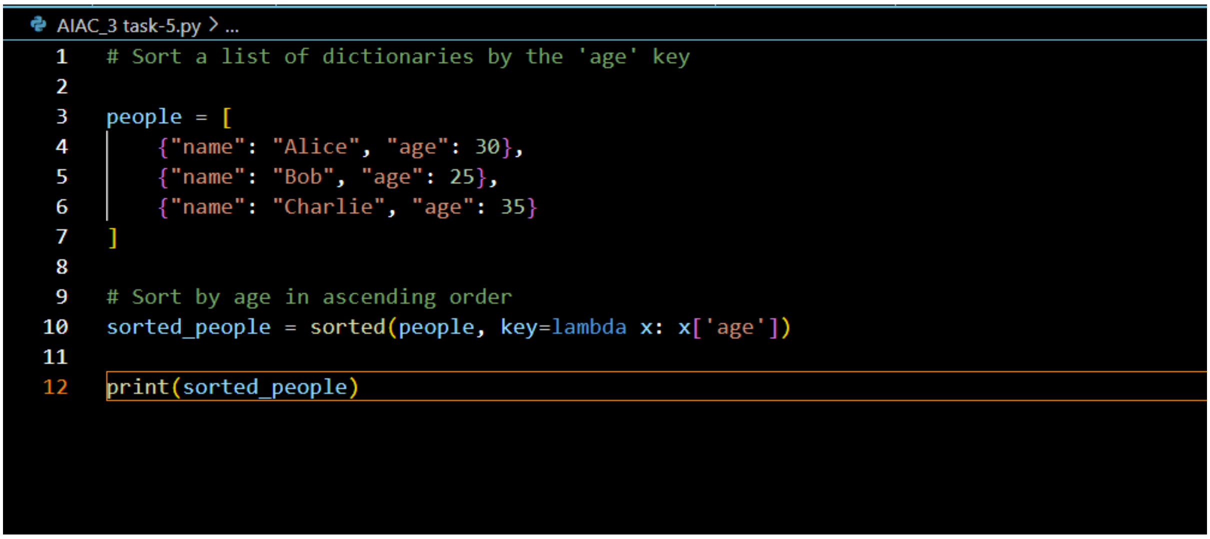
```
# Example list of dictionaries
data = [
    {'name': 'Alice', 'age': 30, 'city': 'New York'},
    {'name': 'Bob', 'age': 22, 'city': 'London'},
    {'name': 'Charlie', 'age': 35, 'city': 'Paris'},
    {'name': 'David', 'age': 28, 'city': 'Tokyo'},
]

# Key to sort by
sort_key = 'age'

# Sort the list of dictionaries by the specified key using the sorted() function and a lambda
# The lambda function extracts the value of the sort_key from each dictionary for comparison.
sorted_data = sorted(data, key=lambda x: x[sort_key])

# Print the sorted list
print(f"Original data: {data}")
print(f"\nSorted data by '{sort_key}':")
for item in sorted_data:
    print(item)
```

OUTPUT:-



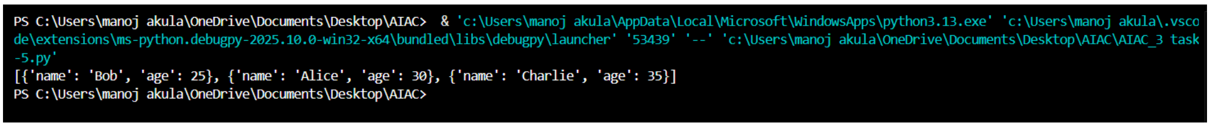
```
Original data: [{'name': 'Alice', 'age': 30, 'city': 'New York'}, {'name': 'Bob', 'age': 22, 'city': 'London'}

Sorted data by 'age':
{'name': 'Bob', 'age': 22, 'city': 'London'}
{'name': 'David', 'age': 28, 'city': 'Tokyo'}
{'name': 'Alice', 'age': 30, 'city': 'New York'}
{'name': 'Charlie', 'age': 35, 'city': 'Paris'}
```

CODE CREATED BY GIT-HUB COPILOT:-



```python
# Sort a list of dictionaries by the 'age' key

people = [
    {"name": "Alice", "age": 30},
    {"name": "Bob", "age": 25},
    {"name": "Charlie", "age": 35}
]

# Sort by age in ascending order
sorted_people = sorted(people, key=lambda x: x['age'])

print(sorted_people)
```

OUTPUT:-



```
PS C:\Users\manoj akula\OneDrive\Documents\Desktop\AIAC>  & 'c:\Users\manoj akula\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\manoj akula\.vsco
de\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '53439' '--' 'c:\Users\manoj akula\OneDrive\Documents\Desktop\AIAC\AIAC_3 task
-5.py'
[{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie', 'age': 35}]
PS C:\Users\manoj akula\OneDrive\Documents\Desktop\AIAC>
```

COMMENTRY:-

Observed that while GitHub Copilot provided better code than Gemini AI in Google Colab .Generated more refined and efficient code by Copilon.

CONCLUSION:- Through this activity, I gained valuable insights into how AI can be effectively leveraged for coding tasks, and my opinion is github copilot provideds small and efficient code than gemini-ai while also reinforcing my understanding of systematic code development.

THANK YOU