AI ASSISTED CODING

**LAB ASSIGNMENT – 2.2**

**NAME :** YUVARAJ REGULA

**HALL.NO :** 2403A52386

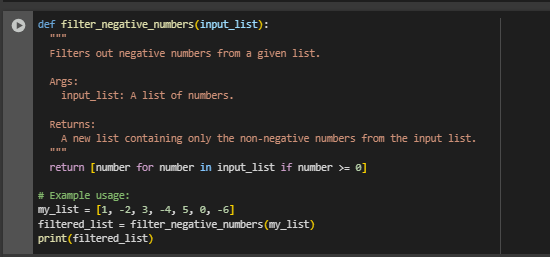
**BATCH.NO :** AIML-14

**ASSIGNMENT NO :** 2.2

**PROMPT 01 :**

i need a pyhton code , unction that filters out all negative numbers from a given list

**CODE :**

****

**OUTPUT :**

****

**CODE EXPLANATION :**

 **Function Definition**: The function filter\_negative\_numbers is defined to take a list of numbers as input.

 **Docstring**: The docstring explains the purpose, input, and output of the function clearly.

 **List Comprehension**: The function uses a list comprehension to iterate through each number in the input list.

 **Condition Check**: Inside the list comprehension, it includes only numbers that are greater than or equal to 0 (i.e., non-negative).

 **Return Value**: It returns a new list that contains only the non-negative numbers.

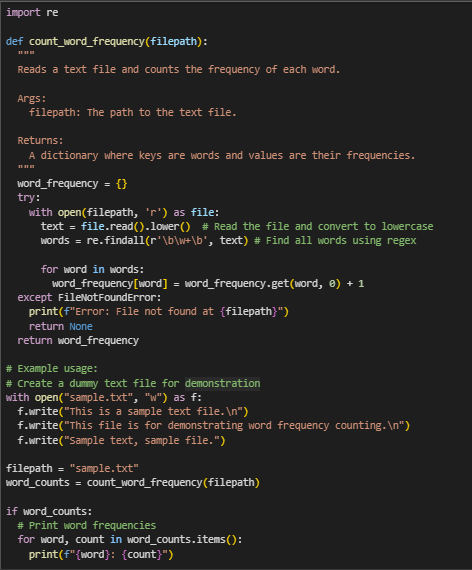
 **Example Usage**: A sample list my\_list is passed to the function, and the filtered result is stored in filtered\_list.

 **Output**: The print(filtered\_list) statement displays the result: [1, 3, 5, 0].

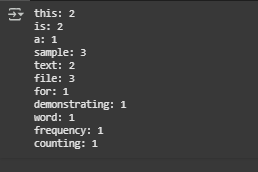
**PROMPT 02 :**

**I** NEED A python code that reads a text file and counts the frequency of each word.

**CODE :**

****

**OUTPUT :**

****

**CODE EXPLANATION :**

 **Function Definition**: The count\_word\_frequency function takes a file path as input and returns a dictionary of word frequencies.

 **Docstring**: Describes the function’s purpose, input (filepath), and output (a dictionary of word counts).

 **Dictionary Initialization**: word\_frequency is initialized as an empty dictionary to store word counts.

 **File Reading**: The file is opened in read mode, and its content is read and converted to lowercase to ensure case-insensitive matching.

 **Regex Word Extraction**: re.findall(r'\b\w+\b', text) extracts all words using regular expressions, where \w+ matches word characters.

 **Word Counting**: A loop iterates through each word, using dict.get() to increment the count for each occurrence.

 **Error Handling**: A try-except block catches FileNotFoundError and prints a message if the file doesn’t exist.

 **Return Value**: The function returns the dictionary of word frequencies if successful.

 **Example Usage**:

A sample file (sample.txt) is created with some text content.

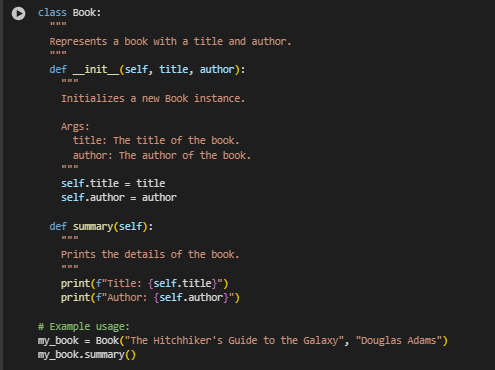
The function is called with this file’s path.

If the result is valid, the script prints the frequency of each word.

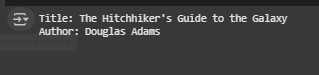
**PROMPT 03 :**

I NEED A PYHTON CODE FOR a Python class called Book with attributes title, author, and a method summary() that prints the details

**CODE :**

****

**OUTPUT :**

****

**CODE EXPLANATION :**

 **Class Definition**: The Book class is defined to represent a book with basic details like title and author.

 **Docstring**: Provides a brief description of what the class represents.

 **Constructor (\_\_init\_\_)**:

* Takes title and author as parameters.
* Initializes instance variables self.title and self.author with the given values.

 **summary() Method**:

* A method that prints the book's title and author in a formatted way.

 **Example Usage**:

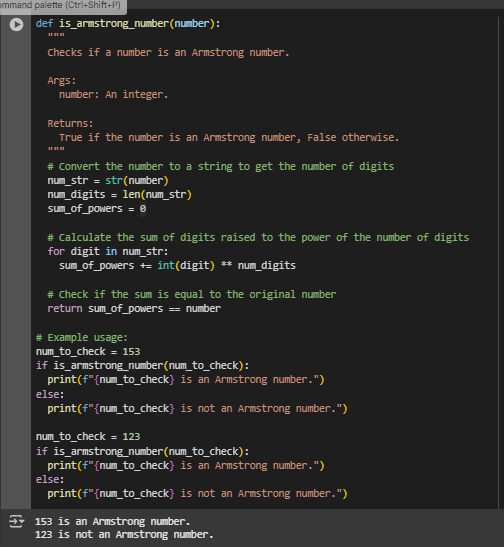
* A Book object is created with a title and author.
* The summary() method is called to display the book’s details.

**PROMPT 04 :**

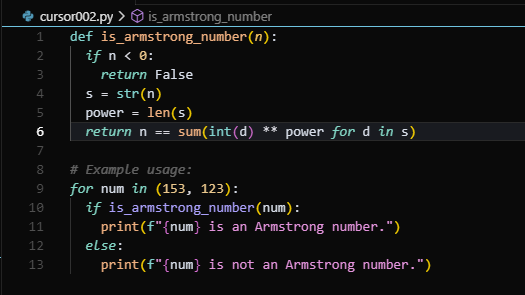
**GEMINI :** i need a code that program that checks whether a number is an Armstrong number

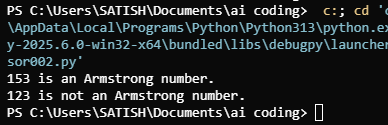
**CURSOR :** modify the given code in simple terms

**CODE & OUTPUT (GEMINI) :**

****

**CODE & OUTPUT (CURSOR) :**

****

****

**CODE EXPLANATION :**

 **Function Definition**: The is\_armstrong\_number function checks whether a given integer is an Armstrong number.

 **Docstring**: Explains the purpose, input (number), and output (True or False) of the function.

 **Digit Count**: Converts the number to a string to easily count how many digits it has.

 **Power Sum Calculation**: Uses a loop to raise each digit to the power of the total number of digits and adds the result to sum\_of\_powers.

 **Comparison**: Compares sum\_of\_powers with the original number to determine if it is an Armstrong number.

 **Return Value**: Returns True if the condition is met; otherwise, returns False.

 **Example Usage**:

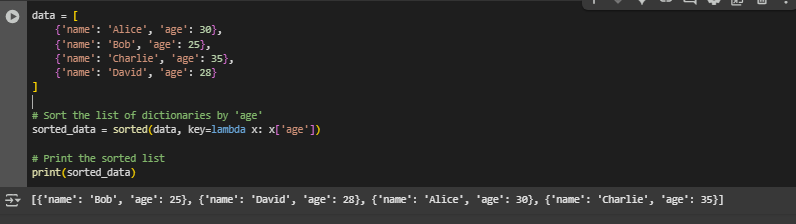
* Checks if 153 is an Armstrong number (which it is).
* Then checks 123, which is not an Armstrong number.
* Prints appropriate messages for both cases.

**PROMPT 05 :**

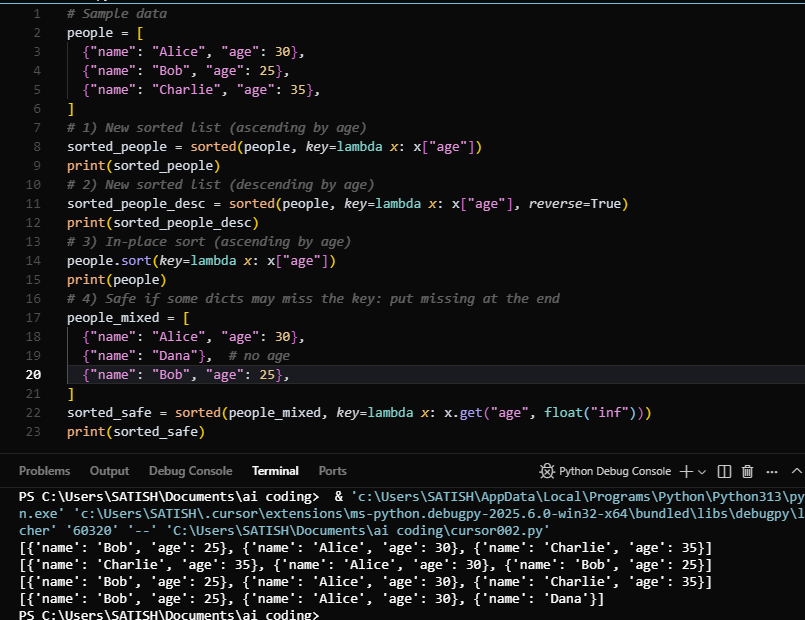
**GEMINI :** generate code for sorting a list of dictionaries by a specific key (e.g., age).

**CURSOR :** generate code for sorting a list of dictionaries by a specific key (e.g., age).

**CODE & OUTPUT (GEMINI) :**

****

**CODE & OUTPUT (CURSOR) :**

****

**CODE EXPLANATION :**

 **Data Structure**: A list named data is created, containing dictionaries. Each dictionary holds a person's name and age.

 **Sorting**:

* The built-in sorted() function is used to sort the list.
* A lambda function is passed as the key, which extracts the 'age' value from each dictionary for comparison.

 **Sorted Result**: The list is sorted in ascending order based on the 'age' field.

 **Output**: The sorted list is printed, showing people ordered from youngest to oldest.