**Task 1**

**Description:** Use Google Gemini in Colab to write a function that reads a CSV file and calculates mean, min, max.

**Expected Output:** Functional code with output and screenshot**.**

**Prerequisites:** Web Browser (Google Chrome, Mozilla FireFox, etc.), Google Colab with Gemini and Linked Google Account, Internet Connection.

**simple\_numbers.csv:**

Value

45

67

23

89

34

78

56

91

12

65

**Prompt:** Write code for a function that reads the CSV file (simple\_numbers.csv) and calculates the mean, min, max

**Gemini (AI) Generated Code:**

import pandas as pd

def analyze\_csv(file\_path):

  """

  Reads a CSV file and calculates the mean, min, and max of the data.

  Args:

    file\_path: The path to the CSV file.

  Returns:

    A dictionary containing the mean, min, and max of the data,

    or None if an error occurs.

  """

  try:

    df = pd.read\_csv(file\_path)

    mean\_value = df.mean().iloc[0]

    min\_value = df.min().iloc[0]

    max\_value = df.max().iloc[0]

    return {"mean": mean\_value, "min": min\_value, "max": max\_value}

  except Exception as e:

    print(f"Error analyzing CSV file: {e}")

    return None

# Example usage (assuming simple\_numbers.csv exists)

results = analyze\_csv('/content/simple\_numbers.csv')

if results:

  print("Analysis Results:")

  print(f"Mean: {results['mean']}")

  print(f"Min: {results['min']}")

  print(f"Max: {results['max']}")

**Output:**

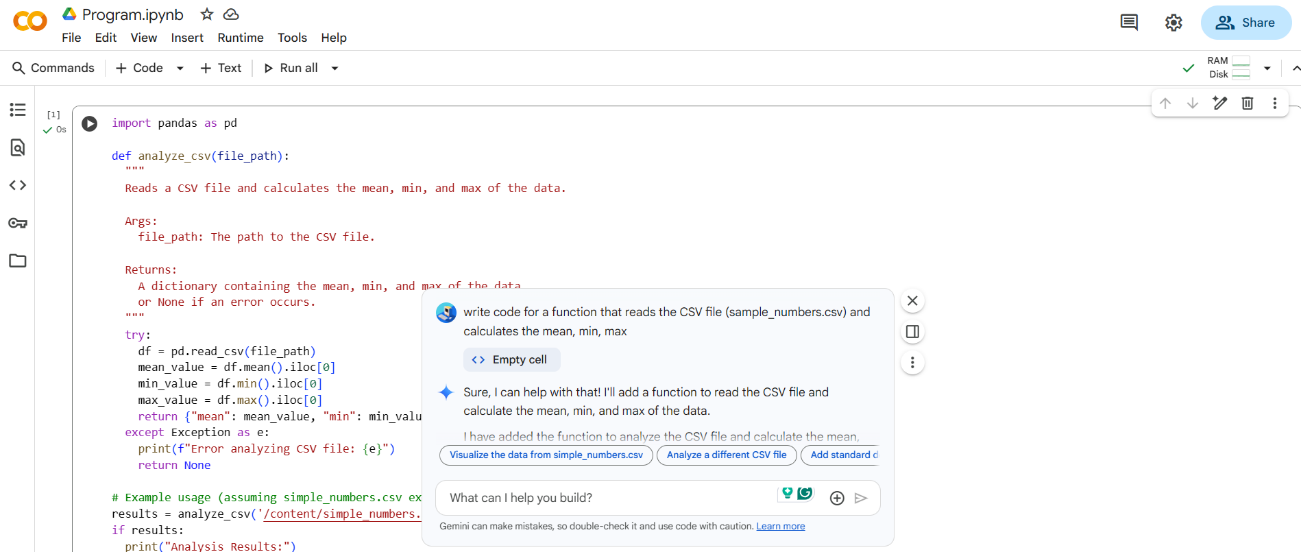
Analysis Results:

Mean: 56.0

Min: 12

Max: 91

**Screenshot:**



**Task 2**

**Description:** Compare Gemini and Copilot outputs for a palindrome check function.

**Expected Output:** Side-by-side comparison and observations**.**

**Prerequisites:** Web Browser (Google Chrome, Mozilla FireFox, etc.), Google Colab with Gemini and Linked Google Account, GitHub Copilot installed in VS Code, Internet Connection.

**Prompt:** Write code for a function that checks for palindrome

**Side-by-side Comparison:**

|  |  |
| --- | --- |
| **Gemini (AI) Generated Code** | **Copilot (AI) Generated Code** |
| def is\_palindrome(s):    """    Checks if a string is a palindrome.    Args:      s: The string to check.    Returns:      True if the string is a palindrome, False otherwise.    """    # Remove non-alphanumeric characters and convert to lowercase    s = ''.join(filter(str.isalnum, s)).lower()    return s == s[::-1]  # Example usage:  print(is\_palindrome("racecar"))  print(is\_palindrome("hello"))  print(is\_palindrome("A man, a plan, a canal: Panama")) | def is\_palindrome(text):      # Remove non-alphanumeric characters and convert to lowercase      cleaned\_text = ''.join(char.lower() for char in text if char.isalnum())        # Compare the string with its reverse      return cleaned\_text == cleaned\_text[::-1]  # Example usage  if \_\_name\_\_ == "\_\_main\_\_":      # Test cases      test\_strings = ["A man, a plan, a canal: Panama", "race a car", "Was it a car or a cat I saw?"]        for text in test\_strings:          if is\_palindrome(text):              print(f'"{text}" is a palindrome')          else:              print(f'"{text}" is not a palindrome') |
| **Gemini (AI) Generated Code Output** | **Copilot (AI) Generated Code Output** |
| True  False  True | "A man, a plan, a canal: Panama" is a palindrome  "race a car" is not a palindrome  "Was it a car or a cat I saw?" is a palindrome |

**Observations:** The code generated by Gemini contains in detail documentation of the is\_palindrome() function in a well-structured format before function implementation and has also handled the non-alphanumeric characters, while the Copilot starts with the function implementation lacking proper documentation and doesn’t handle non-alphanumeric characters. The testing done in Copilot’s code is better compared to Gemini’s code. The output generated by both is almost similar, where Gemini’s code has given the return value of the function as output and Copilot’s code has given a user-friendly structured output.

**Task 3**

**Description:** Ask Gemini to explain a Python function (to calculate the area of various shapes) line by line.

**Expected Output:** Detailed explanation with code snippet**.**

**Prerequisites:** Web Browser (Google Chrome, Mozilla FireFox, etc.), Google Colab with Gemini and Linked Google Account, Internet Connection.

**Prompt:** Explain a Python function (to calculate the area of various shapes) line by line

**Gemini (AI) Generated Code:**

import math

def calculate\_area(shape, dimensions):

  """

  Calculates the area of various shapes.

  Args:

    shape: A string representing the shape (e.g., 'circle', 'square', 'triangle').

    dimensions: A tuple or list of dimensions required for the shape.

  Returns:

    The area of the shape, or None if the shape is not supported or dimensions are incorrect.

  """

  if shape == 'circle':

    # Check if dimensions contain a radius

    if len(dimensions) == 1:

      radius = dimensions[0]

      return math.pi \* radius\*\*2

    else:

      return None # Incorrect dimensions for circle

  elif shape == 'square':

    # Check if dimensions contain a side length

    if len(dimensions) == 1:

      side = dimensions[0]

      return side\*\*2

    else:

      return None # Incorrect dimensions for square

  elif shape == 'triangle':

    # Check if dimensions contain base and height

    if len(dimensions) == 2:

      base, height = dimensions

      return 0.5 \* base \* height

    else:

      return None # Incorrect dimensions for triangle

  else:

    return None # Unsupported shape

# Example usage:

print(calculate\_area('circle', (5,)))

print(calculate\_area('square', (4,)))

print(calculate\_area('triangle', (6, 3)))

print(calculate\_area('rectangle', (5, 4))) # Unsupported shape

**Gemini (AI) Generated Content:**

Here is a line-by-line explanation of the calculate\_area function:

import math

* This line imports the math module, which provides mathematical functions like math.pi for calculating the area of a circle.

def calculate\_area(shape, dimensions):

* This line defines a function named calculate\_area that takes two arguments: shape (a string representing the type of shape) and dimensions (a tuple or list containing the necessary measurements for the shape).

"""..."""

* This is a docstring, which explains what the function does, its arguments, and what it returns. It's good practice for documenting your code.

if shape == 'circle':

* This line starts a conditional block. It checks if the value of the shape argument is exactly the string 'circle'.

if len(dimensions) == 1:

* Inside the 'circle' block, this line checks if the dimensions tuple/list contains exactly one element, which is expected for a circle (the radius).

radius = dimensions[0]

* If the condition is true, this line assigns the first element of the dimensions tuple/list to the variable radius.

return math.pi \* radius\*\*2

* This line calculates the area of the circle using the formula π×radius2 and returns the result. math.pi provides the value of pi, and radius\*\*2 calculates the square of the radius.

else:

* This is the else part of the inner if statement. It executes if the dimensions for a circle are not correct.

return None # Incorrect dimensions for circle

* This line returns None to indicate that the area could not be calculated due to incorrect dimensions for a circle. The comment explains the reason.

elif shape == 'square':

* This line starts another conditional block. It checks if the value of the shape argument is 'square'.

if len(dimensions) == 1:

* Inside the 'square' block, this line checks if the dimensions contain exactly one element (the side length).

side = dimensions[0]

* If the condition is true, this line assigns the first element of dimensions to the variable side.

return side\*\*2

* This line calculates the area of the square using the formula side2 and returns the result.

else:

* This is the else part for the 'square' shape, executed if the dimensions are incorrect.

return None # Incorrect dimensions for square

* This line returns None for incorrect square dimensions.

elif shape == 'triangle':

* This line checks if the shape is 'triangle'.

if len(dimensions) == 2:

* Inside the 'triangle' block, this line checks if the dimensions contain exactly two elements (base and height).

base, height = dimensions

* If the condition is true, this line unpacks the two elements from dimensions into the base and height variables.

return 0.5 \* base \* height

* This line calculates the area of the triangle using the formula 0.5×base×height and returns the result.

else:

* This is the else part for the 'triangle' shape, executed if the dimensions are incorrect.

return None # Incorrect dimensions for triangle

* This line returns None for incorrect triangle dimensions.

else:

* This is the final else block, which is executed if none of the previous elif conditions for shapes ('circle', 'square', 'triangle') were met.

return None # Unsupported shape

* This line returns None to indicate that the provided shape is not supported by the function. The comment clarifies this.

**Task 4**

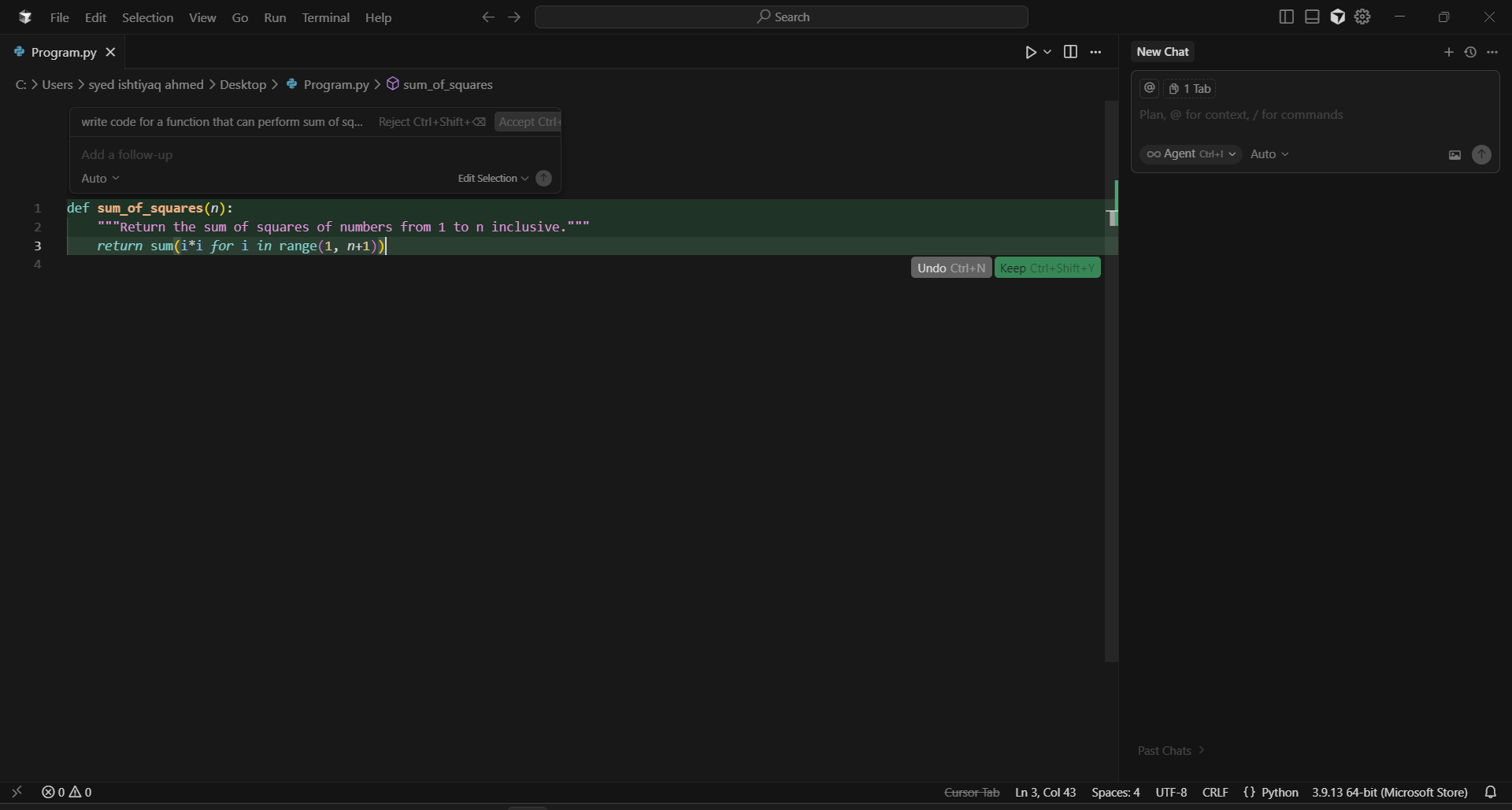
**Description:** Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of squares).

**Expected Output:** Screenshots of working environments with few prompts to generate python code.

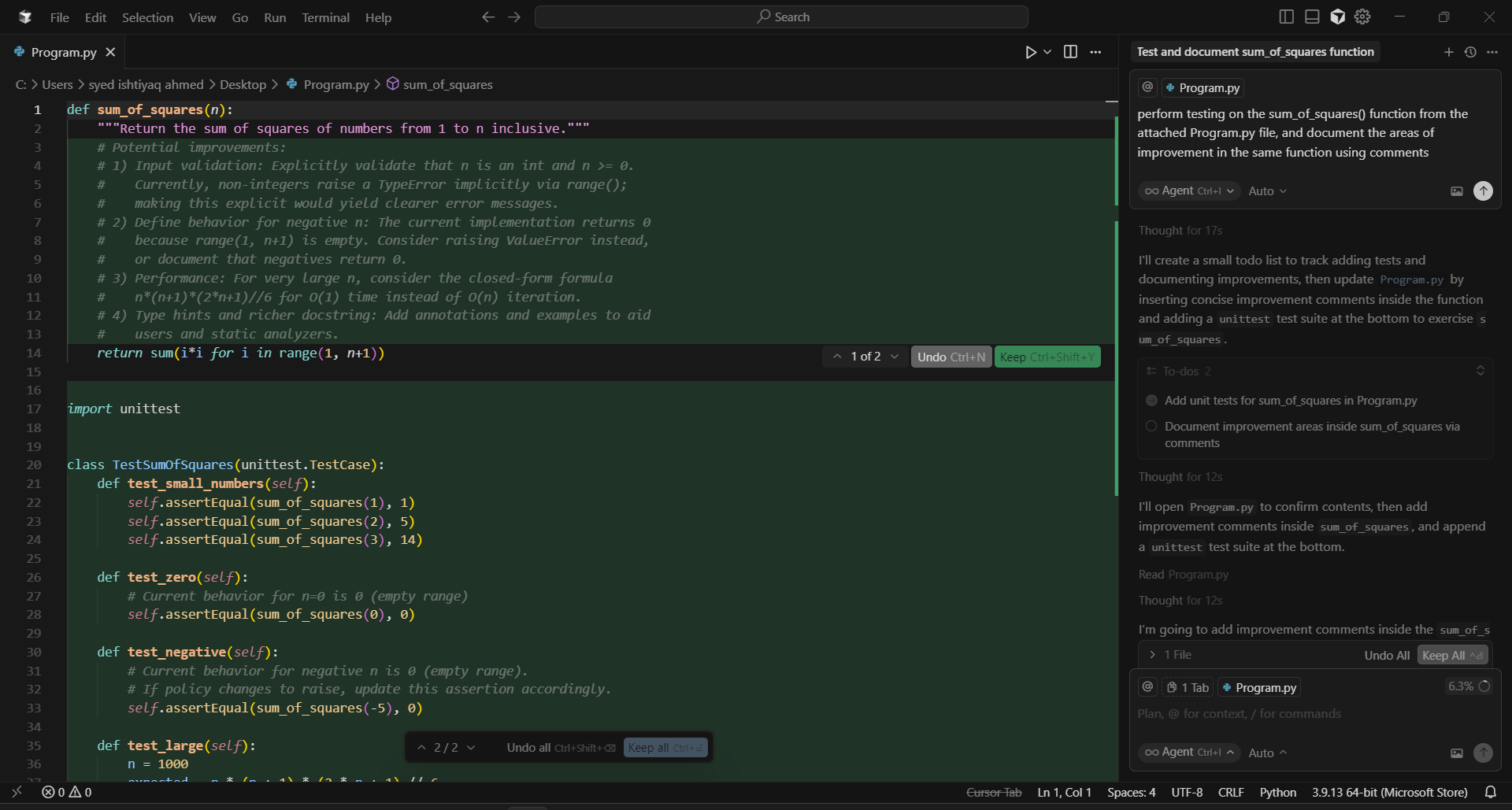
**Prerequisites:** Cursor AI Software, Internet Connection.

**Prompts and Screenshots:**

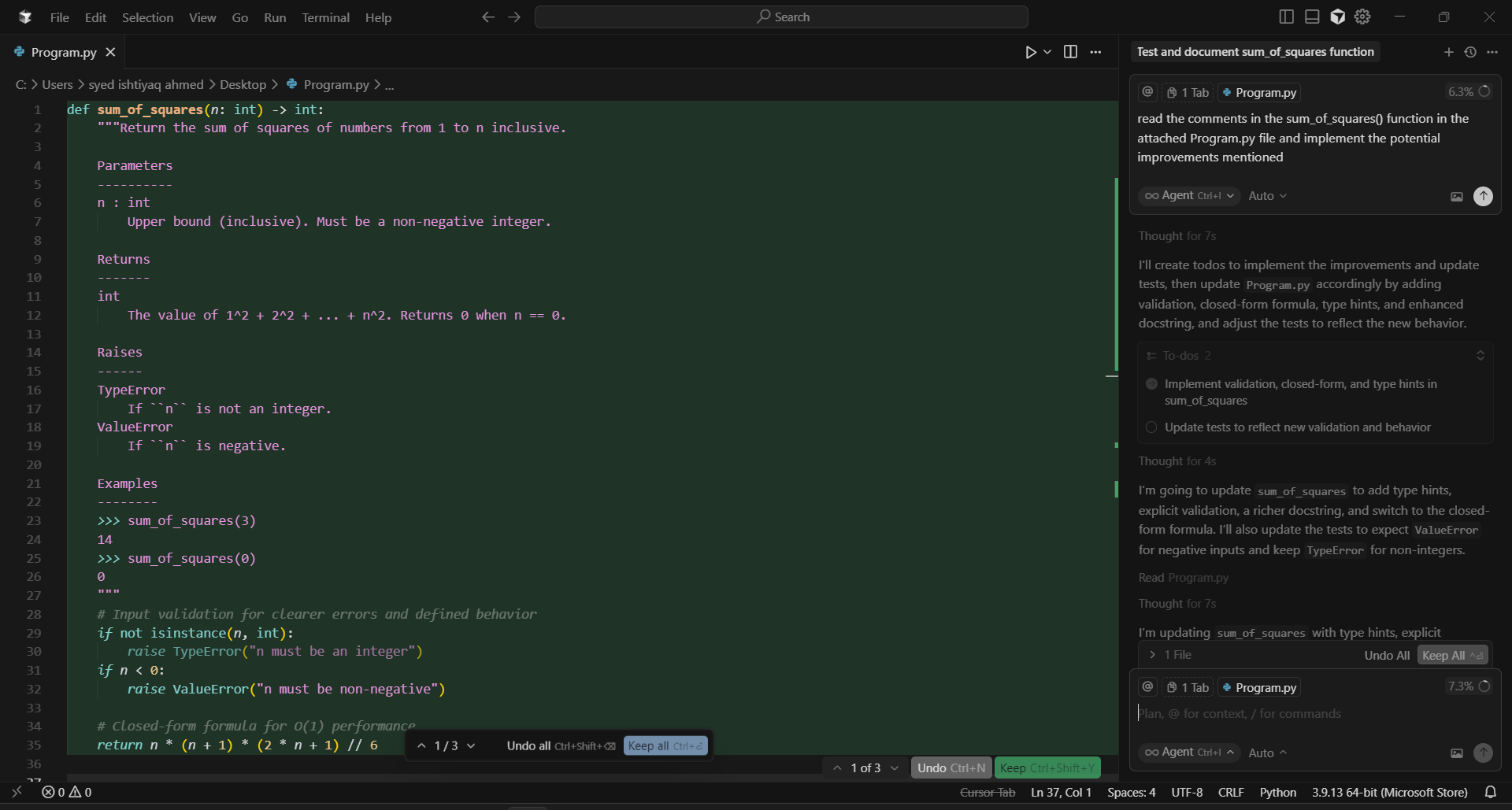
1. Write code for a function that can perform sum of squares upto n numbers in python



1. Perform testing on the sum\_of\_squares() function from the attached Program.py file, and document the areas of improvement in the same function using comments



1. Read the comments in the sum\_of\_squares() function in the attached Program.py file and implement the potential improvements mentioned



**Task 5**

**Description:** Write code to calculate sum of odd numbers and even numbers in the list.

**Expected Output:** Refactored code with improved logic

**Prerequisites:** Cursor AI Software, Internet Connection.

**Code (Program.py):**

def sum\_of\_even\_numbers\_and\_odd\_numbers\_in\_list(list):

even\_numbers\_sum = 0

odd\_numbers\_sum = 0

n = len(list)

for i in range(n):

if list[i] % 2 == 0:

even\_numbers\_sum += list[i]

else:

odd\_numbers\_sum += list[i]

return even\_numbers\_sum, odd\_numbers\_sum

list1 = [1,2,3,5,8,9,13,45]

even\_numbers\_sum, odd\_numbers\_sum = sum\_of\_even\_numbers\_and\_odd\_numbers\_in\_list(list1)

print("Sum of even numbers in the list:", even\_numbers\_sum)

print("Sum of odd numbers in the list:", odd\_numbers\_sum)

**Prompt:** Refactor the code and improve the logic for the function sum\_of\_even\_numbers\_and\_odd\_numbers\_in\_list() in the attached Program.py file

**Cursor (AI) Generated Refactored Code with Improved Logic:**

def sum\_of\_even\_numbers\_and\_odd\_numbers\_in\_list(*values*):

    """Return (sum\_of\_even\_numbers, sum\_of\_odd\_numbers) from an iterable of values.

    Non-integer items are ignored.

    """

    even\_numbers\_sum = sum(v *for* v *in* *values* *if* isinstance(v, int) *and* v % 2 == 0)

    odd\_numbers\_sum = sum(v *for* v *in* *values* *if* isinstance(v, int) *and* v % 2 != 0)

*return* even\_numbers\_sum, odd\_numbers\_sum

list1 = [1,2,3,5,8,9,13,45]

even\_numbers\_sum, odd\_numbers\_sum = sum\_of\_even\_numbers\_and\_odd\_numbers\_in\_list(list1)

print("Sum of even numbers in the list:", even\_numbers\_sum)

print("Sum of odd numbers in the list:", odd\_numbers\_sum)