

## **AI ASSITED CODING**

**NAME : GURRALA SAI SPOORTHY**

**YEAR / SEM : II / I**

**BATCH : 01**

**HALL TICKET NO: 2403A52290**

**COURSE TITLE: AI ASSITED CODING**

**Task 1:**

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

**PROMPT:**

**Write a python function to figure out negative numbers from a user defined list**

**Description:**

**Use Gemini in collab to write a function using python language to negative numbers from all the**

user defined elements in a list must be figured out

## CODE:



The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo, a 'Welcome To Colab' message, and a 'Cannot save changes' warning. The menu bar contains 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The left sidebar has a 'Table of contents' section with links to 'Welcome to Colab!', 'Getting started', 'Data science', 'Machine learning', and 'More Resources'. The main code area contains a Python function `def get_negative_numbers(input_list):` with docstrings and a list comprehension. Below the function, there is a section for getting user input and converting it to a list, with error handling for `ValueError`. The function is then called with the user input, and the results are printed. A tooltip is visible over the function definition, showing its signature and docstring. The bottom of the notebook shows the execution output: 'Enter a list of numbers separated by spaces: 1 2 3 -4 56', 'Original list: [1, 2, 3, -4, 56]', and 'Negative numbers: [-4]'.

```
def get_negative_numbers(input_list):  
    """  
    This function takes a list of numbers and returns a new list containing only the negative numbers.  
    Args:  
        input_list: A list of numbers.  
    Returns:  
        A new list containing only the negative numbers from the input list.  
    """  
    negative_numbers = [num for num in input_list if num < 0]  
    return negative_numbers  
  
# Get input from the user dynamically  
input_string = input("Enter a list of numbers separated by spaces: ")  
  
# Convert the input string to a list of integers  
try:  
    my_list = [int(num) for num in input_string.split()]  
except ValueError:  
    print("Invalid input.")  
    my_list = [] # Assign empty list if input is invalid  
  
negative_numbers_list = get_negative_numbers(my_list)  
print(f"Original list: {my_list}")  
print(f"Negative numbers: {negative_numbers_list}")
```

Enter a list of numbers separated by spaces: 1 2 3 -4 56  
Original list: [1, 2, 3, -4, 56]  
Negative numbers: [-4]

## OUTPUT:

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

```
Enter a list of numbers separated by spaces: 33 44 -55 54 0  
Original list: [33, 44, -55, 54, 0]  
Negative numbers: [-55]
```

## Observation:

**I have given different input for above code .All the test cases are accurate as expected**

## Explanation:

**In the above code,We used Gemini to get the code.We have given the prompt,it has given me the code as per the prompt.**

**We have created an emty list and appended the elements .We used the condition `if element<0`**

**By using the Condition,it is able to find out the negative numbers**

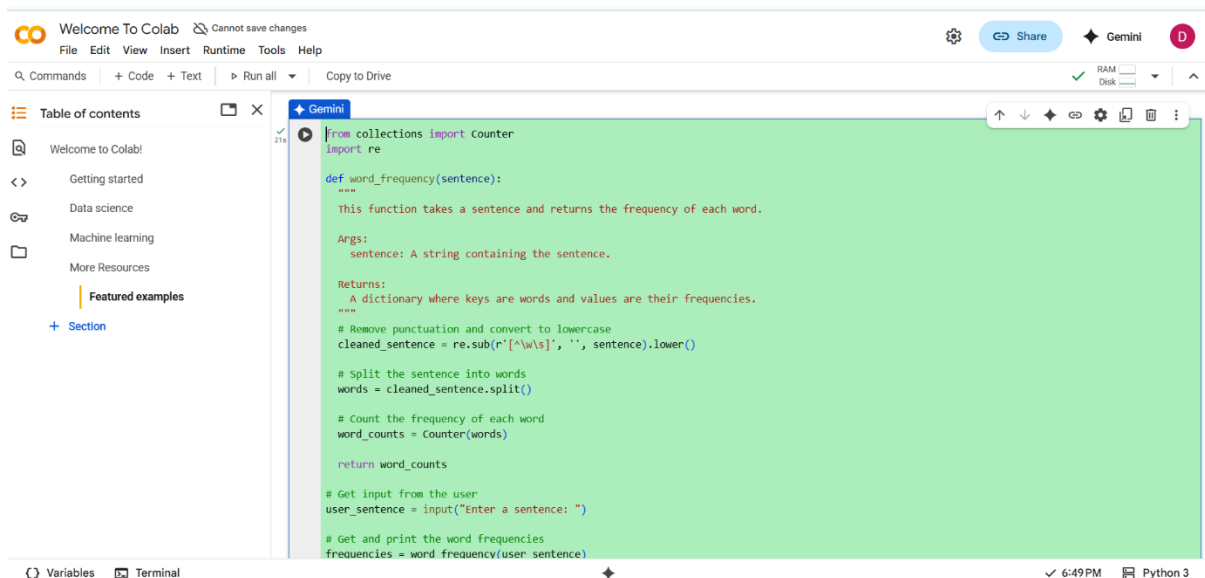
## Task 2:

**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:

**Write a python function which takes a sentence and finds out frequency of each word in it**

## CODE:



The screenshot shows the Google Colab interface. On the left is a sidebar with a 'Table of contents' and a 'Featured examples' section. The main area displays a code editor with Python code generated by Gemini. The code defines a function `word_frequency(sentence)` that uses `collections.Counter` and `re` to count word frequencies in a sentence, removing punctuation and converting to lowercase. Below the function definition, there are comments and code for getting user input and printing the results. The status bar at the bottom indicates '6:49 PM' and 'Python 3'.

```
from collections import Counter
import re

def word_frequency(sentence):
    """
    This function takes a sentence and returns the frequency of each word.

    Args:
        sentence: A string containing the sentence.

    Returns:
        A dictionary where keys are words and values are their frequencies.
    """
    # Remove punctuation and convert to lowercase
    cleaned_sentence = re.sub(r"[^\\w\\s]", "", sentence).lower()

    # Split the sentence into words
    words = cleaned_sentence.split()

    # Count the frequency of each word
    word_counts = Counter(words)

    return word_counts

# Get input from the user
user_sentence = input("Enter a sentence: ")

# Get and print the word frequencies
frequencies = word_frequency(user_sentence)
```

## Output:

**Working code and Gemini's explanation side-by-side.**



Enter a sentence: many years ago there was a poet named named was a poet

output actions frequencies:

many: 1  
years: 1  
ago: 1  
there: 1  
was: 2  
a: 2  
poet: 2  
named: 2

This defines the `word_frequency` function that takes a `sentence` as input. Inside the function, `re.sub(r'^\w\s', '', sentence)` uses a regular expression to remove any characters that are not word characters (alphanumeric + underscore) or whitespace. `.lower()` converts the entire sentence to lowercase, ensuring that "The" and "the" are counted as the same word.

```
# Split the sentence into words  
words = cleaned_sentence.split()
```



What can I help you build?



Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

## Code Explanation:

Here, `Counter(words)` takes the list of words and creates a dictionary-like object where keys are the words and values are their counts (frequencies).

```
return word_counts
```

The function then returns the `word_counts` object.

```
# Get input from the user
user_sentence = input("Enter a sentence: ")
```

This line prompts the user to enter a sentence and stores it in the

◆ What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

```
# Get and print the word frequencies
frequencies = word_frequency(user_sentence)
print("\nWord frequencies:")
for word, count in frequencies.items():
    print(f"{word}: {count}")
```

Finally, this part calls the `word_frequency` function with the user's input, stores the result in the `frequencies` variable, and then iterates through the `frequencies` to print each word and its corresponding count.

◆ What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

**Observation:**

**The outputs are as expected**

## Task 3:

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

## PROMPT:

Write a python class Book with attributes title,author and a method Display which prints information about the class

## CODE:

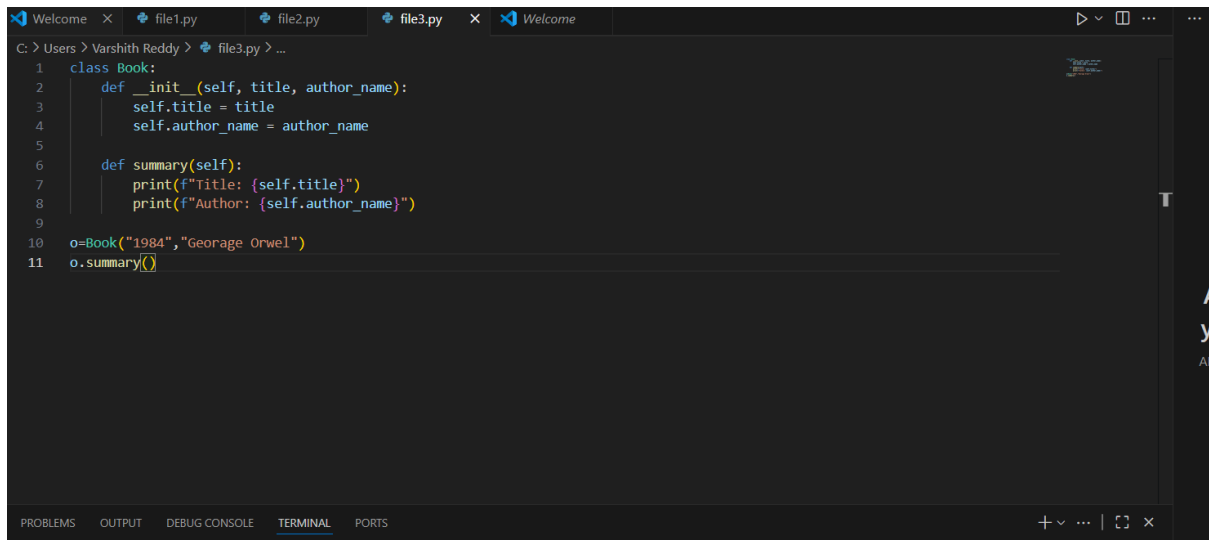
### Code from Gemini in Collab

```
class Book:
    def __init__(self, title, author_name):
        self.title = title
        self.author_name = author_name

    def summary(self):
        print(f>Title: {self.title}")
        print(f>Author: {self.author_name}")

# Example usage:
my_book = Book("The Hitchhiker's Guide to the Galaxy", "Douglas Adams")
my_book.summary()
```

### Code from Copilot



```
C: > Users > Varshith Reddy > file3.py > ...
1 class Book:
2     def __init__(self, title, author_name):
3         self.title = title
4         self.author_name = author_name
5
6     def summary(self):
7         print(f"Title: {self.title}")
8         print(f"Author: {self.author_name}")
9
10 o=Book("1984", "George Orwell")
11 o.summary()
```

## OUTPUT:

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

## Output from Gemini in Collab

---

🔗 Title: The Hitchhiker's Guide to the Galaxy  
Author: Douglas Adams

---

## Output from Copilot



```
Title: 1984
Author: George Orwell
```

## Student Commentary:



**I got the code from both Copilot and Gemini in Collab**

**Performance wise Copilot is too good as it is taking less time to give the code and it is user friendly to use**

**Whereas Gemini in Collab is taking more time than Copilot and there is a delay in displaying of Output**

**However both of them gave the same code i.e static way of coding .No where the code is dynamic**

**Observation:**

**The outputs are as expected**

#### **Task 4:**

**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.**

#### **PROMPT:**


**Write a python function that takes a user defined number and checks whether it is Armstrong or not**

## Description:

Use Python Programming language to take a user defined number and check whether it is Armstrong. Use while loop for better Understanding

## Code:

### Code from Gemini in Collab

```
 def is_armstrong_number(number):  
    """Checks if a number is an Armstrong number."""  
    # Convert the number to a string to easily access its digits  
    num_str = str(number)  
    num_digits = len(num_str)  
    armstrong_sum = 0  
    for digit in num_str:  
        armstrong_sum += int(digit) ** num_digits  
    return armstrong_sum == number  
  
# Get input from the user  
num = int(input("Enter a number: "))  
  
# Check if it's an Armstrong number and print the result  
if is_armstrong_number(num):  
    print(f"{num} is an Armstrong number.")  
else:  
    print(f"{num} is not an Armstrong number.")
```

## Output from Gemini in Collab:

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

```
Enter a number: 9474  
9474 is an Armstrong number.
```

## Code from Copilot:

```
1 def is_armstrong(number):  
2     """  
3     Check if a number is an Armstrong number.  
4  
5     Args:  
6     |   number (int): The number to check.  
7  
8     Returns:  
9     |   bool: True if Armstrong, False otherwise.  
10    """  
11    num_str = str(number)  
12    num_len = len(num_str)  
13    total = sum(int(digit) ** num_len for digit in num_str)  
14    return total == number  
15  
16 # Example usage:  
17 num = int(input("Enter a number: "))  
18 if is_armstrong(num):  
19     print(f"{num} is an Armstrong number.")  
20 else:  
21     print(f"{num} is not an Armstrong number.")
```

## Output from Copilot:

```
Enter a number: 153  
153 is an Armstrong number.
```

## Summary:

**I got two similar codes from Copilot and Gemini in Collab among those two the code from both are good .**

**Coming to their fastness Copilot is a little bit fast  
We can see in above slides**

**Explanation:**

**In above code they are asking us to enter a number to check whether it is an Armstrong or not.As per the conditions the code is checking .**

**Observation:**

**The outputs are as expected**

## **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).**

**CODE:**

**Code from Gemini in Collab**



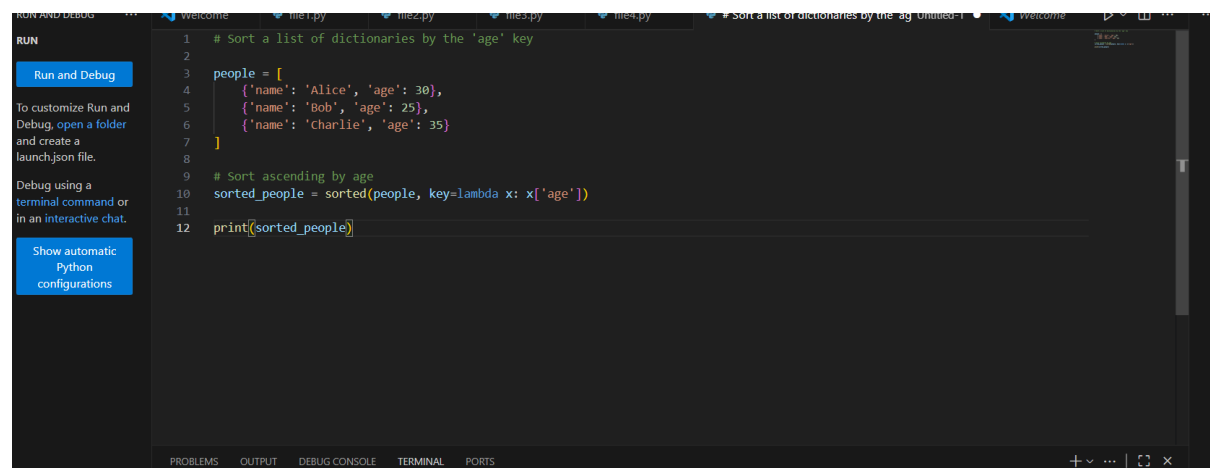
```
list_of_dicts = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35}
]

sorted_list = sorted(list_of_dicts, key=lambda x: x['age'])

print(sorted_list)
```

[[{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie', 'age': 35}]]

**Code from Copilot:**



```
1 # Sort a list of dictionaries by the 'age' key
2
3 people = [
4     {'name': 'Alice', 'age': 30},
5     {'name': 'Bob', 'age': 25},
6     {'name': 'Charlie', 'age': 35}
7 ]
8
9 # Sort ascending by age
10 sorted_people = sorted(people, key=lambda x: x['age'])
11
12 print(sorted_people)
```

**OUTPUT:**

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

### **Output from Gemini in Collab**

```
↵ [{ 'name': 'Bob', 'age': 25}, { 'name': 'Alice', 'age': 30}, { 'name': 'Charlie', 'age': 35}]
```

### **Output from Copilot:**

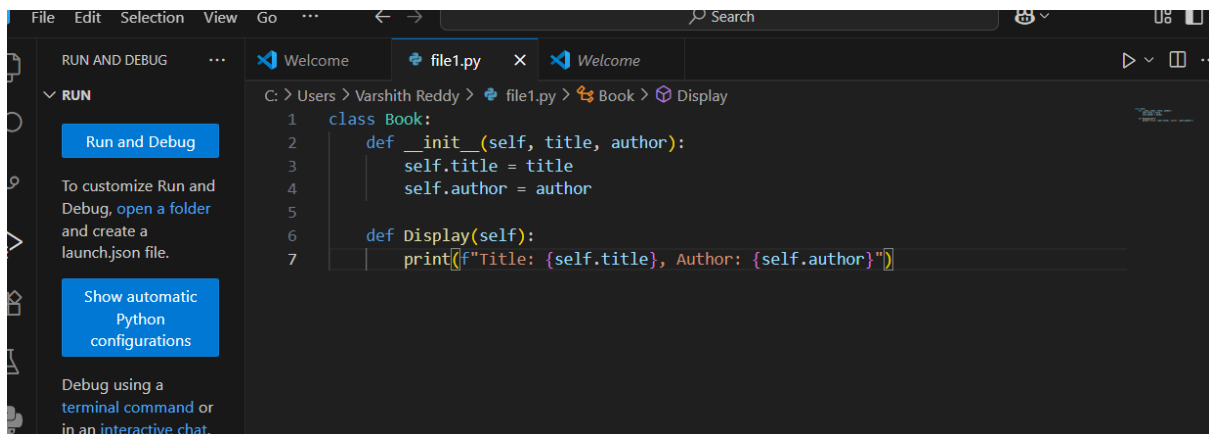
```
[{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie', 'age': 35}]
```

### **Student Commentary:**

**In above two Codes both are giving almost same codes .There is no difference between them.Both of them given static way of code as per the prompt that I have given**

### **Explanation:**

**In the above code they have taken a static list of dictionaries and based on the key output is getting displayed based on the Sorting order**



```
File Edit Selection View Go ... Search
RUN AND DEBUG ... Welcome file1.py X Welcome
▼ RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show automatic Python configurations
Debug using a terminal command or in an interactive chat.
C: > Users > Varshith Reddy > file1.py > Book > Display
1 class Book:
2     def __init__(self, title, author):
3         self.title = title
4         self.author = author
5
6     def Display(self):
7         print(f"Title: {self.title}, Author: {self.author}")
```

**Observation:**

**The outputs are as expected**