LAB ASSIGNEMENT-2.2

< AI Assisted coding >

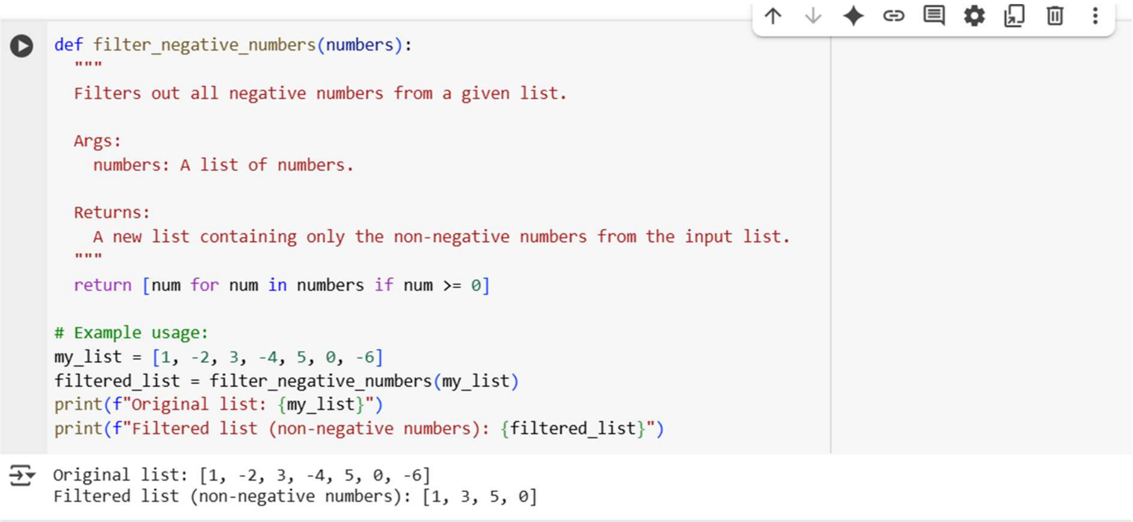
✓ Task Description#1

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

- **Expected Output:**

Functional code with before/after input and output shown in Colab, plus a screenshot.

- **Prompt :** Generate the code of python function that filters out all negative numbers .



```python
def filter_negative_numbers(numbers):
  """
  Filters out all negative numbers from a given list.

  Args:
    numbers: A list of numbers.

  Returns:
    A new list containing only the non-negative numbers from the input list.
  """
  return [num for num in numbers if num >= 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 (non-negative numbers): {filtered_list}")
```

```
Original list: [1, -2, 3, -4, 5, 0, -6]
Filtered list (non-negative numbers): [1, 3, 5, 0]
```

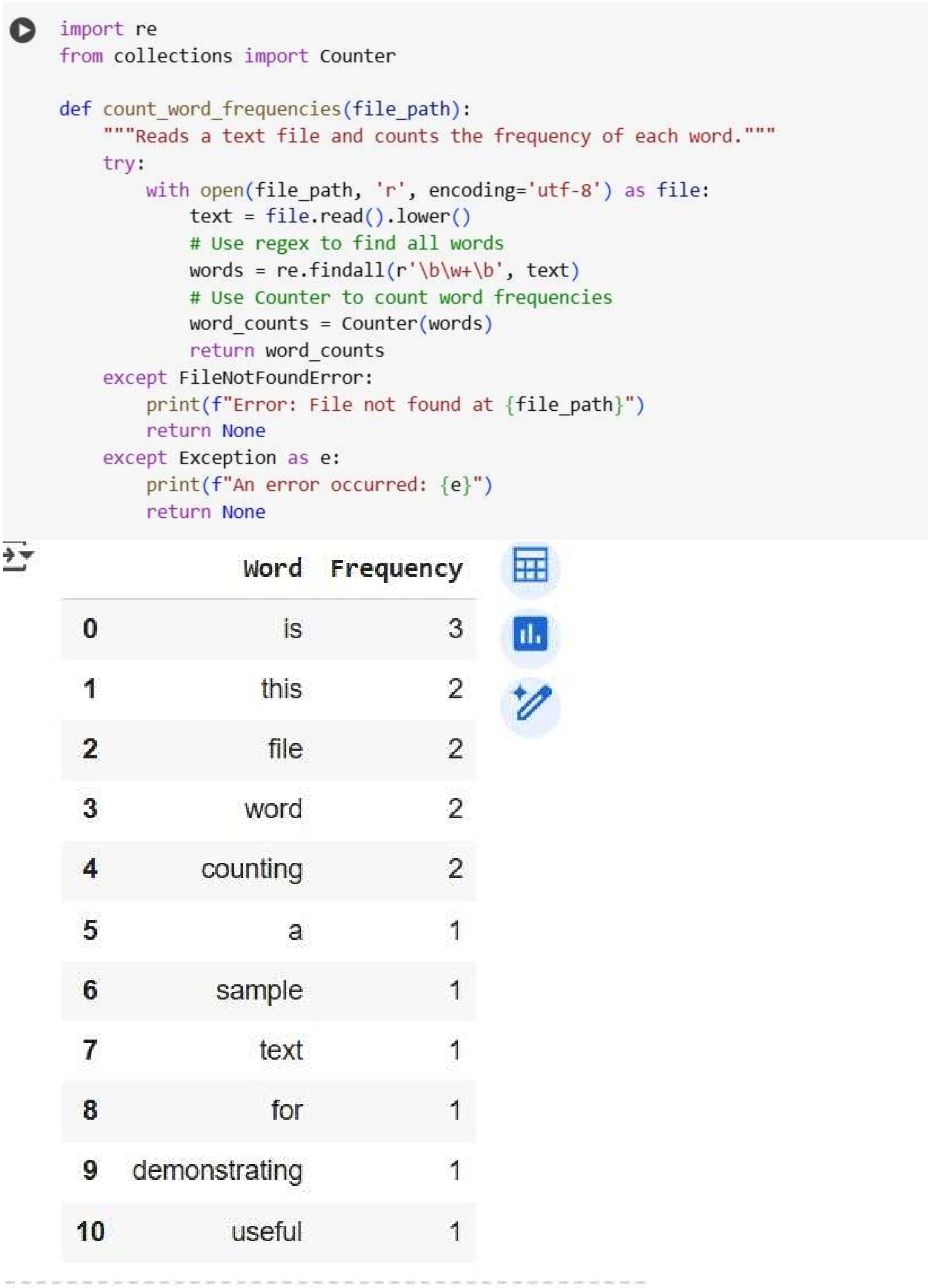
- **Observation:**
  - Gemini understands the task clearly and uses list comprehension, which is both concise and Pythonic.
  - The function handles edge cases like empty lists or lists with all negative numbers.
  - The output is accurate and matches the example provided.
  - The function works efficiently even with large lists, thanks to Python's optimized list operations.

✓ 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.

- **Expected Output:** Working code and Gemini's explanation side-by-side.

- **Prompt:** Generate the python code of that counts the frequency of each word.



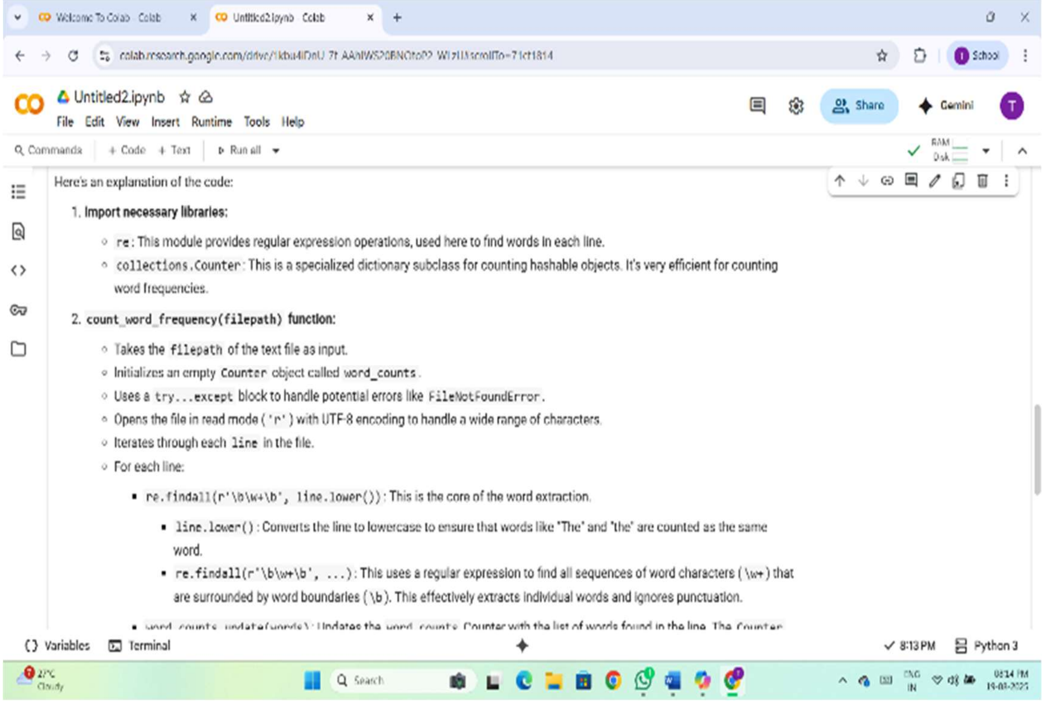
```python
import re
from collections import Counter

def count_word_frequencies(file_path):
    """Reads a text file and counts the frequency of each word."""
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            text = file.read().lower()
            # Use regex to find all words
            words = re.findall(r'\b\w+\b', text)
            # Use Counter to count word frequencies
            word_counts = Counter(words)
            return word_counts
    except FileNotFoundError:
        print(f"Error: File not found at {file_path}")
        return None
    except Exception as e:
        print(f"An error occurred: {e}")
        return None
```

	Word	Frequency
0	is	3
1	this	2
2	file	2
3	word	2
4	counting	2
5	a	1
6	sample	1
7	text	1
8	for	1
9	demonstrating	1
10	useful	1

- **Observation:**

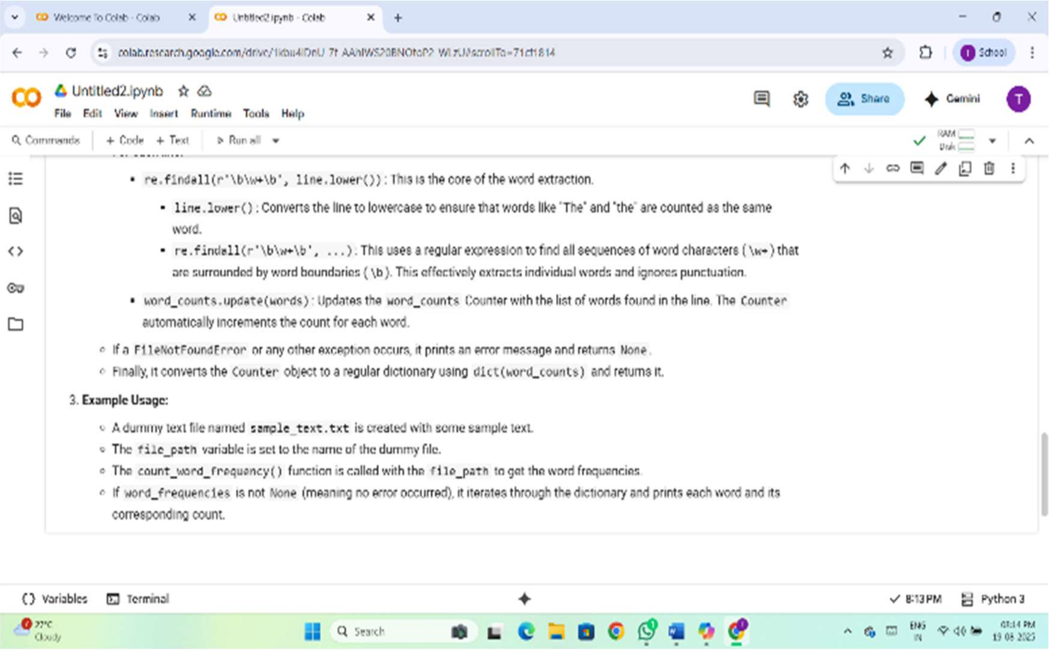
- Gemini correctly uses Python’s built-in `open()` function to read the contents of a text file, often with a context manager (`with open(...) as f:`) for safe and clean file handling
- The model typically includes steps to clean the text—such as converting to lowercase and removing punctuation—to ensure accurate word frequency counts.
- Gemini often uses a dictionary.efficiently count word occurrences, demonstrating knowledge of Python’s standard library.

➢ Explanation of code:



Here's an explanation of the code:

1. **Import necessary libraries:**
    - `re`: This module provides regular expression operations, used here to find words in each line.
    - `collections.Counter`: This is a specialized dictionary subclass for counting hashable objects. It's very efficient for counting word frequencies.
2. **`count_word_frequency(filepath)` function:**
    - Takes the `filepath` of the text file as input.
    - Initializes an empty `Counter` object called `word_counts`.
    - Uses a `try...except` block to handle potential errors like `FileNotFoundError`.
    - Opens the file in read mode (`'r'`) with UTF-8 encoding to handle a wide range of characters.
    - Iterates through each `line` in the file.
    - For each line:
        - `re.findall(r'\b\w+\b', line.lower())`: This is the core of the word extraction.
            - `line.lower()`: Converts the line to lowercase to ensure that words like "The" and "the" are counted as the same word.
            - `re.findall(r'\b\w+\b', ...)`: This uses a regular expression to find all sequences of word characters (`\w+`) that are surrounded by word boundaries (`\b`). This effectively extracts individual words and ignores punctuation.
        - `word_counts.update(words)`: Updates the `word_counts` Counter with the list of words found in the line. The `Counter`



- `re.findall(r'\b\w+\b', line.lower())`: This is the core of the word extraction.
  - `line.lower()`: Converts the line to lowercase to ensure that words like 'The' and 'the' are counted as the same word.
  - `re.findall(r'\b\w+\b', ...)`: This uses a regular expression to find all sequences of word characters (`\w+`) that are surrounded by word boundaries (`\b`). This effectively extracts individual words and ignores punctuation.
- `word_counts.update(words)`: Updates the `word_counts` Counter with the list of words found in the line. The `Counter` automatically increments the count for each word.

- If a `FileNotFoundError` or any other exception occurs, it prints an error message and returns `None`.
- Finally, it converts the `Counter` object to a regular dictionary using `dict(word_counts)` and returns it.

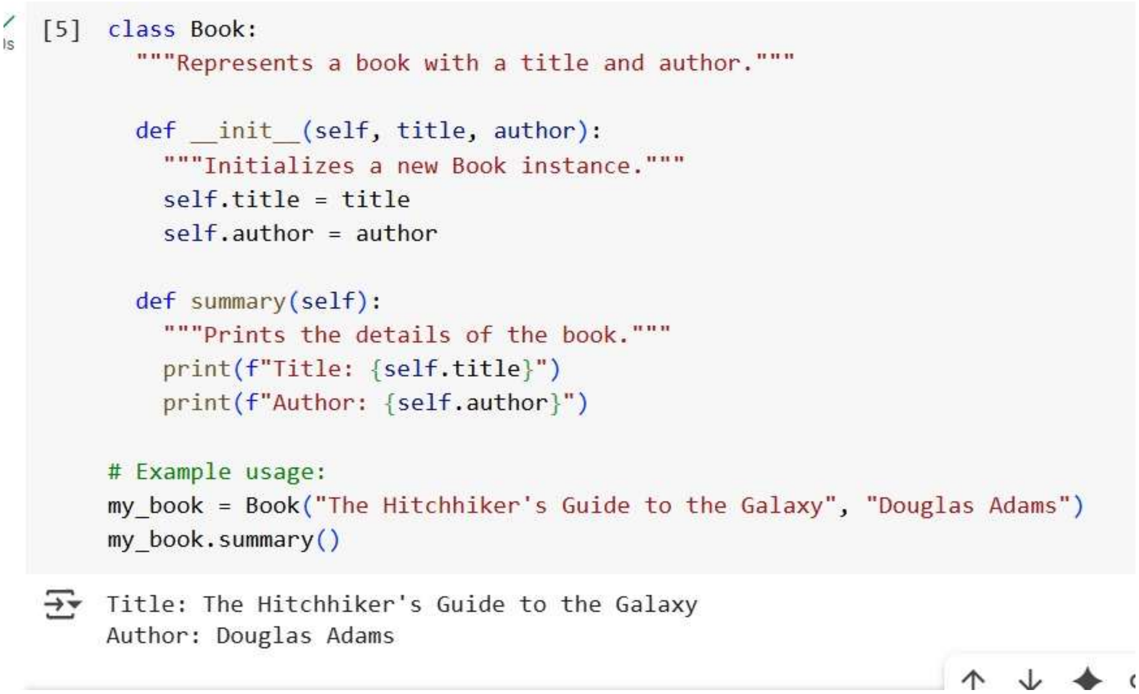
3. **Example Usage:**
   - A dummy text file named `sample_text.txt` is created with some sample text.
   - The `file_path` variable is set to the name of the dummy file.
   - The `count_word_frequency()` function is called with the `file_path` to get the word frequencies.
   - If `word_frequencies` is not `None` (meaning no error occurred), it iterates through the dictionary and prints each word and its corresponding count.

✓ 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.

- **Expected Output#3:** Screenshot comparisons and student commentary on code clarity and performance.

- **Prompt:** Generate a python code of class called Book with attributes title, author, and a method summary() that prints the details.



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

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

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

# Example usage:
my_book = Book("The Hitchhiker's Guide to the Galaxy", "Douglas Adams")
my_book.summary()
```

```
Title: The Hitchhiker's Guide to the Galaxy
Author: Douglas Adams
```

- **Observation:**

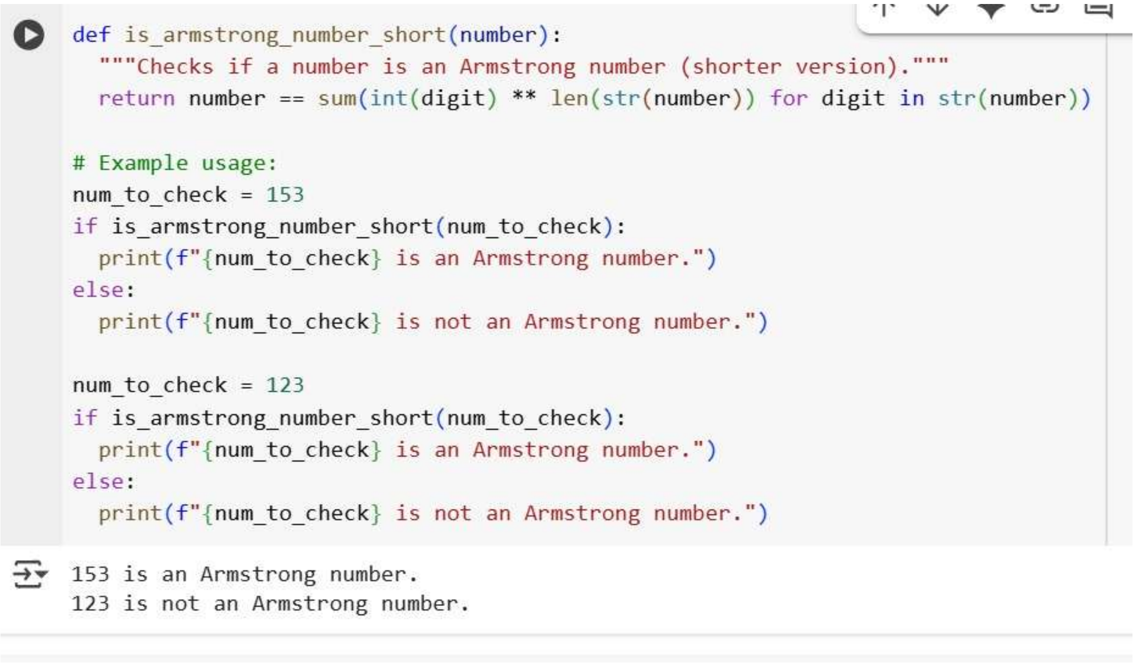
- Cursor AI correctly uses the `class` keyword to define the `Book` class, following Python's object-oriented programming conventions.
- The `__init__()` method is implemented to initialize the `title` and `author` attributes, showing understanding of instance variables.
- The code assigns `self.title` and `self.author` properly, ensuring that each object stores its own data.

✓ 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.

- **Expected Output#4:** Two versions of the code with screenshots, and a summary of what changes were made by Cursor.

- **Prompt:** Generate the python code to check whether a number is Armstrong number.



```
def is_armstrong_number_short(number):
  """Checks if a number is an Armstrong number (shorter version)."""
  return number == sum(int(digit) ** len(str(number)) for digit in str(number))

# Example usage:
num_to_check = 153
if is_armstrong_number_short(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_short(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.
```

➢ **Observation:**

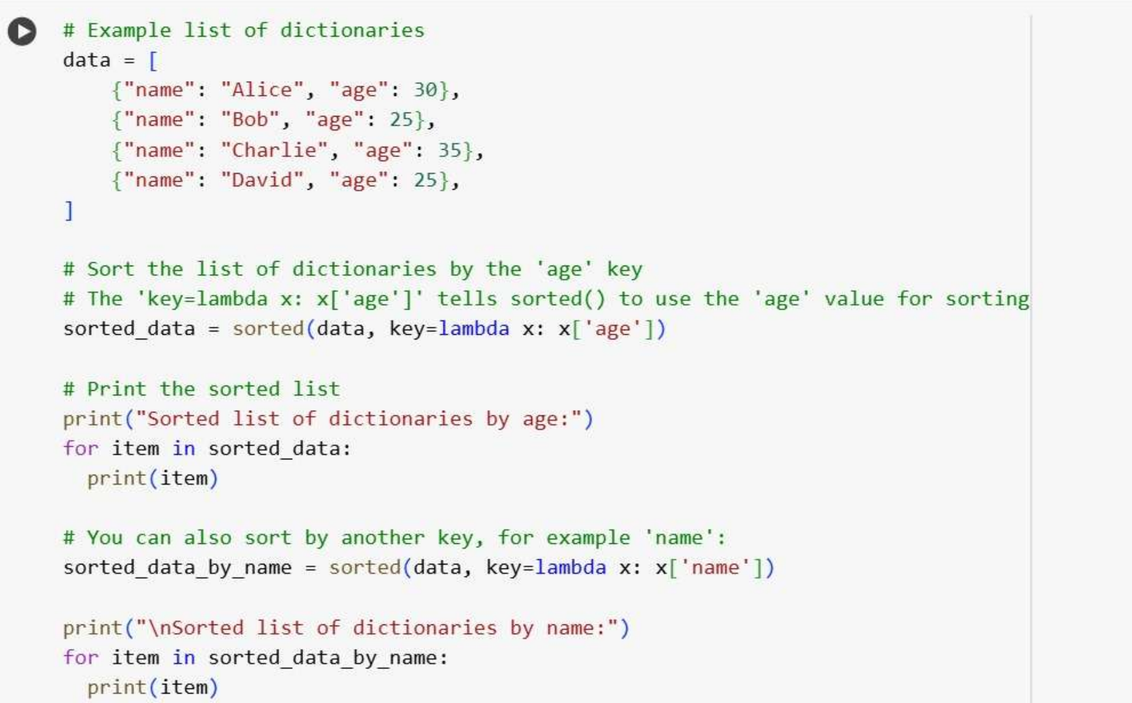
- Gemini included sample inputs like `153` or `9474` and show that the code correctly identifies them as Armstrong numbers.
- Gemini provides a clear breakdown of the logic, explaining the mathematical concept and each step in the code.
- Raising each digit to the power of the number of digits.

✓ *Task Description#5*

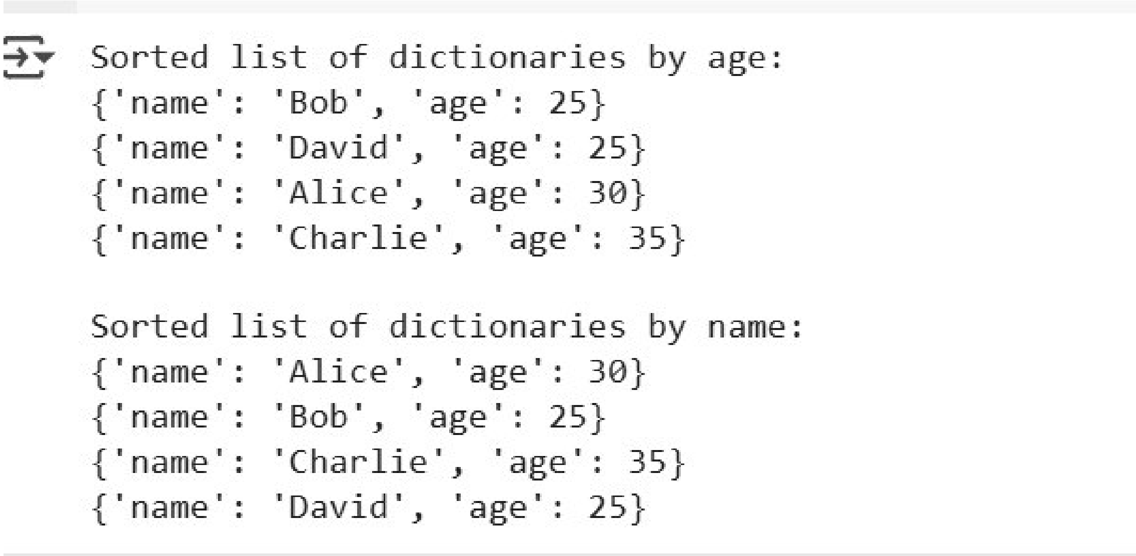
Use both Gemini and Cursor AI to generate code for sorting a list of dictionaries by a specific key (e.g., age).

➢ **Expected Output#5:** Screenshot comparisons and student commentary on code clarity and performance.

➢ **Prompt:** Generate a python code for sorting a list of dictionaries by a specific key.



```python
# Example list of dictionaries
data = [
    {"name": "Alice", "age": 30},
    {"name": "Bob", "age": 25},
    {"name": "Charlie", "age": 35},
    {"name": "David", "age": 25},
]

# Sort the list of dictionaries by the 'age' key
# The 'key=lambda x: x['age']' tells sorted() to use the 'age' value for sorting
sorted_data = sorted(data, key=lambda x: x['age'])

# Print the sorted list
print("Sorted list of dictionaries by age:")
for item in sorted_data:
  print(item)

# You can also sort by another key, for example 'name':
sorted_data_by_name = sorted(data, key=lambda x: x['name'])

print("\nSorted list of dictionaries by name:")
for item in sorted_data_by_name:
  print(item)
```



```
Sorted list of dictionaries by age:
{'name': 'Bob', 'age': 25}
{'name': 'David', 'age': 25}
{'name': 'Alice', 'age': 30}
{'name': 'Charlie', 'age': 35}

Sorted list of dictionaries by name:
{'name': 'Alice', 'age': 30}
{'name': 'Bob', 'age': 25}
{'name': 'Charlie', 'age': 35}
{'name': 'David', 'age': 25}
```

- **Observation:**

- Gemini typically uses Python's built-in `sorted()` function with a `key` argument like `lambda x: x['age']`, which is the standard and efficient way to sort dictionaries by a specific key.
- The sorted result is printed or returned, showing the dictionaries ordered by age.
- Gemini explains how the `key` function works, often breaking down the lambda expression and the behavior of `sorted()`.