

AI ASSISTED CODING

ASSIGNMENT-2

2303A51666-batch-23

Sony Dodla

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

Task 1: Statistical Summary for Survey Data

❖ Scenario:

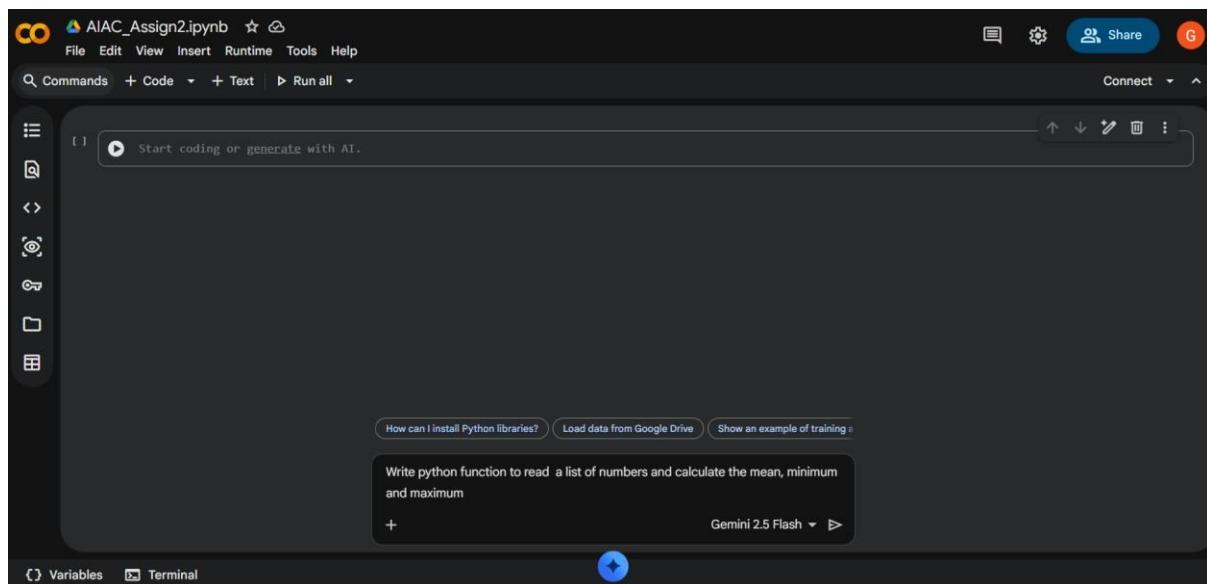
You are a data analyst intern working with survey responses stored as numerical lists.

❖ 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



The screenshot shows a Jupyter Notebook interface with a dark theme. A code cell contains the following Python function:

```

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 (mean, minimum, maximum). Returns (None, None, None) if the input list is empty.
    """
    if not numbers:
        return None, None, None

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

    return mean_val, min_val, max_val

```

Below the code cell is a Gemini AI assistant window. The user has asked it to "Write python function to read a list of numbers and calculate the mean, minimum and maximum". Gemini has responded with the provided code. The AI interface includes a "Accept & Run" button.

The screenshot shows a Jupyter Notebook interface with a dark theme. A code cell contains the following Python code:

```

# 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}")

```

The output pane shows the results of running the code:

```

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

The screenshot shows the Gemini Copilot interface in a dark-themed Jupyter Notebook. A code completion pop-up is open over a cell containing Python code. The pop-up asks for a program to check if a number is Armstrong. It provides a sample implementation and a link to Gemini's GitHub repository.

```
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 to check.

    Returns:
        bool: True if the number is an Armstrong number.
    """
    if not isinstance(number, int):
        return False # Armstrong numbers must be integers

    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 ** n

    return sum_of_powers == number
```

Add standard deviation calculation to analyze_numbers function
Add standard deviation calculation to analyze_numbers function
Explain the analyze_numbers function
Write a python program to read a number and check if it is Armstrong or not
+ Gemini 2.5 Flash ▶

This screenshot shows a more detailed code completion pop-up for the Armstrong number check function. It includes a description of what an Armstrong number is, examples of Armstrong numbers, and a step-by-step breakdown of the function logic.

```
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 to check.

    Returns:
        bool: True if the number is an Armstrong number.
    """
    if not isinstance(number, int):
        return False # Armstrong numbers must be integers

    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 ** n

    return sum_of_powers == number
```

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

This screenshot shows a code completion pop-up for a function named `is_armstrong_number`. It provides several test cases for the function and a link to Gemini's GitHub repository.

```
# Example usage:
print("Is 153 an Armstrong number? " + str(is_armstrong_number(153))) # 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153
print("Is 9 an Armstrong number? " + str(is_armstrong_number(9))) # 9^1 = 9
print("Is 370 an Armstrong number? " + str(is_armstrong_number(370))) # 3^3 + 7^3 + 0^3 = 27 + 343 + 0 = 370
print("Is 10 an Armstrong number? " + str(is_armstrong_number(10))) # 1^2 + 0^2 = 1
print("Is 1634 an Armstrong number? " + str(is_armstrong_number(1634))) # 1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634
print("Is 123 an Armstrong number? " + str(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
+ Gemini 2.5 Flash ▶
Explain the code in the first cell Add more test cases for 'is_armstrong_number' Refactor
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

Github Copilot

The screenshot shows the GitHub Copilot interface. A code completion suggestion is displayed in the center of the screen, asking to "Write a python program to read a number and check if it Armstrong or not". Below the question, there is a button labeled "Add Context...". The code completion suggestion starts with:

```
19
```

The screenshot shows the GitHub Copilot interface with the generated Python code for an Armstrong number checker. The code is as follows:

```
1 def is_armstrong(num):
2     """
3         Check if a number is an Armstrong number.
4         An Armstrong number is equal to the sum of its digits each raised to the power of the number of digits.
5     """
6     num_str = str(num)
7     num_digits = len(num_str)
8     sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
9     return sum_of_powers == num
10
11 # Read input from user
12 number = int(input("Enter a number: "))
13
14 # Check and display result
15 if is_armstrong(number):
16     print(f"{number} is an Armstrong number")
17 else:
18     print(f"{number} is not an Armstrong number")
```

The screenshot shows the GitHub Copilot interface with the terminal output of the generated Python code. The terminal shows the command being run and the resulting output:

```
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 <code>input()</code> | 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 generator expression | with step-by-step loop (beginnerfriendly) |
| 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.

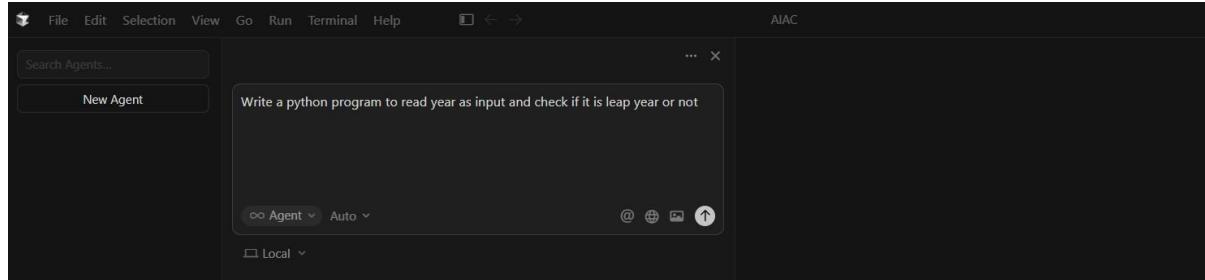
❖ Expected Output:

➤ Two versions of code

➤ Sample inputs/outputs

➤ Brief comparison

Version-1



```
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

```
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.")

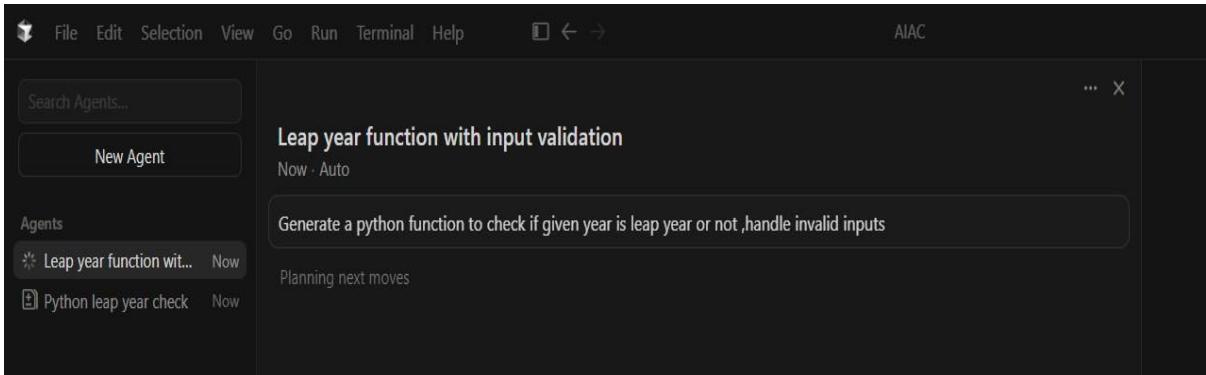
    except ValueError:
        print("Error: Please enter a valid integer for the year.")
```

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

Version-2



File Edit Selection View Go Run Terminal Help AIAC

Search Agents... New Agent

Agents

- Leap year function with ... 1m
- Python leap year check 2m

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
```

```
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 def is_leap_year(year):
2     """
3         Check if a given year is a leap year.
4
5         A leap year is:
6             - Divisible by 400, OR
7             - Divisible by 4 but NOT divisible by 100
8
9         Args:
10            year: The year to check (int, float, or string representation)
11
12         Returns:
13            bool: True if the year is a leap year, False otherwise
14
15         Raises:
16             ValueError: If the input cannot be converted to a valid year
17             TypeError: If the input type is not supported
18
19         # Handle None input
20         if year is None:
21             raise ValueError("Year cannot be None")
22
23         # Try to convert to integer
24         try:
25             # Convert to float first to handle string inputs like "2024.0"
26             year_float = float(year)
27             year_int = int(year_float)
28
29             # Check if conversion was valid (e.g., "2024.5" should be invalid)
30             if year_float != year_int:
31                 raise ValueError(f"Year must be an integer, got: {year}")
32
33         except (ValueError, TypeError) as e:
34             if isinstance(e, ValueError) and "invalid literal" not in str(e).lower():
35                 raise
36             raise ValueError(f"Invalid year input: {year}. Year must be a number.")
```

1 of 1 Undo File Keep File Ctrl+D

```

New Agent
Agents
② Leap year function with ... 2m
② Python leap year check 3m

1 def is_leap_year(year):
2     # Check if year is positive (leap year calculation doesn't make sense for negative years)
3     if year_int < 0:
4         raise ValueError(f"Year must be a positive number, got: {year_int}")
5
6     # Leap year logic:
7     # 1. If divisible by 400, it's a Leap year
8     # 2. Else if divisible by 100, it's NOT a Leap year
9     # 3. Else if divisible by 4, it's a leap year
10    # 4. Otherwise, it's not a leap year
11
12    if year_int % 400 == 0:
13        return True
14    elif year_int % 100 == 0:
15        return False
16    elif year_int % 4 == 0:
17        return True
18    else:
19        return False
20
21
22 # Example usage and test cases
23 if __name__ == "__main__":
24     # Test cases for valid leap years
25     test_cases = [
26         (2000, True), # Divisible by 400
27         (2004, True), # Divisible by 4, not by 100
28         (2020, True), # Divisible by 4, not by 100
29         (2024, True), # Divisible by 4, not by 100
30         (1900, False), # Divisible by 100 but not by 400
31         (2001, False), # Not divisible by 4
32         (2023, False), # Not divisible by 4
33     ]
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

```

```

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}")

```

Undo Ctrl+N Keep Ctrl+Shift+Y

```

leap_year_checker.py > ...
Review Next File

1 def is_leap_year(year):
2     # Handle None input
3     if year is None:
4         raise ValueError("Year cannot be None")
5
6     # Try to convert to integer
7     try:
8         # Convert to float first to handle string inputs like "2024.0"
9         year_float = float(year)
10        year_int = int(year_float)
11
12        # Check if conversion was valid (e.g., "2024.5" should be invalid)
13        if year_float != year_int:
14            raise ValueError(f"Year must be an integer, got: {year}")
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

```

Problems Output Debug Console Terminal Ports

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)

Ctrl+K to generate command

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

The screenshot shows the AIAC IDE interface. The code editor window displays the following Python script:

```
22 #Sum of even and odd numbers in tuple
23 t=tuple(map(int,input().split()))
24 es=0
25 os=0
26 for i in t:
27     if i%2==0:
28         | es+=i
29     else:
30         | os+=i
31 print("sum of even numbers is ",es)
32 print("sum of odd numbers is ",os)
33 
```

The terminal window shows the output of running the script with input "3 6 1 0 3 2 4 1". The output is:

```
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

The screenshot shows the AIAC IDE interface with a refactored version of the Python script. The code editor window displays the same script as above, but the code itself has been modified. To the right, a large panel titled "IMPROVING INPUT HANDLING AND EDGE CASE MANAGEMENT" provides a detailed explanation of the changes:

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

```
# 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: ")
            # 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")
```

The panel also includes a note about calculating the sum of even and odd numbers:

```
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
```

The bottom of the panel says "Explore and understand your code" and "Ask Auto".

```
37 # Sum of even and odd numbers in tuple
38 def get_tuple_input():
39     """Get and validate tuple input from user"""
40     while True:
41         try:
42             user_input = input("Enter numbers separated by spaces: ").strip()
43             # Handle empty input
44             if not user_input:
45                 print("Error: Please enter at least one number.")
46                 continue
47             # Convert to tuple of integers
48             t = tuple(map(int, user_input.split()))
49             return t
50         except ValueError:
51             print("Error: Please enter valid integers only.")
52 def calculate_sums(t):
53     """Calculate sum of even and odd numbers"""
54     es = 0 # even sum
55     os = 0 # odd sum
56     for i in t:
57         if i % 2 == 0:
58             es += i
59         else:
60             os += i
61     return es, os
62 # Main program
63 if __name__ == "__main__":
64     t = get_tuple_input()
65     if len(t) == 0:
66         print("Error: Tuple is empty.")
67     else:
68         es, os = calculate_sums(t)
69         print(f"Sum of even numbers: {es}")
70         print(f"Sum of odd numbers: {os}")
71         print(f"Total numbers: {len(t)}")
```

```
37 # Sum of even and odd numbers in tuple
38 def get_tuple_input():
39     """Get and validate tuple input from user"""
40     while True:
41         try:
42             user_input = input("Enter numbers separated by spaces: ").strip()
43             # Handle empty input
44             if not user_input:
45                 print("Error: Please enter at least one number.")
46                 continue
47             # Convert to tuple of integers
48             t = tuple(map(int, user_input.split()))
49             return t
50         except ValueError:
51             print("Error: Please enter valid integers only.")
52 def calculate_sums(t):
53     """Calculate sum of even and odd numbers"""
54     es = 0 # even sum
55     os = 0 # odd sum
56     for i in t:
57         if i % 2 == 0:
58             es += i
59         else:
60             os += i
61     return es, os
62 # Main program
63 if __name__ == "__main__":
64     t = get_tuple_input()
65     if len(t) == 0:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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