AI ASSISTED CODING

ASSIGNMENT-2

Name: Harshitha Guda

H.T.No:2303A51102

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.

❖ Task:

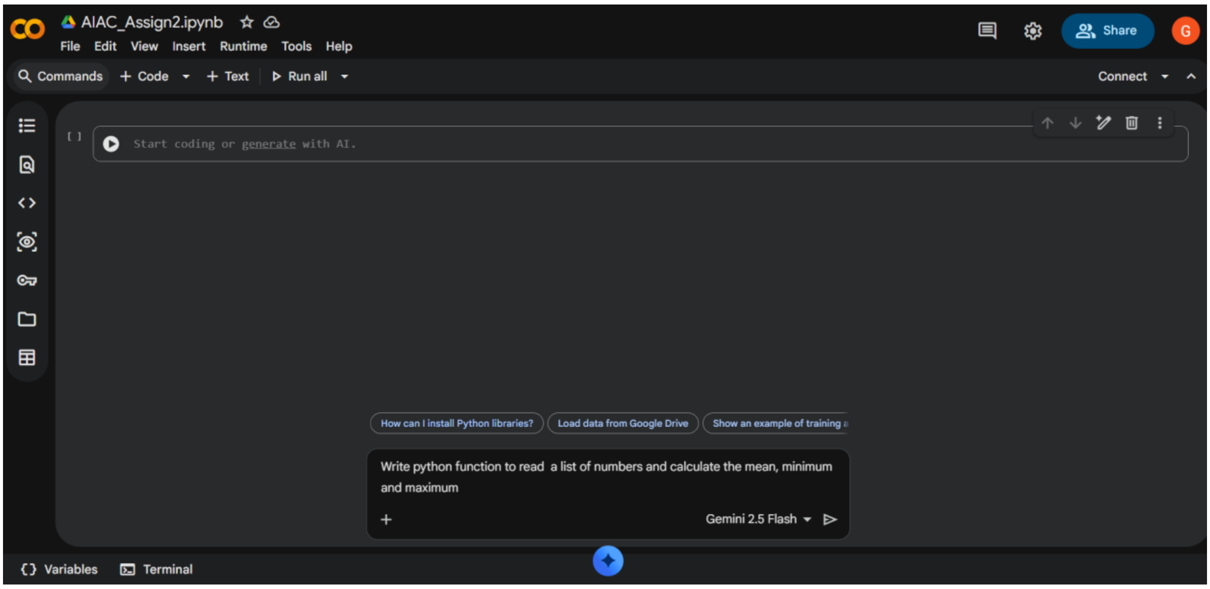
Use Google Gemini in Colab to generate a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.

❖ Expected Output:

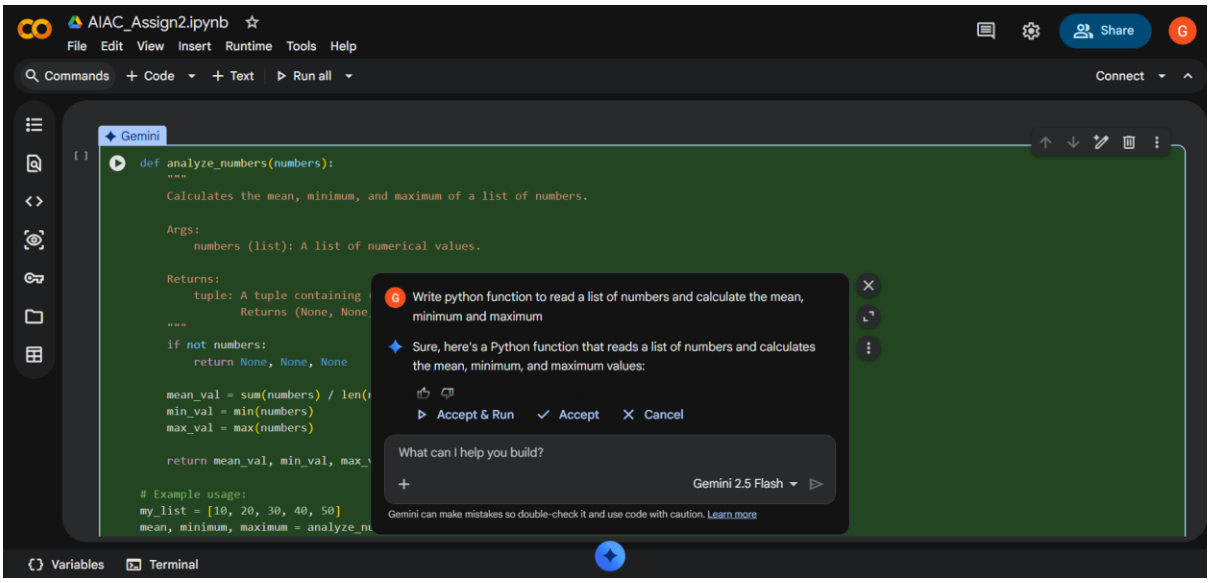
➢ Correct Python function

➢ Output shown in Colab

➢ Screenshot of Gemini prompt and result


AIAC_Assign2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text ▷ Run all
Connect
Start coding or generate with AI.
How can I install Python libraries?
Load data from Google Drive
Show an example of training a
Gemini 2.5 Flash
Variables
Terminal

Write python function to read a list of numbers and calculate the mean, minimum and maximum

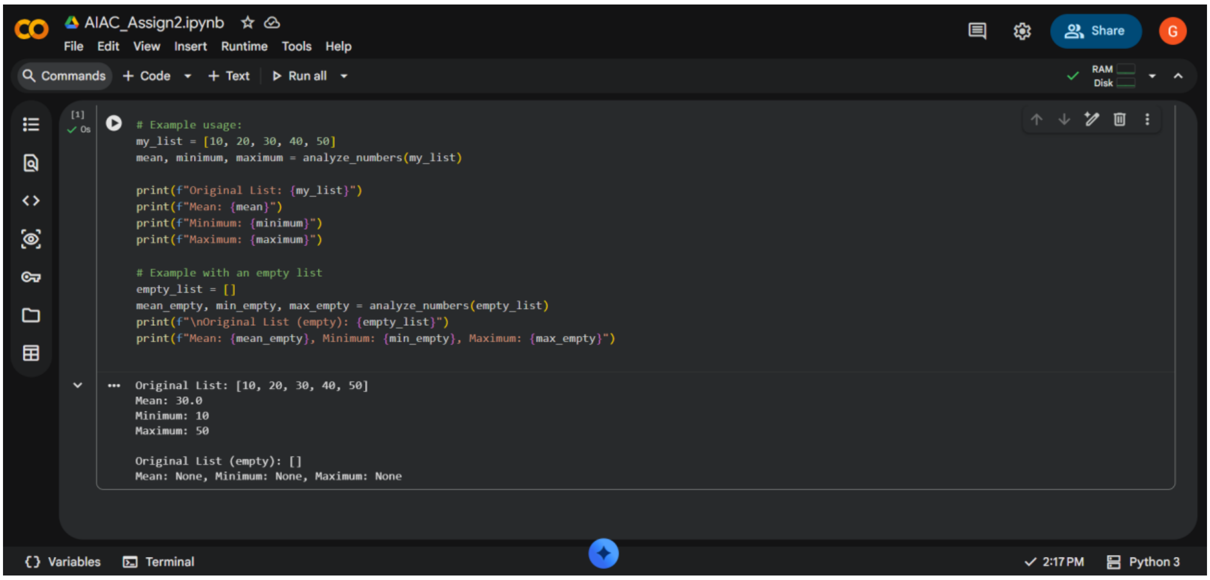

AIAC_Assign2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Connect
Share
Gemini
Variables Terminal

```
def analyze_numbers(numbers):
    """
    Calculates the mean, minimum, and maximum of a list of numbers.

    Args:
        numbers (list): A list of numerical values.

    Returns:
        tuple: A tuple containing 
               Returns (None, None
    """
    if not numbers:
        return None, None, None

    mean_val = sum(numbers) / len(
    min_val = min(numbers)
    max_val = max(numbers)

    return mean_val, min_val, max_

# Example usage:
my_list = [10, 20, 30, 40, 50]
mean, minimum, maximum = analyze_nu
```

Write python function to read a list of numbers and calculate the mean, minimum and maximum

Sure, here's a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values:

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


AIAC_Assign2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
RAM
Disk
Share
[1]
0s
Variables Terminal
2:17 PM
Python 3

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

print(f"Original List: {my_list}")
print(f"Mean: {mean}")
print(f"Minimum: {minimum}")
print(f"Maximum: {maximum}")

# Example with an empty list
empty_list = []
mean_empty, min_empty, max_empty = analyze_numbers(empty_list)
print(f"\nOriginal List (empty): {empty_list}")
print(f"Mean: {mean_empty}, Minimum: {min_empty}, Maximum: {max_empty}")
```

```
Original List: [10, 20, 30, 40, 50]
Mean: 30.0
Minimum: 10
Maximum: 50

Original List (empty): []
Mean: None, Minimum: None, Maximum: None
```

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.

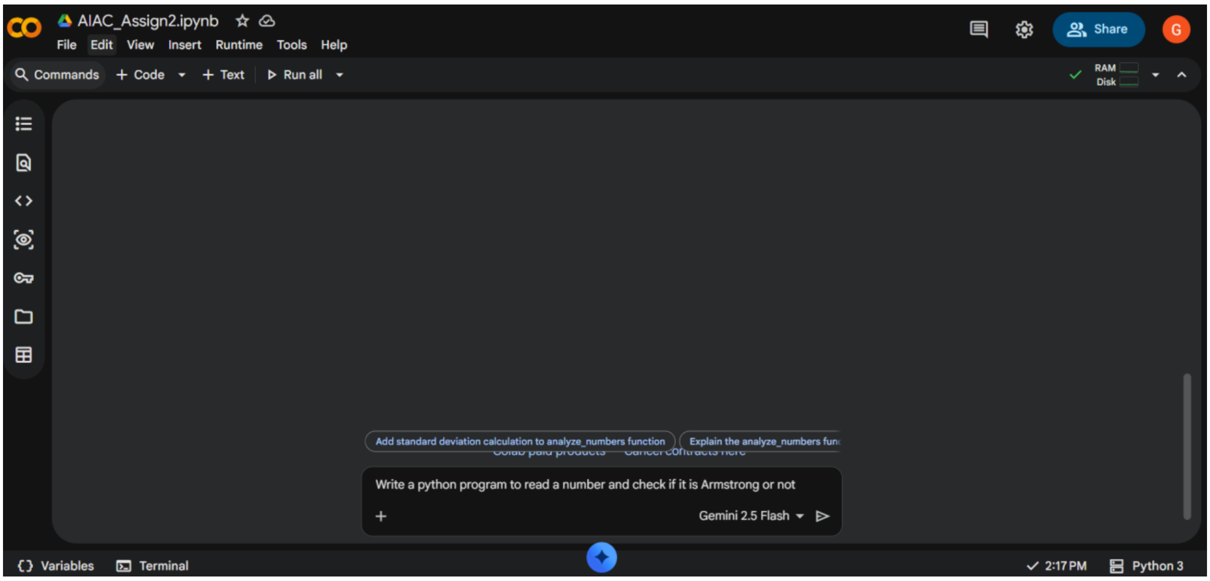
Compare their outputs, logic style, and clarity.

❖ Expected Output:

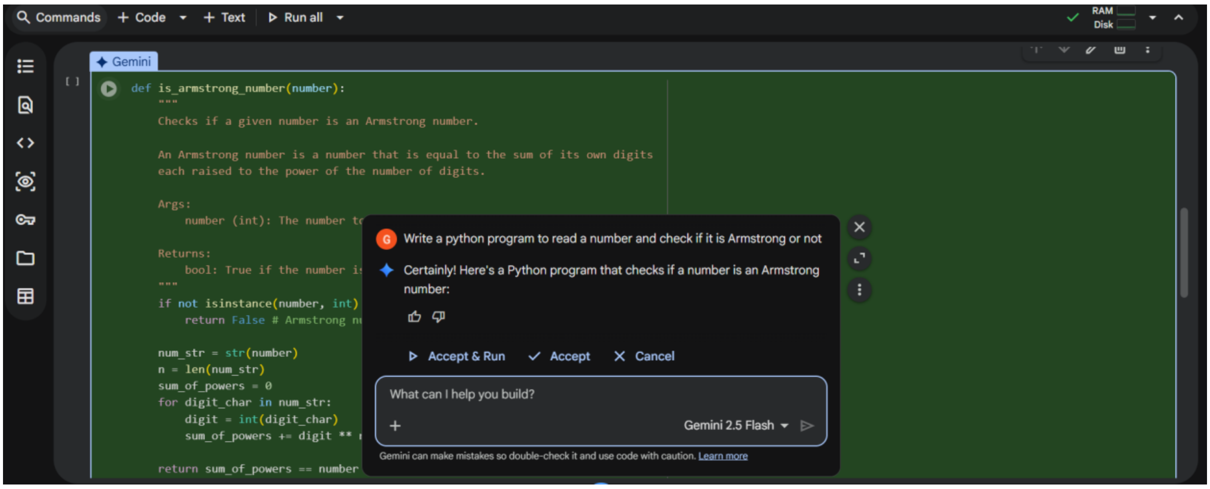
➢ Side-by-side comparison table

➢ Screenshots of prompts and generated code

Gemini Copilot


AIAC_Assign2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands
+ Code
+ Text
Run all
RAM
Disk
Add standard deviation calculation to analyze_numbers function
Explain the analyze_numbers func
Gemini 2.5 Flash
Variables
Terminal
2:17 PM
Python 3

Write a python program to read a number and check if it is Armstrong or not



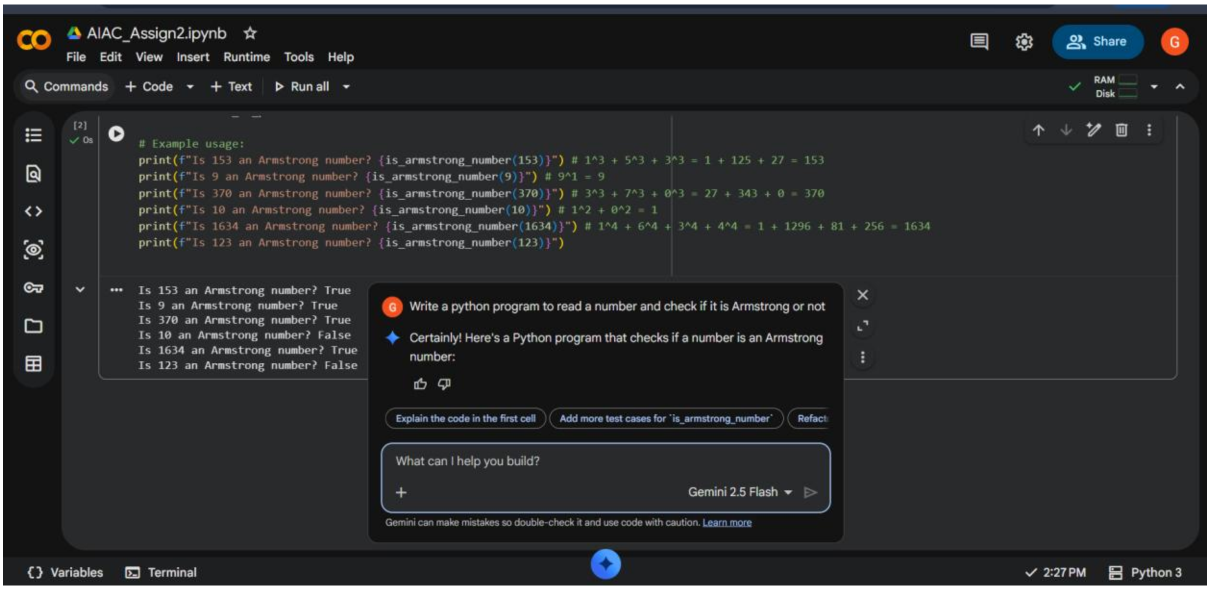
```
def is_armstrong_number(number):
    """
    Checks if a given number is an Armstrong number.

    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 number t

    Returns:
        bool: True if the number i
    """
    if not isinstance(number, int)
        return False # Armstrong n

    num_str = str(number)
    n = len(num_str)
    sum_of_powers = 0
    for digit_char in num_str:
        digit = int(digit_char)
        sum_of_powers += digit ** r

    return sum_of_powers == number
```

Write a python program to read a number and check if it is Armstrong or not

Certainly! Here's a Python program that checks if a number is an Armstrong number:

What can I help you build?

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


AIAC_Assign2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands
+ Code
+ Text
Run all
RAM
Disk
[2]
0s
Explain the code in the first cell
Add more test cases for `is_armstrong_number`
Refact
Gemini 2.5 Flash
Variables
Terminal
2:27 PM
Python 3

```
# Example usage:
print(f"Is 153 an Armstrong number? {is_armstrong_number(153)}") # 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153
print(f"Is 9 an Armstrong number? {is_armstrong_number(9)}") # 9^1 = 9
print(f"Is 370 an Armstrong number? {is_armstrong_number(370)}") # 3^3 + 7^3 + 0^3 = 27 + 343 + 0 = 370
print(f"Is 10 an Armstrong number? {is_armstrong_number(10)}") # 1^2 + 0^2 = 1
print(f"Is 1634 an Armstrong number? {is_armstrong_number(1634)}") # 1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634
print(f"Is 123 an Armstrong number? {is_armstrong_number(123)}")
```

```
Is 153 an Armstrong number? True
Is 9 an Armstrong number? True
Is 370 an Armstrong number? True
Is 10 an Armstrong number? False
Is 1634 an Armstrong number? True
Is 123 an Armstrong number? False
```

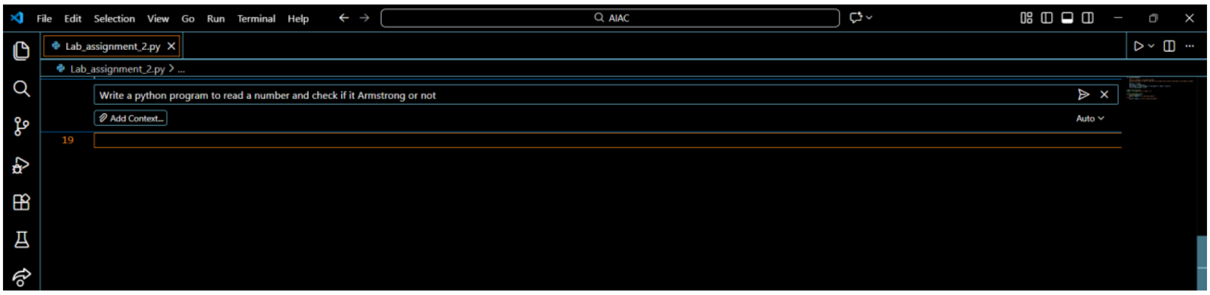
Write a python program to read a number and check if it is Armstrong or not

Certainly! Here's a Python program that checks if a number is an Armstrong number:

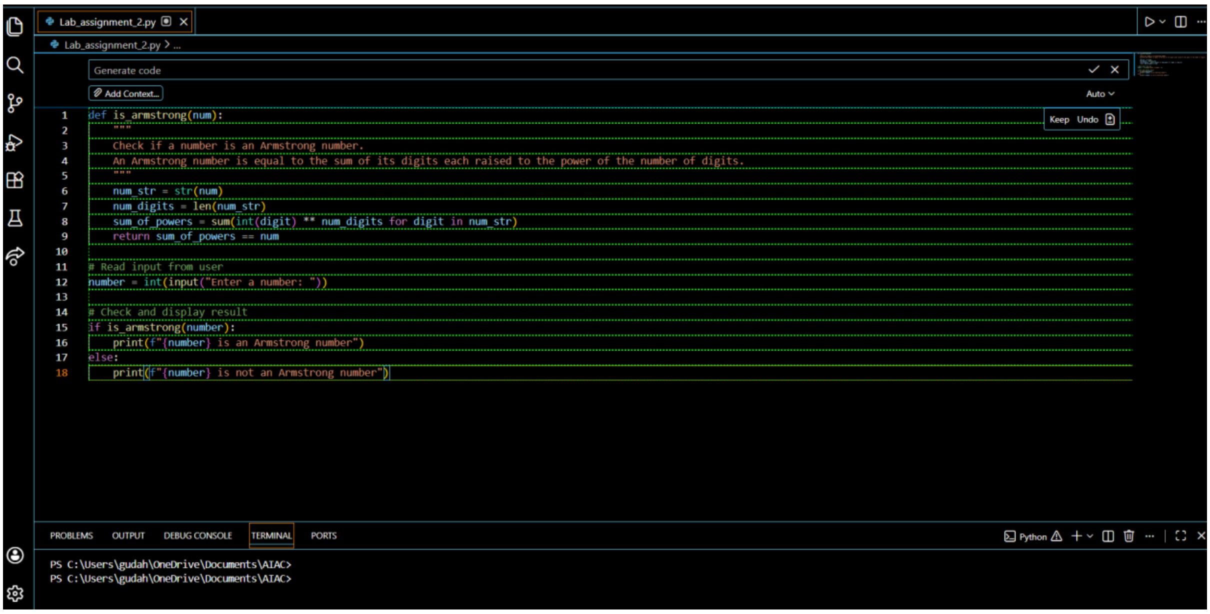
What can I help you build?

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

Github Copilot

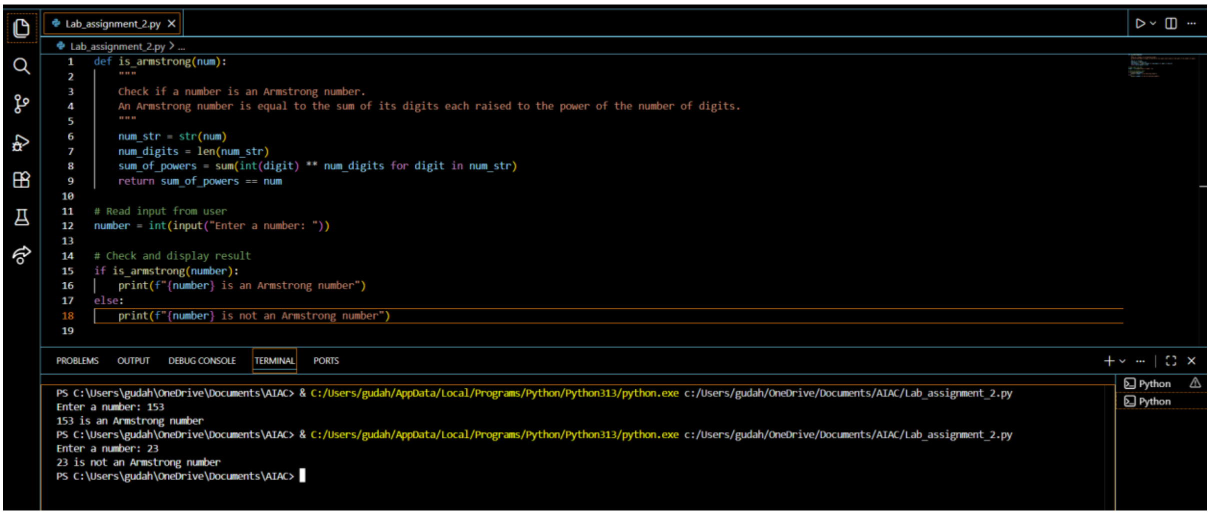


Write a python program to read a number and check if it Armstrong or not



```python
def is_armstrong(num):
    """
    Check if a number is an Armstrong number.
    An Armstrong number is equal to the sum of its digits each raised to the power of the number of digits.
    """
    num_str = str(num)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == num

# Read input from user
number = int(input("Enter a number: "))

# Check and display result
if is_armstrong(number):
    print(f"{number} is an Armstrong number")
else:
    print(f"{number} is not an Armstrong number")
```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC>
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```



```python
def is_armstrong(num):
    """
    Check if a number is an Armstrong number.
    An Armstrong number is equal to the sum of its digits each raised to the power of the number of digits.
    """
    num_str = str(num)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == num

# Read input from user
number = int(input("Enter a number: "))

# Check and display result
if is_armstrong(number):
    print(f"{number} is an Armstrong number")
else:
    print(f"{number} is not an Armstrong number")

```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_2.py
Enter a number: 153
153 is an Armstrong number
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_2.py
Enter a number: 23
23 is not an Armstrong number
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

Comparison: Armstrong Number Programs

Aspect	**GitHub Copilot Code**	**Google Colab Gemini Copilot Code**
Purpose	Checks if a number is an Armstrong number	Checks if a number is an Armstrong number
Input Handling	Takes input from the user using input()	Uses predefined test values (no user input)
Output Style	Prints a clear message (is / is not an Armstrong number)	Prints True or False with explanation comments
Return Value	Returns True or False internally	Returns True or False
Logic Style	Uses **Pythonic one-line sum with generator expression**	Uses **step-by-step loop (beginner-friendly)**
Code Length	Shorter and more compact	Slightly longer and more detailed
Readability	Clean, but may feel compact for beginners	Very clear and easy to understand
Validation Checks	No validation for negative or non-integer input	Checks for non-integer and negative numbers
Documentation	Short docstring	Detailed docstring with Args & Returns
Examples / Testing	Depends on user input	Multiple example test cases included
Best For	Quick use, competitive coding, clean scripts	Learning, teaching, and clarity
Error Safety	May crash if input is invalid	Safer due to input validation

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.

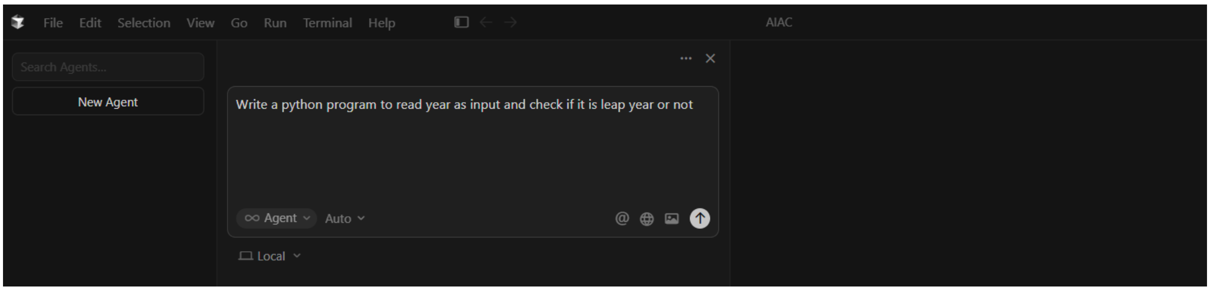
❖ Expected Output:

➢ Two versions of code

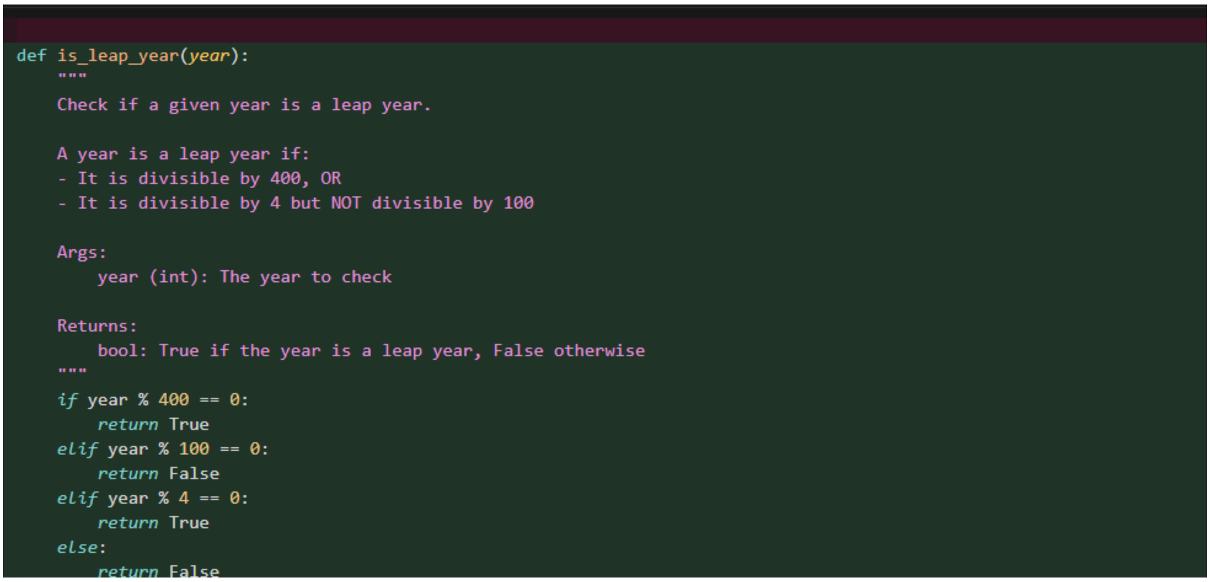
➢ Sample inputs/outputs

➢ Brief comparison

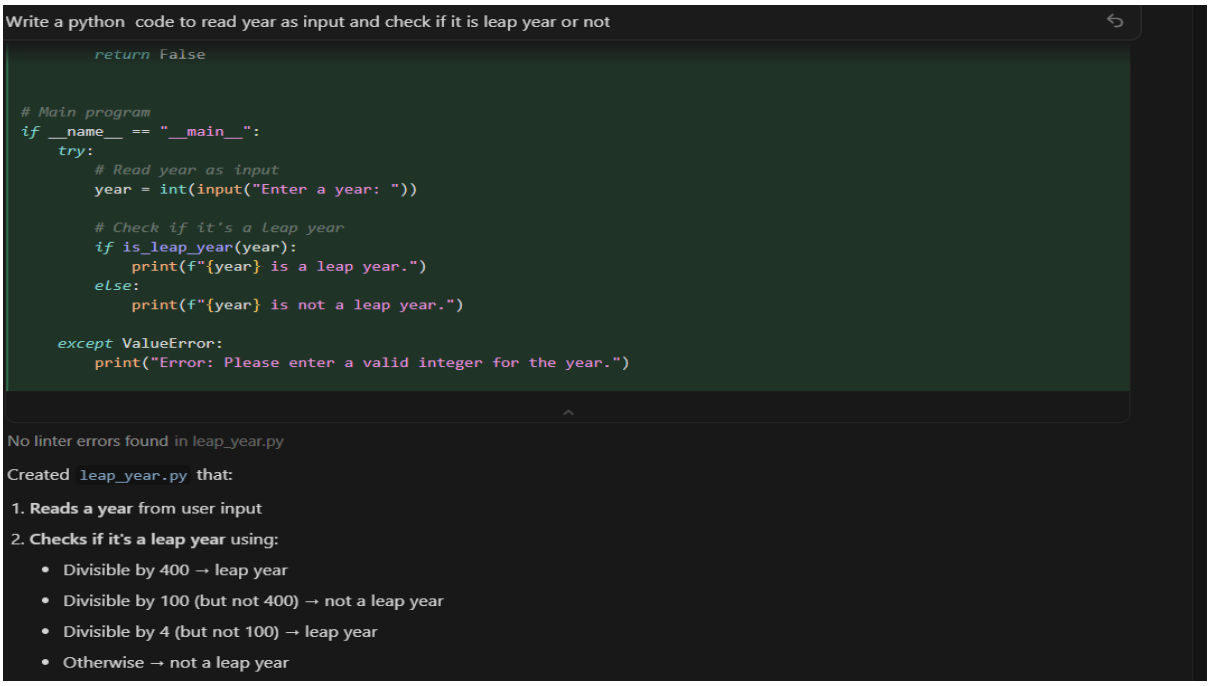
Version-1



Write a python program to read year as input and check if it is leap year or not



```python
def is_leap_year(year):
    """
    Check if a given year is a leap year.

    A year is a leap year if:
    - It is divisible by 400, OR
    - It is divisible by 4 but NOT divisible by 100

    Args:
        year (int): The year to check

    Returns:
        bool: True if the year is a leap year, False otherwise
    """
    if year % 400 == 0:
        return True
    elif year % 100 == 0:
        return False
    elif year % 4 == 0:
        return True
    else:
        return False
```



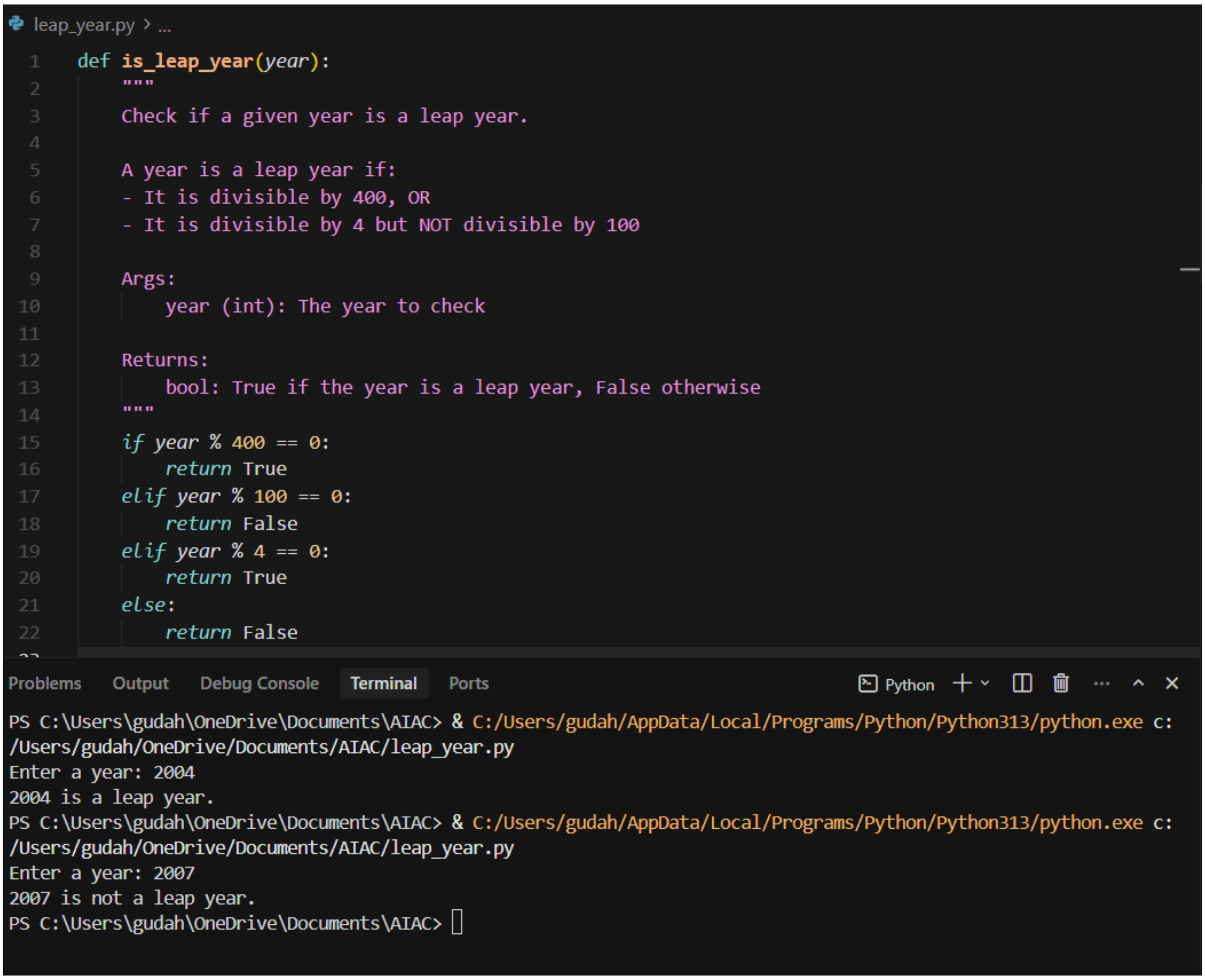
Write a python code to read year as input and check if it is leap year or not

```python
        return False


# Main program
if __name__ == "__main__":
    try:
        # Read year as input
        year = int(input("Enter a year: "))

        # Check if it's a leap year
        if is_leap_year(year):
            print(f"{year} is a leap year.")
        else:
            print(f"{year} is not a leap year.")

    except ValueError:
        print("Error: Please enter a valid integer for the year.")
```

No linter errors found in leap_year.py

Created `leap_year.py` that:

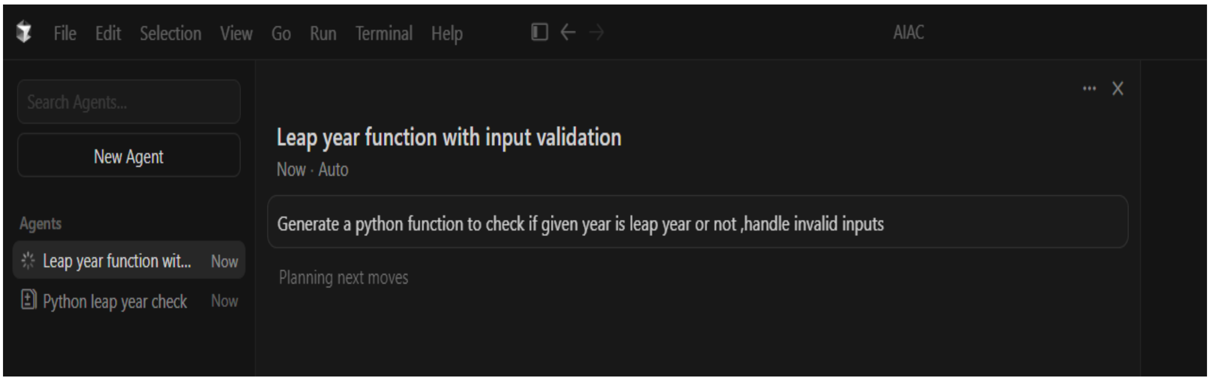
1. **Reads a year** from user input
2. **Checks if it's a leap year** using:
    - Divisible by 400 → leap year
    - Divisible by 100 (but not 400) → not a leap year
    - Divisible by 4 (but not 100) → leap year
    - Otherwise → not a leap year



```python
def is_leap_year(year):
    """
    Check if a given year is a leap year.

    A year is a leap year if:
    - It is divisible by 400, OR
    - It is divisible by 4 but NOT divisible by 100

    Args:
        year (int): The year to check

    Returns:
        bool: True if the year is a leap year, False otherwise
    """
    if year % 400 == 0:
        return True
    elif year % 100 == 0:
        return False
    elif year % 4 == 0:
        return True
    else:
        return False
```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:
/Users/gudah/OneDrive/Documents/AIAC/leap_year.py
Enter a year: 2004
2004 is a leap year.
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:
/Users/gudah/OneDrive/Documents/AIAC/leap_year.py
Enter a year: 2007
2007 is not a leap year.
PS C:\Users\gudah\OneDrive\Documents\AIAC> 
```

Version-2

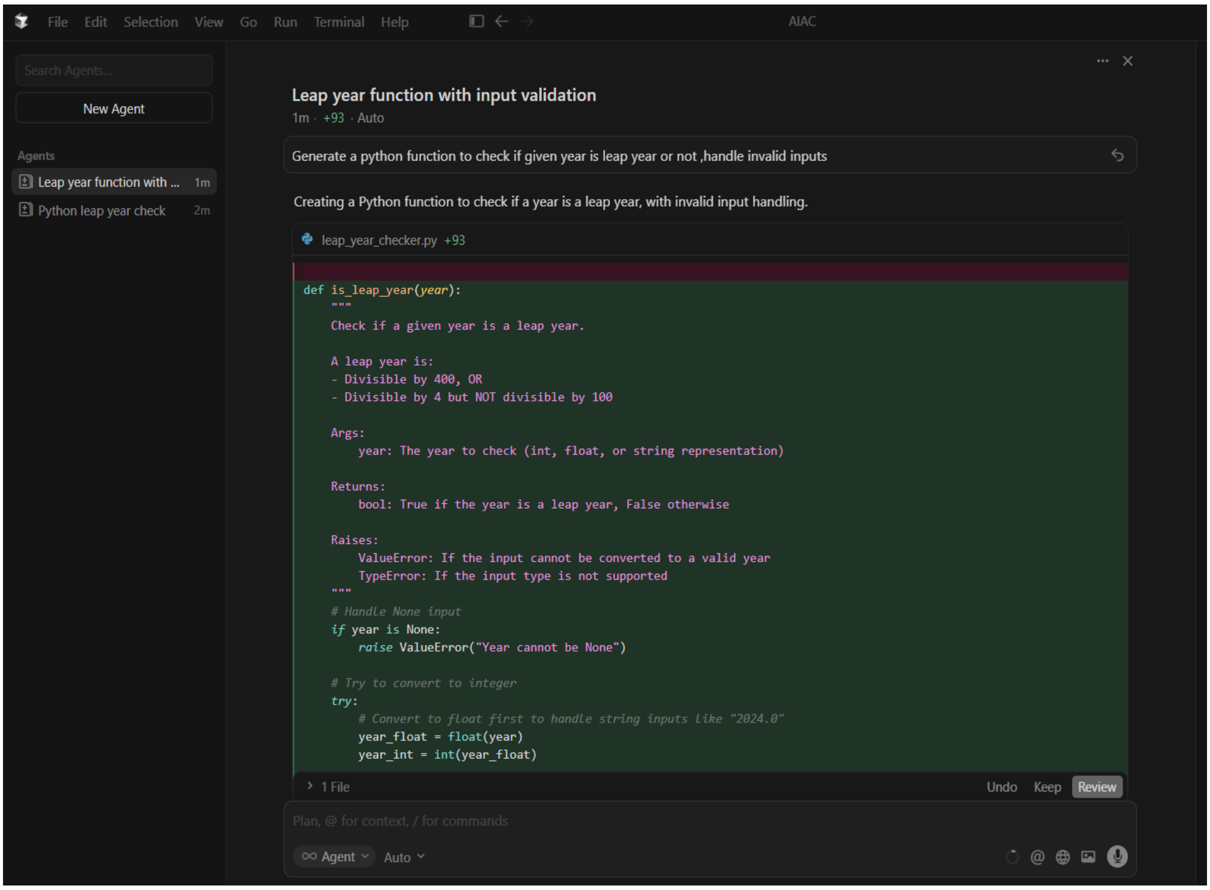


Leap year function with input validation

Now · Auto

Generate a python function to check if given year is leap year or not ,handle invalid inputs

Planning next moves



Leap year function with input validation

1m · +93 · Auto

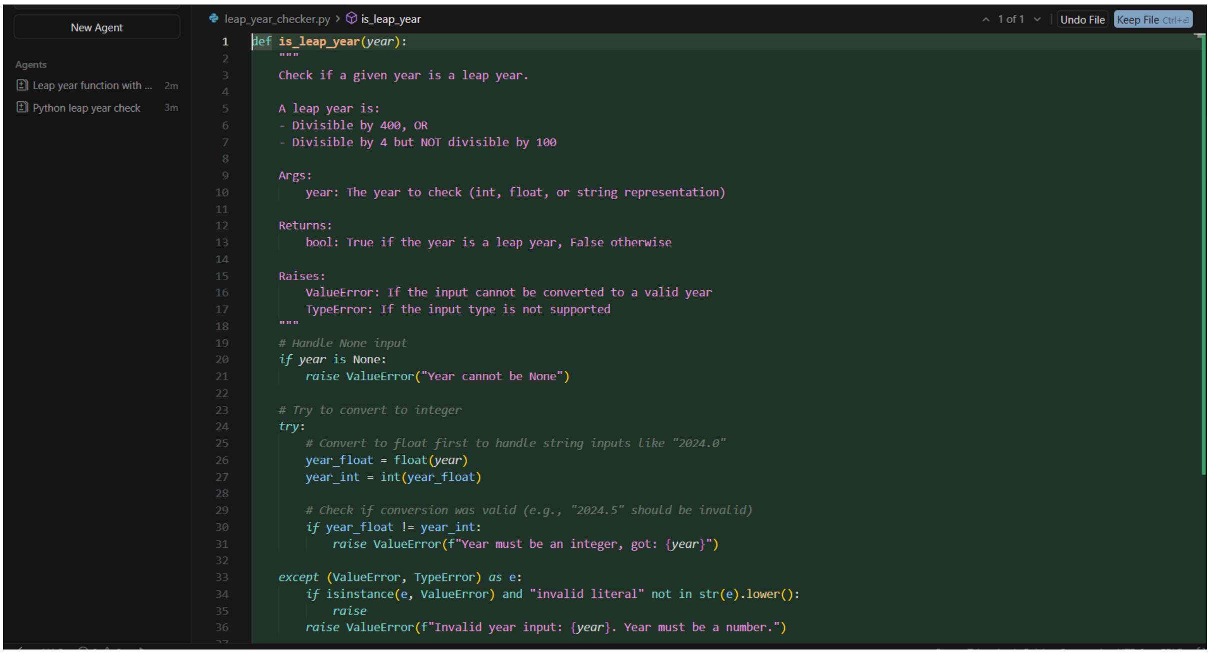
Generate a python function to check if given year is leap year or not ,handle invalid inputs

Creating a Python function to check if a year is a leap year, with invalid input handling.

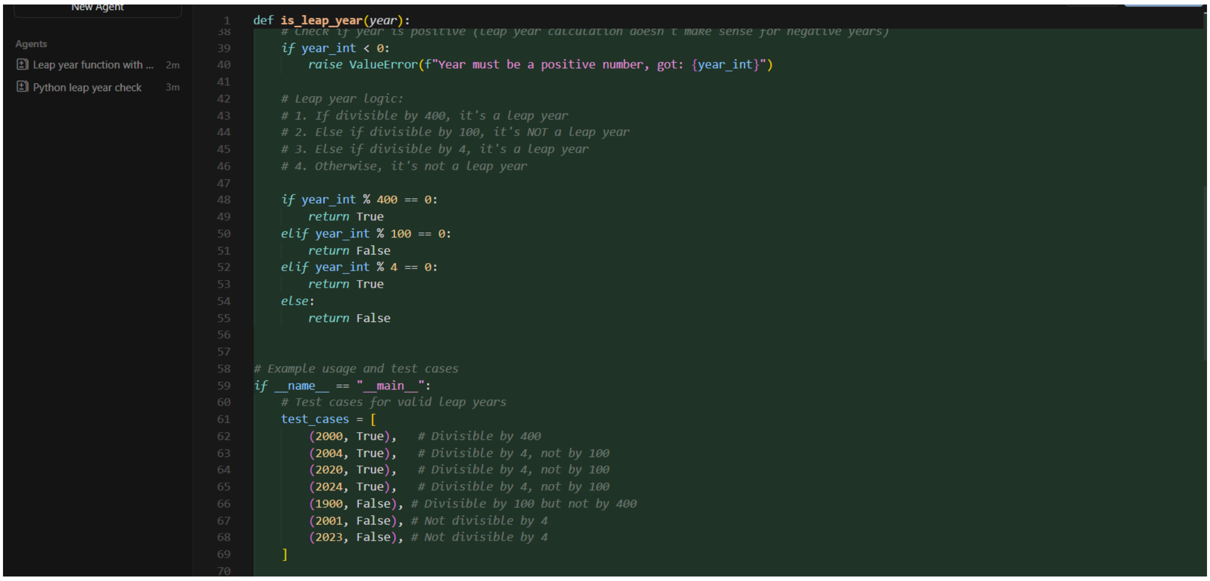
leap_year_checker.py +93

```python
def is_leap_year(year):
    """
    Check if a given year is a leap year.

    A leap year is:
    - Divisible by 400, OR
    - Divisible by 4 but NOT divisible by 100

    Args:
        year: The year to check (int, float, or string representation)

    Returns:
        bool: True if the year is a leap year, False otherwise

    Raises:
        ValueError: If the input cannot be converted to a valid year
        TypeError: If the input type is not supported
    """
    # Handle None input
    if year is None:
        raise ValueError("Year cannot be None")

    # Try to convert to integer
    try:
        # Convert to float first to handle string inputs like "2024.0"
        year_float = float(year)
        year_int = int(year_float)
```

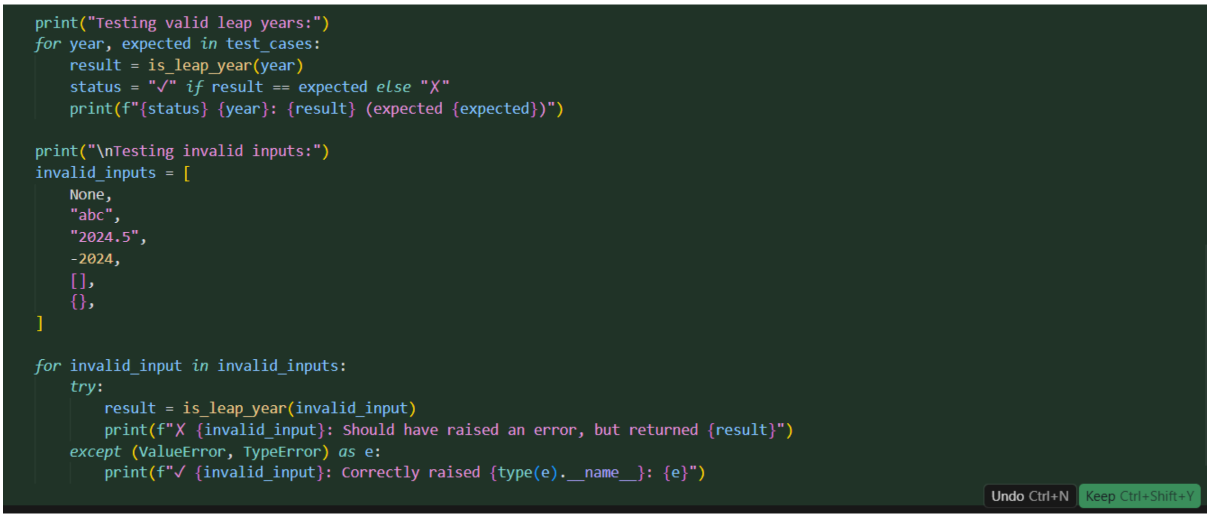
1 File
Undo
Keep
Review
Plan, @ for context, / for commands
Agent
Auto


New Agent
Agents
Leap year function with ... 2m
Python leap year check 3m
leap_year_checker.py > is_leap_year
1 of 1
Undo File
Keep File Ctrl+⏎

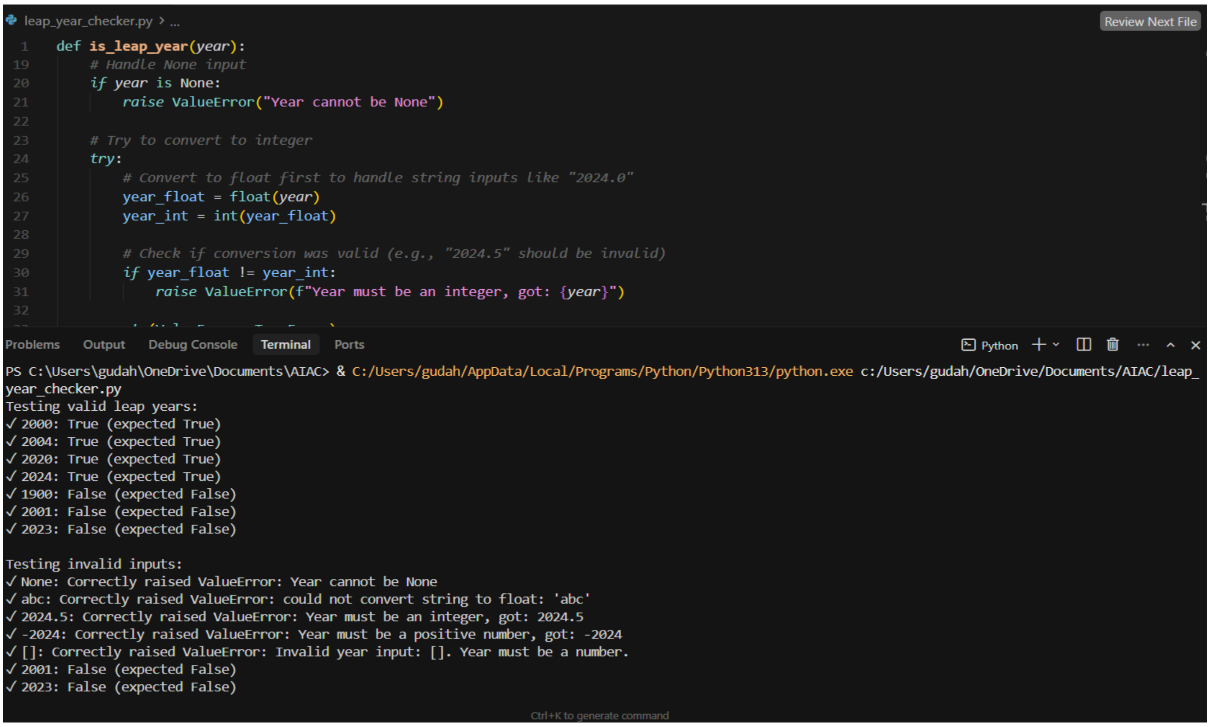
```python
def is_leap_year(year):
    """
    Check if a given year is a leap year.

    A leap year is:
    - Divisible by 400, OR
    - Divisible by 4 but NOT divisible by 100

    Args:
        year: The year to check (int, float, or string representation)

    Returns:
        bool: True if the year is a leap year, False otherwise

    Raises:
        ValueError: If the input cannot be converted to a valid year
        TypeError: If the input type is not supported
    """
    # Handle None input
    if year is None:
        raise ValueError("Year cannot be None")

    # Try to convert to integer
    try:
        # Convert to float first to handle string inputs like "2024.0"
        year_float = float(year)
        year_int = int(year_float)

        # Check if conversion was valid (e.g., "2024.5" should be invalid)
        if year_float != year_int:
            raise ValueError(f"Year must be an integer, got: {year}")

    except (ValueError, TypeError) as e:
        if isinstance(e, ValueError) and "invalid literal" not in str(e).lower():
            raise
        raise ValueError(f"Invalid year input: {year}. Year must be a number.")
```



```python
def is_leap_year(year):
    # Check if year is positive (leap year calculation doesn't make sense for negative years)
    if year_int < 0:
        raise ValueError(f"Year must be a positive number, got: {year_int}")

    # Leap year logic:
    # 1. If divisible by 400, it's a leap year
    # 2. Else if divisible by 100, it's NOT a leap year
    # 3. Else if divisible by 4, it's a leap year
    # 4. Otherwise, it's not a leap year

    if year_int % 400 == 0:
        return True
    elif year_int % 100 == 0:
        return False
    elif year_int % 4 == 0:
        return True
    else:
        return False


# Example usage and test cases
if __name__ == "__main__":
    # Test cases for valid leap years
    test_cases = [
        (2000, True),   # Divisible by 400
        (2004, True),   # Divisible by 4, not by 100
        (2020, True),   # Divisible by 4, not by 100
        (2024, True),   # Divisible by 4, not by 100
        (1900, False), # Divisible by 100 but not by 400
        (2001, False), # Not divisible by 4
        (2023, False), # Not divisible by 4
    ]
```



```python
    print("Testing valid leap years:")
    for year, expected in test_cases:
        result = is_leap_year(year)
        status = "✓" if result == expected else "✗"
        print(f"{status} {year}: {result} (expected {expected})")

    print("\nTesting invalid inputs:")
    invalid_inputs = [
        None,
        "abc",
        "2024.5",
        -2024,
        [],
        {},
    ]

    for invalid_input in invalid_inputs:
        try:
            result = is_leap_year(invalid_input)
            print(f"✗ {invalid_input}: Should have raised an error, but returned {result}")
        except (ValueError, TypeError) as e:
            print(f"✓ {invalid_input}: Correctly raised {type(e).__name__}: {e}")
```



```python
def is_leap_year(year):
    # Handle None input
    if year is None:
        raise ValueError("Year cannot be None")

    # Try to convert to integer
    try:
        # Convert to float first to handle string inputs like "2024.0"
        year_float = float(year)
        year_int = int(year_float)

        # Check if conversion was valid (e.g., "2024.5" should be invalid)
        if year_float != year_int:
            raise ValueError(f"Year must be an integer, got: {year}")
```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/leap_
year_checker.py
Testing valid leap years:
✓ 2000: True (expected True)
✓ 2004: True (expected True)
✓ 2020: True (expected True)
✓ 2024: True (expected True)
✓ 1900: False (expected False)
✓ 2001: False (expected False)
✓ 2023: False (expected False)

Testing invalid inputs:
✓ None: Correctly raised ValueError: Year cannot be None
✓ abc: Correctly raised ValueError: could not convert string to float: 'abc'
✓ 2024.5: Correctly raised ValueError: Year must be an integer, got: 2024.5
✓ -2024: Correctly raised ValueError: Year must be a positive number, got: -2024
✓ []: Correctly raised ValueError: Invalid year input: []. Year must be a number.
✓ 2001: False (expected False)
✓ 2023: False (expected False)
```

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.

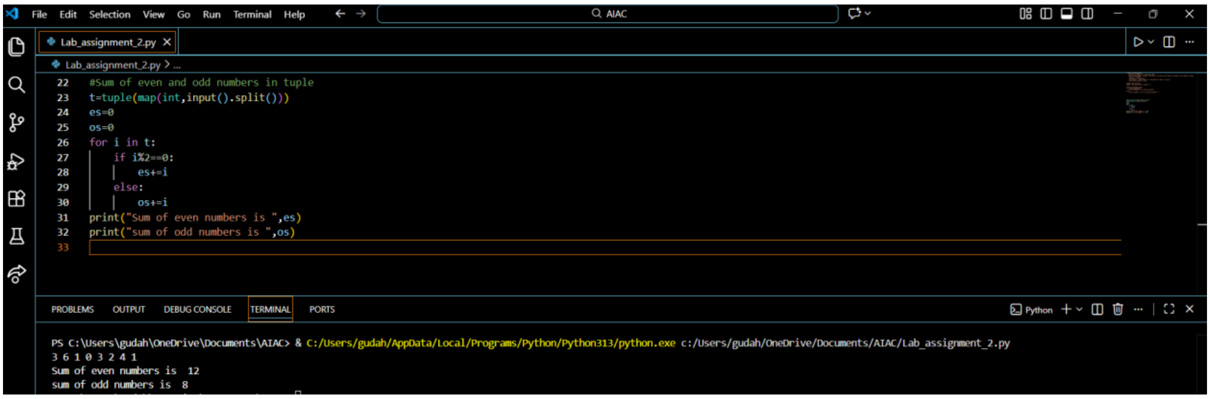
❖ Expected Output:

➢ Original code

➢ Refactored code

➢ Explanation of improvements

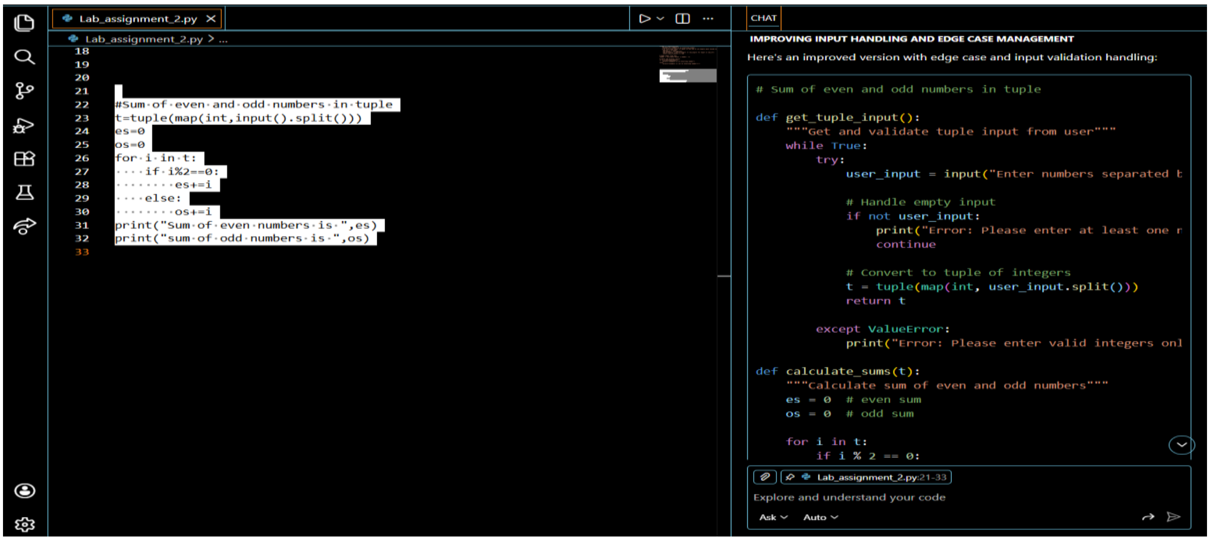
Original code


File Edit Selection View Go Run Terminal Help
AIAC
Lab_assignment_2.py
Lab_assignment_2.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python

```python
#Sum of even and odd numbers in tuple
t=tuple(map(int,input().split()))
es=0
os=0
for i in t:
    if i%2==0:
        es+=i
    else:
        os+=i
print("Sum of even numbers is ",es)
print("sum of odd numbers is ",os)
```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_2.py
3 6 1 0 3 2 4 1
Sum of even numbers is  12
sum of odd numbers is  8
```

Refactored Code

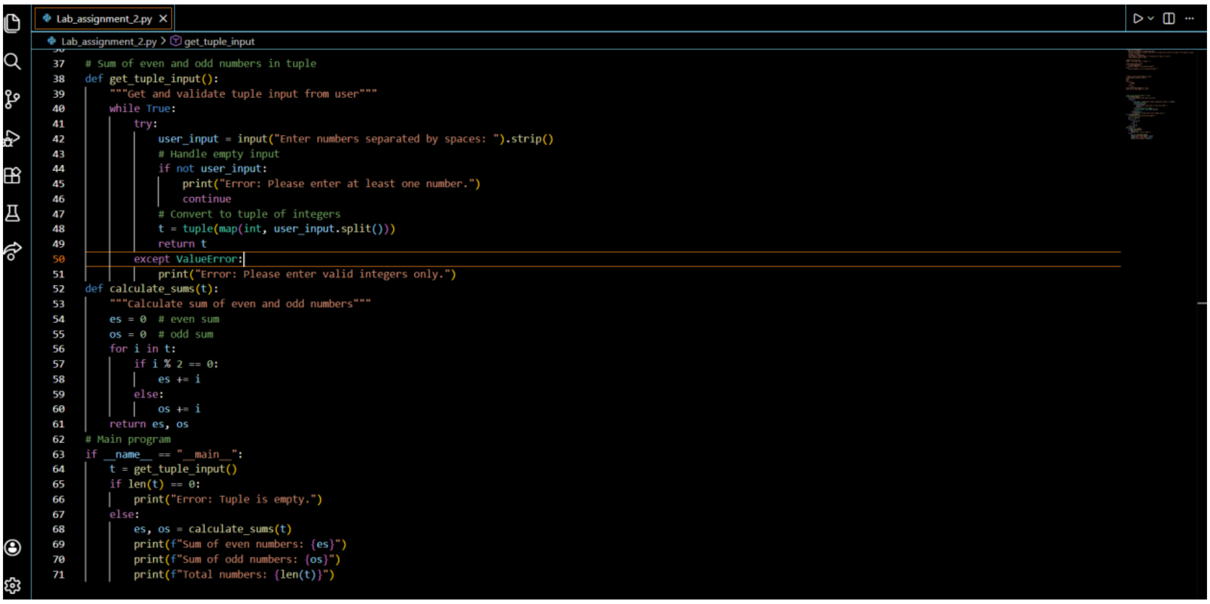

Lab_assignment_2.py
Lab_assignment_2.py > ...
CHAT
Lab_assignment_2.py:21-33
Explore and understand your code
Ask
Auto

```python
#Sum of even and odd numbers in tuple
t=tuple(map(int,input().split()))
es=0
os=0
for i in t:
    if i%2==0:
        es+=i
    else:
        os+=i
print("Sum of even numbers is ",es)
print("sum of odd numbers is ",os)
```

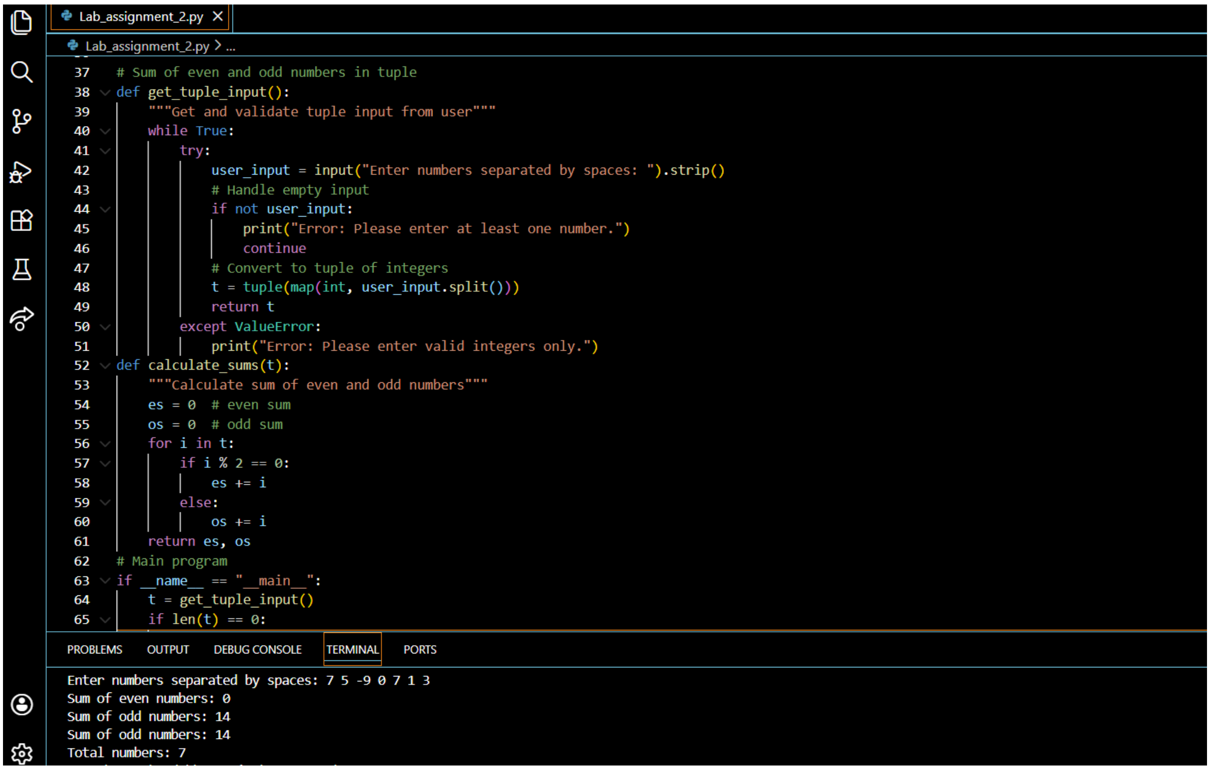
IMPROVING INPUT HANDLING AND EDGE CASE MANAGEMENT

Here's an improved version with edge case and input validation handling:

```python
# Sum of even and odd numbers in tuple

def get_tuple_input():
    """Get and validate tuple input from user"""
    while True:
        try:
            user_input = input("Enter numbers separated t

            # Handle empty input
            if not user_input:
                print("Error: Please enter at least one r
                continue

            # Convert to tuple of integers
            t = tuple(map(int, user_input.split()))
            return t

        except ValueError:
            print("Error: Please enter valid integers onl

def calculate_sums(t):
    """Calculate sum of even and odd numbers"""
    es = 0  # even sum
    os = 0  # odd sum

    for i in t:
        if i % 2 == 0:
```



```python
# Sum of even and odd numbers in tuple
def get_tuple_input():
    """Get and validate tuple input from user"""
    while True:
        try:
            user_input = input("Enter numbers separated by spaces: ").strip()
            # Handle empty input
            if not user_input:
                print("Error: Please enter at least one number.")
                continue
            # Convert to tuple of integers
            t = tuple(map(int, user_input.split()))
            return t
        except ValueError:
            print("Error: Please enter valid integers only.")
def calculate_sums(t):
    """Calculate sum of even and odd numbers"""
    es = 0  # even sum
    os = 0  # odd sum
    for i in t:
        if i % 2 == 0:
            es += i
        else:
            os += i
    return es, os
# Main program
if __name__ == "__main__":
    t = get_tuple_input()
    if len(t) == 0:
        print("Error: Tuple is empty.")
    else:
        es, os = calculate_sums(t)
        print(f"Sum of even numbers: {es}")
        print(f"Sum of odd numbers: {os}")
        print(f"Total numbers: {len(t)}")
```



```python
# Sum of even and odd numbers in tuple
def get_tuple_input():
    """Get and validate tuple input from user"""
    while True:
        try:
            user_input = input("Enter numbers separated by spaces: ").strip()
            # Handle empty input
            if not user_input:
                print("Error: Please enter at least one number.")
                continue
            # Convert to tuple of integers
            t = tuple(map(int, user_input.split()))
            return t
        except ValueError:
            print("Error: Please enter valid integers only.")
def calculate_sums(t):
    """Calculate sum of even and odd numbers"""
    es = 0  # even sum
    os = 0  # odd sum
    for i in t:
        if i % 2 == 0:
            es += i
        else:
            os += i
    return es, os
# Main program
if __name__ == "__main__":
    t = get_tuple_input()
    if len(t) == 0:
```

```
Enter numbers separated by spaces: 7 5 -9 0 7 1 3
Sum of even numbers: 0
Sum of odd numbers: 14
Sum of odd numbers: 14
Total numbers: 7
```

Key improvements:

- **Input validation**: Checks for empty input and non-integer values
- **Error handling**: Try-except block catches invalid inputs
- **Retry logic**: Prompts user again if input is invalid
- **Modular functions**: Separated logic into reusable functions
- **Better formatting**: Uses f-strings for cleaner output
- **Edge cases handled**: Empty tuples, zero values, negative numbers
- **User-friendly messages**: Clear error and result messages