AI ASSISTED CODING ASSIGNMENT - 2.1

2303a51019

Jangam srijith Rao

Task 1: Statistical Summary for Survey Data

• Scenario

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

• Task Description

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 Deliverables

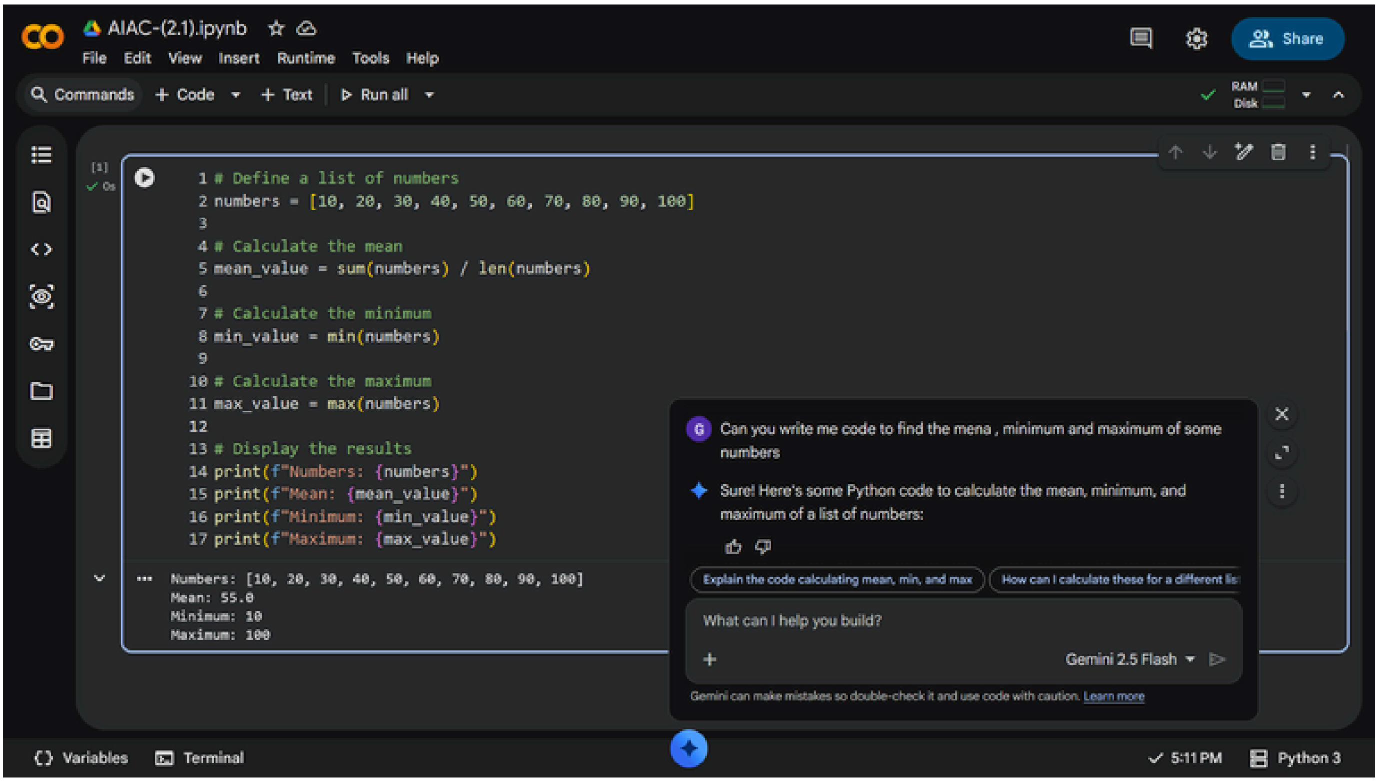
➢ Correct Python function

- Yes, it is generated by Gemini

➢ Output shown in Colab

- Yes, mean, minimum and maximum are formed

➢ Screenshot of Gemini prompt and result



```python
# Define a list of numbers
numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Calculate the mean
mean_value = sum(numbers) / len(numbers)

# Calculate the minimum
min_value = min(numbers)

# Calculate the maximum
max_value = max(numbers)

# Display the results
print(f"Numbers: {numbers}")
print(f"Mean: {mean_value}")
print(f"Minimum: {min_value}")
print(f"Maximum: {max_value}")
```

```
Numbers: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Mean: 55.0
Minimum: 10
Maximum: 100
```

Can you write me code to find the mena , minimum and maximum of some numbers

Sure! Here's some Python code to calculate the mean, minimum, and maximum of a list of numbers:

Explain the code calculating mean, min, and max
How can I calculate these for a different lis
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

Task 2: Armstrong Number – AI Comparison

• Scenario

You are evaluating AI tools for numeric validation logic.

• Task Description

Generate Armstrong numbers checker using Gemini and GitHub Copilot.

Compare their outputs, logic style, and clarity.

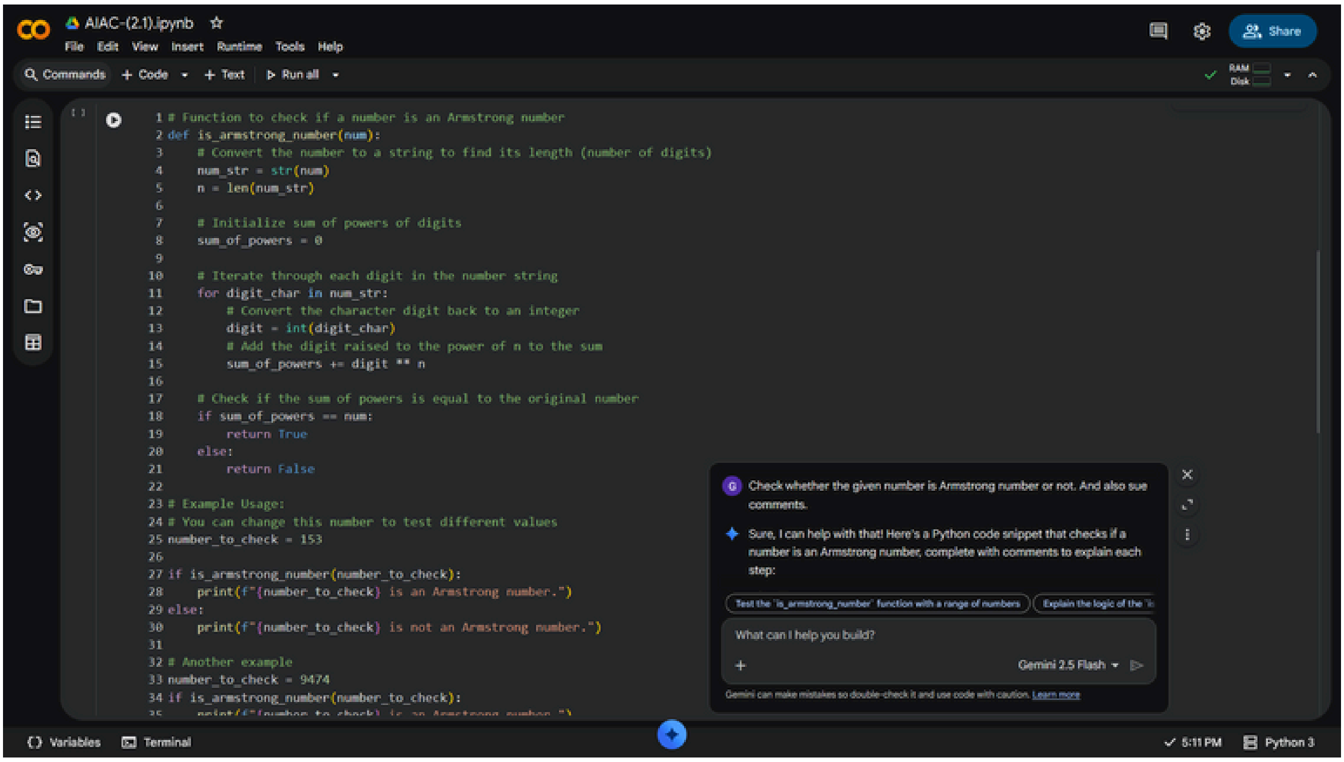
• Expected Deliverables

➢ Side-by-side comparison

Feature	Google Gemini	GitHub Co-pilot
Code Correctness	Correct	Correct
Logic Style	Straightforward, beginner-friendly	Slightly optimized, cleaner variables
Readability	Good	Very good
Comments	Basic comments	Clear inline comments
Ease of Generation	Easy in chat	Very fast while typing
Output	Correct	Correct

➢ Screenshots of prompts and generated code

#Check whether the given number is Armstrong number or not. And also sue comments.


AIAC-(2.1).ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
RAM
Disk
Variables
Terminal
5:11 PM
Python 3

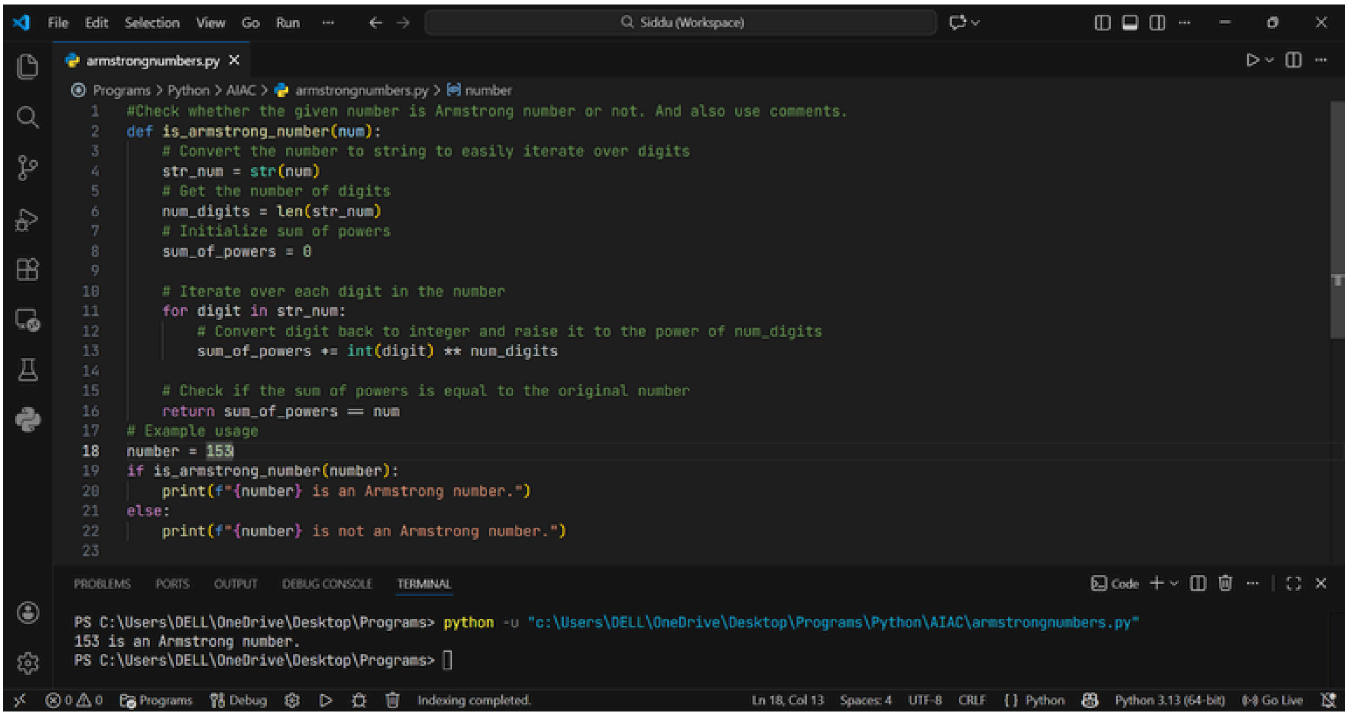
```python
# Function to check if a number is an Armstrong number
def is_armstrong_number(num):
    # Convert the number to a string to find its length (number of digits)
    num_str = str(num)
    n = len(num_str)

    # Initialize sum of powers of digits
    sum_of_powers = 0

    # Iterate through each digit in the number string
    for digit_char in num_str:
        # Convert the character digit back to an integer
        digit = int(digit_char)
        # Add the digit raised to the power of n to the sum
        sum_of_powers += digit ** n

    # Check if the sum of powers is equal to the original number
    if sum_of_powers == num:
        return True
    else:
        return False

# Example Usage:
# You can change this number to test different values
number_to_check = 153

if is_armstrong_number(number_to_check):
    print(f"{number_to_check} is an Armstrong number.")
else:
    print(f"{number_to_check} is not an Armstrong number.")

# Another example
number_to_check = 9474
if is_armstrong_number(number_to_check):
    print(f"{number_to_check} is an Armstrong number.")
```

Check whether the given number is Armstrong number or not. And also sue comments.

Sure, I can help with that! Here's a Python code snippet that checks if a number is an Armstrong number, complete with comments to explain each step:

Test the `is_armstrong_number` function with a range of numbers
Explain the logic of the `i
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

#Check whether the given number is Armstrong number or not. And also use comments.


File Edit Selection View Go Run ···
Siddu (Workspace)
armstrongnumbers.py
Programs > Python > AIAC > armstrongnumbers.py > number
PROBLEMS PORTS OUTPUT DEBUG CONSOLE TERMINAL
Code
Programs
Debug
Indexing completed.
Ln 18, Col 13 Spaces: 4 UTF-8 CRLF {} Python Python 3.13 (64-bit) Go Live

```python
#Check whether the given number is Armstrong number or not. And also use comments.
def is_armstrong_number(num):
    # Convert the number to string to easily iterate over digits
    str_num = str(num)
    # Get the number of digits
    num_digits = len(str_num)
    # Initialize sum of powers
    sum_of_powers = 0

    # Iterate over each digit in the number
    for digit in str_num:
        # Convert digit back to integer and raise it to the power of num_digits
        sum_of_powers += int(digit) ** num_digits

    # Check if the sum of powers is equal to the original number
    return sum_of_powers == num
# Example usage
number = 153
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

```
PS C:\Users\DELL\OneDrive\Desktop\Programs> python -u "c:\Users\DELL\OneDrive\Desktop\Programs\Python\AIAC\armstrongnumbers.py"
153 is an Armstrong number.
PS C:\Users\DELL\OneDrive\Desktop\Programs> 
```

Task 3: Leap Year Validation Using Cursor AI

• Scenario

You are validating a calendar module for a backend system.

• Task Description

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 Deliverables

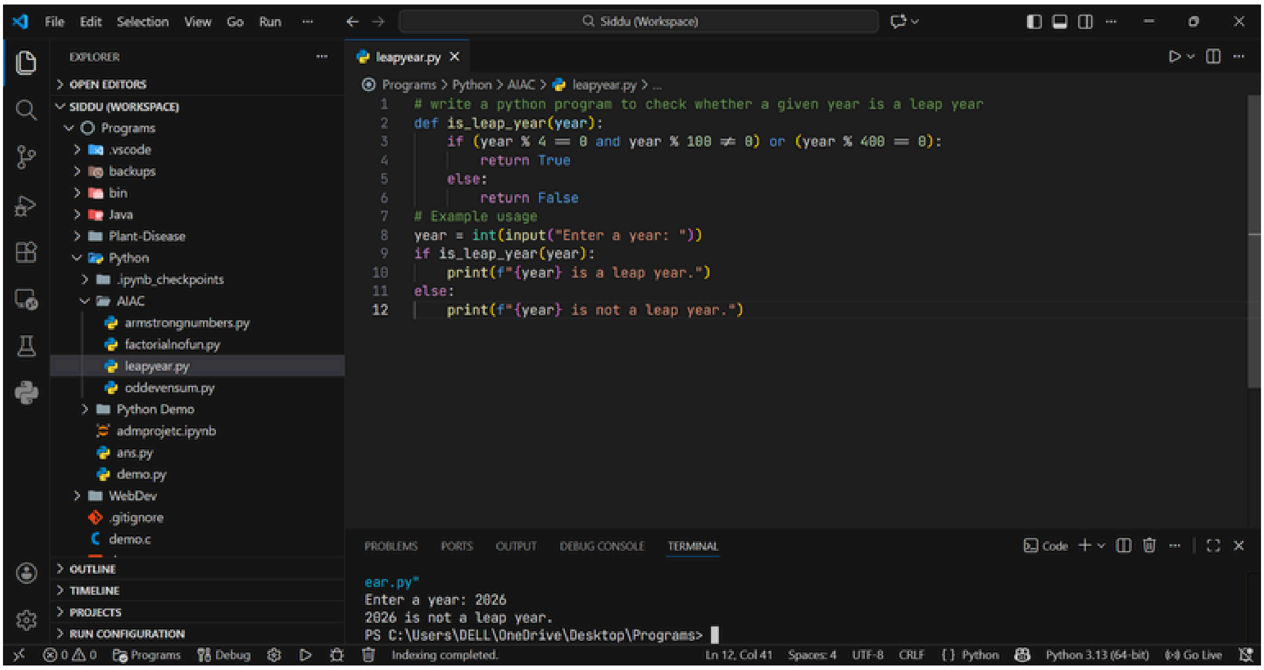
➢ Two versions of code

➢ Sample inputs/outputs

➢ Brief comparison

Aspect	**Version 1**	**Version 2**
Prompt Type	Simple	Detailed
Readability	Good	Very Good
Variable Naming	Inline condition	Uses is leap year
Comments	Minimal	Clear explanation
Maintainability	Moderate	High
Output	Correct	Correct

#Write a python program to check whether a given year is a leap year

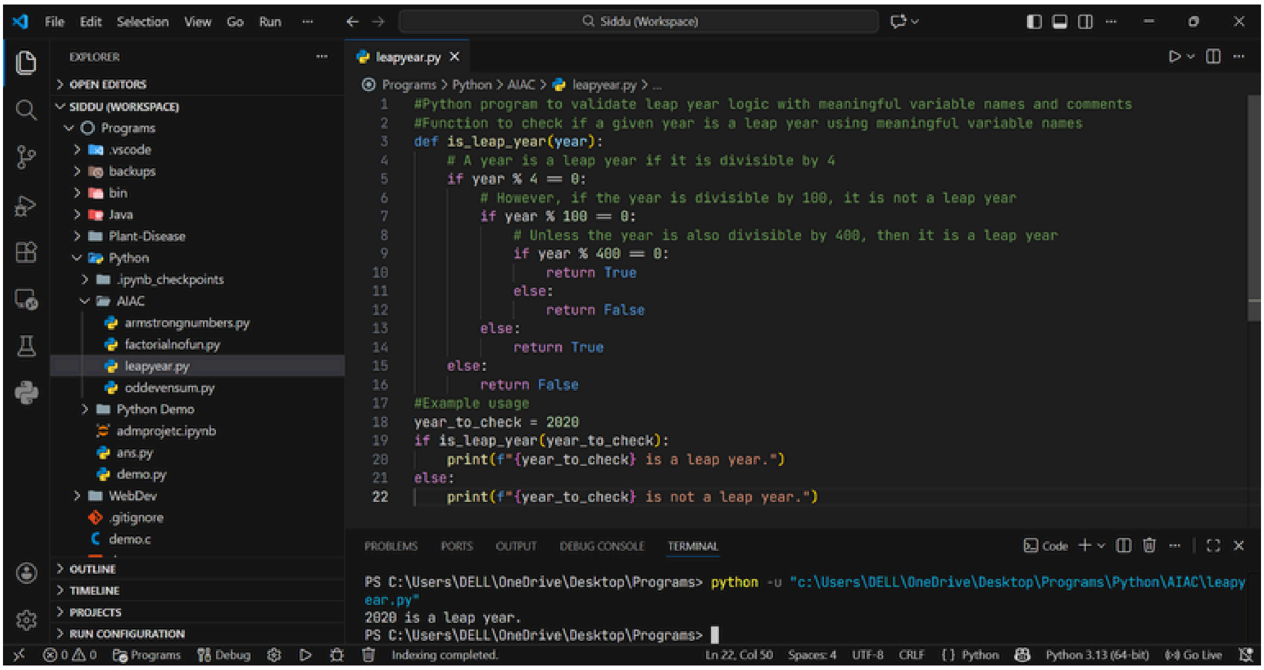


```python
# write a python program to check whether a given year is a leap year
def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False
# Example usage
year = int(input("Enter a year: "))
if is_leap_year(year):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

```
ear.py"
Enter a year: 2026
2026 is not a leap year.
PS C:\Users\DELL\OneDrive\Desktop\Programs>
```

#Python program to validate leap year logic with meaningful variable names and comments

#Function to check if a given year is a leap year using meaningful variable names



```python
#Python program to validate leap year logic with meaningful variable names and comments
#Function to check if a given year is a leap year using meaningful variable names
def is_leap_year(year):
    # A year is a leap year if it is divisible by 4
    if year % 4 == 0:
        # However, if the year is divisible by 100, it is not a leap year
        if year % 100 == 0:
            # Unless the year is also divisible by 400, then it is a leap year
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
#Example usage
year_to_check = 2020
if is_leap_year(year_to_check):
    print(f"{year_to_check} is a leap year.")
else:
    print(f"{year_to_check} is not a leap year.")
```

```
PS C:\Users\DELL\OneDrive\Desktop\Programs> python -u "c:\Users\DELL\OneDrive\Desktop\Programs\Python\AIAC\leapy
ear.py"
2020 is a leap year.
PS C:\Users\DELL\OneDrive\Desktop\Programs>
```

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

• Scenario

Company policy requires developers to write logic before using AI.

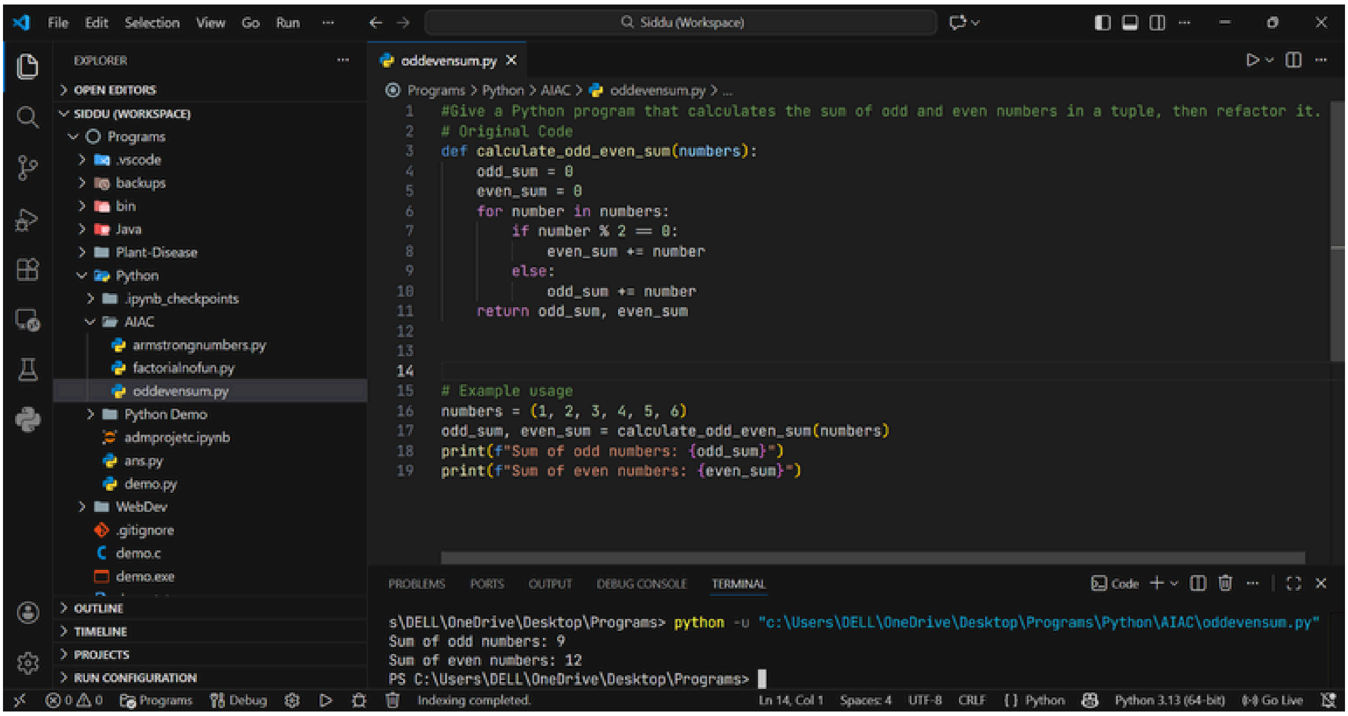
• Task Description

Write a Python program that calculates the sum of odd and even numbers in a tuple, then refactor it using any AI tool.

• Expected Deliverables

➢ Original Code

#Give a Python program that calculates the sum of odd and even numbers in a tuple, then refactor it using any AI tool.

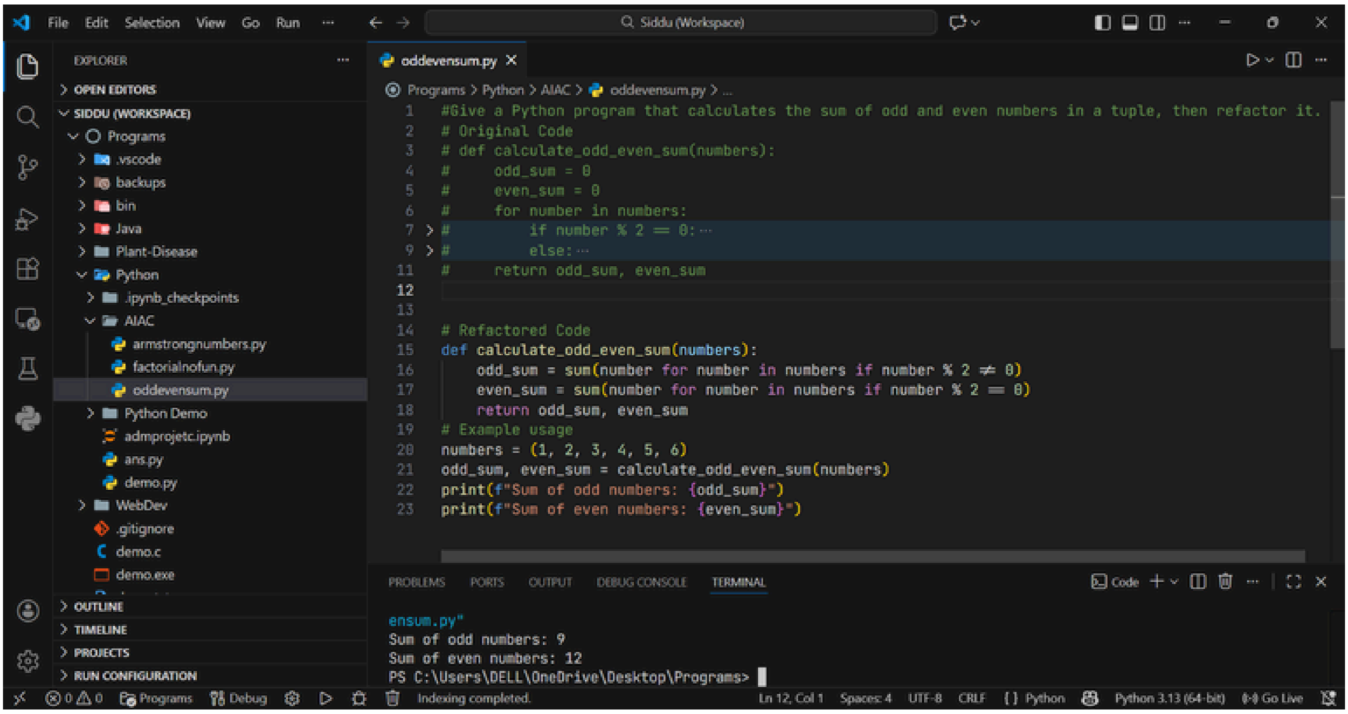


```python
#Give a Python program that calculates the sum of odd and even numbers in a tuple, then refactor it.
# Original Code
def calculate_odd_even_sum(numbers):
    odd_sum = 0
    even_sum = 0
    for number in numbers:
        if number % 2 == 0:
            even_sum += number
        else:
            odd_sum += number
    return odd_sum, even_sum



# Example usage
numbers = (1, 2, 3, 4, 5, 6)
odd_sum, even_sum = calculate_odd_even_sum(numbers)
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
```

```
s\DELL\OneDrive\Desktop\Programs> python -u "c:\Users\DELL\OneDrive\Desktop\Programs\Python\AIAC\oddevensum.py"
Sum of odd numbers: 9
Sum of even numbers: 12
PS C:\Users\DELL\OneDrive\Desktop\Programs>
```

➢ Refactored Code

# Refactored Version (using list comprehensions)



```python
#Give a Python program that calculates the sum of odd and even numbers in a tuple, then refactor it.
# Original Code
# def calculate_odd_even_sum(numbers):
#     odd_sum = 0
#     even_sum = 0
#     for number in numbers:
#         if number % 2 == 0:···
#         else:···
#     return odd_sum, even_sum


# Refactored Code
def calculate_odd_even_sum(numbers):
    odd_sum = sum(number for number in numbers if number % 2 != 0)
    even_sum = sum(number for number in numbers if number % 2 == 0)
    return odd_sum, even_sum
# Example usage
numbers = (1, 2, 3, 4, 5, 6)
odd_sum, even_sum = calculate_odd_even_sum(numbers)
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
```

```
ensum.py"
Sum of odd numbers: 9
Sum of even numbers: 12
PS C:\Users\DELL\OneDrive\Desktop\Programs>
```

- Explanation of improvements

Aspect	Original Code	Refactored Code
Logic Style	Uses explicit loop and condition	Uses Python built-in sum()
Readability	Clear but longer	Shorter and cleaner
Performance	Multiple operations in loop	Optimized with generator expressions
Maintainability	More lines to manage	Easier to modify and review
Code Quality	Basic procedural style	Pythonic and concise