AI ASSISTED CODING

Lab Assignment-2.1

Name: Thulasi Shylasri

HTNO:2303A51876

Batch-14(LAB-2)

Date: 12-01-2026

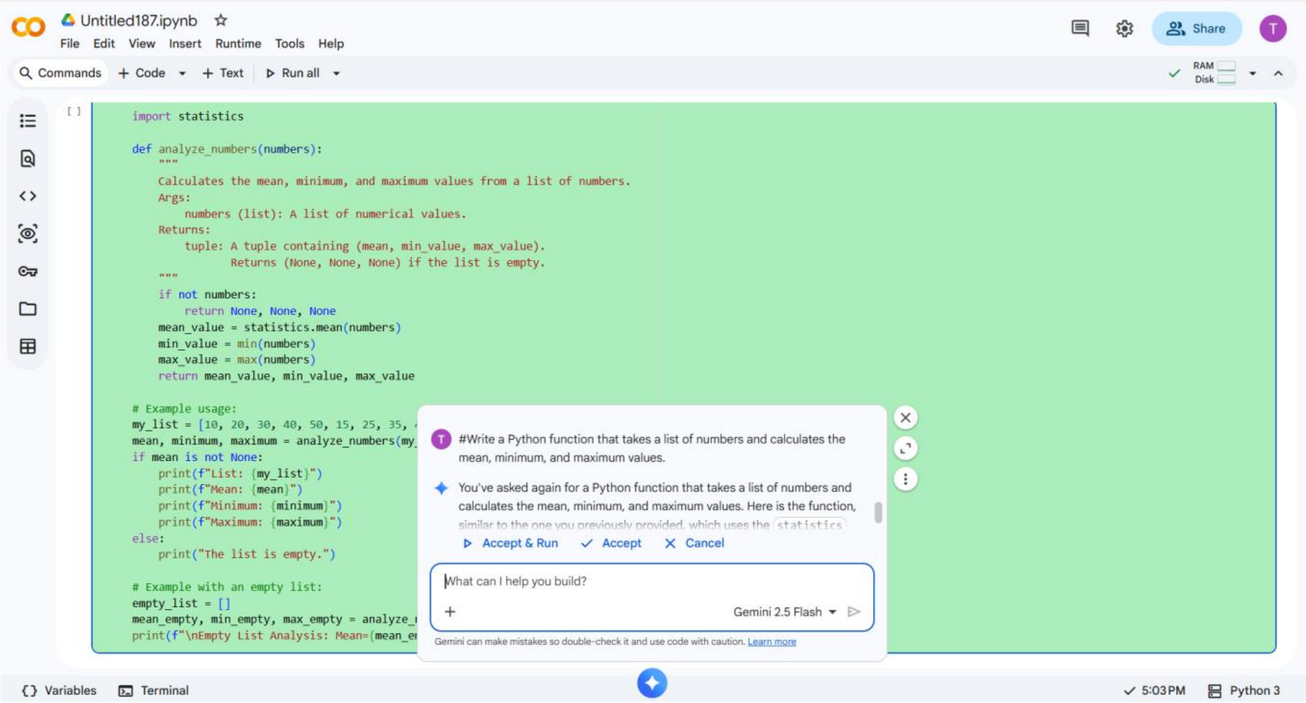
1Q) Task 1: Statistical Summary for Survey Data

- **Scenario:**
  You are a data analyst intern working with survey responses stored as numerical lists.
- **Task:**
  Use Google Gemini in Colab to generate a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.
- **Expected Output:**
  - **Correct Python function**
  - **Output shown in Colab**
  - **Screenshot of Gemini prompt and result**

❖ Prompt:

#Write a Python function that takes a list of numbers and calculates the mean, minimum, and maximum values.

Screenshot:



Untitled187.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk
Variables Terminal
5:03PM
Python 3

```
import statistics

def analyze_numbers(numbers):
    """
    Calculates the mean, minimum, and maximum values from a list of numbers.
    Args:
        numbers (list): A list of numerical values.
    Returns:
        tuple: A tuple containing (mean, min_value, max_value).
               Returns (None, None, None) if the list is empty.
    """
    if not numbers:
        return None, None, None
    mean_value = statistics.mean(numbers)
    min_value = min(numbers)
    max_value = max(numbers)
    return mean_value, min_value, max_value

# Example usage:
my_list = [10, 20, 30, 40, 50, 15, 25, 35,
mean, minimum, maximum = analyze_numbers(my_
if mean is not None:
    print(f"List: {my_list}")
    print(f"Mean: {mean}")
    print(f"Minimum: {minimum}")
    print(f"Maximum: {maximum}")
else:
    print("The list is empty.")

# Example with an empty list:
empty_list = []
mean_empty, min_empty, max_empty = analyze_
print(f"\nEmpty List Analysis: Mean={mean_e
```

#Write a Python function that takes a list of numbers and calculates the mean, minimum, and maximum values.

You've asked again for a Python function that takes a list of numbers and calculates the mean, minimum, and maximum values. Here is the function, similar to the one you previously provided, which uses the `statistics`

Accept & Run
Accept
Cancel
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more

Code:

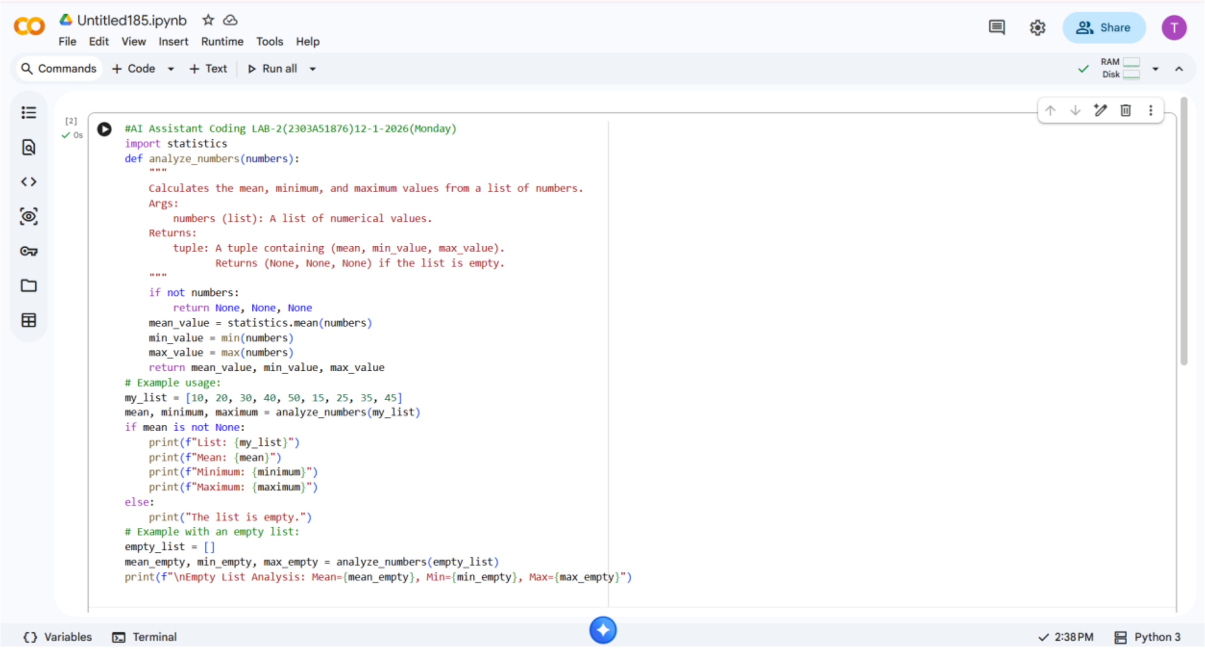
```
#AI Assistant Coding LAB-2(2303A51876)12-1-2026(Monday)

import statistics

def analyze_numbers(numbers):

  """

  Calculates the mean, minimum, and maximum values from a list of numbers.

  Args:

    numbers (list): A list of numerical values.

  Returns:

    tuple: A tuple containing (mean, min_value, max_value).

        Returns (None, None, None) if the list is empty.

  """

  if not numbers:

    return None, None, None

  mean_value = statistics.mean(numbers)
```

```
    min_value = min(numbers)
    max_value = max(numbers)
    return mean_value, min_value, max_value
# Example usage:
my_list = [10, 20, 30, 40, 50, 15, 25, 35, 45]
mean, minimum, maximum = analyze_numbers(my_list)
if mean is not None:
    print(f"List: {my_list}")
    print(f"Mean: {mean}")
    print(f"Minimum: {minimum}")
    print(f"Maximum: {maximum}")
else:
    print("The list is empty.")
# Example with an empty list:
empty_list = []
mean_empty, min_empty, max_empty = analyze_numbers(empty_list)
print(f"\nEmpty List Analysis: Mean={mean_empty}, Min={min_empty}, Max={max_empty}")
```

Scenario:

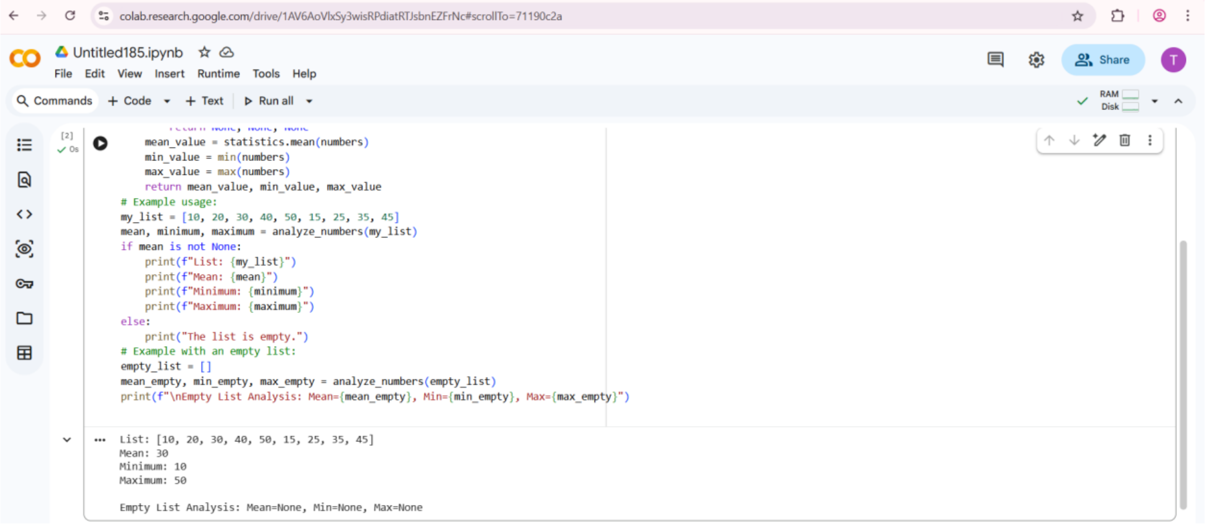
Survey responses are stored as numerical values, and statistical measures are required for analysis.

Screenshot:



```
#AI Assistant Coding LAB-2(2303A51876)12-1-2026(Monday)
import statistics
def analyze_numbers(numbers):
    """
    Calculates the mean, minimum, and maximum values from a list of numbers.
    Args:
        numbers (list): A list of numerical values.
    Returns:
        tuple: A tuple containing (mean, min_value, max_value).
               Returns (None, None, None) if the list is empty.
    """
    if not numbers:
        return None, None, None
    mean_value = statistics.mean(numbers)
    min_value = min(numbers)
    max_value = max(numbers)
    return mean_value, min_value, max_value
# Example usage:
my_list = [10, 20, 30, 40, 50, 15, 25, 35, 45]
mean, minimum, maximum = analyze_numbers(my_list)
if mean is not None:
    print(f"List: {my_list}")
    print(f"Mean: {mean}")
    print(f"Minimum: {minimum}")
    print(f"Maximum: {maximum}")
else:
    print("The list is empty.")
# Example with an empty list:
empty_list = []
mean_empty, min_empty, max_empty = analyze_numbers(empty_list)
print(f"\nEmpty List Analysis: Mean={mean_empty}, Min={min_empty}, Max={max_empty}")
```

Output:


colab.research.google.com/drive/1AV6AoVlxSy3wisRPdiatRTJsbnEZFrNc#scrollTo=71190c2a
Untitled185.ipynb
File Edit View Insert Runtime Tools Help
Commands
+ Code
+ Text
Run all
Share
RAM
Disk
[2]
0s

```
    mean_value = statistics.mean(numbers)
    min_value = min(numbers)
    max_value = max(numbers)
    return mean_value, min_value, max_value
# Example usage:
my_list = [10, 20, 30, 40, 50, 15, 25, 35, 45]
mean, minimum, maximum = analyze_numbers(my_list)
if mean is not None:
    print(f"List: {my_list}")
    print(f"Mean: {mean}")
    print(f"Minimum: {minimum}")
    print(f"Maximum: {maximum}")
else:
    print("The list is empty.")
# Example with an empty list:
empty_list = []
mean_empty, min_empty, max_empty = analyze_numbers(empty_list)
print(f"\nEmpty List Analysis: Mean={mean_empty}, Min={min_empty}, Max={max_empty}")
```

```
List: [10, 20, 30, 40, 50, 15, 25, 35, 45]
Mean: 30
Minimum: 10
Maximum: 50

Empty List Analysis: Mean=None, Min=None, Max=None
```

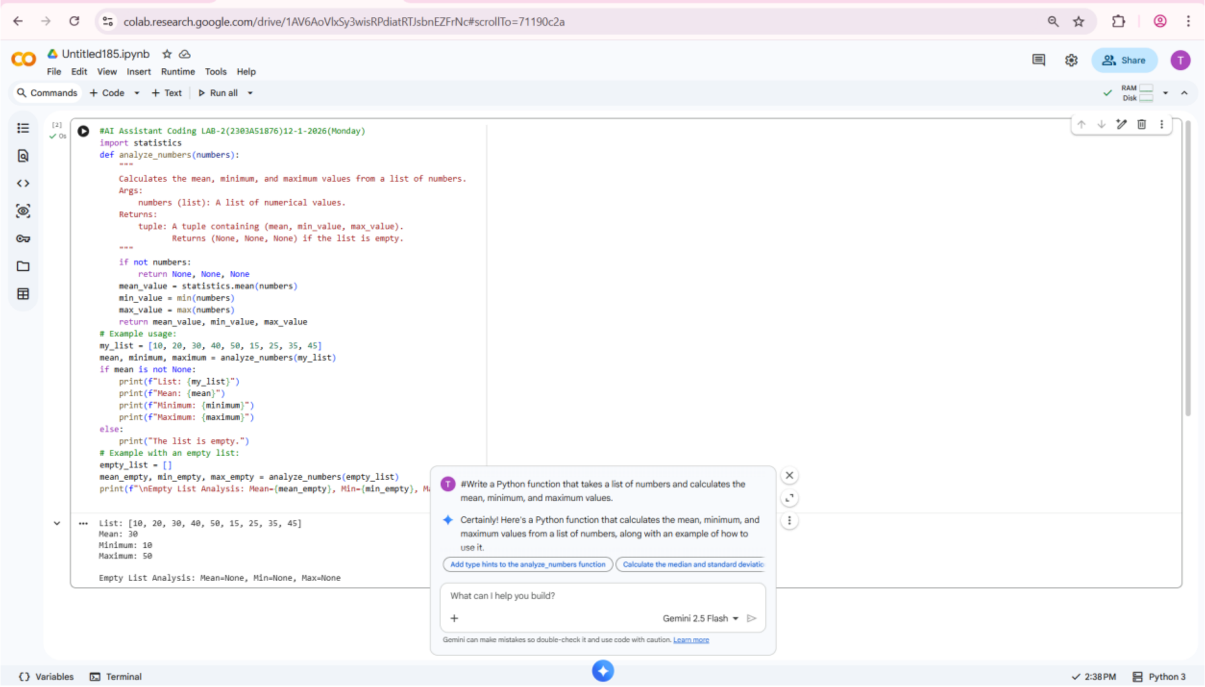
Explanation:

- The code is a Python function that takes a list of numbers and calculates the mean, minimum, and maximum values.
- The function first checks if the list is empty and returns None, None, None if it is.
- Then it calculates the mean by summing up all the numbers and dividing by the number of numbers.
- Then it calculates the minimum by finding the smallest number in the list.
- Then it calculates the maximum by finding the largest number in the list.
- Finally it returns the mean, minimum, and maximum values.

Justification:

Survey data analysis is a common real-world task in data analytics. Calculating mean, minimum, and maximum values helps summarize user responses and identify trends. Using Google Gemini in Colab demonstrates how AI can quickly generate accurate statistical functions.

Task-1


colab.research.google.com/drive/1AV6AoVlxSy3wisRPdiatRTJsbnEZFrNc#scrollTo=71190c2a
Untitled185.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk
Variables Terminal
2:38 PM Python 3

```python
#AI Assistant Coding LAB-2(2303A51876)12-1-2026(Monday)
import statistics
def analyze_numbers(numbers):
    """
    Calculates the mean, minimum, and maximum values from a list of numbers.
    Args:
        numbers (list): A list of numerical values.
    Returns:
        tuple: A tuple containing (mean, min_value, max_value).
               Returns (None, None, None) if the list is empty.
    """
    if not numbers:
        return None, None, None
    mean_value = statistics.mean(numbers)
    min_value = min(numbers)
    max_value = max(numbers)
    return mean_value, min_value, max_value
# Example usage:
my_list = [10, 20, 30, 40, 50, 15, 25, 35, 45]
mean, minimum, maximum = analyze_numbers(my_list)
if mean is not None:
    print(f"List: {my_list}")
    print(f"Mean: {mean}")
    print(f"Minimum: {minimum}")
    print(f"Maximum: {maximum}")
else:
    print("The list is empty.")
# Example with an empty list:
empty_list = []
mean_empty, min_empty, max_empty = analyze_numbers(empty_list)
print(f"\nEmpty List Analysis: Mean={mean_empty}, Min={min_empty}, M
```

```
List: [10, 20, 30, 40, 50, 15, 25, 35, 45]
Mean: 30
Minimum: 10
Maximum: 50

Empty List Analysis: Mean=None, Min=None, Max=None
```

#Write a Python function that takes a list of numbers and calculates the mean, minimum, and maximum values.

Certainly! Here's a Python function that calculates the mean, minimum, and maximum values from a list of numbers, along with an example of how to use it.

Add type hints to the analyze_numbers function
Calculate the median and standard deviatic
What can I help you build?
Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. Learn more

2Q) Task 2: Armstrong Number – AI Comparison

❖ **Scenario:**

You are evaluating AI tools for numeric validation logic.

❖ **Task:**

Generate an Armstrong number checker using Gemini and GitHub Copilot.

Compare their outputs, logic style, and clarity.

❖ **Expected Output:**

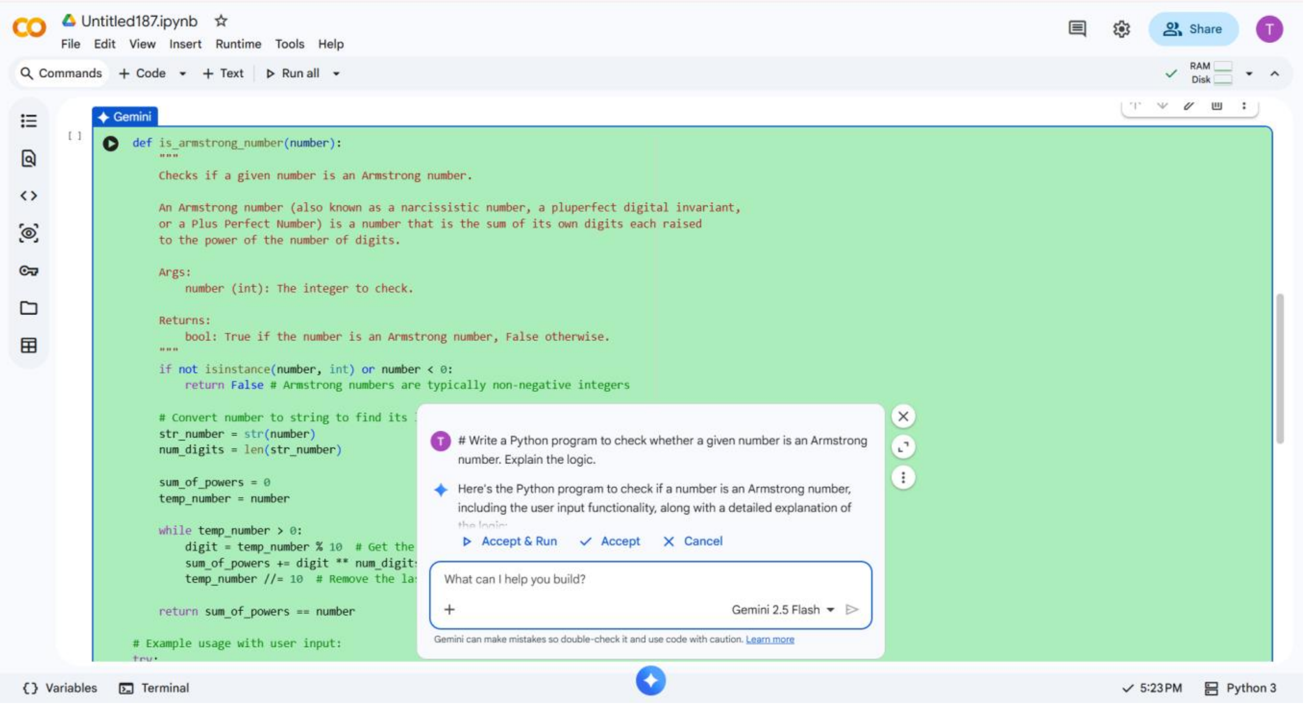
- **Side-by-side comparison table**
- **Screenshots of prompts and generated code**

Prompt for Google Gemini:

# Write a Python program to check whether a given number is an Armstrong number.

Explain the logic.

Screenshot:



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

    An Armstrong number (also known as a narcissistic number, a pluperfect digital invariant,
    or a Plus Perfect Number) is a number that is the sum of its own digits each raised
    to the power of the number of digits.

    Args:
        number (int): The integer to check.

    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    if not isinstance(number, int) or number < 0:
        return False # Armstrong numbers are typically non-negative integers

    # Convert number to string to find its
    str_number = str(number)
    num_digits = len(str_number)

    sum_of_powers = 0
    temp_number = number

    while temp_number > 0:
        digit = temp_number % 10  # Get the
        sum_of_powers += digit ** num_digit
        temp_number //= 10  # Remove the la

    return sum_of_powers == number

# Example usage with user input:
try:
```

# Write a Python program to check whether a given number is an Armstrong number. Explain the logic.

Here's the Python program to check if a number is an Armstrong number, including the user input functionality, along with a detailed explanation of the logic:

Gemini can make mistakes so double-check it and use code with caution. Learn more

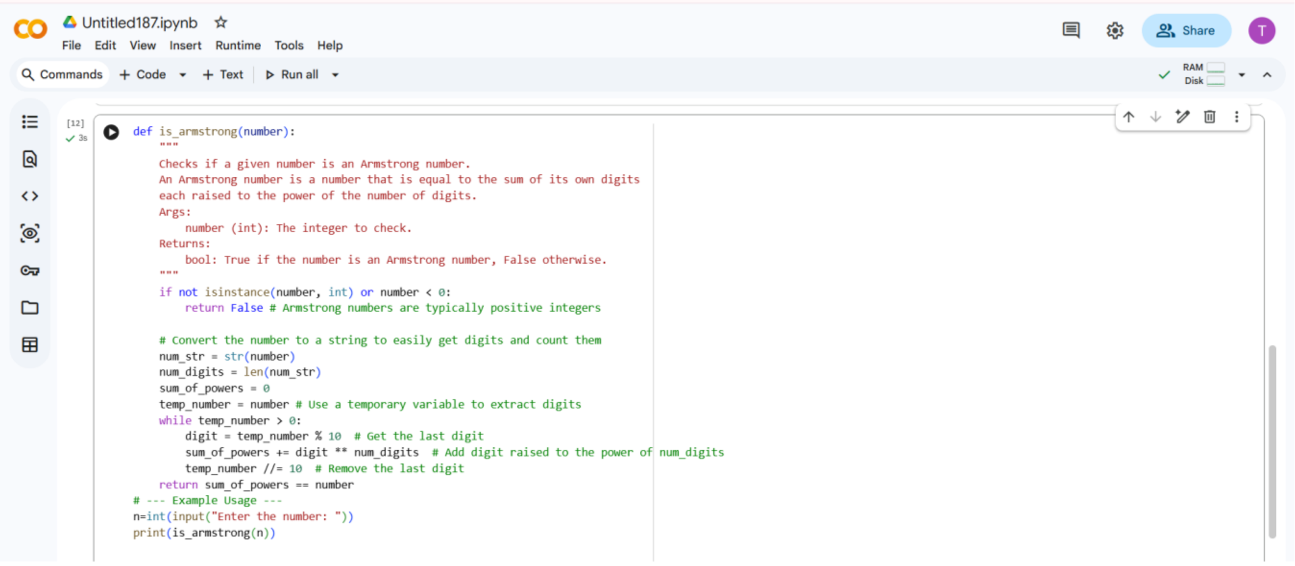
Code:

```
def is_armstrong(number):
    """
    Checks if a given number is an Armstrong number.
    An Armstrong number is a number that is equal to the sum of its own digits
    each raised to the power of the number of digits.
    Args:
        number (int): The integer to check.
    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    if not isinstance(number, int) or number < 0:
        return False # Armstrong numbers are typically positive integers
    # Convert the number to a string to easily get digits and count them
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0
    temp_number = number # Use a temporary variable to extract digits
    while temp_number > 0:
        digit = temp_number % 10  # Get the last digit
        sum_of_powers += digit ** num_digits  # Add digit raised to the power of num_digits
        temp_number //= 10  # Remove the last digit
    return sum_of_powers == number
# --- Example Usage ---
n=int(input("Enter the number: "))
print(is_armstrong(n))
```

Scenario:

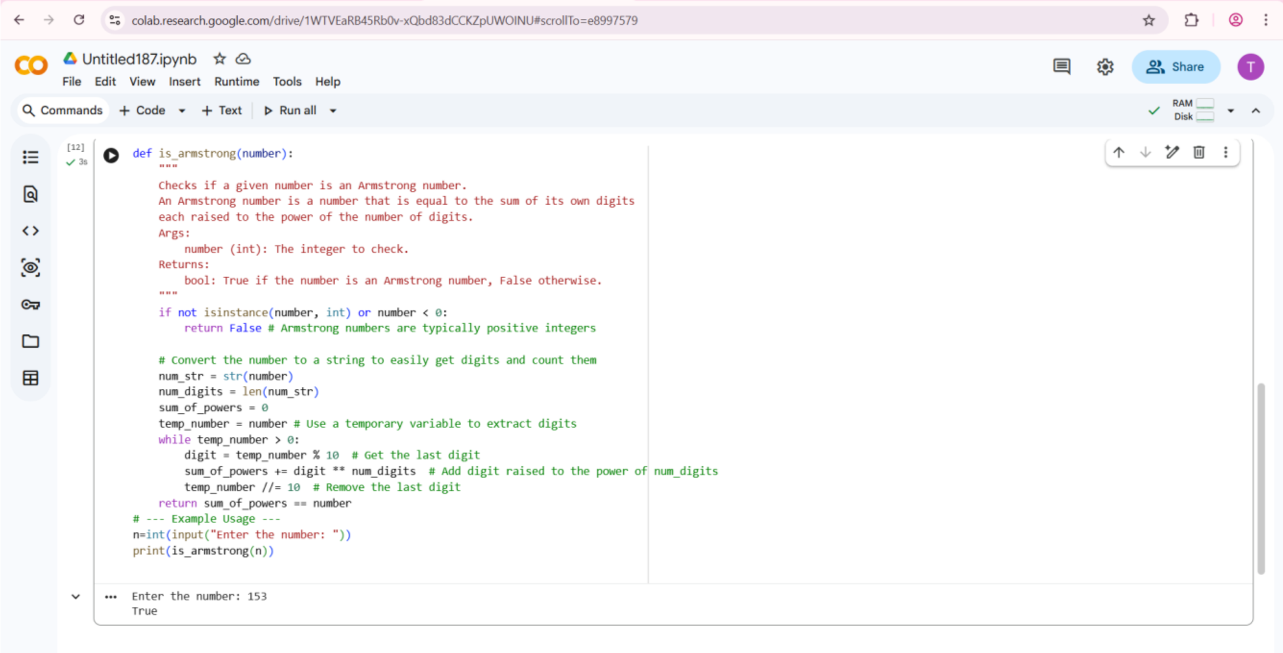
Armstrong number logic is used to test numeric validation across AI tools.

Code Screenshot:



```
def is_armstrong(number):
    """
    Checks if a given number is an Armstrong number.
    An Armstrong number is a number that is equal to the sum of its own digits
    each raised to the power of the number of digits.
    Args:
        number (int): The integer to check.
    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    if not isinstance(number, int) or number < 0:
        return False # Armstrong numbers are typically positive integers

    # Convert the number to a string to easily get digits and count them
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0
    temp_number = number # Use a temporary variable to extract digits
    while temp_number > 0:
        digit = temp_number % 10  # Get the last digit
        sum_of_powers += digit ** num_digits  # Add digit raised to the power of num_digits
        temp_number //= 10  # Remove the last digit
    return sum_of_powers == number
# --- Example Usage ---
n=int(input("Enter the number: "))
print(is_armstrong(n))
```

Output:


colab.research.google.com/drive/1WTVEaRB45Rb0v-xQbd83dCCKZpUWOINU#scrollTo=e8997579
Untitled187.ipynb
File Edit View Insert Runtime Tools Help
Commands
+ Code
+ Text
Run all
Share
RAM
Disk
[12]
3s

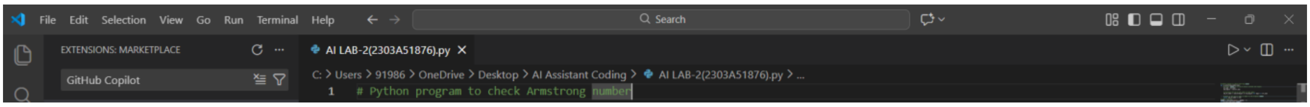
```
def is_armstrong(number):
    """
    Checks if a given number is an Armstrong number.
    An Armstrong number is a number that is equal to the sum of its own digits
    each raised to the power of the number of digits.
    Args:
        number (int): The integer to check.
    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    if not isinstance(number, int) or number < 0:
        return False # Armstrong numbers are typically positive integers

    # Convert the number to a string to easily get digits and count them
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0
    temp_number = number # Use a temporary variable to extract digits
    while temp_number > 0:
        digit = temp_number % 10  # Get the last digit
        sum_of_powers += digit ** num_digits  # Add digit raised to the power of num_digits
        temp_number //= 10  # Remove the last digit
    return sum_of_powers == number
# --- Example Usage ---
n=int(input("Enter the number: "))
print(is_armstrong(n))
```

```
Enter the number: 153
True
```

Prompt for GitHub Copilot:

# Python program to check Armstrong number

Screenshot:


File Edit Selection View Go Run Terminal Help
Search
EXTENSIONS: MARKETPLACE
GitHub Copilot
AI LAB-2(2303A51876).py
C: > Users > 91986 > OneDrive > Desktop > AI Assistant Coding > AI LAB-2(2303A51876).py > ...

```
# Python program to check Armstrong number
```

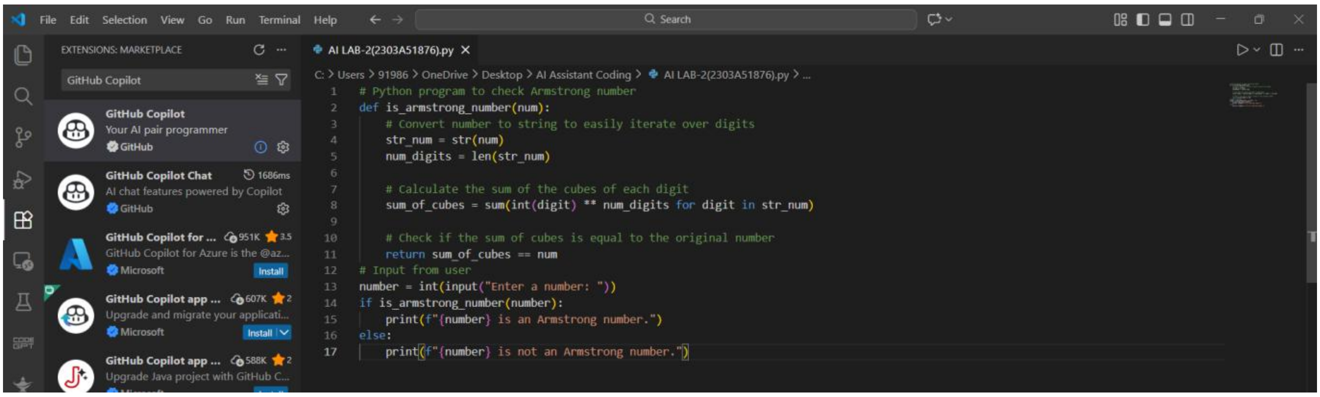
Code:

```
def is_armstrong_number(num):

    # Convert number to string to easily iterate over digits

    str_num = str(num)

    num_digits = len(str_num)

    # Calculate the sum of the cubes of each digit

    sum_of_cubes = sum(int(digit) ** num_digits for digit in str_num)

    # Check if the sum of cubes is equal to the original number
    return sum_of_cubes == num
# Input from user
number = int(input("Enter a number: "))
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

Scenario:

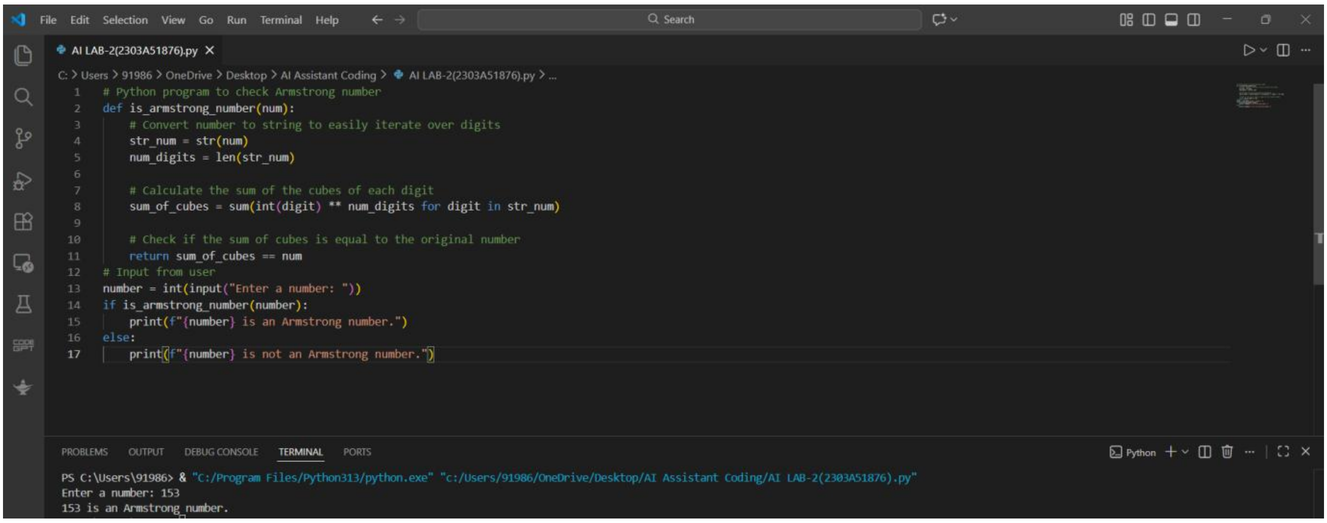
Armstrong number logic is used to test numeric validation across AI tools.

Code Screenshot:


File Edit Selection View Go Run Terminal Help
Search
EXTENSIONS: MARKETPLACE
GitHub Copilot
GitHub Copilot
Your AI pair programmer
GitHub
GitHub Copilot Chat
AI chat features powered by Copilot
GitHub
GitHub Copilot for ...
GitHub Copilot for Azure is the @az...
Microsoft
Install
GitHub Copilot app ...
Upgrade and migrate your applicati...
Microsoft
Install
GitHub Copilot app ...
Upgrade Java project with GitHub C...
AI LAB-2(2303A51876).py
C: > Users > 91986 > OneDrive > Desktop > AI Assistant Coding > AI LAB-2(2303A51876).py > ...

```python
# Python program to check Armstrong number
def is_armstrong_number(num):
    # Convert number to string to easily iterate over digits
    str_num = str(num)
    num_digits = len(str_num)

    # Calculate the sum of the cubes of each digit
    sum_of_cubes = sum(int(digit) ** num_digits for digit in str_num)

    # Check if the sum of cubes is equal to the original number
    return sum_of_cubes == num
# Input from user
number = int(input("Enter a number: "))
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

Output:


File Edit Selection View Go Run Terminal Help
Search
AI LAB-2(2303A51876).py
C: > Users > 91986 > OneDrive > Desktop > AI Assistant Coding > AI LAB-2(2303A51876).py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python

```python
# Python program to check Armstrong number
def is_armstrong_number(num):
    # Convert number to string to easily iterate over digits
    str_num = str(num)
    num_digits = len(str_num)

    # Calculate the sum of the cubes of each digit
    sum_of_cubes = sum(int(digit) ** num_digits for digit in str_num)

    # Check if the sum of cubes is equal to the original number
    return sum_of_cubes == num
# Input from user
number = int(input("Enter a number: "))
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

```
PS C:\Users\91986> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/AI LAB-2(2303A51876).py"
Enter a number: 153
153 is an Armstrong number.
```

Comparison Table:

Feature	**Google Gemini**	**GitHub Copilot**
Code Accuracy	High	High
Explanation	Detailed	Minimal
Readability	Very clear	Concise
Beginner Friendly	Yes	Moderate

Explanation:

Both **Google Gemini** and **GitHub Copilot** successfully generate correct logic for checking an Armstrong number, but they differ in their approach and purpose.

- **Google Gemini** provides:
    - A complete Python program.
    - Step-by-step explanation of the logic involved.
    - Clear variable naming and comments.
    - Beginner-friendly reasoning that helps students understand *why* each step is used.
- **GitHub Copilot** focuses mainly on:
    - Fast code generation and auto-completion.
    - Minimal or no explanation of the logic.
    - Assisting experienced developers who already understand the concept.
    - Improving coding speed rather than conceptual learning.

Thus, while both tools produce correct results, **Gemini emphasizes learning and clarity**, whereas **Copilot emphasizes productivity and speed**.

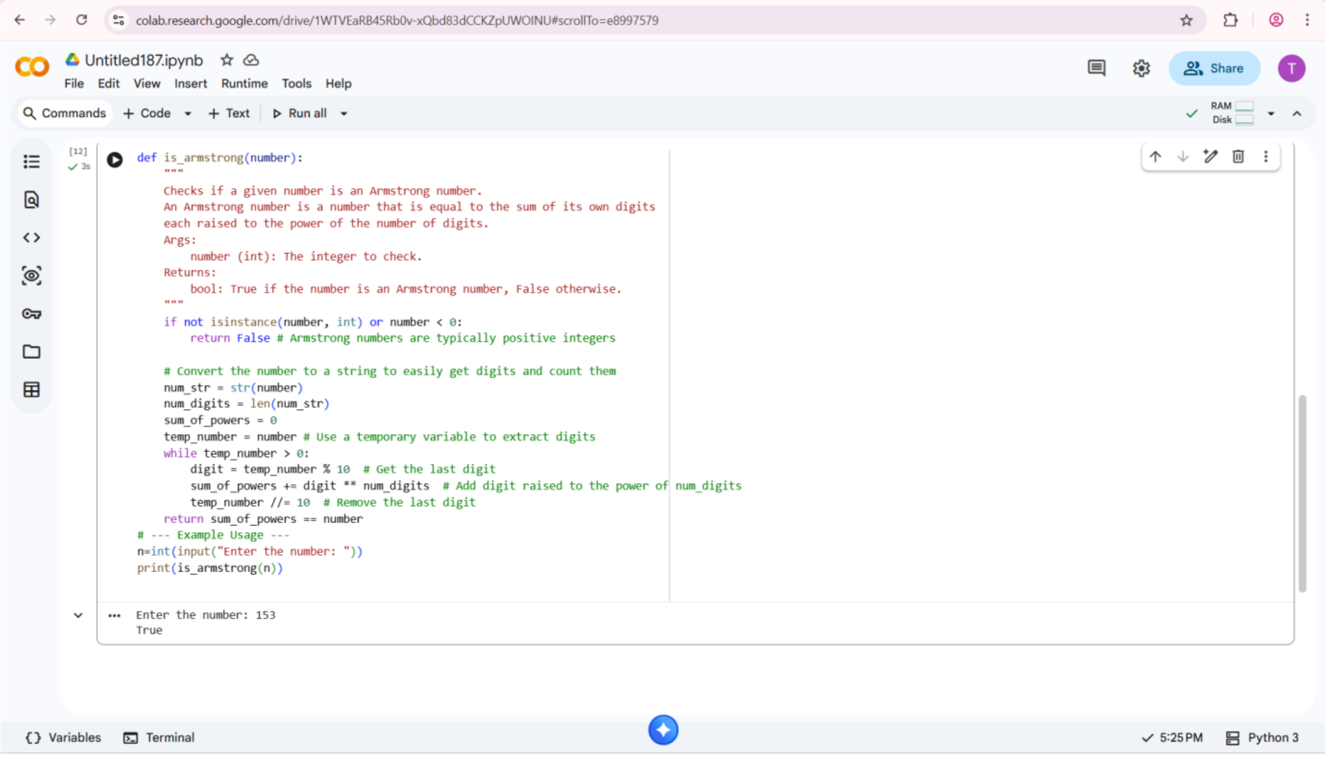
Justification:

Using multiple AI tools allows developers to select tools based on learning or productivity needs.

By comparing multiple AI tools, developers and students can:

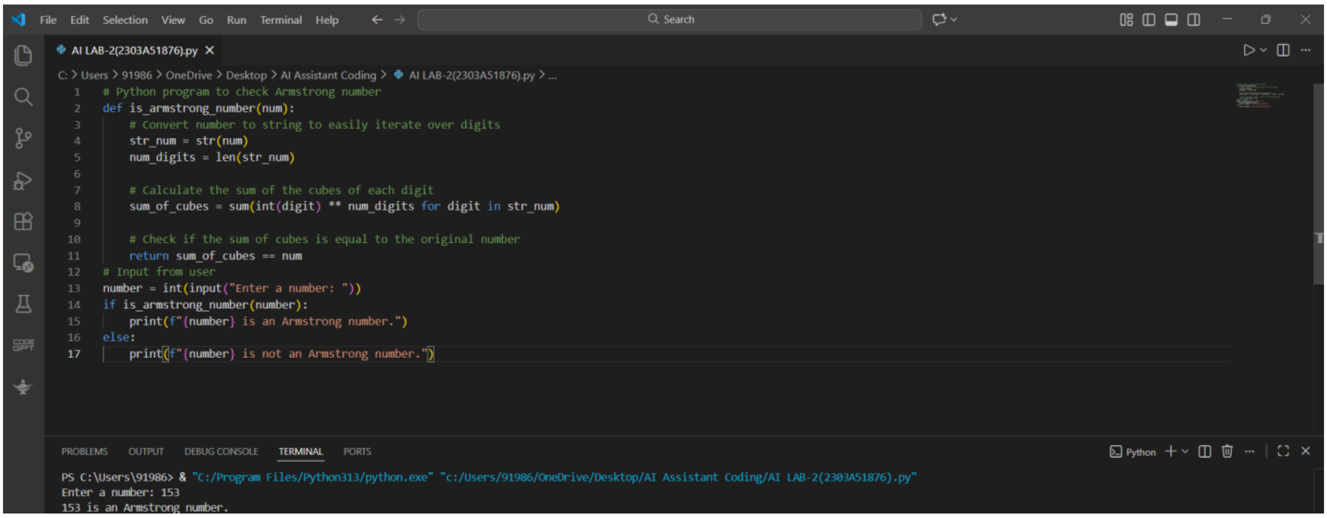
- Choose the right tool based on their learning or productivity requirements.
- Understand the strengths and limitations of each AI platform.
- Improve both conceptual understanding and coding efficiency.

Task-2


colab.research.google.com/drive/1WTVEaRB45Rb0v-xQbd83dCCKZpUWOINU#scrollTo=e8997579
Untitled187.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM
Disk
[12]
3s
Variables
Terminal
5:25 PM
Python 3

```python
def is_armstrong(number):
    """
    Checks if a given number is an Armstrong number.
    An Armstrong number is a number that is equal to the sum of its own digits
    each raised to the power of the number of digits.
    Args:
        number (int): The integer to check.
    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    if not isinstance(number, int) or number < 0:
        return False # Armstrong numbers are typically positive integers

    # Convert the number to a string to easily get digits and count them
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0
    temp_number = number # Use a temporary variable to extract digits
    while temp_number > 0:
        digit = temp_number % 10  # Get the last digit
        sum_of_powers += digit ** num_digits  # Add digit raised to the power of num_digits
        temp_number //= 10  # Remove the last digit
    return sum_of_powers == number
# --- Example Usage ---
n=int(input("Enter the number: "))
print(is_armstrong(n))
```

```
Enter the number: 153
True
```


File Edit Selection View Go Run Terminal Help
Search
AI LAB-2(2303A51876).py
C: > Users > 91986 > OneDrive > Desktop > AI Assistant Coding > AI LAB-2(2303A51876).py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python

```python
# Python program to check Armstrong number
def is_armstrong_number(num):
    # Convert number to string to easily iterate over digits
    str_num = str(num)
    num_digits = len(str_num)

    # Calculate the sum of the cubes of each digit
    sum_of_cubes = sum(int(digit) ** num_digits for digit in str_num)

    # Check if the sum of cubes is equal to the original number
    return sum_of_cubes == num
# Input from user
number = int(input("Enter a number: "))
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

```
PS C:\Users\91986> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/AI LAB-2(2303A51876).py"
Enter a number: 153
153 is an Armstrong number.
```

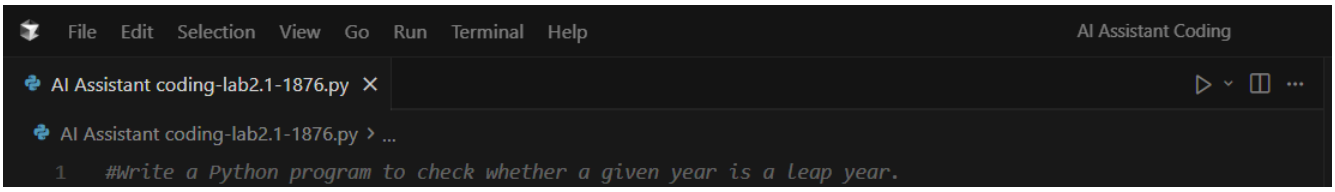
3Q) Task 3: Leap Year Validation Using Cursor AI

- **Scenario:**
  You are validating a calendar module for a backend system.
- **Task:**
  Use Cursor AI to generate a Python program that checks whether a given year is a leap year.
  Use at least two different prompts and observe changes in code.
- **Expected Output:**
  - **Two versions of code**
  - **Sample inputs/outputs**
  - **Brief comparison**

Prompt 1 (Cursor AI):

#Write a Python program to check whether a given year is a leap year.

Screenshot:


File
Edit
Selection
View
Go
Run
Terminal
Help
AI Assistant Coding
AI Assistant coding-lab2.1-1876.py
AI Assistant coding-lab2.1-1876.py > ...

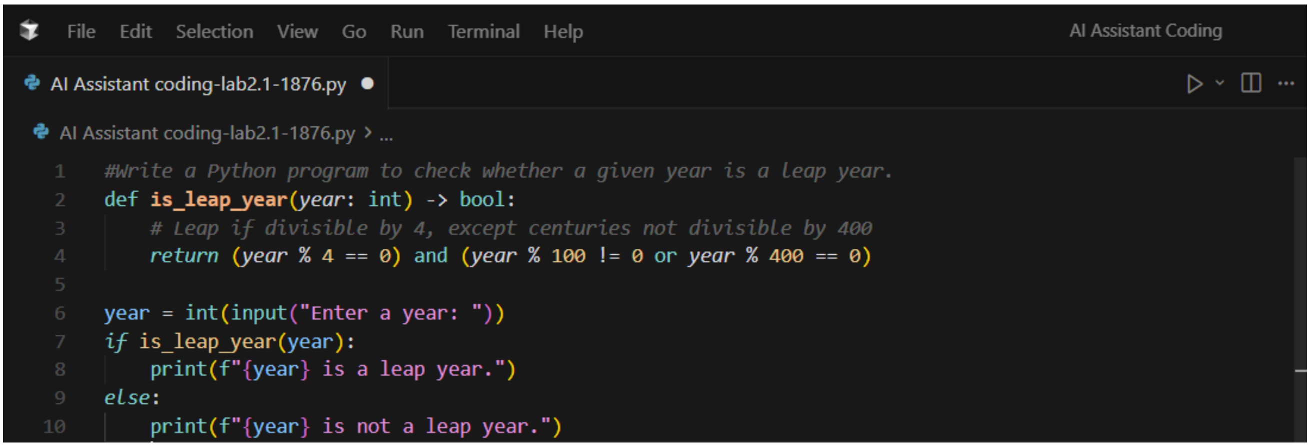
```
#Write a Python program to check whether a given year is a leap year.
```

Code:

```
def is_leap_year(year: int) -> bool:
    # Leap if divisible by 4, except centuries not divisible by 400
    return (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)
year = int(input("Enter a year: "))
if is_leap_year(year):
    print(f"{year} is a leap year.")
```

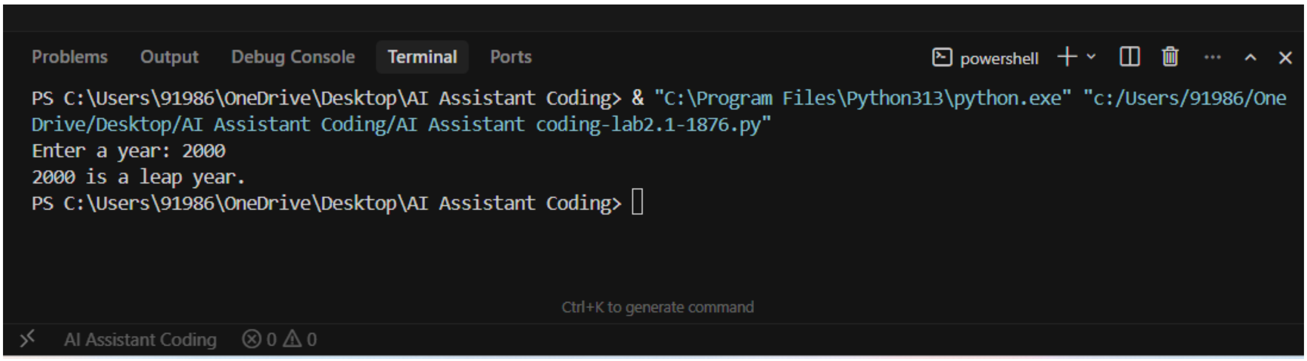
```
else:

    print(f"{year} is not a leap year.")
```

Code Screenshot:


File
Edit
Selection
View
Go
Run
Terminal
Help
AI Assistant Coding
AI Assistant coding-lab2.1-1876.py
AI Assistant coding-lab2.1-1876.py > ...

```python
#Write a Python program to check whether a given year is a leap year.
def is_leap_year(year: int) -> bool:
    # Leap if divisible by 4, except centuries not divisible by 400
    return (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)

year = int(input("Enter a year: "))
if is_leap_year(year):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

Output:

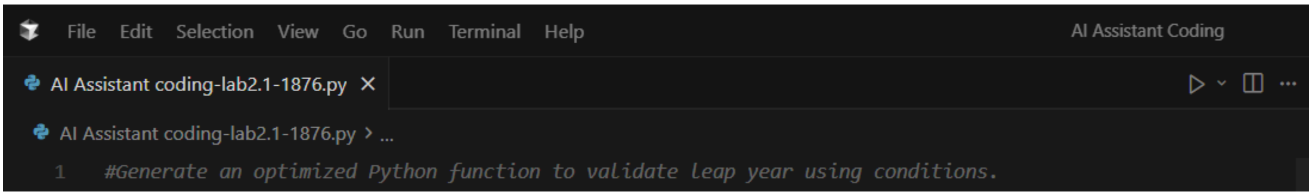


```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python313\python.exe" "c:/Users/91986/One
Drive/Desktop/AI Assistant Coding/AI Assistant coding-lab2.1-1876.py"
Enter a year: 2000
2000 is a leap year.
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> 
```

Prompt 2 (Cursor AI):

#Generate an optimized Python function to validate leap year using conditions.

Screenshot:

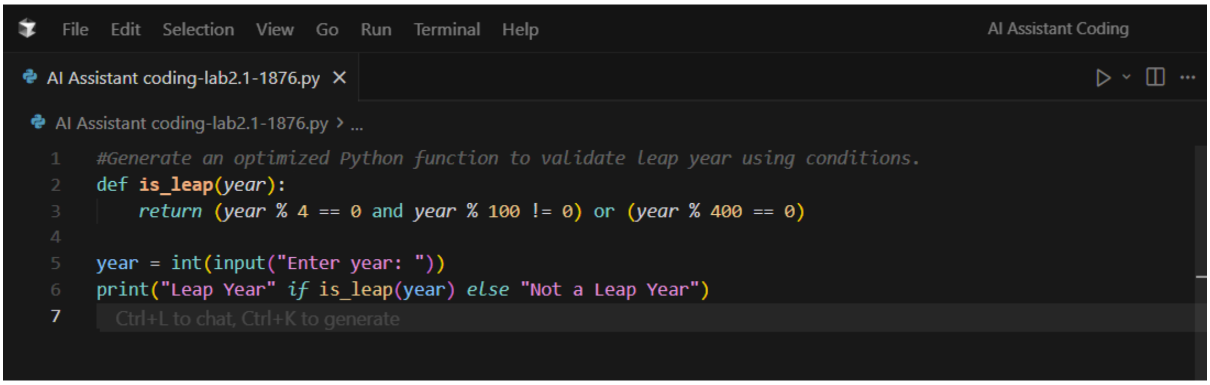

File
Edit
Selection
View
Go
Run
Terminal
Help
AI Assistant Coding
AI Assistant coding-lab2.1-1876.py
AI Assistant coding-lab2.1-1876.py > ...

```python
#Generate an optimized Python function to validate leap year using conditions.
```

Code:

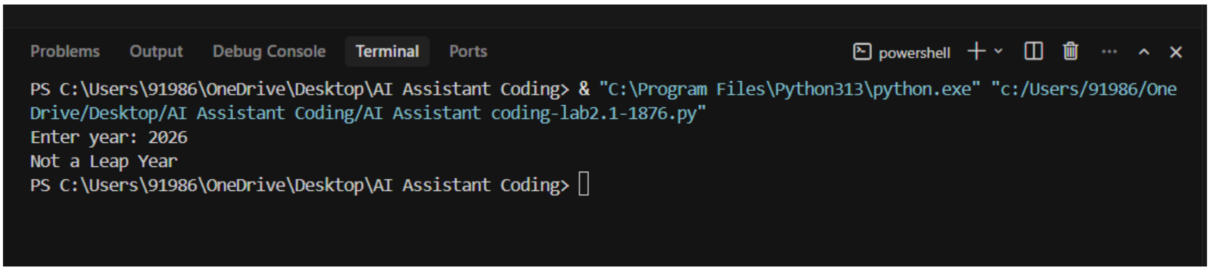
```
def is_leap(year):

    return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

year = int(input("Enter year: "))

print("Leap Year" if is_leap(year) else "Not a Leap Year")
```

Code Screenshot:


File
Edit
Selection
View
Go
Run
Terminal
Help
AI Assistant Coding
AI Assistant coding-lab2.1-1876.py
AI Assistant coding-lab2.1-1876.py > ...
Ctrl+L to chat, Ctrl+K to generate

```
#Generate an optimized Python function to validate leap year using conditions.
def is_leap(year):
    return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

year = int(input("Enter year: "))
print("Leap Year" if is_leap(year) else "Not a Leap Year")
```

Output:



```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python313\python.exe" "c:/Users/91986/One
Drive/Desktop/AI Assistant Coding/AI Assistant coding-lab2.1-1876.py"
Enter year: 2026
Not a Leap Year
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> 
```

Scenario:

Leap year validation is required in calendar systems.

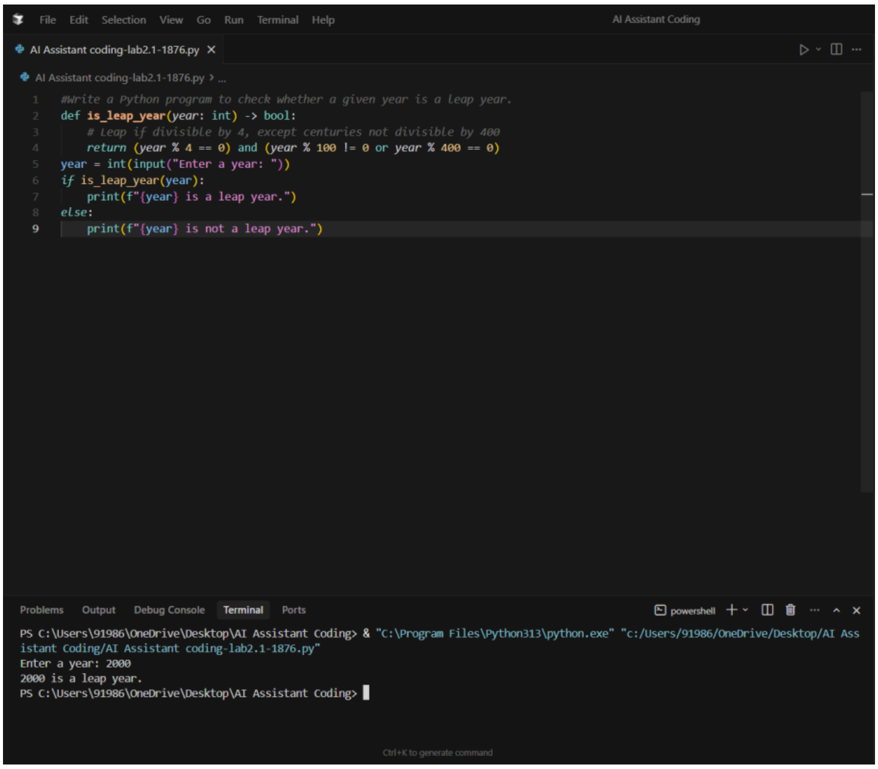
Explanation:

Leap year validation requires multiple conditions for accuracy.
The first prompt generates basic logic, while the second prompt improves structure and correctness.
Cursor AI adapts the code based on prompt detail, demonstrating prompt engineering effectiveness.

Justification:

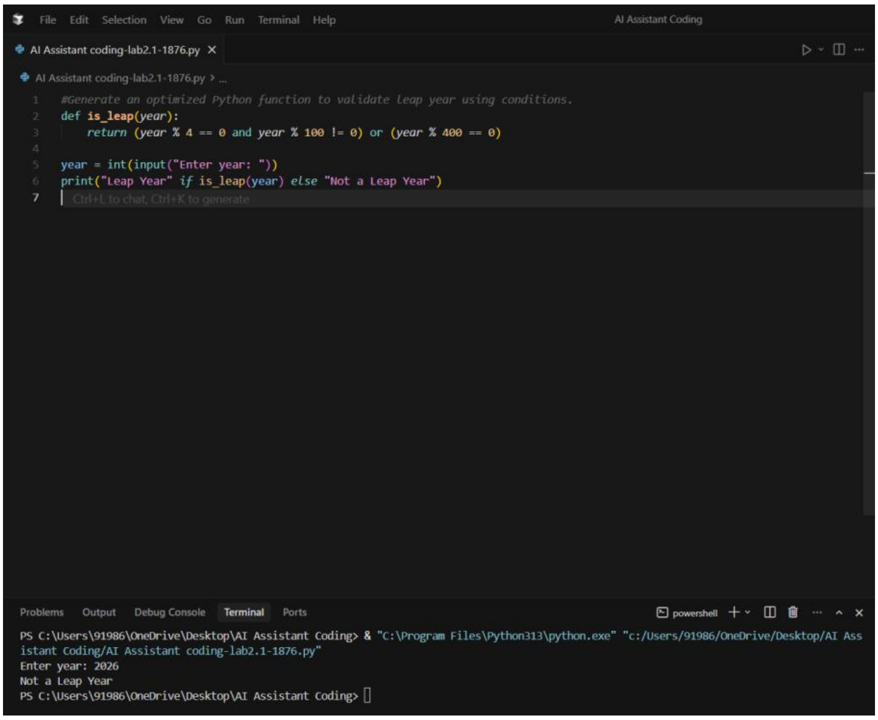
This task highlights how AI improves code quality when prompts are refined. Cursor AI supports refactoring and optimization, making it useful for professional development environments.

Task-3



```python
#Write a Python program to check whether a given year is a leap year.
def is_leap_year(year: int) -> bool:
    # Leap if divisible by 4, except centuries not divisible by 400
    return (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)
year = int(input("Enter a year: "))
if is_leap_year(year):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python313\python.exe" "c:/Users/91986/OneDrive/Desktop/AI Ass
istant Coding/AI Assistant coding-lab2.1-1876.py"
Enter a year: 2000
2000 is a leap year.
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```


File
Edit
Selection
View
Go
Run
Terminal
Help
AI Assistant Coding
AI Assistant coding-lab2.1-1876.py
AI Assistant coding-lab2.1-1876.py › ...
Ctrl+L to chat, Ctrl+K to generate
Problems
Output
Debug Console
Terminal
Ports
powershell

```python
#Generate an optimized Python function to validate leap year using conditions.
def is_leap(year):
    return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

year = int(input("Enter year: "))
print("Leap Year" if is_leap(year) else "Not a Leap Year")
```

```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python313\python.exe" "c:/Users/91986/OneDrive/Desktop/AI Ass
istant Coding/AI Assistant coding-lab2.1-1876.py"
Enter year: 2026
Not a Leap Year
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

Task 4: Student Logic + AI Refactoring (Odd/Even Sum)

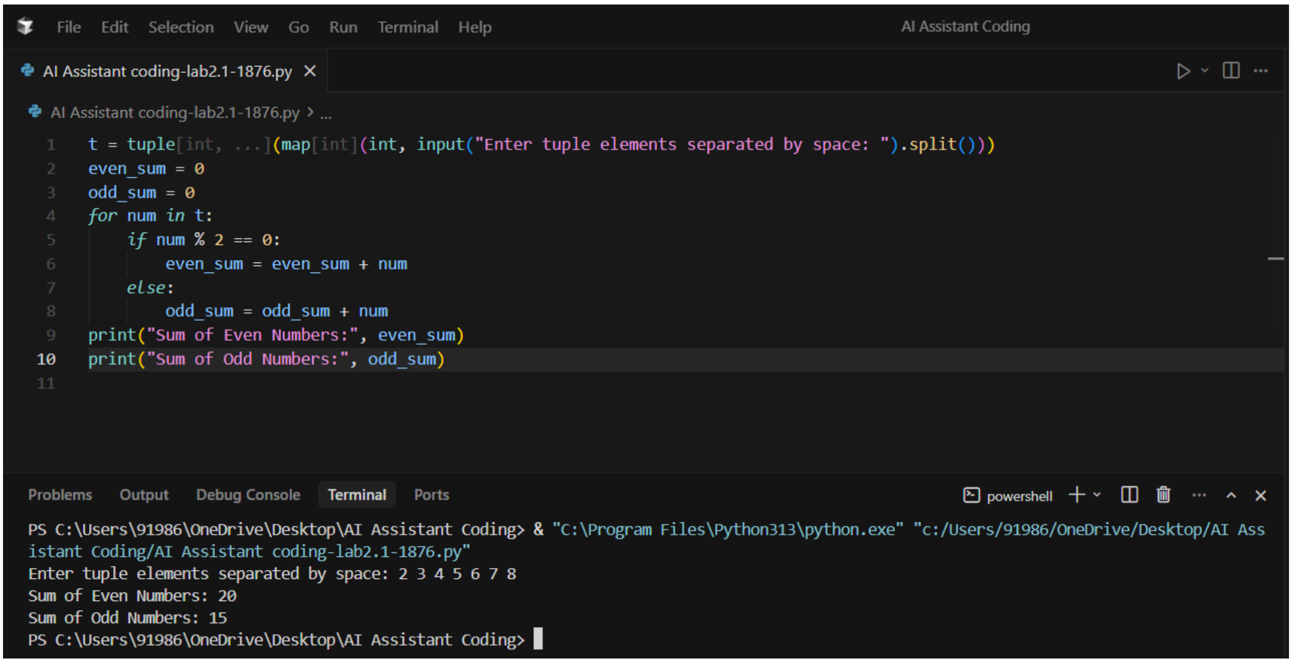
- **Scenario:**
  Company policy requires developers to write logic before using AI.
- **Task:**
  Write a Python program that calculates the sum of odd and even numbers in a tuple, then refactor it using any AI tool.
- **Expected Output:**
  - Original code
  - Refactored code
  - Explanation of improvements

Prompt 1 (For Student-Written Logic – Self Coding):

Problem:

Calculate the sum of odd and even numbers present in a tuple.

Screenshot:



```python
t = tuple[int, ...](map[int](int, input("Enter tuple elements separated by space: ").split()))
even_sum = 0
odd_sum = 0
for num in t:
    if num % 2 == 0:
        even_sum = even_sum + num
    else:
        odd_sum = odd_sum + num
print("Sum of Even Numbers:", even_sum)
print("Sum of Odd Numbers:", odd_sum)
```

```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python313\python.exe" "c:/Users/91986/OneDrive/Desktop/AI Ass
istant Coding/AI Assistant coding-lab2.1-1876.py"
Enter tuple elements separated by space: 2 3 4 5 6 7 8
Sum of Even Numbers: 20
Sum of Odd Numbers: 15
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> 
```

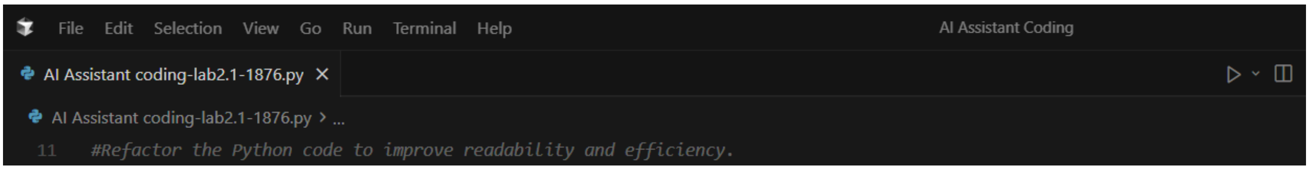
Code:

```
t = tuple(map(int, input("Enter tuple elements separated by space: ").split()))
even_sum = 0
odd_sum = 0
for num in t:
    if num % 2 == 0:
        even_sum = even_sum + num
    else:
        odd_sum = odd_sum + num
print("Sum of Even Numbers:", even_sum)
print("Sum of Odd Numbers:", odd_sum)
```

AI Refactoring Prompt:

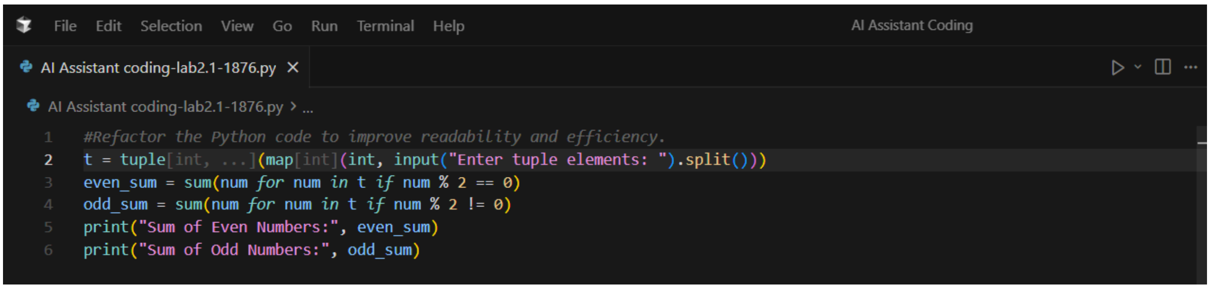
#Refactor the Python code to improve readability and efficiency.

Screenshot:


File
Edit
Selection
View
Go
Run
Terminal
Help
AI Assistant Coding
AI Assistant coding-lab2.1-1876.py
AI Assistant coding-lab2.1-1876.py > ...

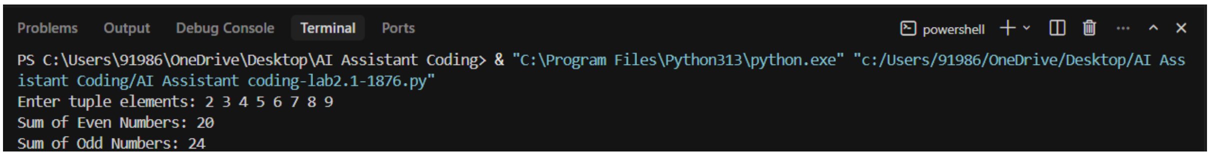
```
#Refactor the Python code to improve readability and efficiency.
```

Code Screenshot:


File
Edit
Selection
View
Go
Run
Terminal
Help
AI Assistant Coding
AI Assistant coding-lab2.1-1876.py
AI Assistant coding-lab2.1-1876.py > ...

```
#Refactor the Python code to improve readability and efficiency.
t = tuple[int, ...](map[int](int, input("Enter tuple elements: ").split()))
even_sum = sum(num for num in t if num % 2 == 0)
odd_sum = sum(num for num in t if num % 2 != 0)
print("Sum of Even Numbers:", even_sum)
print("Sum of Odd Numbers:", odd_sum)
```

Output:

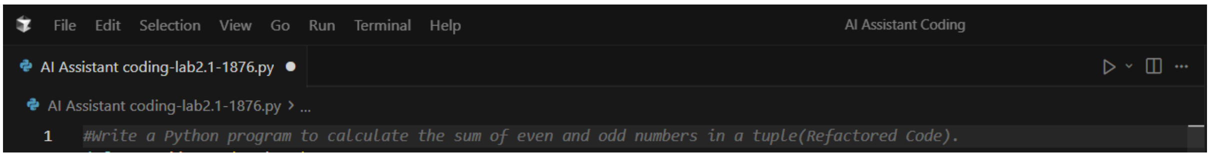


```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python313\python.exe" "c:/Users/91986/OneDrive/Desktop/AI Ass
istant Coding/AI Assistant coding-lab2.1-1876.py"
Enter tuple elements: 2 3 4 5 6 7 8 9
Sum of Even Numbers: 20
Sum of Odd Numbers: 24
```

Refactored Code Prompt:

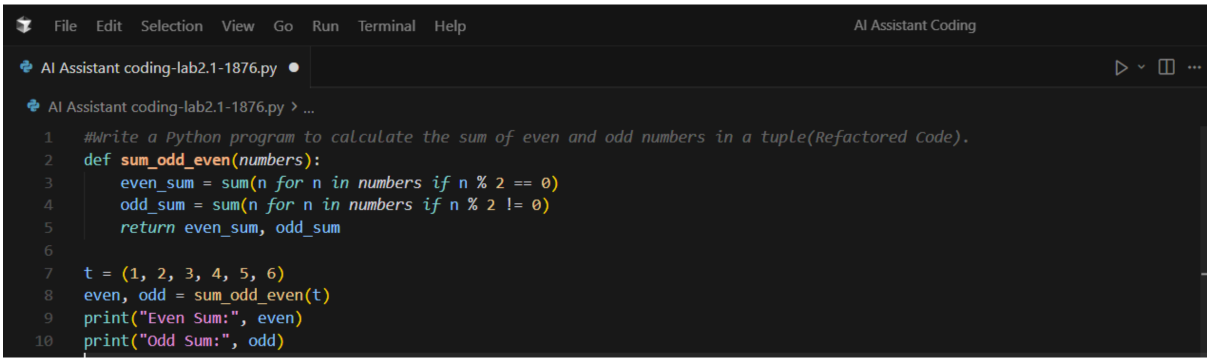
#Write a Python program to calculate the sum of even and odd numbers in a tuple(Refactored Code).

Screenshot:


File Edit Selection View Go Run Terminal Help
AI Assistant Coding
AI Assistant coding-lab2.1-1876.py
AI Assistant coding-lab2.1-1876.py > ...

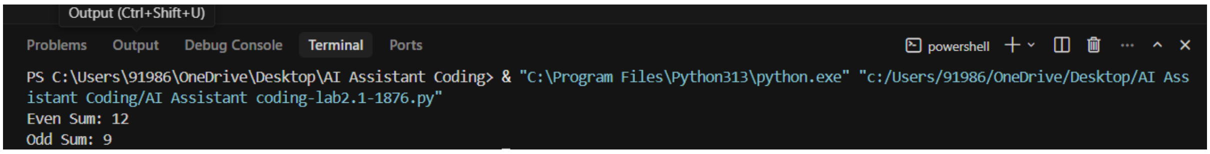
```
#Write a Python program to calculate the sum of even and odd numbers in a tuple(Refactored Code).
```

Code Screenshot:


File Edit Selection View Go Run Terminal Help
AI Assistant Coding
AI Assistant coding-lab2.1-1876.py
AI Assistant coding-lab2.1-1876.py > ...

```python
#Write a Python program to calculate the sum of even and odd numbers in a tuple(Refactored Code).
def sum_odd_even(numbers):
    even_sum = sum(n for n in numbers if n % 2 == 0)
    odd_sum = sum(n for n in numbers if n % 2 != 0)
    return even_sum, odd_sum

t = (1, 2, 3, 4, 5, 6)
even, odd = sum_odd_even(t)
print("Even Sum:", even)
print("Odd Sum:", odd)
```

Output:



```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python313\python.exe" "c:/Users/91986/OneDrive/Desktop/AI Ass
istant Coding/AI Assistant coding-lab2.1-1876.py"
Even Sum: 12
Odd Sum: 9
```

Scenario

Developers must write logic before AI refactoring.

Explanation:

The original code uses loops and conditional statements to separate odd and even values.
The AI-refactored code uses Python built-in functions and generator expressions.
This results in cleaner, more efficient, and reusable code without changing the logic.

Justification:

Writing logic manually ensures conceptual understanding.
AI refactoring improves code readability and performance.
This approach aligns with industry standards where AI assists but does not replace developer thinking.