AI ASSISTED CODING

Program :B.tech(CSE)

Name :ANANTHA MANIDEEP

En No. :2403A52078

Batch No. :02

LAB ASSIGNMENT-02

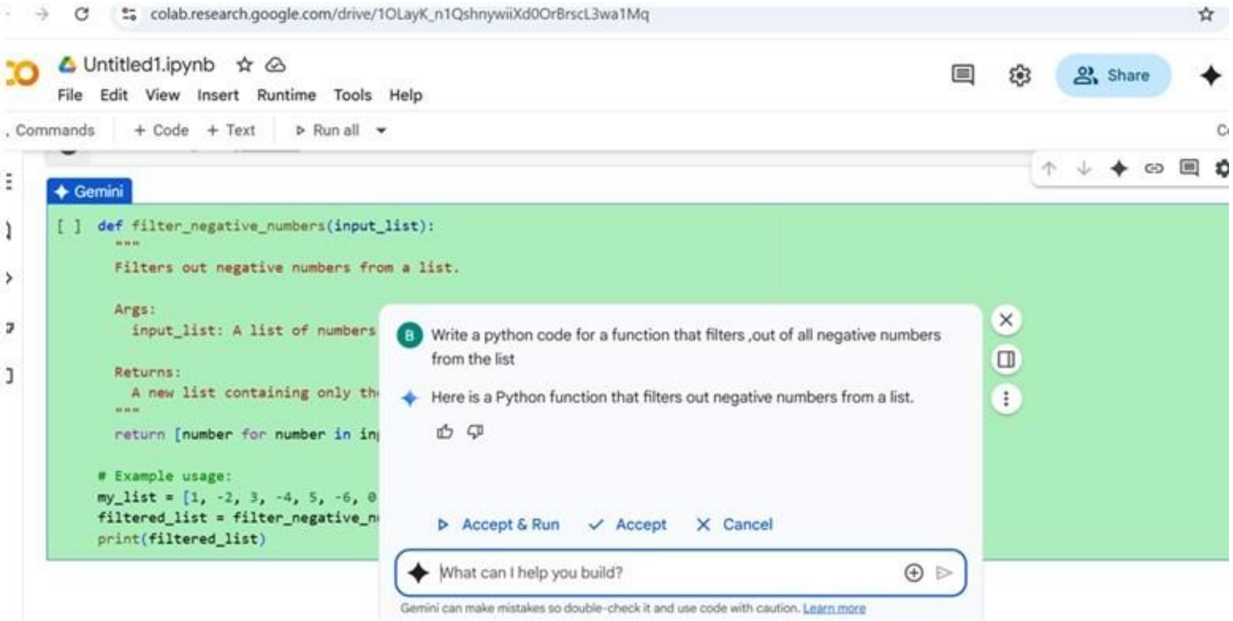
Task Description -1:

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

PROMPT:

Write a python code for a function, out of all negative numbers from the list.

Question:

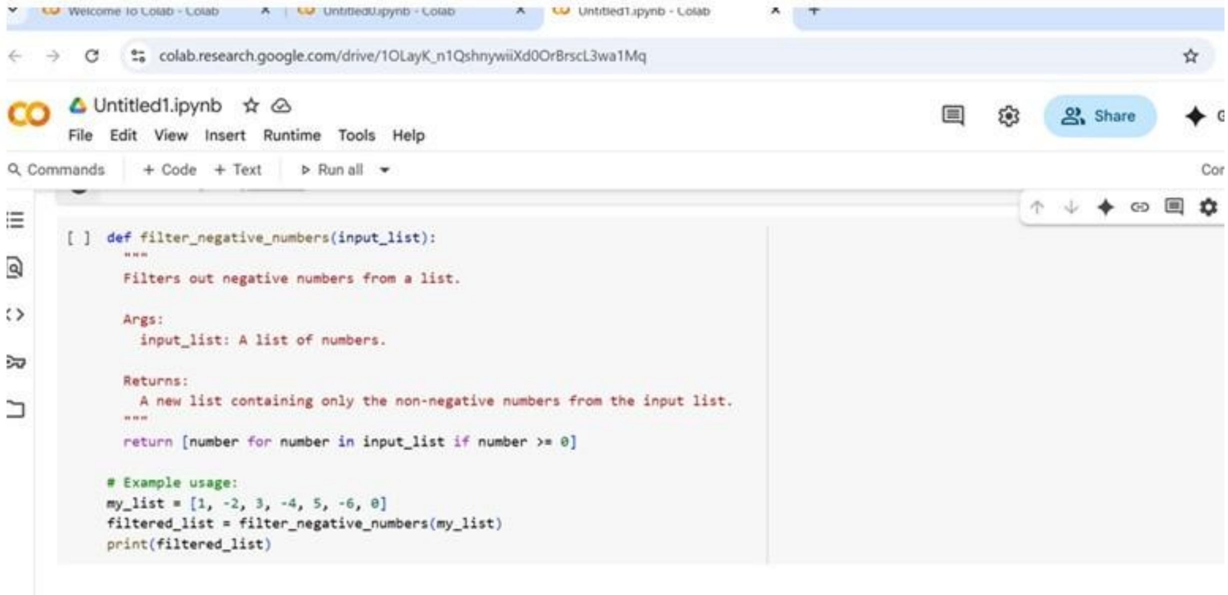

colab.research.google.com/drive/1OLayK_n1QshnywiiXd0OrBrscL3wa1Mq
Untitled1.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Share
Commands
+ Code
+ Text
Run all
Gemini
Accept & Run
Accept
Cancel
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more

```
[ ] def filter_negative_numbers(input_list):
      """
      Filters out negative numbers from a list.

      Args:
        input_list: A list of numbers

      Returns:
        A new list containing only th
      """
      return [number for number in in

    # Example usage:
    my_list = [1, -2, 3, -4, 5, -6, 0
    filtered_list = filter_negative_n
    print(filtered_list)
```

Write a python code for a function that filters ,out of all negative numbers from the list

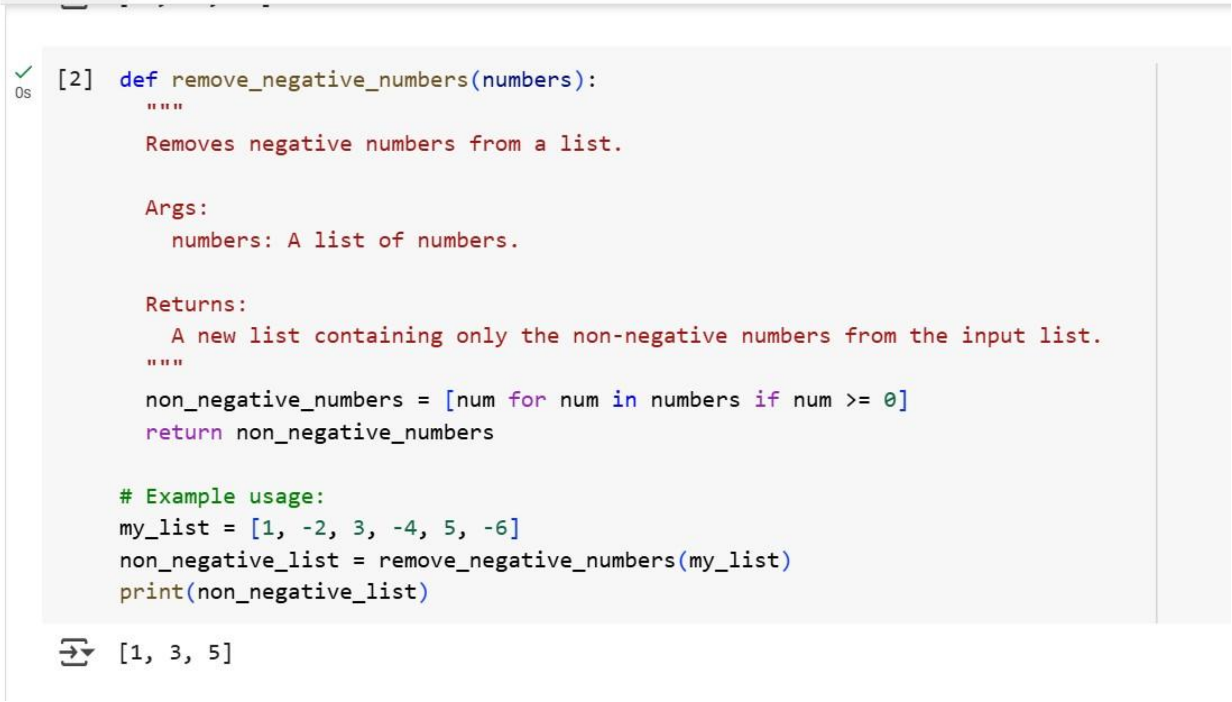
Here is a Python function that filters out negative numbers from a list.

Code:


Welcome To Colab - Colab
Untitled0.ipynb - Colab
Untitled1.ipynb - Colab
colab.research.google.com/drive/1OLayK_n1QshnywiiXd0OrBrscL3wa1Mq
Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share

```
[ ] def filter_negative_numbers(input_list):
      """
      Filters out negative numbers from a list.

      Args:
        input_list: A list of numbers.

      Returns:
        A new list containing only the non-negative numbers from the input list.
      """
      return [number for number in input_list if number >= 0]

    # Example usage:
    my_list = [1, -2, 3, -4, 5, -6, 0]
    filtered_list = filter_negative_numbers(my_list)
    print(filtered_list)
```

Output:



```
[2] def remove_negative_numbers(numbers):
      """
      Removes negative numbers from a list.

      Args:
        numbers: A list of numbers.

      Returns:
        A new list containing only the non-negative numbers from the input list.
      """
      non_negative_numbers = [num for num in numbers if num >= 0]
      return non_negative_numbers

    # Example usage:
    my_list = [1, -2, 3, -4, 5, -6]
    non_negative_list = remove_negative_numbers(my_list)
    print(non_negative_list)
```

```
[1, 3, 5]
```

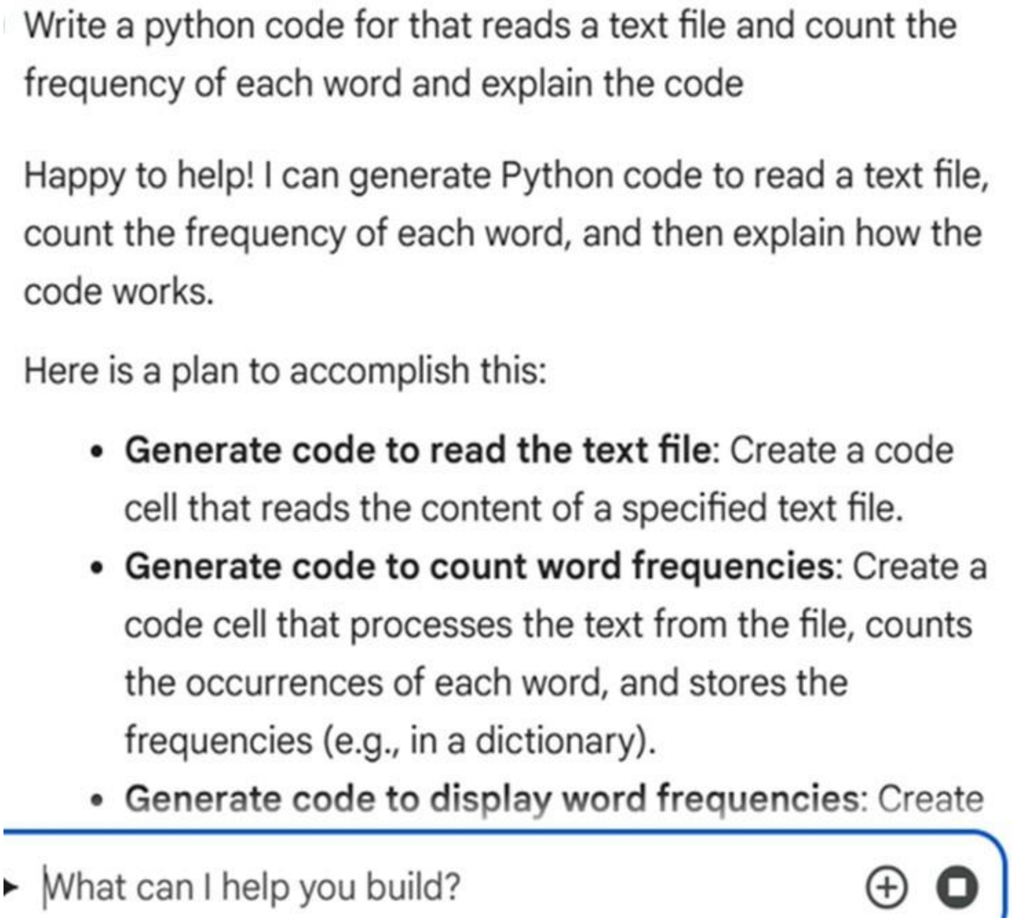
TASK Description-2:

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-2:

Write a python code that reads a text file and count the frequency of each word and explain the code.

Question-2:



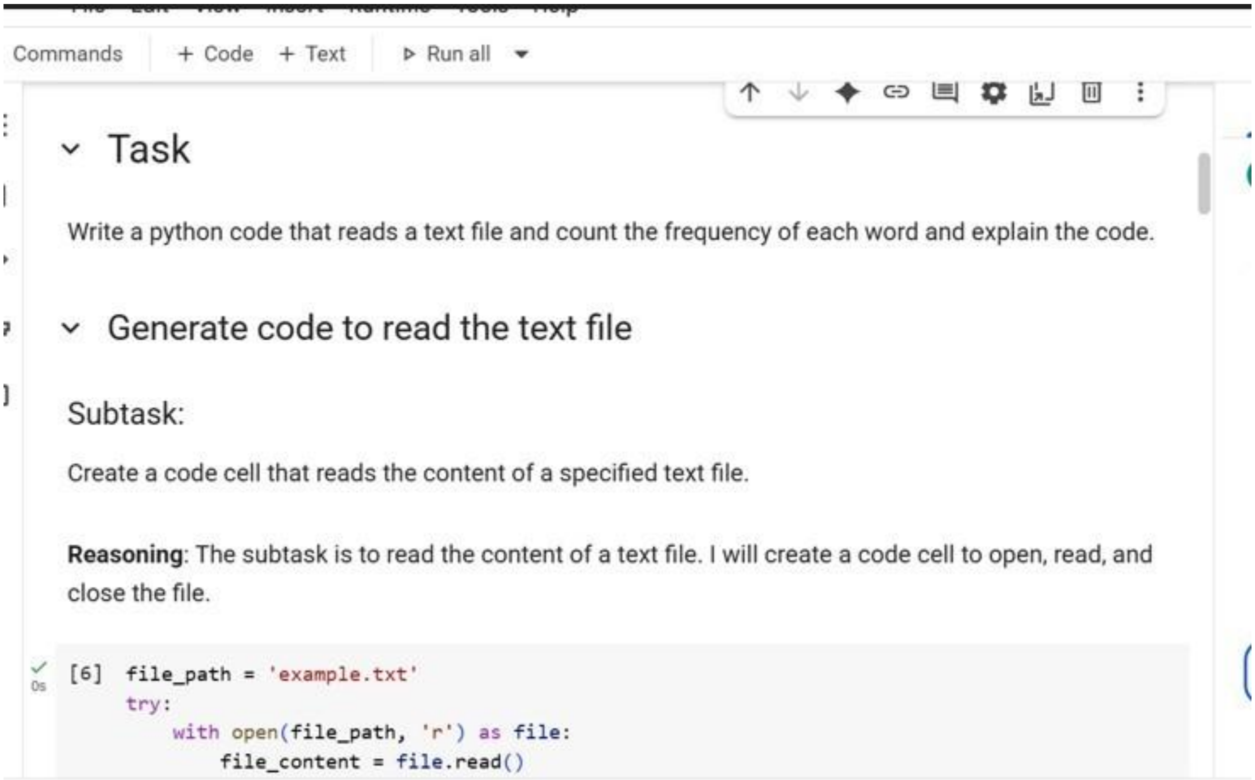
Write a python code for that reads a text file and count the frequency of each word and explain the code

Happy to help! I can generate Python code to read a text file, count the frequency of each word, and then explain how the code works.

Here is a plan to accomplish this:

- **Generate code to read the text file**: Create a code cell that reads the content of a specified text file.
- **Generate code to count word frequencies**: Create a code cell that processes the text from the file, counts the occurrences of each word, and stores the frequencies (e.g., in a dictionary).
- **Generate code to display word frequencies**: Create

Code:



Task

Write a python code that reads a text file and count the frequency of each word and explain the code.

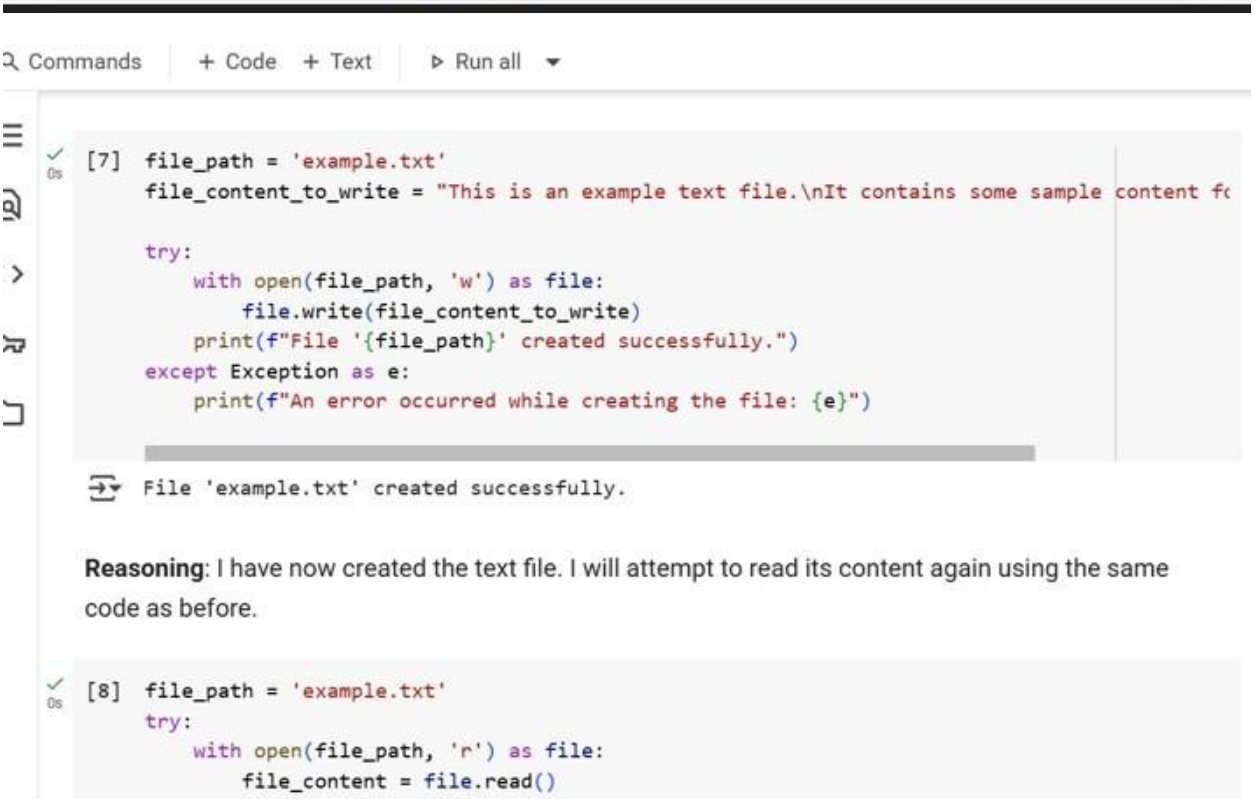
Generate code to read the text file

Subtask:

Create a code cell that reads the content of a specified text file.

Reasoning: The subtask is to read the content of a text file. I will create a code cell to open, read, and close the file.

```
[6] file_path = 'example.txt'
    try:
        with open(file_path, 'r') as file:
            file_content = file.read()
```

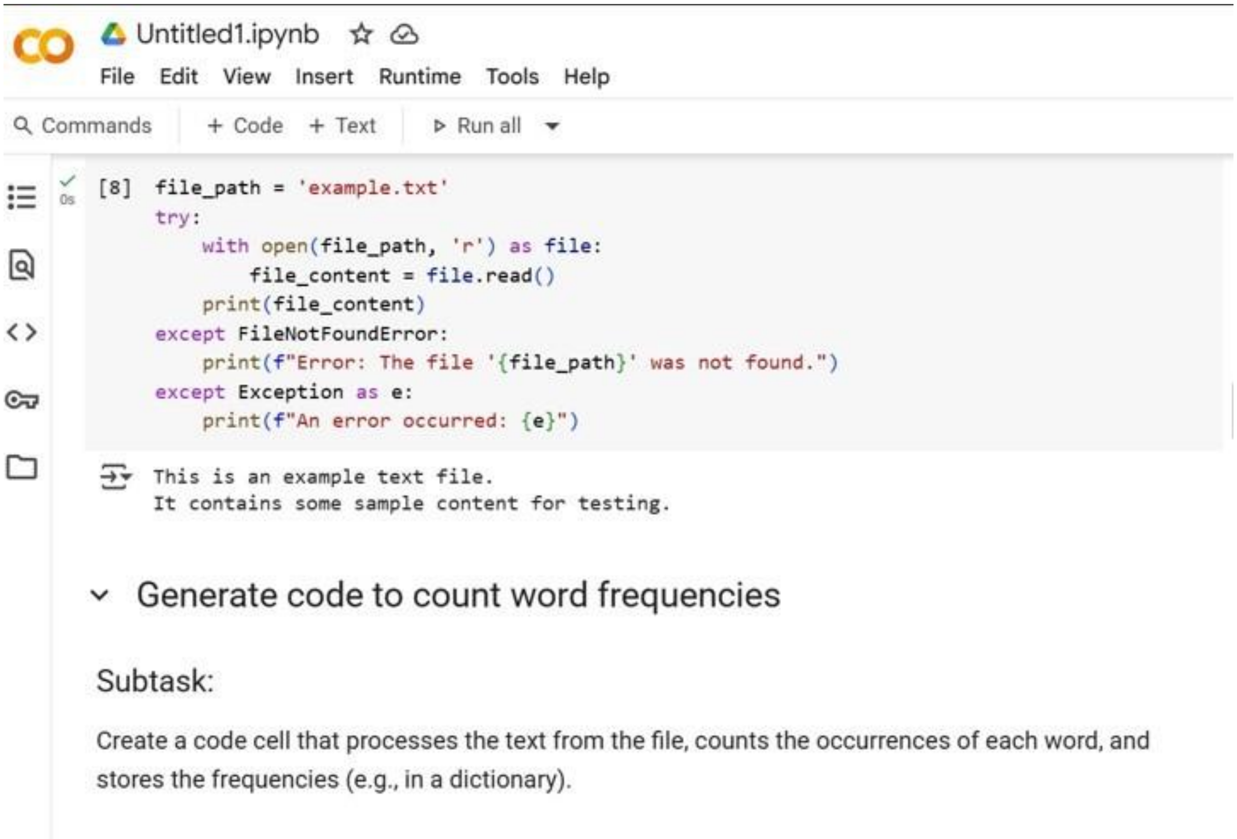


```
[7] file_path = 'example.txt'
    file_content_to_write = "This is an example text file.\nIt contains some sample content f

    try:
        with open(file_path, 'w') as file:
            file.write(file_content_to_write)
        print(f"File '{file_path}' created successfully.")
    except Exception as e:
        print(f"An error occurred while creating the file: {e}")
```

```
File 'example.txt' created successfully.
```

Reasoning: I have now created the text file. I will attempt to read its content again using the same code as before.

```
[8] file_path = 'example.txt'
    try:
        with open(file_path, 'r') as file:
            file_content = file.read()
```



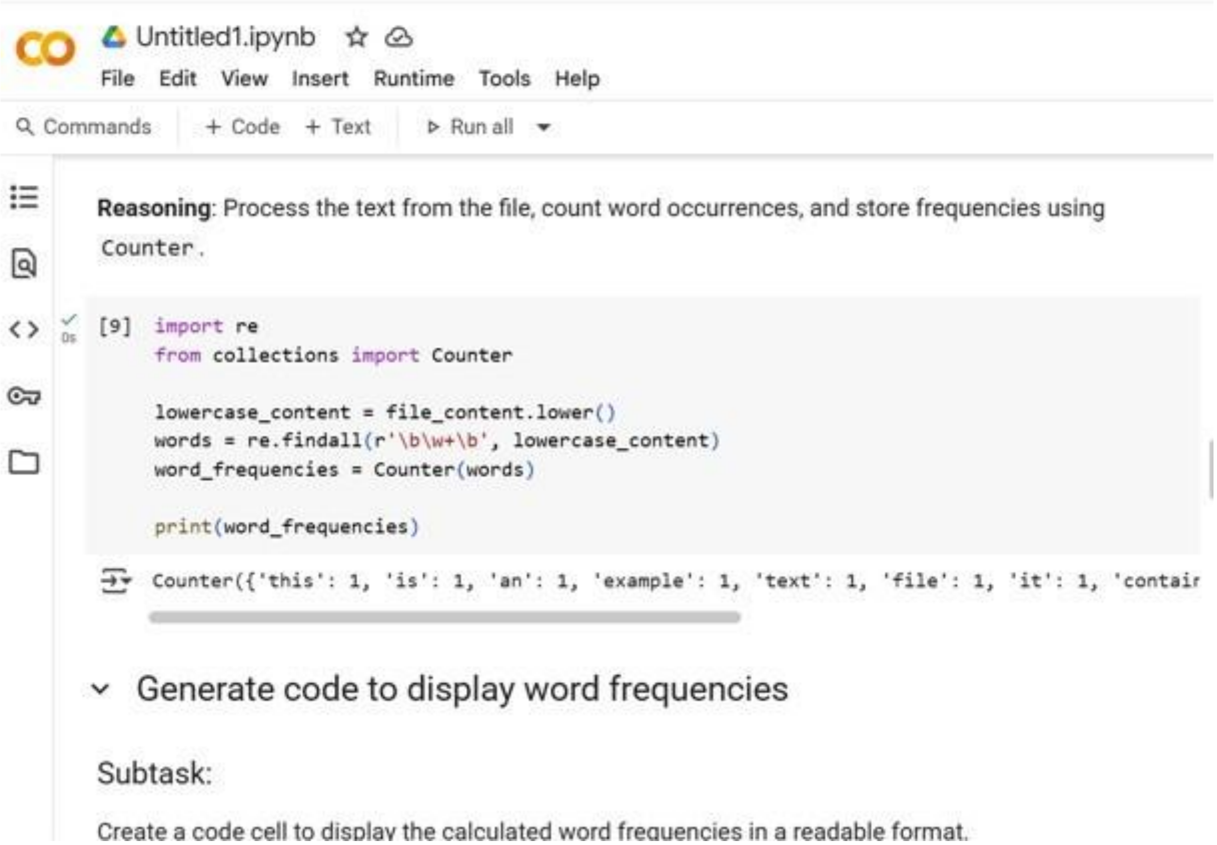
```
file_path = 'example.txt'
try:
    with open(file_path, 'r') as file:
        file_content = file.read()
    print(file_content)
except FileNotFoundError:
    print(f"Error: The file '{file_path}' was not found.")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
This is an example text file.
It contains some sample content for testing.
```

Generate code to count word frequencies

Subtask:

Create a code cell that processes the text from the file, counts the occurrences of each word, and stores the frequencies (e.g., in a dictionary).



Reasoning: Process the text from the file, count word occurrences, and store frequencies using `Counter`.

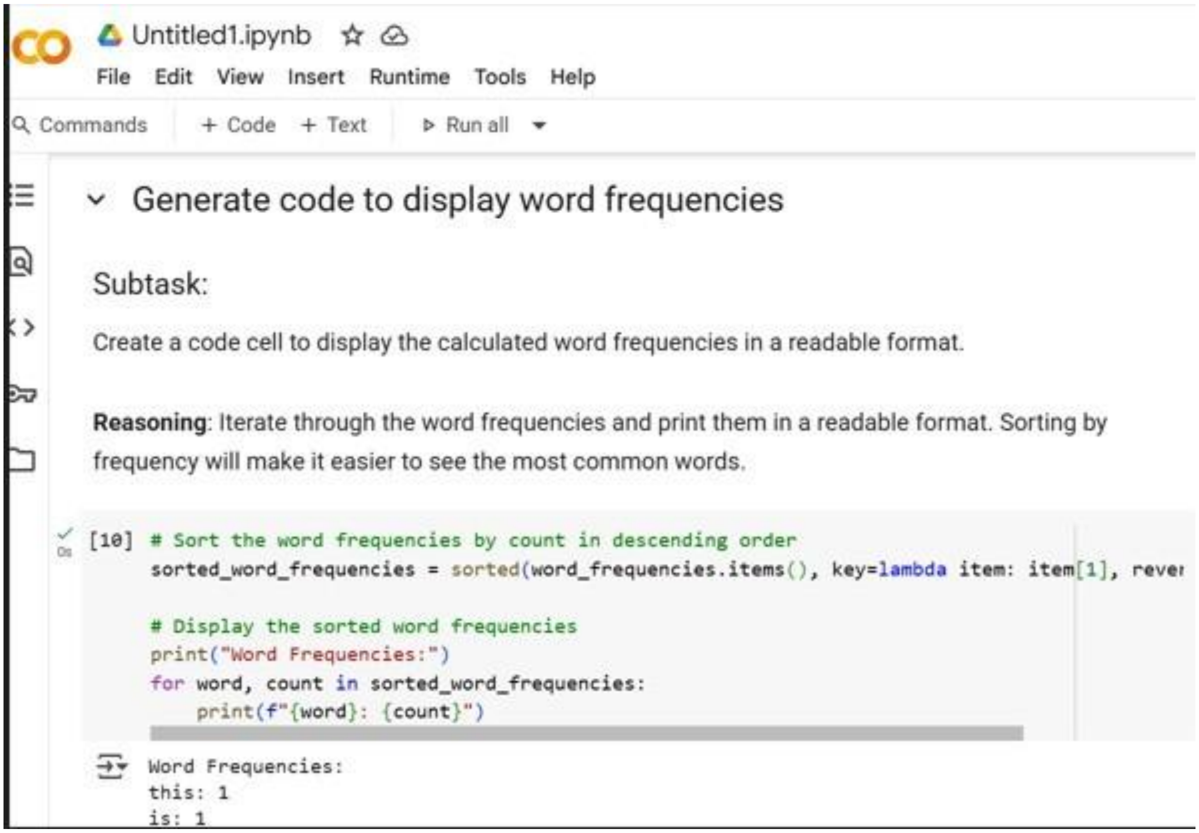
```python
import re
from collections import Counter

lowercase_content = file_content.lower()
words = re.findall(r'\b\w+\b', lowercase_content)
word_frequencies = Counter(words)

print(word_frequencies)
```

```
Counter({'this': 1, 'is': 1, 'an': 1, 'example': 1, 'text': 1, 'file': 1, 'it': 1, 'contai
```

Generate code to display word frequencies

Subtask:

Create a code cell to display the calculated word frequencies in a readable format.



Generate code to display word frequencies

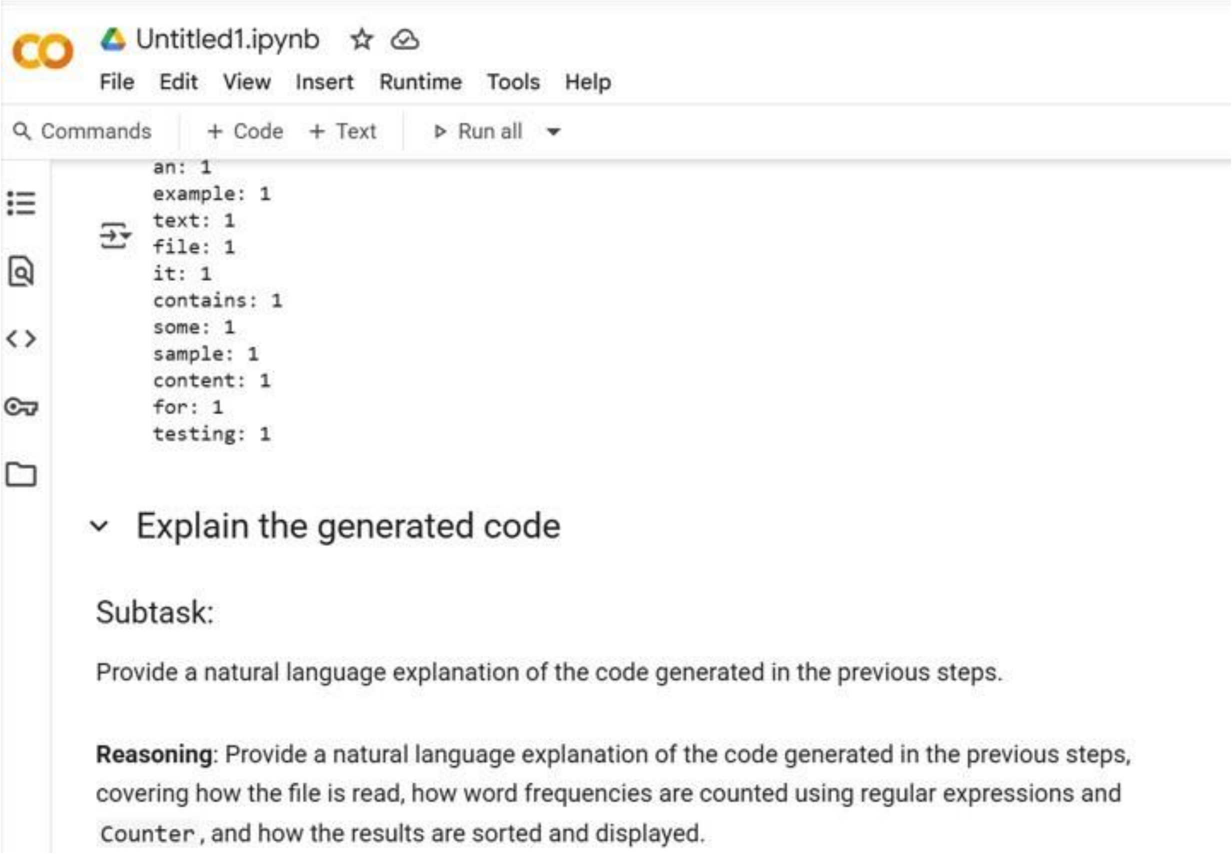
Subtask:

Create a code cell to display the calculated word frequencies in a readable format.

Reasoning: Iterate through the word frequencies and print them in a readable format. Sorting by frequency will make it easier to see the most common words.

```python
# Sort the word frequencies by count in descending order
sorted_word_frequencies = sorted(word_frequencies.items(), key=lambda item: item[1], rever

# Display the sorted word frequencies
print("Word Frequencies:")
for word, count in sorted_word_frequencies:
    print(f"{word}: {count}")
```

```
Word Frequencies:
this: 1
is: 1
```



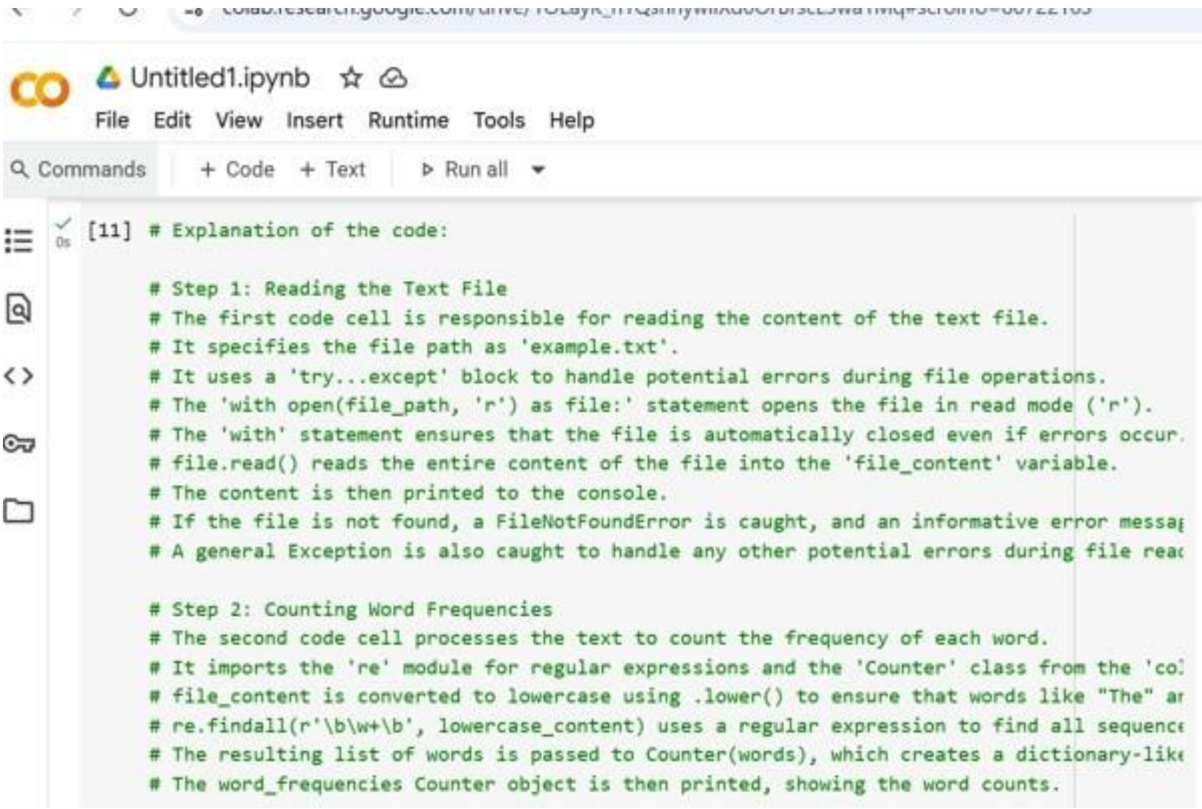
```
an: 1
example: 1
text: 1
file: 1
it: 1
contains: 1
some: 1
sample: 1
content: 1
for: 1
testing: 1
```

Explain the generated code

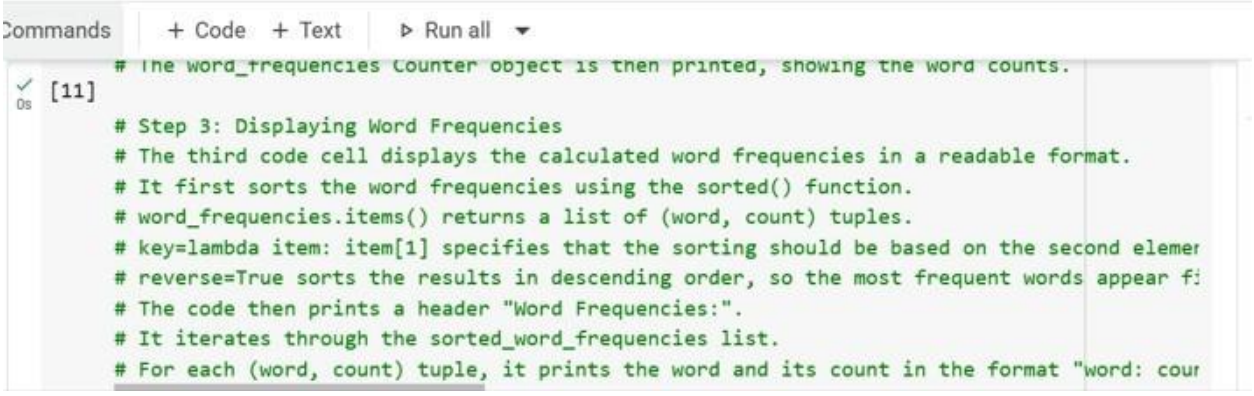
Subtask:

Provide a natural language explanation of the code generated in the previous steps.

Reasoning: Provide a natural language explanation of the code generated in the previous steps, covering how the file is read, how word frequencies are counted using regular expressions and `Counter`, and how the results are sorted and displayed.



```
# Explanation of the code:

# Step 1: Reading the Text File
# The first code cell is responsible for reading the content of the text file.
# It specifies the file path as 'example.txt'.
# It uses a 'try...except' block to handle potential errors during file operations.
# The 'with open(file_path, 'r') as file:' statement opens the file in read mode ('r').
# The 'with' statement ensures that the file is automatically closed even if errors occur.
# file.read() reads the entire content of the file into the 'file_content' variable.
# The content is then printed to the console.
# If the file is not found, a FileNotFoundError is caught, and an informative error messa
# A general Exception is also caught to handle any other potential errors during file rea

# Step 2: Counting Word Frequencies
# The second code cell processes the text to count the frequency of each word.
# It imports the 're' module for regular expressions and the 'Counter' class from the 'co
# file_content is converted to lowercase using .lower() to ensure that words like "The" ar
# re.findall(r'\b\w+\b', lowercase_content) uses a regular expression to find all sequenc
# The resulting list of words is passed to Counter(words), which creates a dictionary-lik
# The word_frequencies Counter object is then printed, showing the word counts.
```



```
# The word_frequencies Counter object is then printed, showing the word counts.

# Step 3: Displaying Word Frequencies
# The third code cell displays the calculated word frequencies in a readable format.
# It first sorts the word frequencies using the sorted() function.
# word_frequencies.items() returns a list of (word, count) tuples.
# key=lambda item: item[1] specifies that the sorting should be based on the second eleme
# reverse=True sorts the results in descending order, so the most frequent words appear fi
# The code then prints a header "Word Frequencies:".
# It iterates through the sorted_word_frequencies list.
# For each (word, count) tuple, it prints the word and its count in the format "word: cou
```

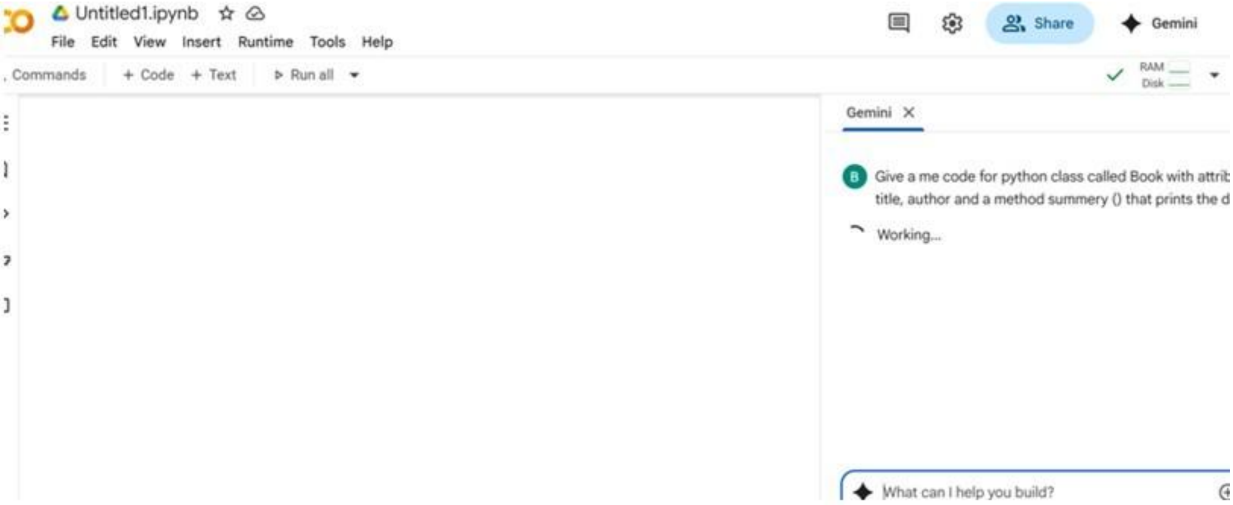
Task Desciption-3:

Use Cursor AI to generate a python class called book with attributes title, author, and a method Summery() that prints the details.

Prompt-3:

Give me a code for python class Book with attributes title, author and a method summery() that prints the details

QUESTION:

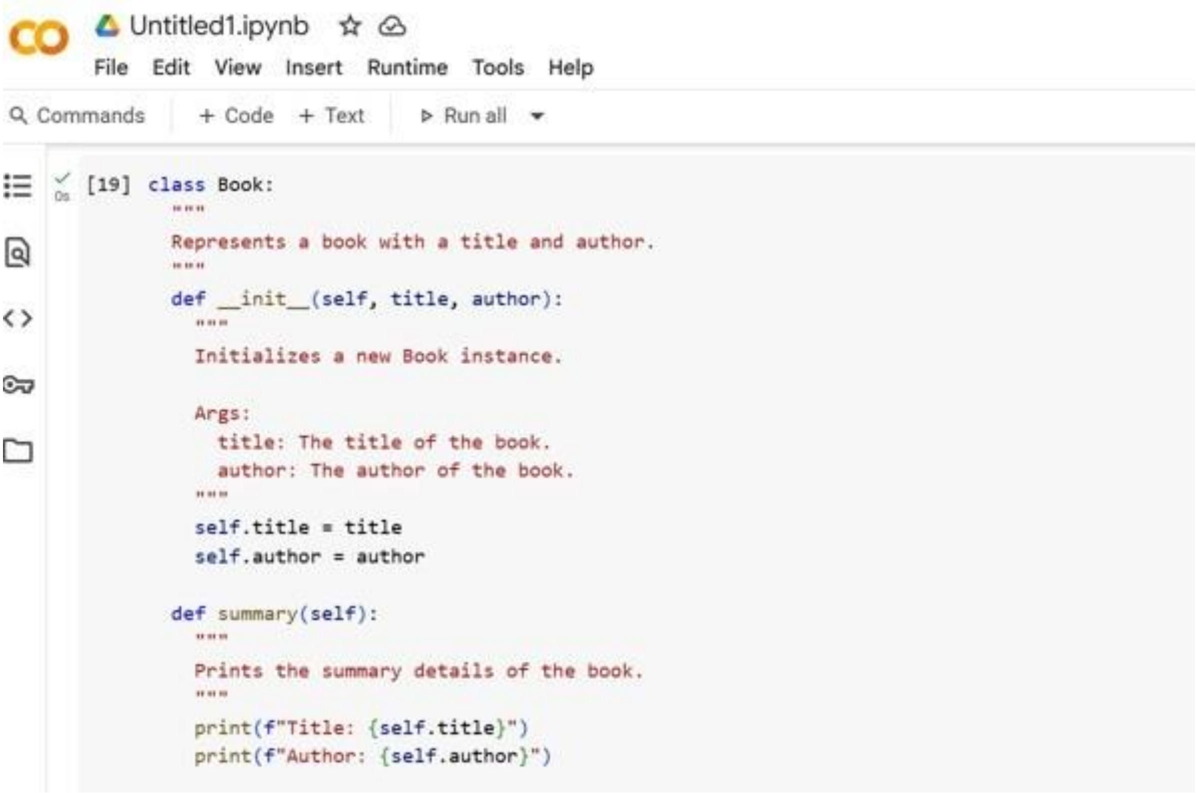


Give a me code for python class called Book with attrib
title, author and a method summery () that prints the d

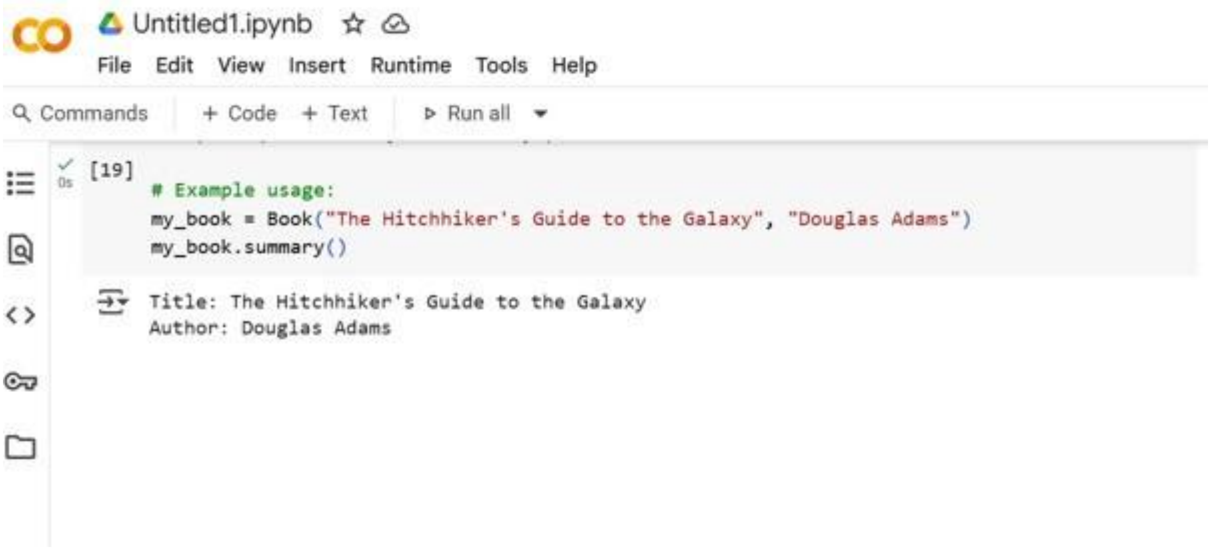
Working...

What can I help you build?

CODE:



```
[19] class Book:
       """
       Represents a book with a title and author.
       """
       def __init__(self, title, author):
         """
         Initializes a new Book instance.

         Args:
           title: The title of the book.
           author: The author of the book.
         """
         self.title = title
         self.author = author

       def summary(self):
         """
         Prints the summary 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
```

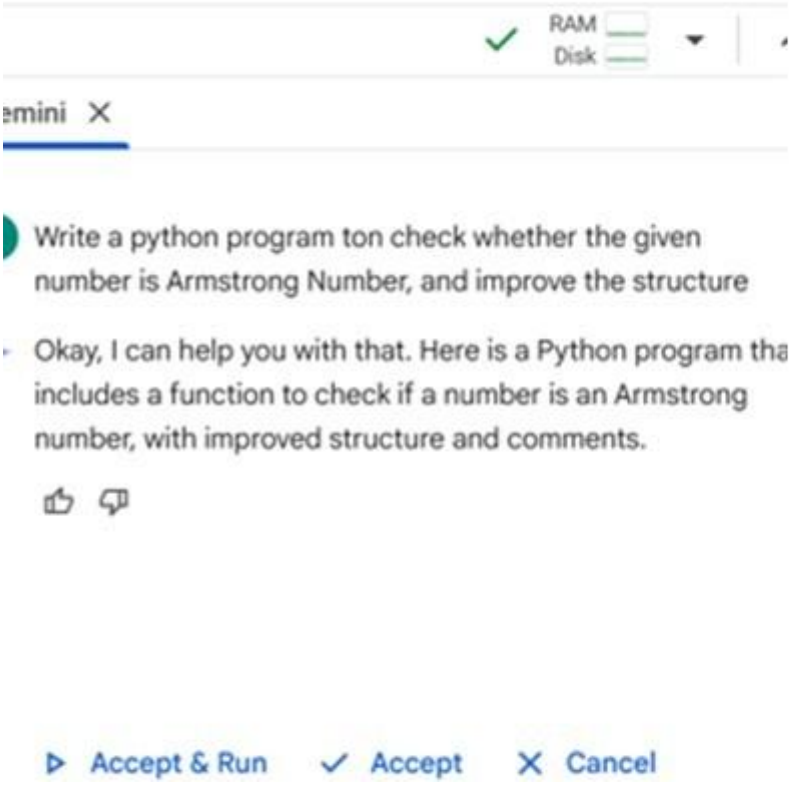
Task Description-4 :

Ask Gemini to write a program that checks whether Number is an Armstrong number ,and then modify using cursor Ai to improve performance or structure.

Prompt-4:

Write a python to check whether the given number is Armstrong Number, and improve the Structure.

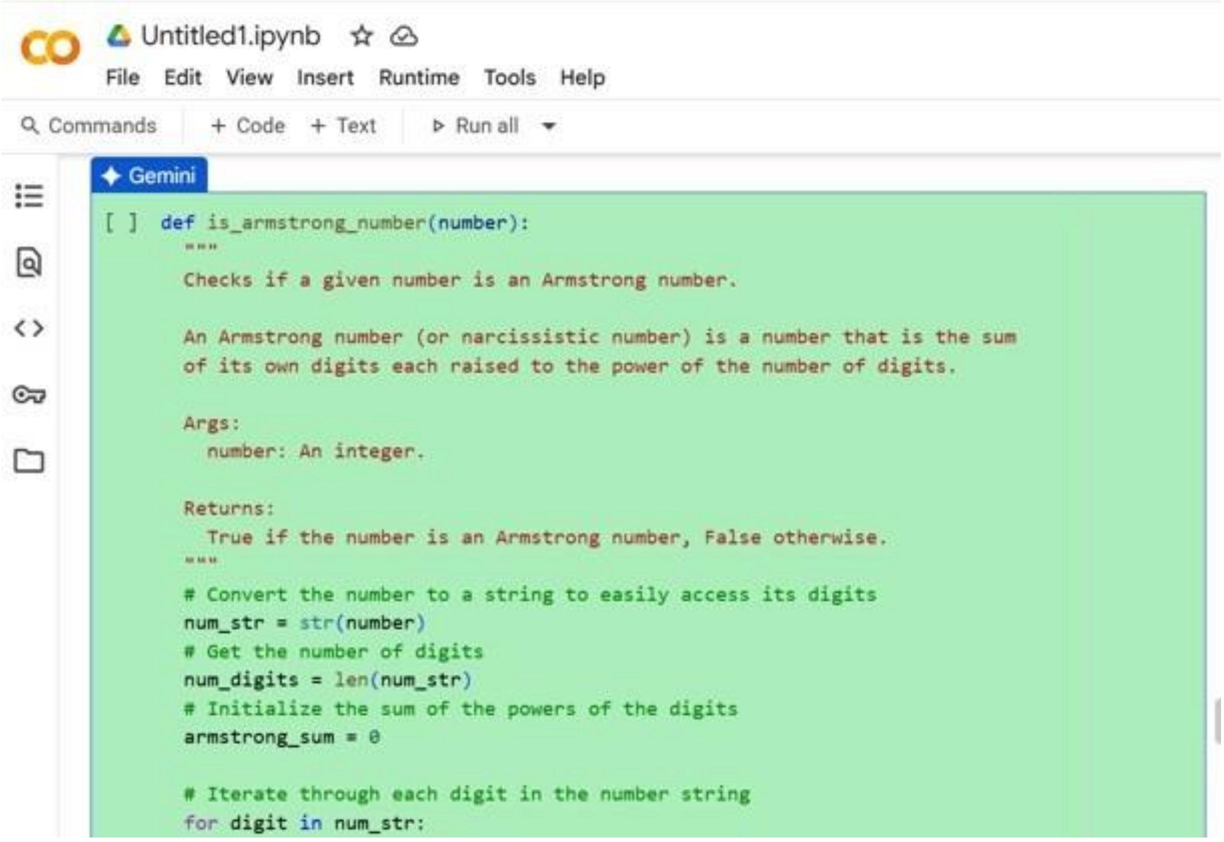
QUESTION:



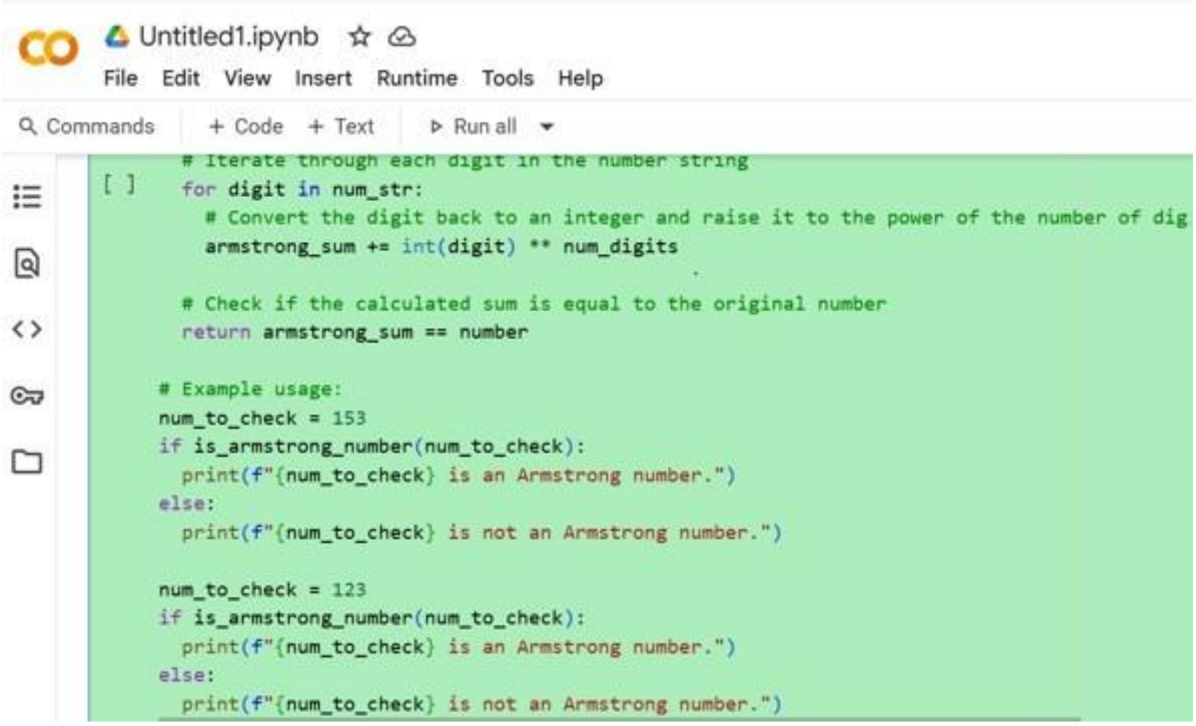
Write a python program ton check whether the given number is Armstrong Number, and improve the structure

Okay, I can help you with that. Here is a Python program tha includes a function to check if a number is an Armstrong number, with improved structure and comments.

CODE:

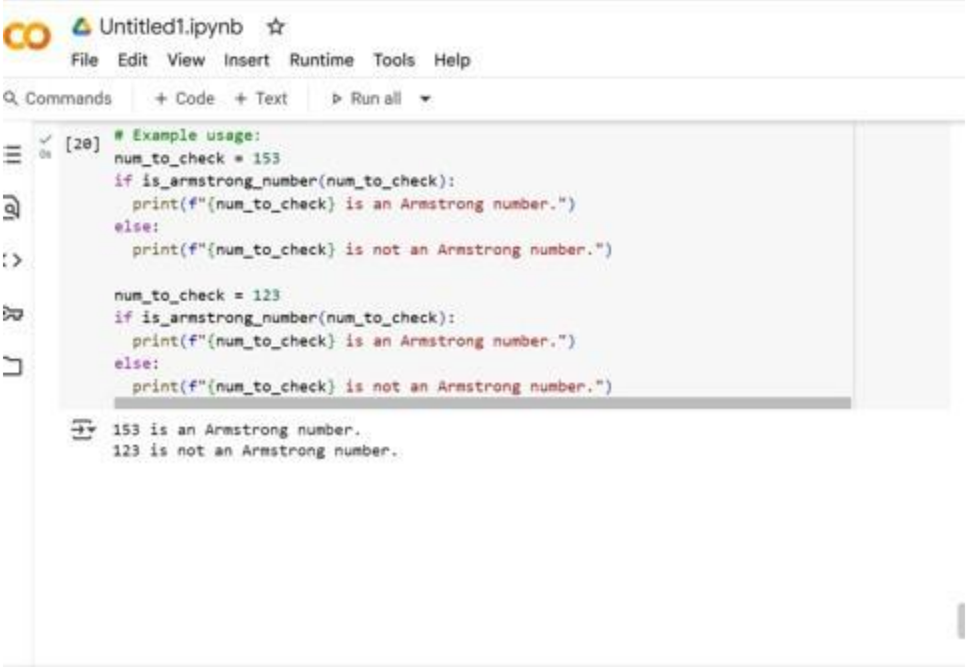


```python
def is_armstrong_number(number):
  """
  Checks if a given number is an Armstrong number.

  An Armstrong number (or narcissistic number) is a number that is the sum
  of its own digits each raised to the power of the number of digits.

  Args:
    number: An integer.

  Returns:
    True if the number is an Armstrong number, False otherwise.
  """
  # Convert the number to a string to easily access its digits
  num_str = str(number)
  # Get the number of digits
  num_digits = len(num_str)
  # Initialize the sum of the powers of the digits
  armstrong_sum = 0

  # Iterate through each digit in the number string
  for digit in num_str:
```



```python
  # Iterate through each digit in the number string
  for digit in num_str:
    # Convert the digit back to an integer and raise it to the power of the number of dig
    armstrong_sum += int(digit) ** num_digits

  # Check if the calculated sum is equal to the original number
  return armstrong_sum == number

# Example usage:
num_to_check = 153
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.")

num_to_check = 123
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.")
```

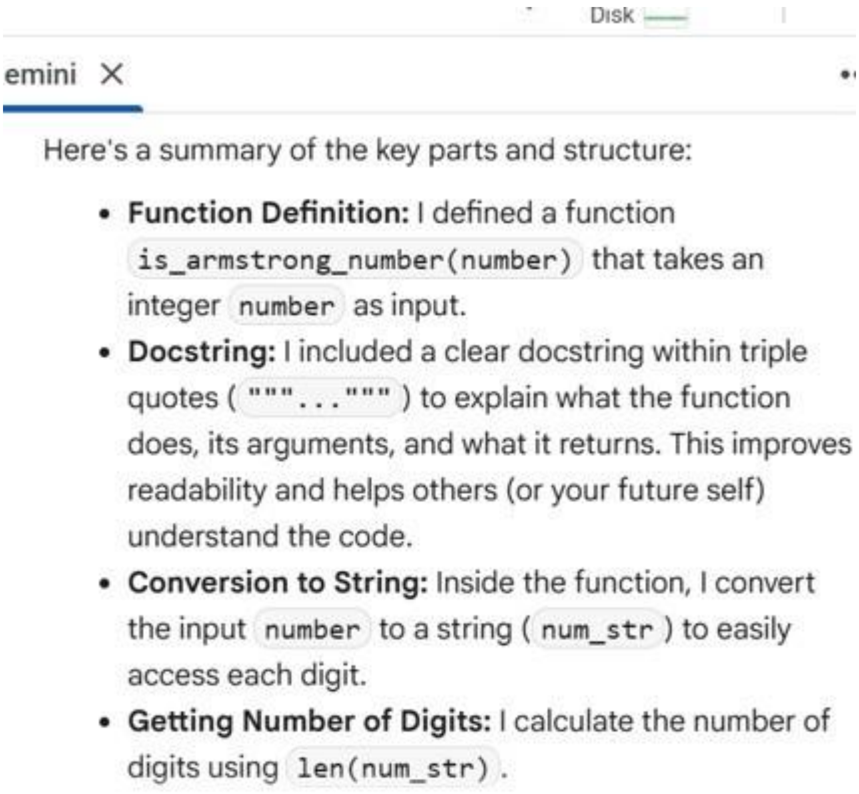
OUTPUT:



```python
# Example usage:
num_to_check = 153
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.")

num_to_check = 123
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.")
```

```
153 is an Armstrong number.
123 is not an Armstrong number.
```

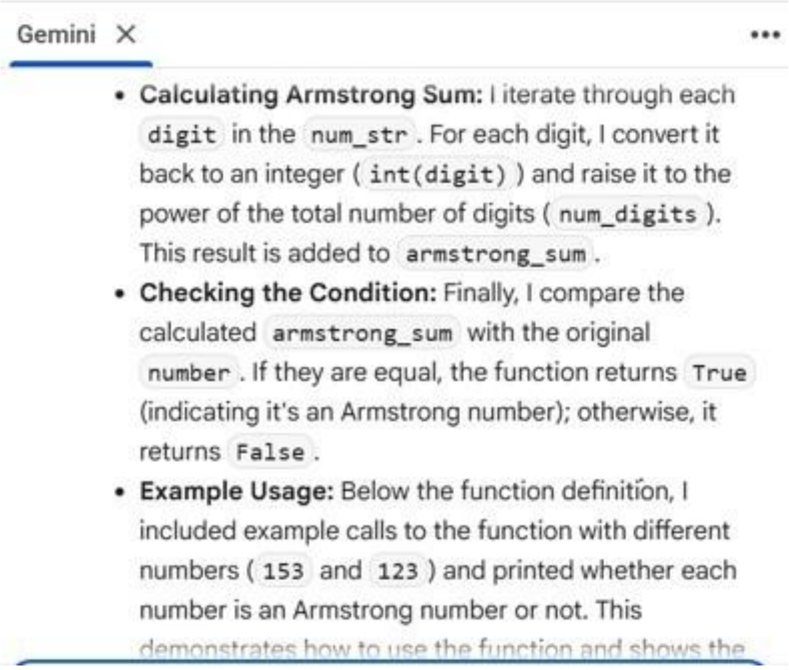
Summery:



Here's a summary of the key parts and structure:

- **Function Definition:** I defined a function `is_armstrong_number(number)` that takes an integer `number` as input.
- **Docstring:** I included a clear docstring within triple quotes (`"""..."""`) to explain what the function does, its arguments, and what it returns. This improves readability and helps others (or your future self) understand the code.
- **Conversion to String:** Inside the function, I convert the input `number` to a string (`num_str`) to easily access each digit.
- **Getting Number of Digits:** I calculate the number of digits using `len(num_str)`.

CODE:



- **Calculating Armstrong Sum:** I iterate through each `digit` in the `num_str`. For each digit, I convert it back to an integer (`int(digit)`) and raise it to the power of the total number of digits (`num_digits`). This result is added to `armstrong_sum`.
- **Checking the Condition:** Finally, I compare the calculated `armstrong_sum` with the original `number`. If they are equal, the function returns `True` (indicating it's an Armstrong number); otherwise, it returns `False`.
- **Example Usage:** Below the function definition, I included example calls to the function with different numbers (`153` and `123`) and printed whether each number is an Armstrong number or not. This demonstrates how to use the function and shows the

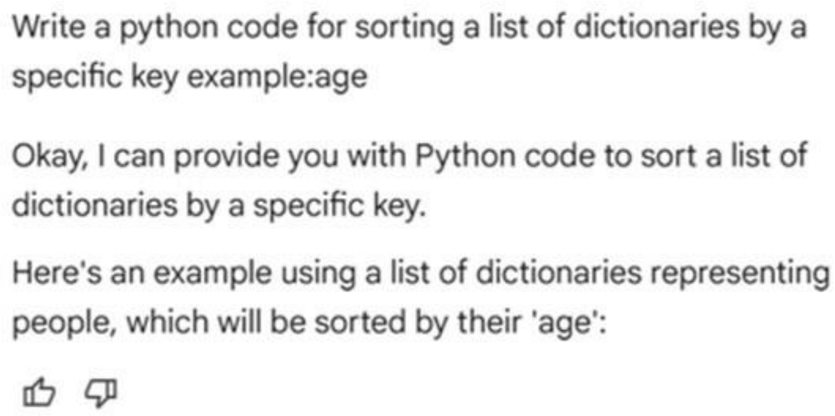
Task Description-6:

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

Prompt-6:

Write a Python code for sorting a list of dictionaries by a specific key example:age.

QUESTION:

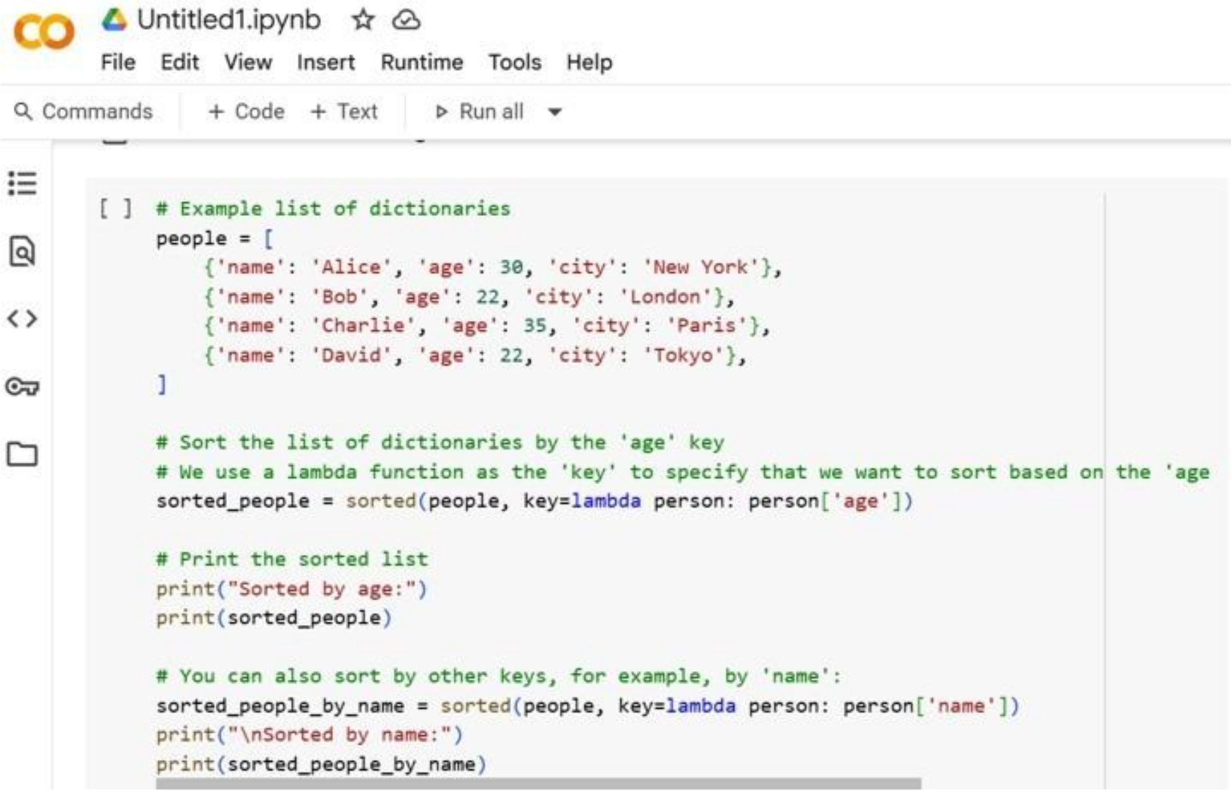


Write a python code for sorting a list of dictionaries by a specific key example:age

Okay, I can provide you with Python code to sort a list of dictionaries by a specific key.

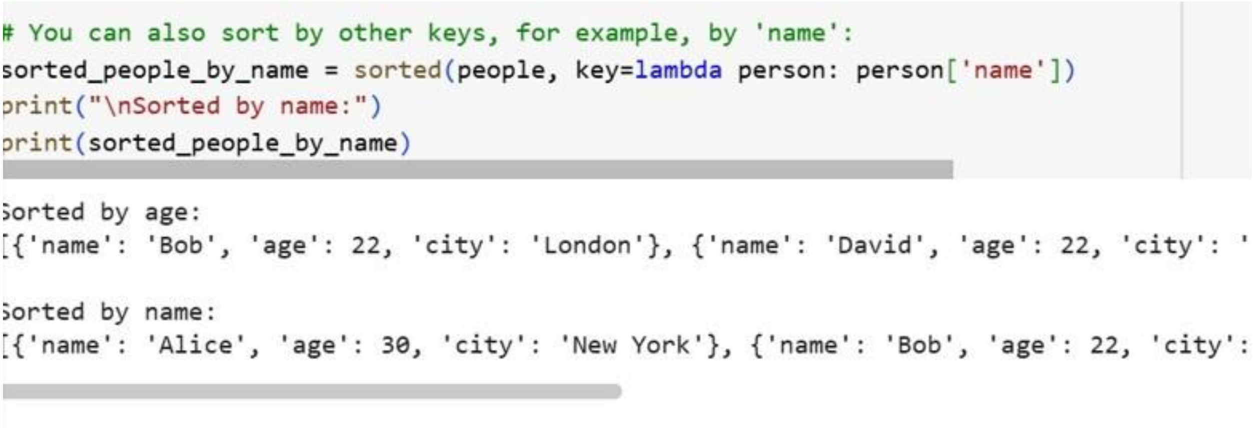
Here's an example using a list of dictionaries representing people, which will be sorted by their 'age':

CODE:



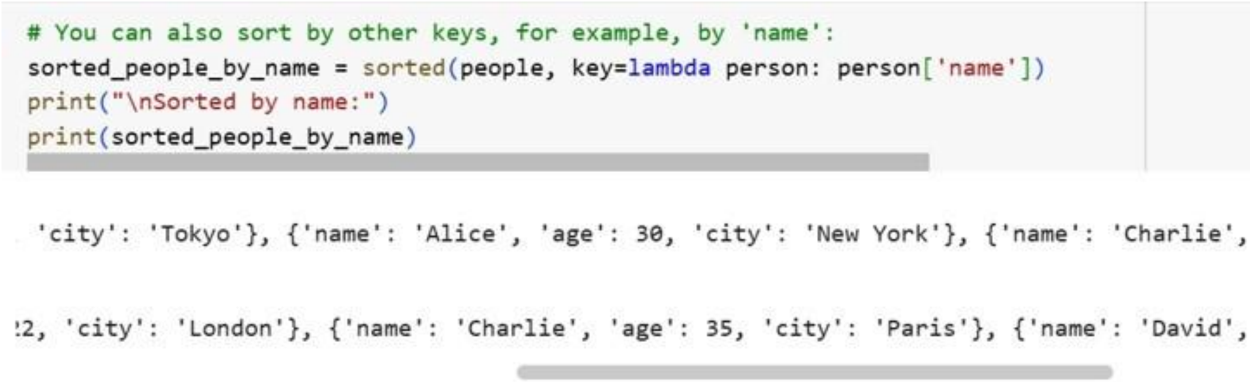
```
# Example list of dictionaries
people = [
    {'name': 'Alice', 'age': 30, 'city': 'New York'},
    {'name': 'Bob', 'age': 22, 'city': 'London'},
    {'name': 'Charlie', 'age': 35, 'city': 'Paris'},
    {'name': 'David', 'age': 22, 'city': 'Tokyo'},
]

# Sort the list of dictionaries by the 'age' key
# We use a lambda function as the 'key' to specify that we want to sort based on the 'age
sorted_people = sorted(people, key=lambda person: person['age'])

# Print the sorted list
print("Sorted by age:")
print(sorted_people)

# You can also sort by other keys, for example, by 'name':
sorted_people_by_name = sorted(people, key=lambda person: person['name'])
print("\nSorted by name:")
print(sorted_people_by_name)
```

OUTPUT:



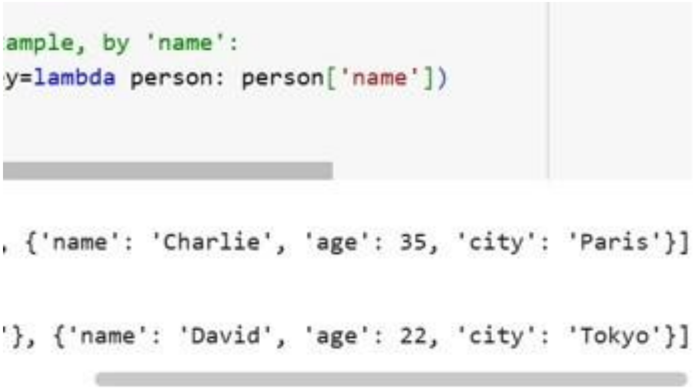
```
# You can also sort by other keys, for example, by 'name':
sorted_people_by_name = sorted(people, key=lambda person: person['name'])
print("\nSorted by name:")
print(sorted_people_by_name)
```

```
Sorted by age:
[{'name': 'Bob', 'age': 22, 'city': 'London'}, {'name': 'David', 'age': 22, 'city': '

Sorted by name:
[{'name': 'Alice', 'age': 30, 'city': 'New York'}, {'name': 'Bob', 'age': 22, 'city':
```



```
# You can also sort by other keys, for example, by 'name':
sorted_people_by_name = sorted(people, key=lambda person: person['name'])
print("\nSorted by name:")
print(sorted_people_by_name)
```

```
 'city': 'Tokyo'}, {'name': 'Alice', 'age': 30, 'city': 'New York'}, {'name': 'Charlie',

2, 'city': 'London'}, {'name': 'Charlie', 'age': 35, 'city': 'Paris'}, {'name': 'David',
```



```
ample, by 'name':
y=lambda person: person['name'])
```

```
, {'name': 'Charlie', 'age': 35, 'city': 'Paris'}]

'}, {'name': 'David', 'age': 22, 'city': 'Tokyo'}]
```

---END---