School of Computer Science and Artificial Intelligence

<u>**Lab Assignment # 02**</u>

Course Title : **AI Assistant Coding**
Name of Student : **BG Sreevani**
Enrollment No. : **2303A54066**
Batch No. : **48**

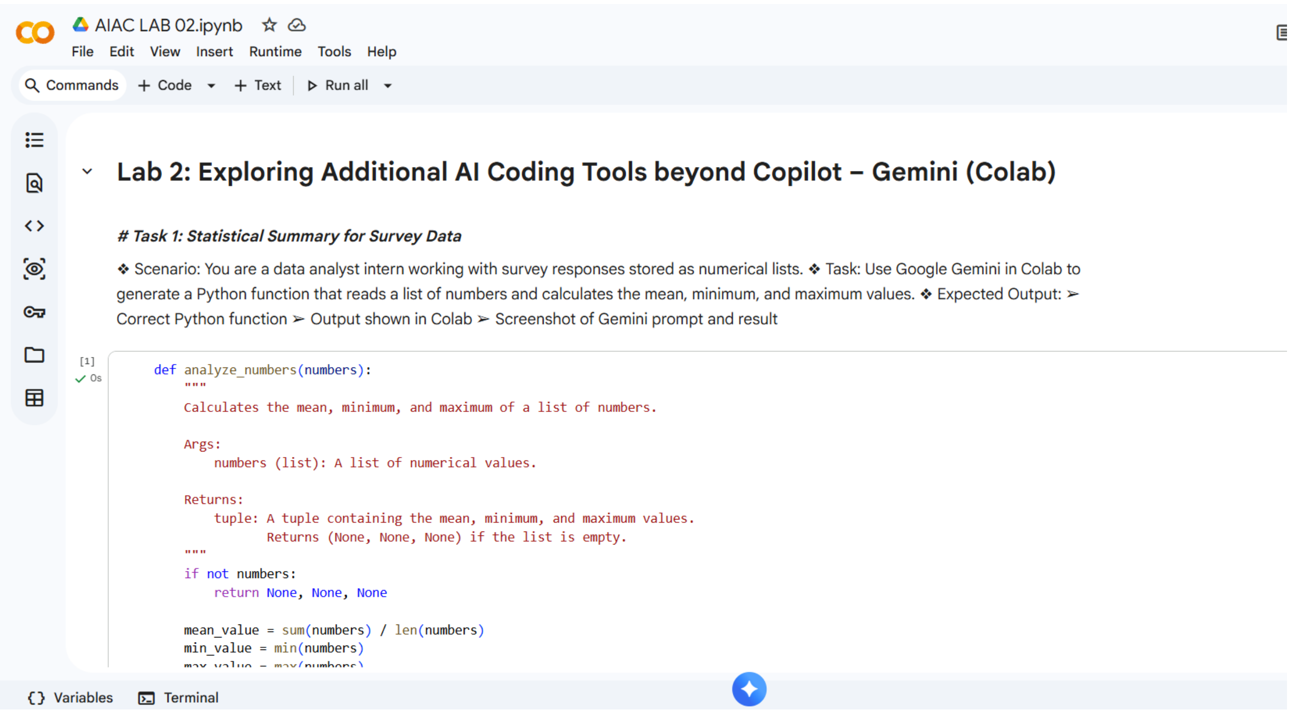
Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI

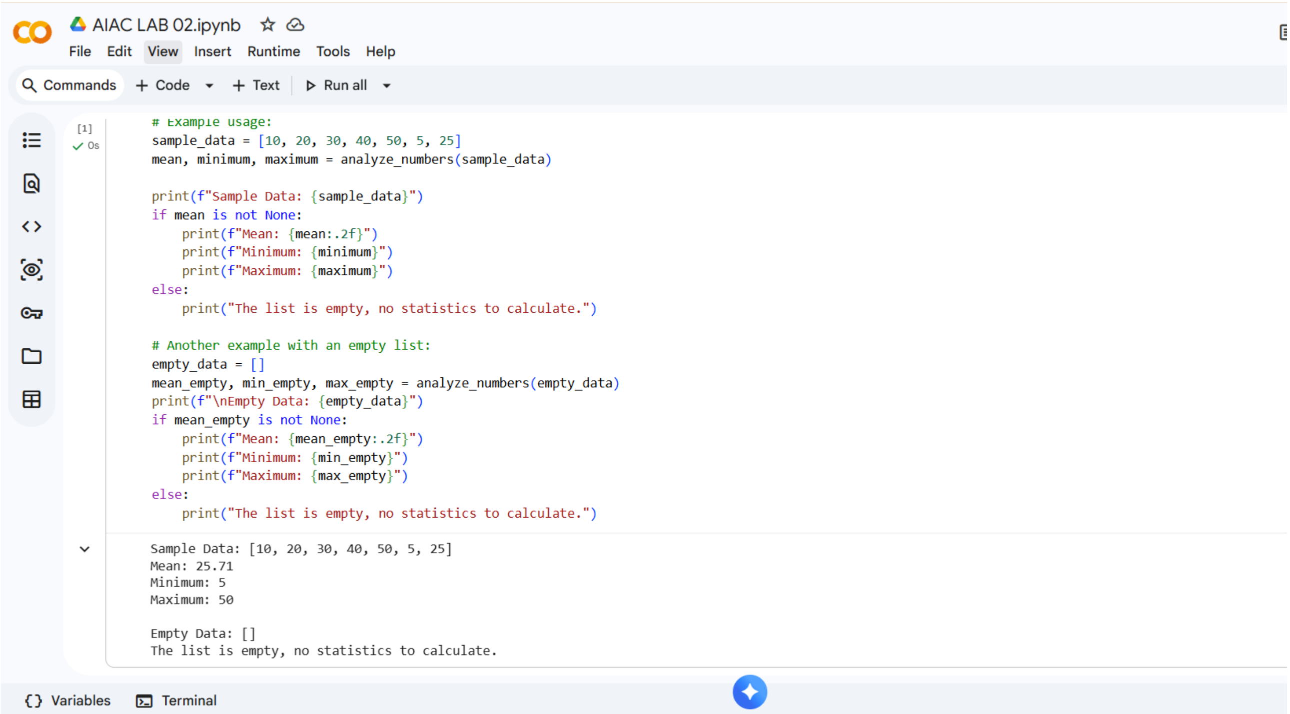
Task 1: Statistical Summary for Survey Data

❖ ***Scenario: You are a data analyst intern working with survey responses stored as numerical lists.***

• **Prompt used:** Write a Python function that takes a list of numbers and returns the mean, minimum, and maximum values.

• **Screenshot of Generated Code:**






```python
# Example usage:
sample_data = [10, 20, 30, 40, 50, 5, 25]
mean, minimum, maximum = analyze_numbers(sample_data)

print(f"Sample Data: {sample_data}")
if mean is not None:
    print(f"Mean: {mean:.2f}")
    print(f"Minimum: {minimum}")
    print(f"Maximum: {maximum}")
else:
    print("The list is empty, no statistics to calculate.")

# Another example with an empty list:
empty_data = []
mean_empty, min_empty, max_empty = analyze_numbers(empty_data)
print(f"\nEmpty Data: {empty_data}")
if mean_empty is not None:
    print(f"Mean: {mean_empty:.2f}")
    print(f"Minimum: {min_empty}")
    print(f"Maximum: {max_empty}")
else:
    print("The list is empty, no statistics to calculate.")
```

```
Sample Data: [10, 20, 30, 40, 50, 5, 25]
Mean: 25.71
Minimum: 5
Maximum: 50

Empty Data: []
The list is empty, no statistics to calculate.
```

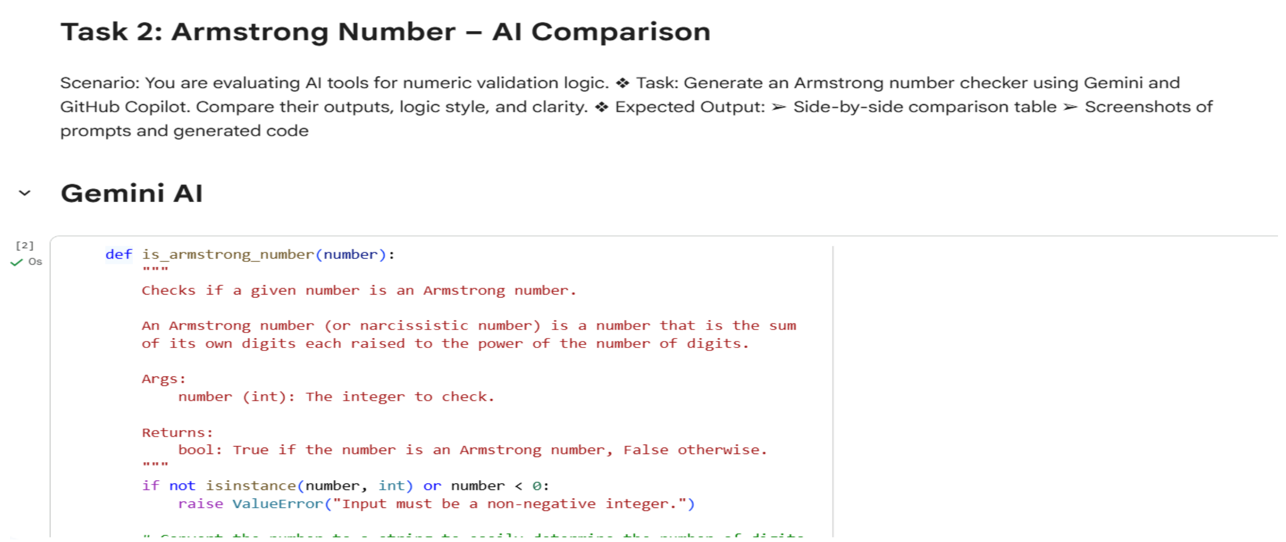

• Short Explanation of Logic:
The mean represents the average of a set of numbers. It is obtained by adding all the values in the list and dividing the sum by the total number of elements. The minimum value is the smallest number present in the list, and the maximum value is the largest number present in the list. These values are calculated using Python's built-in min() and max() functions. The function accepts a list of numerical values as input, computes the mean, minimum.

Task 2: Armstrong Number – AI Comparison

❖ ***Scenario: You are evaluating AI tools for numeric validation logic.***

• Prompt used (Gemini): "Write a Python program to check whether a given number is an Armstrong number."
• Prompt used (Copilot): "Write a Python program to check whether a given number is an Armstrong number."

• Screenshot of Generated Code:



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

Gemini AI

```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 (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:
        raise ValueError("Input must be a non-negative integer.")
```

AIAC LAB 02.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```python
    num_str = str(number)
    n_digits = len(num_str)

    # Calculate the sum of each digit raised to the power of the total number of digits
    sum_of_powers = 0
    for digit_char in num_str:
        digit = int(digit_char)
        sum_of_powers += digit ** n_digits

    # Compare this sum with the original number
    return sum_of_powers == number

# Example Usage:
print("Checking Armstrong numbers:")
print(f"153 is Armstrong: {is_armstrong_number(153)}") # Expected: True (1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153)
print(f"9 is Armstrong: {is_armstrong_number(9)}")    # Expected: True (9^1 = 9)
print(f"370 is Armstrong: {is_armstrong_number(370)}")  # Expected: True (3^3 + 7^3 + 0^3 = 27 + 343 + 0 = 370)
print(f"1634 is Armstrong: {is_armstrong_number(1634)}") # Expected: True (1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634)
print(f"123 is Armstrong: {is_armstrong_number(123)}")  # Expected: False (1^3 + 2^3 + 3^3 = 1 + 8 + 27 = 36 != 123)
print(f"54748 is Armstrong: {is_armstrong_number(54748)}") # Expected: True
```

```
Checking Armstrong numbers:
153 is Armstrong: True
9 is Armstrong: True
370 is Armstrong: True
1634 is Armstrong: True
123 is Armstrong: False
54748 is Armstrong: True
```

Variables Terminal 8:24 PM

AIAC LAB 02.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Git Hub Co-pilot code

```python
def is_armstrong(number: int) -> bool:
    """
    Check if a given number is an Armstrong number.

    An Armstrong number of 'n' digits is a number such that
    the sum of its digits raised to the power 'n' equals the number itself.
    """
    # Convert number to string to easily iterate over digits
    digits = str(number)
    num_digits = len(digits)

    # Compute sum of each digit raised to the power of num_digits
    total = sum(int(digit) ** num_digits for digit in digits)

    return total == number


# Example usage
if __name__ == "__main__":
    test_numbers = [153, 9474, 123, 407]
    for num in test_numbers:
        if is_armstrong(num):
            print(f"{num} is an Armstrong number.")
        else:
            print(f"{num} is NOT an Armstrong number.")
```

Variables Terminal

```
153 is an Armstrong number.
9474 is an Armstrong number.
123 is NOT an Armstrong number.
407 is an Armstrong number.
```

• Side-by-Side Comparison Table:

Feature	Google Gemini	GitHub Copilot
Code structure	Uses a separate function	Written directly in main logic
Readability	Very clear and modular	Simple and straightforward
Logic clarity	Easy to understand with function	Easy but less modular
Beginner friendly	High	Medium
Explanation support	Provides explanation along with code	No explanation, only code
Best use case	Learning and documentation	Fast coding inside editor

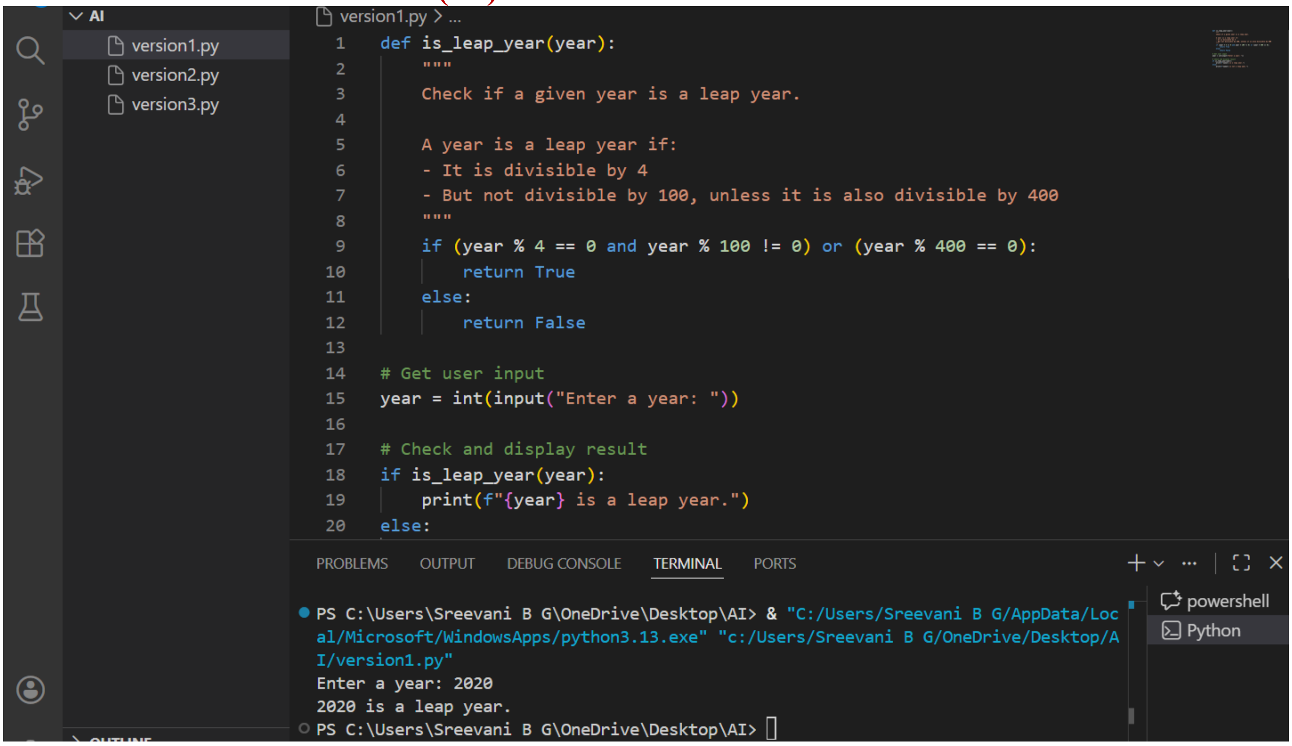
Task 3: Leap Year Validation Using Cursor AI

❖ ***Scenario: You are validating a calendar module for a backend system.***

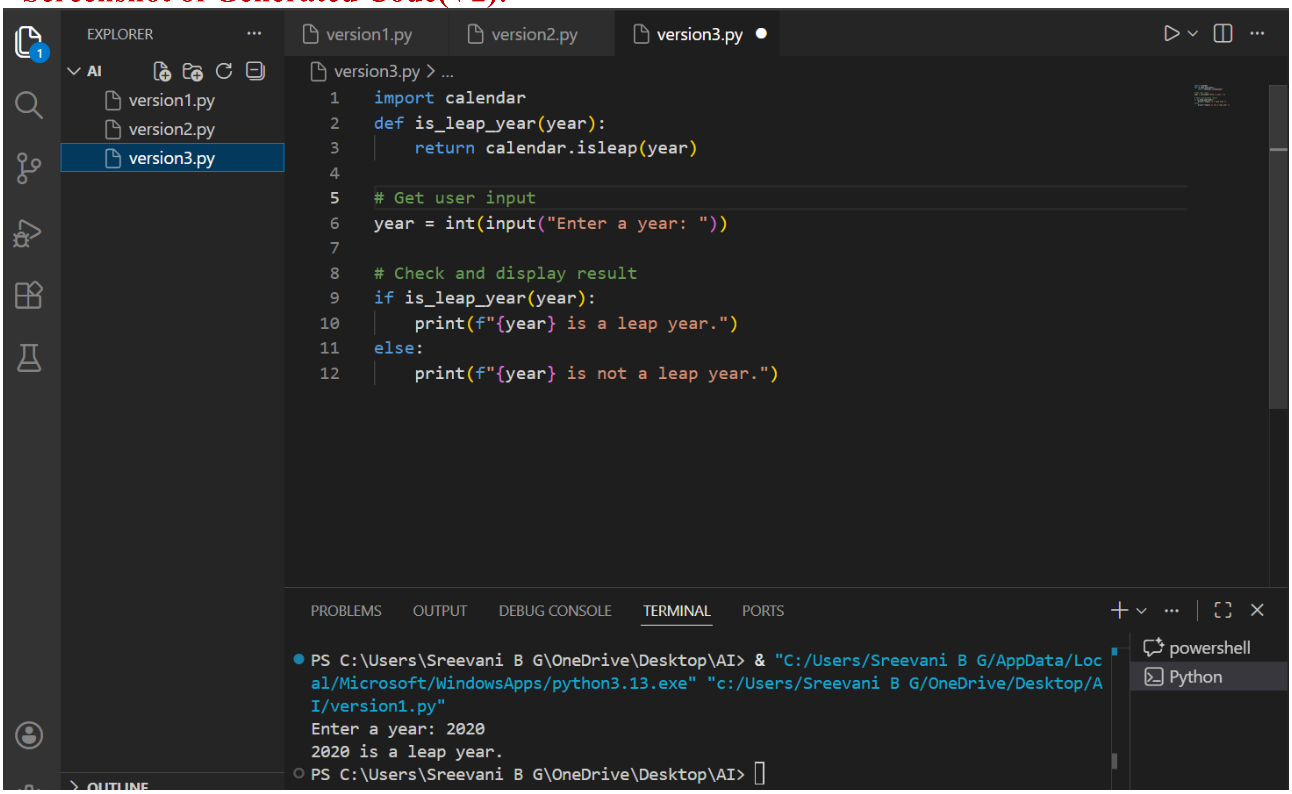
• **Prompt 1:** "Write a Python program to check whether a given year is a leap year."

• **Prompt 2:** "Write an optimized Python program with proper conditions and comments to check whether a year is a leap year."

• **Screenshot of Generated Code(V1):**



• **Screenshot of Generated Code(V2):**



- **Sample Input:** Enter a year: 2026
- **Sample Output:** 2026 is not a leap year.

- **Short Explanation of Logic:**

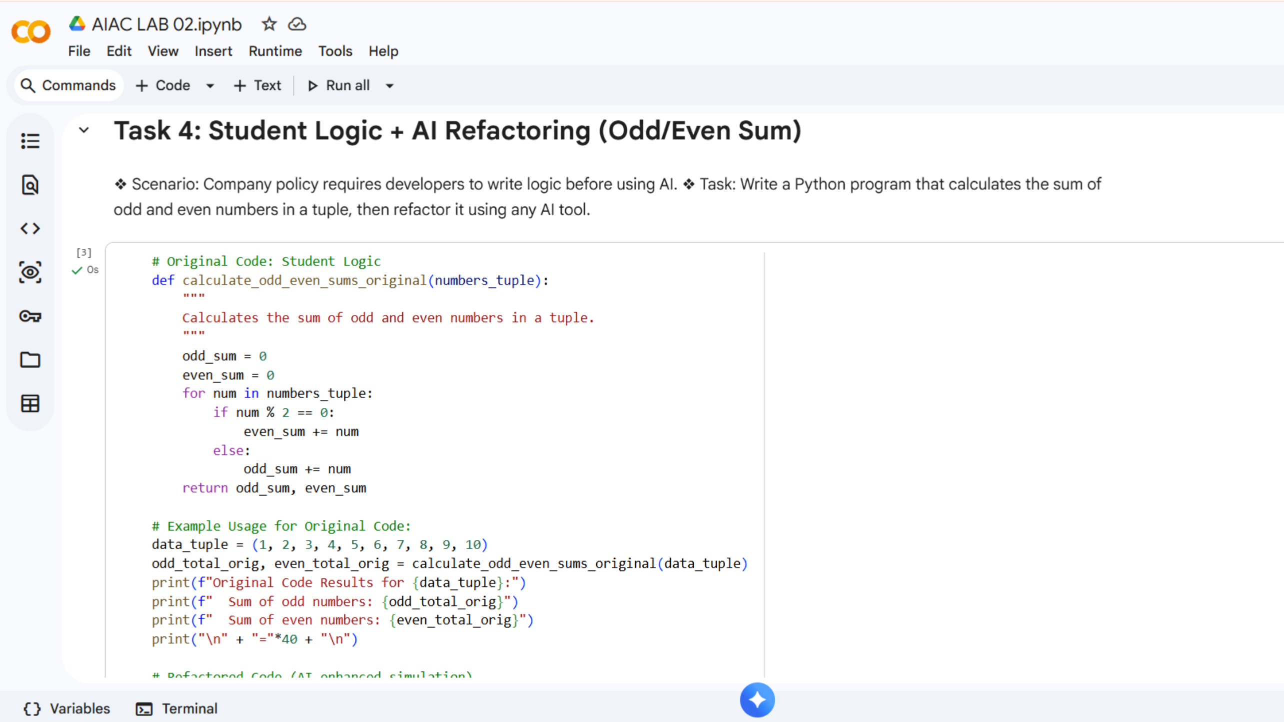
version1.py manually checks leap year rules using modulo arithmetic, making the logic explicit and educational with no dependencies. version3.py uses Python's built-in calendar.isleap(), which is shorter, cleaner, and production-ready but depends on the standard library. Both are fast and correct: the first emphasizes learning the rules, while the second emphasizes simplicity.

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

❖ ***Scenario: Company policy requires developers to write logic before using AI.***

- **Prompt used:** "Refactor this Python code to improve readability and efficiency."

- **Screenshot of Generated Code**



AIAC LAB 02.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

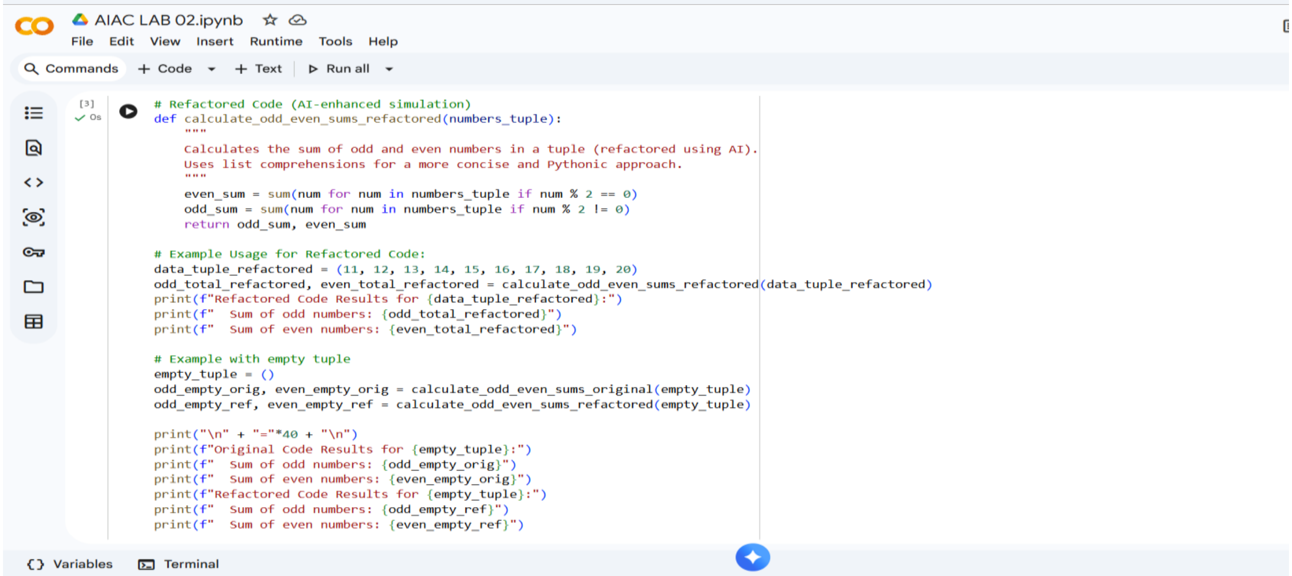
Task 4: Student Logic + AI Refactoring (Odd/Even Sum)

❖ 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.

```
# Original Code: Student Logic
def calculate_odd_even_sums_original(numbers_tuple):
    """
    Calculates the sum of odd and even numbers in a tuple.
    """
    odd_sum = 0
    even_sum = 0
    for num in numbers_tuple:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num
    return odd_sum, even_sum

# Example Usage for Original Code:
data_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
odd_total_orig, even_total_orig = calculate_odd_even_sums_original(data_tuple)
print(f"Original Code Results for {data_tuple}:")
print(f"  Sum of odd numbers: {odd_total_orig}")
print(f"  Sum of even numbers: {even_total_orig}")
print("\n" + "="*40 + "\n")
```

Variables Terminal



AIAC LAB 02.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
# Refactored Code (AI-enhanced simulation)
def calculate_odd_even_sums_refactored(numbers_tuple):
    """
    Calculates the sum of odd and even numbers in a tuple (refactored using AI).
    Uses list comprehensions for a more concise and Pythonic approach.
    """
    even_sum = sum(num for num in numbers_tuple if num % 2 == 0)
    odd_sum = sum(num for num in numbers_tuple if num % 2 != 0)
    return odd_sum, even_sum

# Example Usage for Refactored Code:
data_tuple_refactored = (11, 12, 13, 14, 15, 16, 17, 18, 19, 20)
odd_total_refactored, even_total_refactored = calculate_odd_even_sums_refactored(data_tuple_refactored)
print(f"Refactored Code Results for {data_tuple_refactored}:")
print(f"  Sum of odd numbers: {odd_total_refactored}")
print(f"  Sum of even numbers: {even_total_refactored}")

# Example with empty tuple
empty_tuple = ()
odd_empty_orig, even_empty_orig = calculate_odd_even_sums_original(empty_tuple)
odd_empty_ref, even_empty_ref = calculate_odd_even_sums_refactored(empty_tuple)

print("\n" + "="*40 + "\n")
print(f"Original Code Results for {empty_tuple}:")
print(f"  Sum of odd numbers: {odd_empty_orig}")
print(f"  Sum of even numbers: {even_empty_orig}")
print(f"Refactored Code Results for {empty_tuple}:")
print(f"  Sum of odd numbers: {odd_empty_ref}")
print(f"  Sum of even numbers: {even_empty_ref}")
```

Variables Terminal

AIAC LAB 02.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
print(f"  Sum of even numbers: {even_empty_ref}")
```

```
Original Code Results for (1, 2, 3, 4, 5, 6, 7, 8, 9, 10):
  Sum of odd numbers: 25
  Sum of even numbers: 30

========================================

Refactored Code Results for (11, 12, 13, 14, 15, 16, 17, 18, 19, 20):
  Sum of odd numbers: 75
  Sum of even numbers: 80

========================================

Original Code Results for ():
  Sum of odd numbers: 0
  Sum of even numbers: 0
Refactored Code Results for ():
  Sum of odd numbers: 0
  Sum of even numbers: 0
```

• Explanation

write Explanation of improvements for original code and refracted code

Empty cell

Certainly! Here's a summary of the improvements between the original and refactored code for calculating the sum of odd and even numbers:

Original Code (`calculate_odd_even_sums_original`):

- **Explicit Loop and Conditionals:** Uses a traditional `for` loop with `if-else` statements to iterate and check each number, explicitly managing `odd_sum` and `even_sum` variables.
- **Readability (Beginner):** Easy to understand for those new to programming due to its step-by-step nature.
- **Verbosity:** More lines of code are required to achieve the desired outcome.

Refactored Code (`calculate_odd_even_sums_refactored`):

- **Conciseness & Pythonic Style:** Achieves the same result in significantly fewer lines of code by using generator expressions within the built-in `sum()` function. This is considered a more idiomatic and expressive Python approach.
- **Readability (Experienced Pythonista):** For experienced Python developers, this version is often more readable as it clearly states *what* is being aggregated (sum of even numbers, sum of odd numbers) rather than *how* the iteration and conditional logic are handled.