***ASSIGNMENT 2.2***

***NAME: BHAWWANA SHRE***

***H.NO: 2403A52311***

***SUBJECT: AI ASSISTANT CODING***

***BATCH: 01***

*T****ASK DESCRIPTION 1:***

***QUESTION:***

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

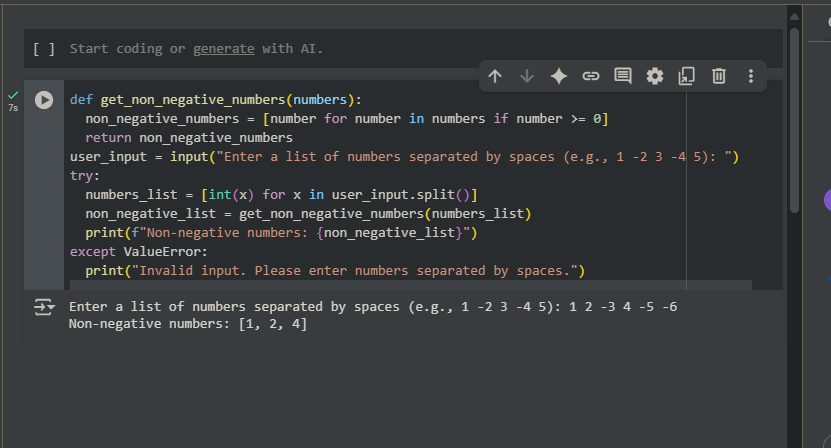
***PROMPT:***

*write a code in python in which input of list is taken from user and create a function which separates the negative numbers from list and return filtered list*

***EXPECTED OUTPUT:***

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

***CODE WITH OUTPUT:***

******

***OBSERVATION:*** *IT IS OBSERVED THAT CODE IS GENERATED WITH NO TIME AND WITH BETTER OUTPUT*

***EXPLAINATION:***

*This code takes a list of numbers entered by the user, separates them into negative and non-negative numbers, and then prints both lists.*

***TASK DESCRIPTION 2:***

***QUESTION:***

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

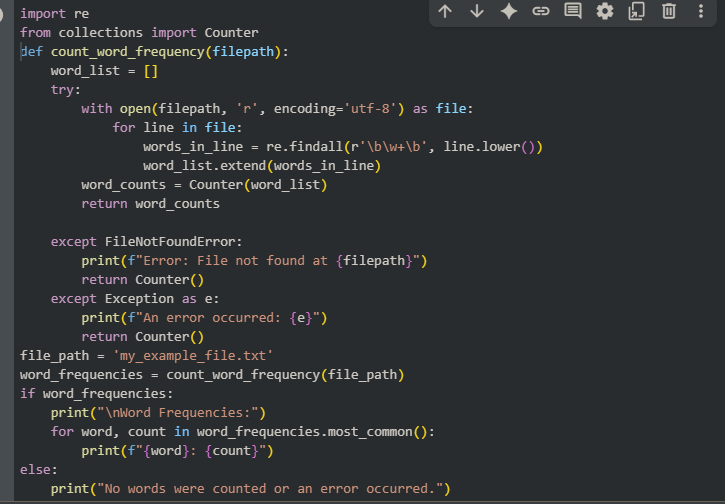
***PROMPT:***

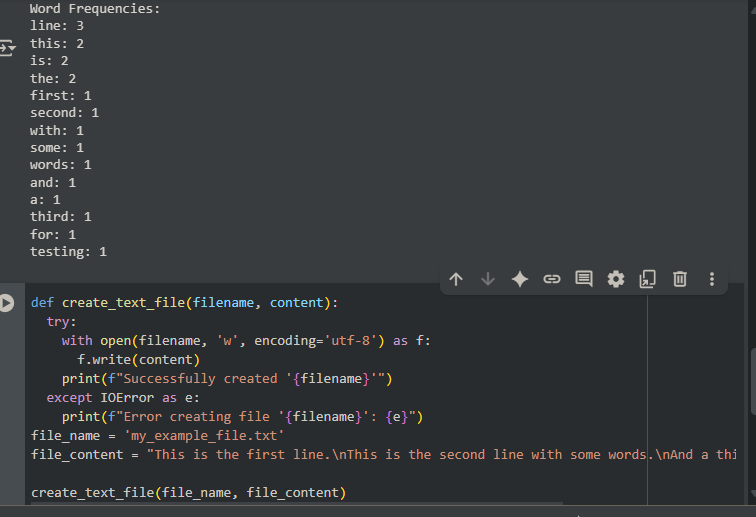
*generate code in python that reads a text file and counts the frequency of each word*

***EXPECTED OUTPUT:***

*Working code and Gemini’s explanation side-by-side*

***CODE WITH OUTPUT:***

**

**

***OBSERVATION:***

*The provided code is a good balance of simplicity and efficiency for creating a text file and counting word frequencies. It uses standard, well-optimized Python libraries and techniques. While more advanced methods exist for handling truly massive datasets, this code is efficient and perfectly suitable for many common text processing tasks.*

***EXPLAINATION:***

*In summary, this code provides a robust way to read a text file, extract words, and count how many times each word appears, handling potential file-related errors.*

***TASK DESCRIPTION 3:***

***QUESTION:***

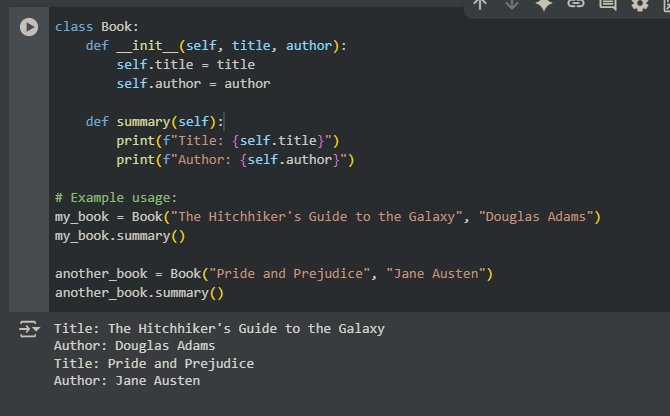
*Use Cursor AI to generate a Python class called Book with attributes title, author, and  
a method summary() that prints the details.*

***PROMPT IN GEMINI:***

*generate a code in Python class called Book with attributes title, author, and a method summary() that prints the details.*

***EXPECTED OUTPUT:***

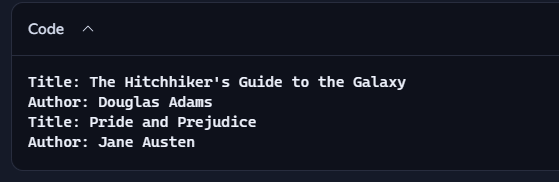
*Screenshot comparisons and student commentary on code clarity and performance*

**

***PROMPT IN COPILOT:***

*generate a code in Python class called Book with attributes title, author, and a method summary() that prints the details.*

**

**

***COMMENTRY ON CODE CLARITY AND PERFORMANCE:***

* ***Clarity:****COPILOT’s code is like a simple, well-written instruction manual. I can quickly understand how to make a "book" object and get its basic information. There are no confusing steps or jargon.*
* ***Performance:****It's like a lightning-fast little machine. When I ask it to create a book or tell you it’s summary, it does it almost instantly without breaking a sweat. It's not doing any unnecessary work.*

***EXPLAINATION****:*

* *This code defines a template for creating "Book" objects, each holding its own title and author, and provides a way to display that information.*

***TASK DESCRIPTION 4:***

***QUESTION:***

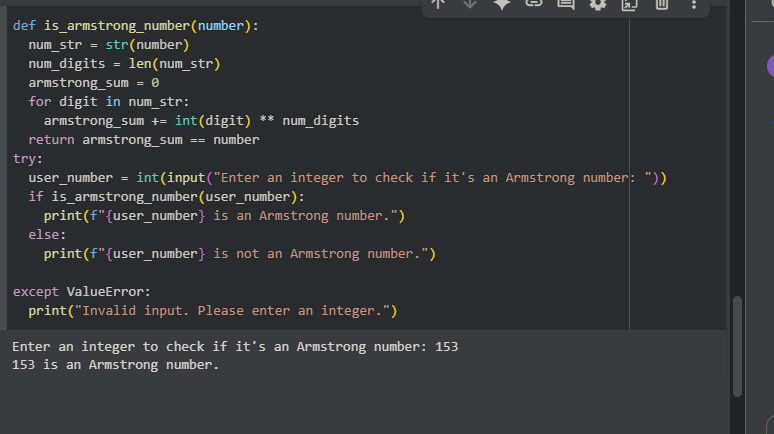
*Ask Gemini to write a program that checks whether a number is an Armstrong  
number, and then modify it using Cursor AI to improve performance or structure*

***EXPECTED OUTPUT****:*

*Two versions of the code with screenshots, and a summary of what changes were  
made by Cursor.*

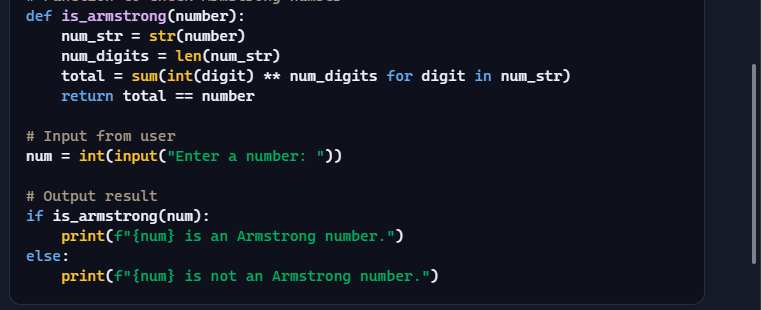
***PROMPT IN GEMINI:***

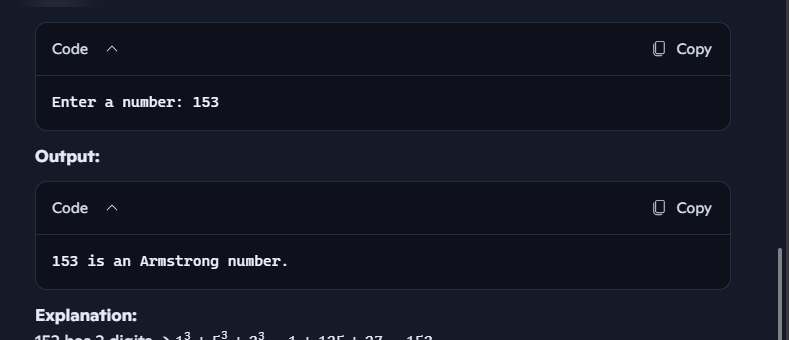
*generate a code when a number is given it should check whether a number is an Armstrong number*

**

***PROMPT IN COPILOT:***

*generate a code when a number is given it should check whether a number is an Armstrong number*

**

**

***SUMMARY:***

* ***Clarity:****COPILOT’s code is like a clear recipe for checking Armstrong numbers. Most of the steps are obvious, and even the slightly more compact calculation step is a common way to do things efficiently in Python.*
* ***Performance:****It's like a quick little calculator for this specific job. For the numbers you'd usually test, it gives you the answer almost instantly. It handles bigger numbers gracefully; it just takes a tiny bit longer as the number gets longer, but it's not like it suddenly grinds to a halt. For performance it is Gemini AI which performed in very good manner.*

***TASK DESCRIPTION 5:***

***QUESTIO****N:*

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

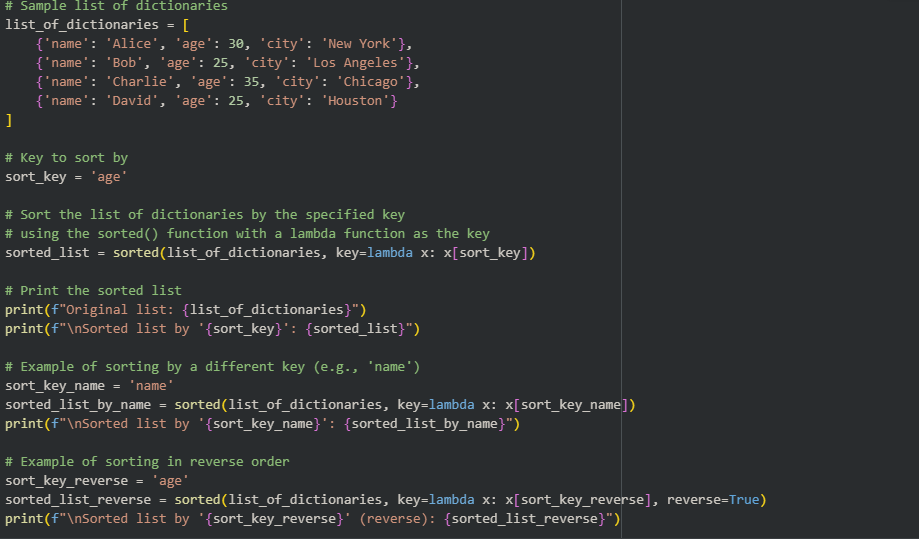
***PROMPT IN GEMINI:***

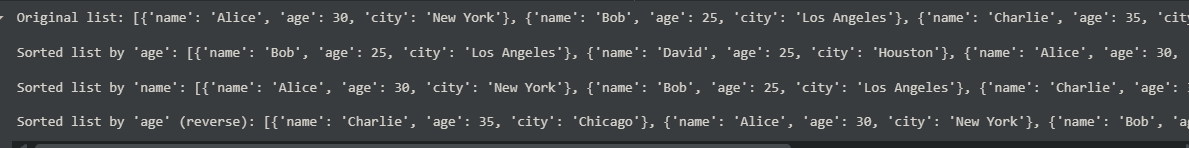
*generate a code for sorting a list of dictionaries by a specific key.*

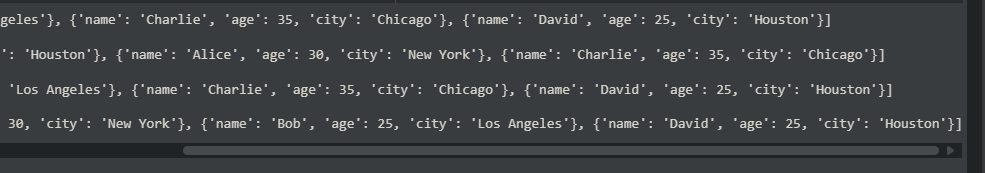
***EXPECTED OUTPUT:***

*Screenshot comparisons and student commentary on code clarity and performance*

***CODE WITH OUTPUT IN GEMINI:***

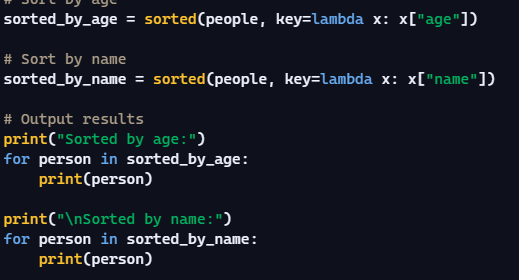
******

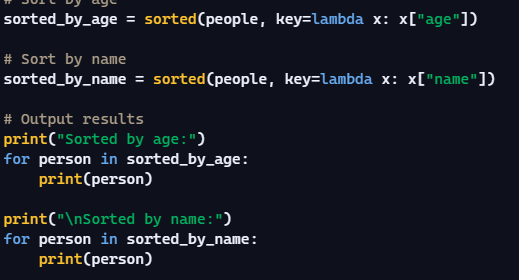
**

**

***CODE AND OUTPUT WITH COPILOT:***

**

**



***SUMMARY:***

* ***Clarity:****COPILOT is like having a simple instruction: "Take this stack of papers (your list of dictionaries) and sort them based on the age written on each paper (the 'age' key)." It's direct and easy to follow. The lambda is just a quick way to peek at the right piece of information on each paper for sorting.*
* ***Performance:****It's like having a really efficient sorting machine. I give it the stack, and it quickly arranges them in the order I asked for, whether it's by age or name. It doesn't waste time and can handle quite a few papers before slowing down noticeably.*