

---

<b>Program</b>	:B.tech(CSE)
<b>Specialization</b>	:AIML
<b>Course Title</b>	:AI Assisted Coding
<b>Course Code</b>	:24CS002PC215
<b>Semester</b>	:3 <sup>rd</sup> sem
<b>Name of Student</b>	: Kaveti Manohar
<b>En. No.</b>	:2403A52079
<b>Batch No.</b>	:02
<b>Date</b>	:19/08/2025

---

## #LAB ASSIGNMENT

---

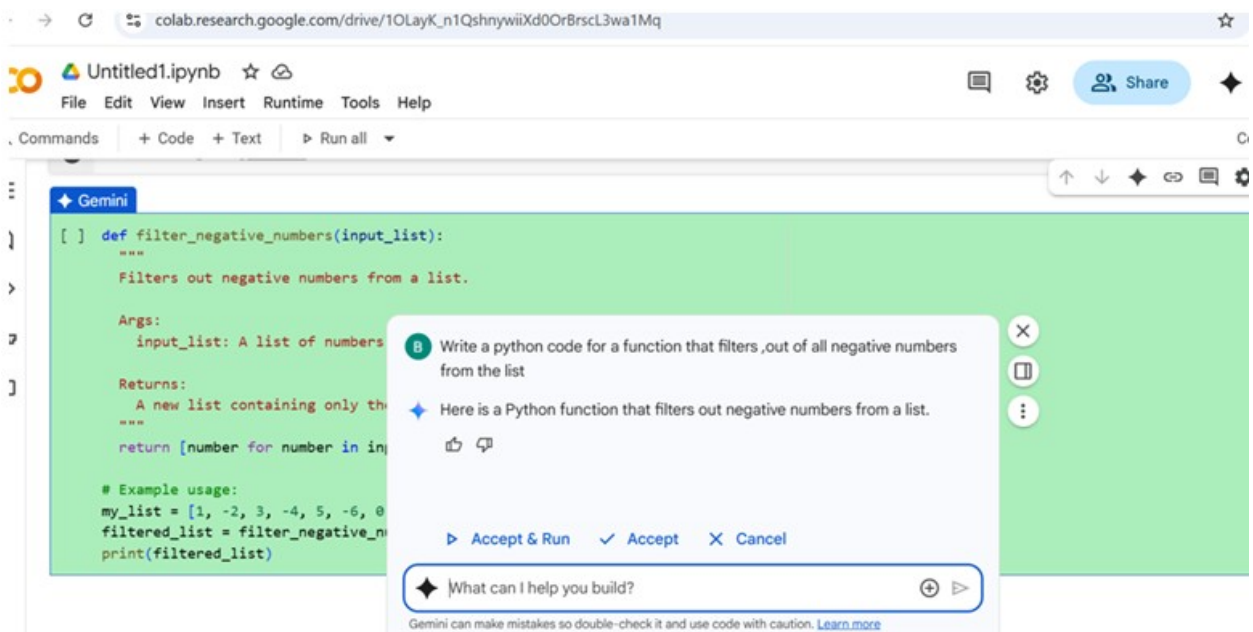
### Task Description -1:

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:



Code:



Output:

```
✓ [2] def remove_negative_numbers(numbers):  
0s  """  
    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

---

► What can I help you build?



Code:

File Edit View Insert Running Tools Help

Commands + Code + Text ▶ Run all ▼

↑ ↓ ↻ 🔗 🗨 ⚙ 📄 🗑 ⋮

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()
```

🔍 Commands + Code + Text ▶ Run all ▼

☰ 📄 > ⚙ 📄 🗑 ⋮

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

    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()
```



Untitled1.ipynb ☆ ☁

File Edit View Insert Runtime Tools Help

Q Commands

+ Code + Text

▶ Run all ▼



0s



```
[8] 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).

 Untitled1.ipynb ☆ ☁

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all ▼



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

```
[9] 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, 'contair
```

▼ Generate code to display word frequencies






Subtask:

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

 Untitled1.ipynb ☆ ☁

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all ▼



▼ 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.

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






      # 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
```

  Untitled1.ipynb  

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all ▼





```
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.

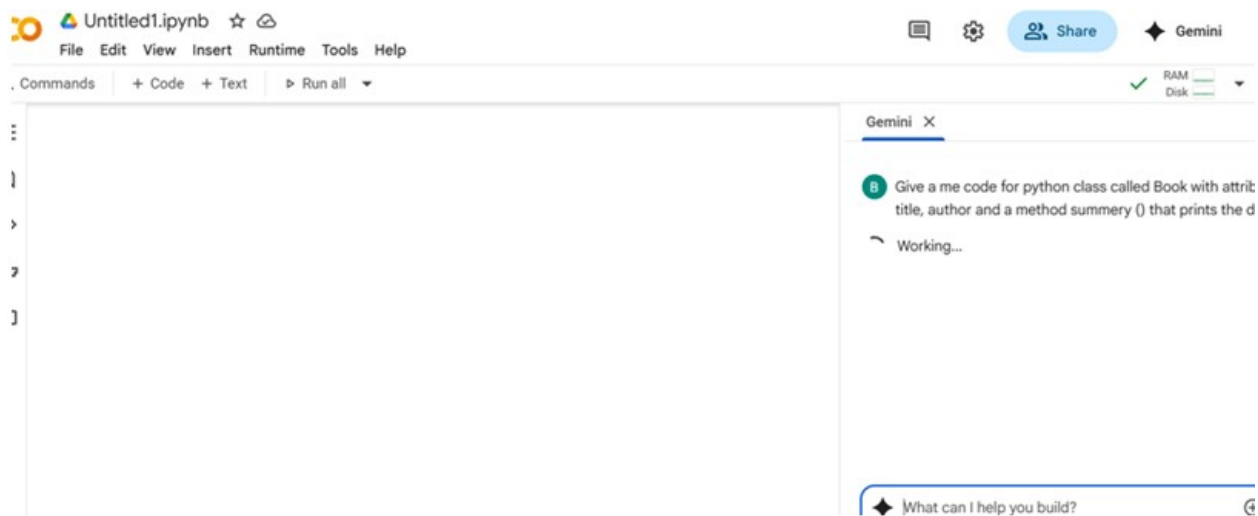




### Prompt-3

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

### QUESTION:



### CODE:

CO

Untitled1.ipynb

☆

☁

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all ▼

☰

✓

0s

[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}")
```

CO

Untitled1.ipynb

☆

☁

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all ▼

☰

✓

0s

[19]

```
# 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

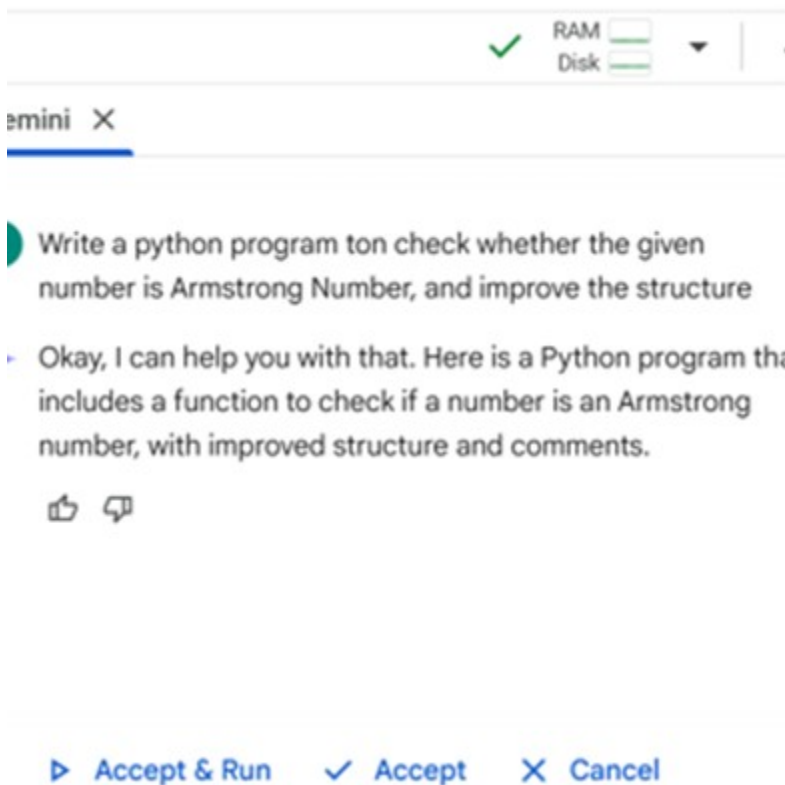
## Task Description-4

- 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:








#### CODE:

 Untitled1.ipynb ☆ ☁


File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all ▼








Gemini

```
[ ] 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:
```

 Untitled1.ipynb ☆ ☁

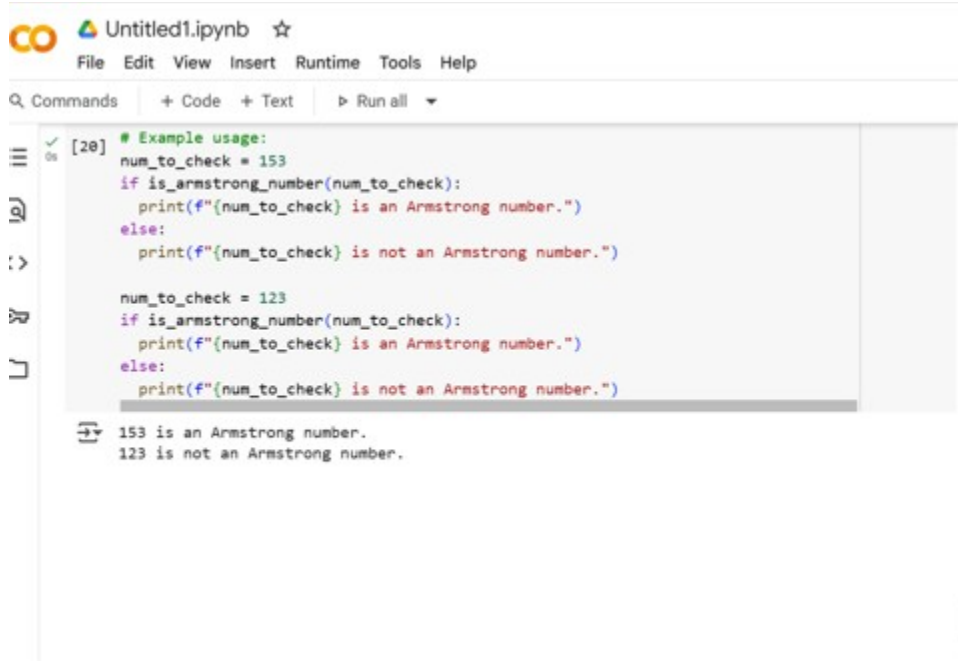
File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all ▼



```
        # 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 digits  
        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:



The screenshot shows a Jupyter Notebook titled 'Untitled1.ipynb'. The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar with options like 'Commands', '+ Code', '+ Text', and 'Run all'. The code cell contains the following Python code:

```
[20] # 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.")
```

The output of the code is displayed below the cell:

```
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)`.



- **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:

A screenshot of a Jupyter Notebook interface. The title bar shows 'Untitled1.ipynb' with a star and a refresh icon. Below the title bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Underneath the menu bar is a toolbar with 'Commands', '+ Code', '+ Text', and 'Run all' with a dropdown arrow. On the left side, there is a sidebar with icons for a menu, search, expand/collapse, key, and folder. The main area contains a code cell with the following Python 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': 'Tokyo'}, {'name': 'Alice', 'age': 30, 'city': 'New York'}, {'name': 'Charlie', 'age': 35, 'city': 'Paris'}]
```

Sorted by name:

```
[{'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'}]
```

```
# 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)
```

```
[{'name': 'Alice', 'age': 30, 'city': 'New York'}, {'name': 'Bob', 'age': 22, 'city': 'Tokyo'}, {'name': 'Charlie', 'age': 35, 'city': 'Paris'}, {'name': 'David', 'age': 22, 'city': 'London'}]
```

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

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

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