# AI Assisted Coding

## Assignment 2.5

Name: V.Harivamsh

Hall ticket no: 2303A51266

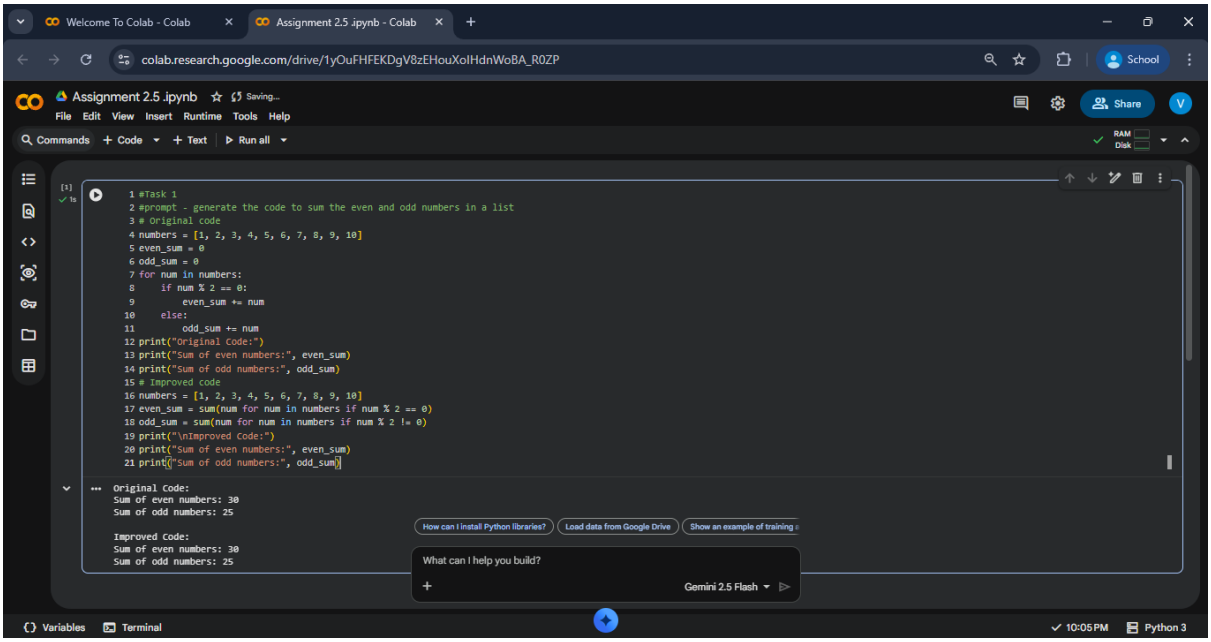
Batch no: 19

Task 1:

Prompt:

Generate the code to sum of even and odd numbers in a list

Code& Output:




Explanation:

The original code uses a standard for-loop combined with if/else checks to go through the list and add up values. It works correctly but ends up being longer and more manual.

The updated version takes advantage of Python's built-in features:

• **Generator Expression:** Instead of building temporary lists, it uses a generator (number for number in my_list if condition) directly inside `sum()`, which saves memory because values are produced as needed.

• **sum() Function:** Python's `sum()` is a fast, built-in tool designed specifically for adding up elements from an iterable.

Overall, this rewritten method is cleaner, shorter, and more in line with typical Python style. It's also easier to read for experienced Python users, especially when dealing with simple calculations like this.

Task 2:

Ask Gemini to explain a function that calculates the area of different shapes.

Prompt:

Explain a function that calculates the area of different shapes.

Code:

```
import math  # We need this module to use the value of Pi (3.14159...)


def calculate_area(shape_type, dimension1, dimension2=0):
    """
    Function to calculate the area of a Circle, Rectangle, or Triangle.

    Parameters:
    - shape_type (str): The name of the shape (e.g., 'circle', 'rectangle', 'triangle').
    - dimension1 (float): The first dimension (Radius for Circle, Length for Rectangle, Base for Triangle).
    - dimension2 (float): The second dimension (Width for Rectangle, Height for Triangle). Default is 0 for Circle.
    """

    area = 0  # Initialize variable to store the result

    # 1. Logic for Circle
    # Formula: Area = Pi * r^2
    if shape_type == "circle":
        radius = dimension1
        area = math.pi * (radius ** 2)
```

```
        print(f"Calculated area for Circle with radius {radius}")

    # 2. Logic for Rectangle
    # Formula: Area = Length * Width
    elif shape_type == "rectangle":
        length = dimension1
        width = dimension2
        area = length * width
        print(f"Calculated area for Rectangle with length {length} and width {width}")

    # 3. Logic for Triangle
    # Formula: Area = 0.5 * Base * Height
    elif shape_type == "triangle":
        base = dimension1
        height = dimension2
        area = 0.5 * base * height
        print(f"Calculated area for Triangle with base {base} and height {height}")

    # 4. Error handling for unknown shapes
    else:
        print(f"Error: The shape '{shape_type}' is not supported.")
        return None

    return area

# --- Driver Code (Testing the function) ---
print("--- Area Calculator Results ---")
```

```
# Test Case 1: Circle
# We only pass one dimension (radius) because dimension2 defaults to 0
circle_area = calculate_area("circle", 5)
print(f"Result: {circle_area:.2f}\n") # .2f restricts output to 2 decimal places

# Test Case 2: Rectangle
# We pass length (10) and width (5)
rect_area = calculate_area("rectangle", 10, 5)
print(f"Result: {rect_area:.2f}\n")

# Test Case 3: Triangle
# We pass base (10) and height (5)
tri_area = calculate_area("triangle", 10, 5)
print(f"Result: {tri_area:.2f}\n")
```

**Output:**

```
--- Area Calculator Results ---
Calculated area for Circle with radius 5
Result: 78.54
Calculated area for Rectangle with length 10 and width 5
Result: 50.00
Calculated area for Triangle with base 10 and height 5
Result: 25.00
```

**Explanation:**

1. The `calculate_area` function is used to find the area of circles, rectangles, and triangles using just one function instead of writing three separate ones.
2. The `math` module is imported so we can use `math.pi`, which gives us the value of Pi for circle area calculations.
3. Depending on the shape you give, the function applies the right formula — circles use $\pi \times \text{radius}^2$, rectangles use length×width, and triangles use 0.5×base×height.

4. If the user enters a shape that the function doesn't recognize, it shows an error message instead of crashing.
5. The sample test cases at the end show how the function works, and the results are printed with two decimal places to look clean.

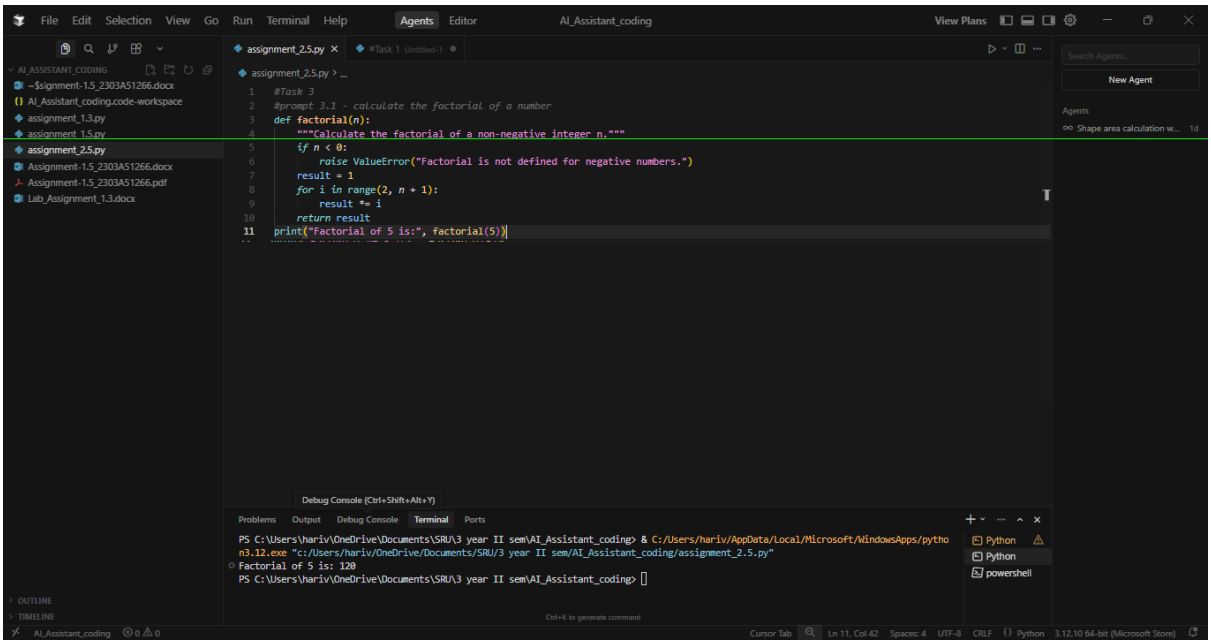
**Task 3:**

Use Cursor AI with different prompts for the same problem and observe code changes.

**Prompt 3.1:**

Calculate the factorial of a number

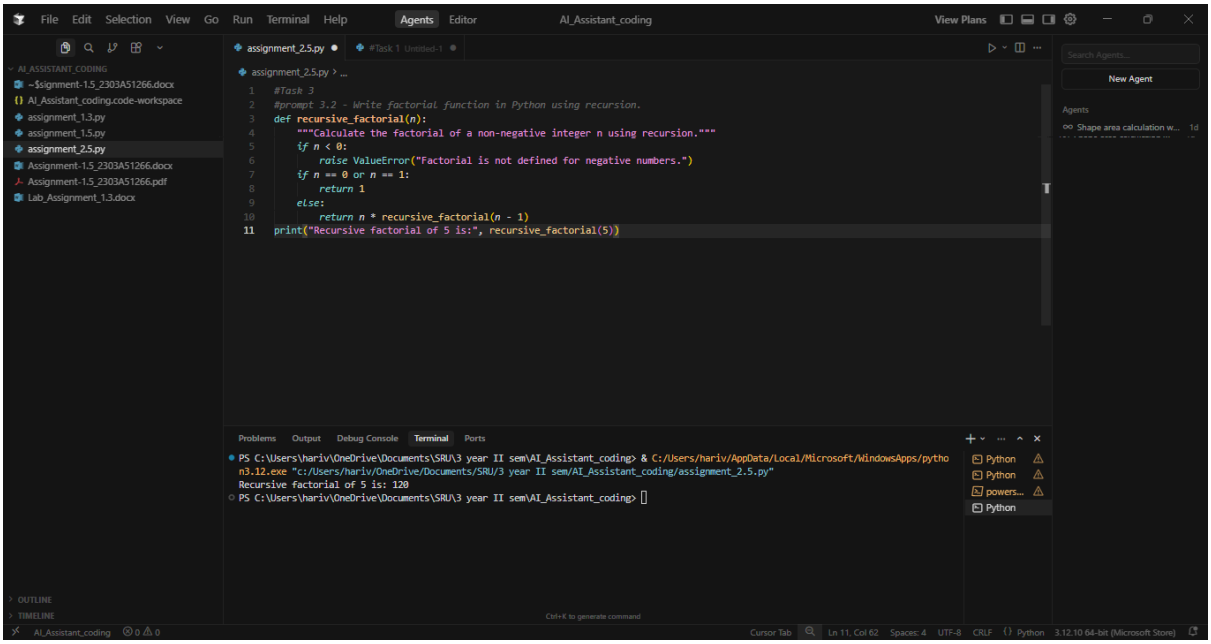
**Code& Output:**




**Prompt 3.2 :**

Write factorial function in python using recursion

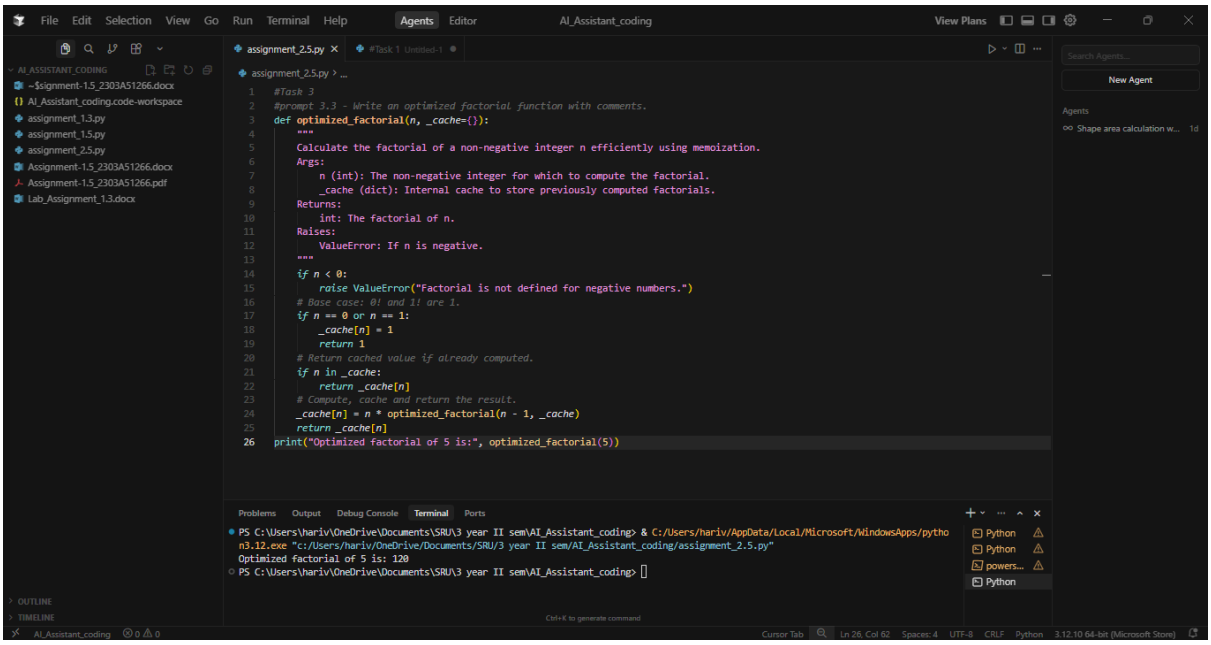
**Code & Output:**




**Prompt 3.3:**

Write an optional factorial function with comments.

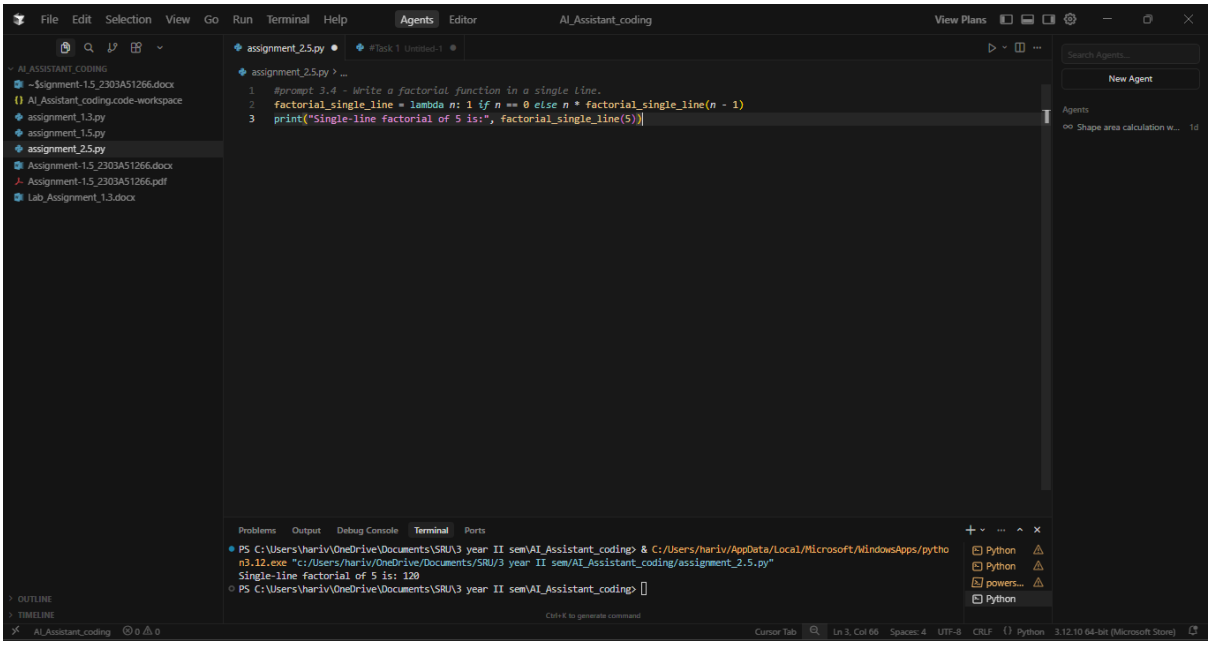
**Code & Output:**




**Prompt 3.4:**

Write a factorial function in a single line.

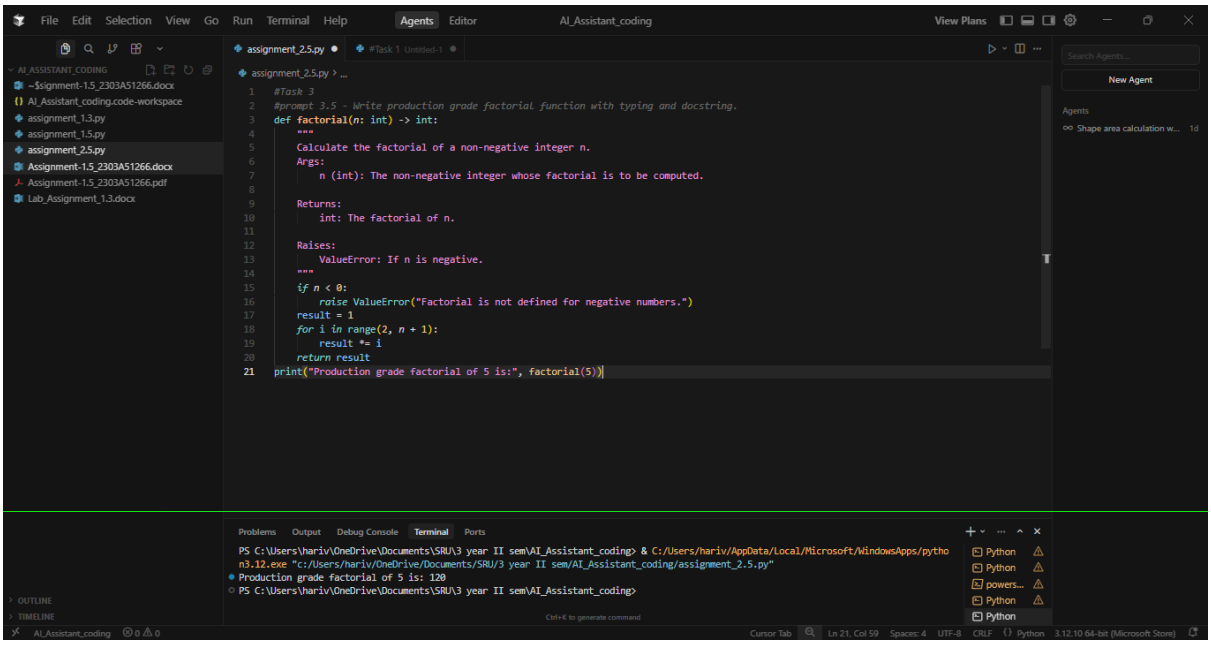
**Code & Output:**




**Prompt 3.5:**

Write production grade factorial function with typing and docstring.

**Code & Output:**




**Explanation:**

In this task, I asked Cursor to generate the factorial program using different prompts. Even though the problem was the same, the tool produced different versions of the code depending on how I asked. First, it gave a simple loop-based factorial, then a recursive version when I mentioned recursion, and later an optimized one when I asked for better performance. It also handled a single-line lambda version and finally a production-style version with typing and

docstrings. This shows that the tool understands prompt details and adjusts the code style, structure, and features based on how clearly the prompt is written.

**Task 4:**

Compare Gemini, Copilot, and Cursor AI for usability and code quality.

Short reflection:

Gemini, Copilot, and Cursor AI all support programmers, but they each focus on different strengths. Gemini works well as a learning assistant because it explains concepts clearly and provides clean, understandable examples. However, it tends to stay within basic, beginner-friendly patterns and doesn't easily adapt to more advanced coding styles or specialized requirements, so it fits educational use more than serious development work.

GitHub Copilot focuses on speed inside the code editor. It predicts and completes lines or entire blocks of code as you type, which makes development faster. The downside is that it doesn't check whether the overall logic is correct—if the user's approach is flawed, Copilot typically continues following that mistake without fixing it.

Cursor AI stands out because it can adjust the way it writes code depending on how the problem is described. The same prompt can produce beginner-style code, optimized code, or production-ready code, showing that Cursor understands both context and intention. This leads to cleaner structure and more practical output for real projects.

For these reasons, Cursor AI offers the most well-rounded experience. It combines ease of use with strong code quality and adaptability, making it the most suitable tool for real-world software development.