Lab Assignment

Name : Saini.Deepthi

Hall ticket No. : 2403A52014

Batch No. : 24BTCAIAIB02

Course Title : Ai Assisted Coding

TASK -1

Description:

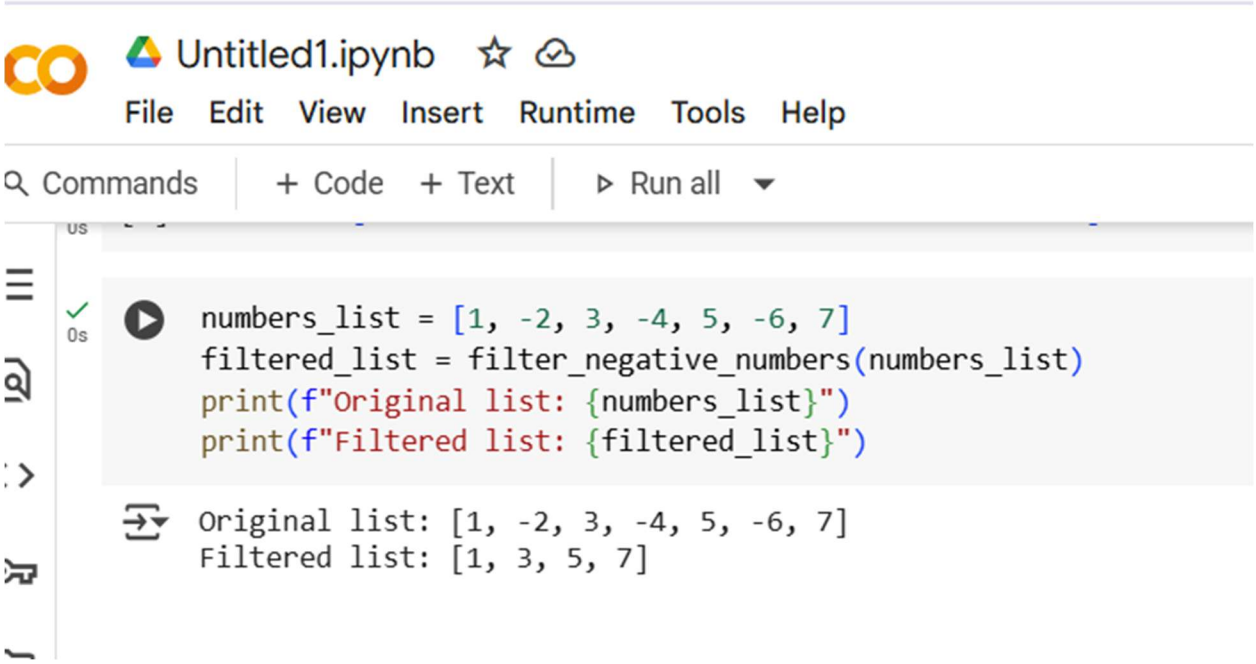
>>Use Gemini in Colab to write a function that filters out all negative numbers from a given list.

Expected Output:

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

Prompt:

>> write a code for function that filters out all negative numbers from a given list.



Obsevation:

>>The code successfully filters out the negative numbers from the numbers_list, resulting in a filtered_list containing only the non-negative numbers: [1, 3, 5, 7]. This shows that the filter_negative_numbers function is working as expected.

Explanation:

>> The code in the last cell demonstrates the usage of the filter_negative_numbers function. It begins by defining a list called numbers_list that contains a mix of positive and negative integers. This list is then passed as an argument to the filter_negative_numbers function, which processes the list and returns a new list containing only the non-negative numbers. This resulting list is stored in the variable filtered_list. Finally, the code

prints both the original list (numbers_list) and the filtered list (filtered_list) to the console, clearly showing the effect of the function and confirming that it successfully removed the negative values.

TASK -2

Description:

>>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 code that reads a text file and 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

Obsevation:

>> The function reads a text file and converts text to lowercase for consistent counting.It extracts words using regular expressions, ignoring punctuation.collections.Counter efficiently tallies word frequencies.Basic error handling is included for file not found issues.The output provides a count of each unique word in the text**.**

Explanation:

>> This code has a function that reads a text file you give it. It goes through the text, turns everything into lowercase so that "Word" and "word" are counted together, and finds all the actual words. Then, it counts how many times each word shows up. If it can't find the file

or runs into a problem, it will tell you there was an error. Finally, it gives you back a list of all the words and how many times they appeared.

TASK -3

Description:

>>Use Cursor AI to generate a 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:

>> Generate a Python class called Book with attributes title, author, and a method summary() that prints the details.

```
[5]  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:

>>The Book class is a basic example of how to use object-oriented programming in Python. It has two attributes: title and author, which store the name of the book and the writer. The summary() method prints these details in a clear format. This class is easy to understand and can be expanded later to include more features like genre or publication year. It's a good starting point for learning how to create and use classes in Python.

Explanation:

>>A class in Python is like a blueprint for creating objects. In this case, the Book class helps you create book objects. Each book has a title and an author. When you create a book using this class, you give it those two details. The class also has a method called summary()—this is like a small function that prints out the book's

title and author when you call it. So, if you want to store and show book information in your program, this class makes it easy and organized.

TASK -4

Description:

>>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:

>> write a program that checks whether a number is an Armstrong number or not.

```python
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:

>> This program checks if a number is an Armstrong number by breaking it into digits, raising each digit to the power of the number of digits, and adding them up. If the result equals the original number, it’s an Armstrong number. The logic is easy to follow and works for any number of digits. It’s a good example of using loops and math in Python.

Explanation:

This program checks if a number is an Armstrong number. First, it converts the number into a string to get each digit. Then it counts how many digits there are. For each digit, it raises it to the power of the total number of digits and adds them all together. If the final sum is equal to the original number, it means the number is an Armstrong number. For example, 9474 has 4 digits, and 9^4 + 4^4 + 7^4 + 4^4 = 9474, so it's an Armstrong number.

TASK -5

Description:

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

Expected Output:

>> Screenshot comparisons and student commentary on code clarity and performance.

Prompt:

>> Generate 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:

>> This program uses the sorted() function with a lambda to sort the list of dictionaries by the "age" key. It's a clean and efficient way to organize data based on a specific value. You can change "age" to any other key like "name" or "score" to sort differently. It's a useful trick when working with structured data like records or JSON.

Explanation:

When you have a list of dictionaries in Python—each containing keys like "name" and "age"—you can sort them using the sorted() function. To tell Python which key to sort by, you use key=lambda **x:** x["age"], which means "look at the value of the "age" key in each dictionary." Python then arranges the dictionaries in order based on that value, from lowest to highest. This is useful when you want to organize data clearly, like sorting people by age, students by marks, or tasks by priority. It's a clean and readable way to handle structured data.