AI Assisted Coding Ass-2.5

Name: Ch.Sanju

Roll Number: 2303A52313

Batch: 45

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

Lab Objectives:

- To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab.
- To understand and use Cursor AI for code generation, explanation, and refactoring.
- To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI.
- To perform code optimization and documentation using AI tools.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Generate Python code using Google Gemini in Google Colab.
- Analyze the effectiveness of code explanations and suggestions by Gemini.
- Set up and use Cursor AI for AI-powered coding assistance.
- Evaluate and refactor code using Cursor AI features.
- Compare AI tool behavior and code quality across different platforms.

Task 1: Refactoring Odd/Even Logic (List Version)

- **Scenario:**
You are improving legacy code.
- **Task:**
Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI.
- **Expected Output:**
- Original and improved code

Code:

Gemini AI:

 # generate a program to calculate the sum of odd and even numbers in a list

```
def sum_odd_even(numbers):
    """
    Calculates the sum of odd and even numbers in a list.
```

```
    Args:
      numbers (list): A list of integers.

    Returns:
      tuple: A tuple containing two integers: (sum_of_even_numbers, sum_of_odd_numbers).
    """
    sum_even = 0
sum_odd = 0     for
number in numbers:
if number % 2 == 0:
          sum_even += number
      else:
          sum_odd += number
return sum_even, sum_odd

# Example usage:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum, odd_sum = sum_odd_even(my_list)

print(f"Original list: {my_list}")
print(f"Sum of even numbers: {even_sum}") print(f"Sum
of odd numbers: {odd_sum}")

# Another example
list_2 = [11, 22, 33, 44, 55]
even_sum_2, odd_sum_2 = sum_odd_even(list_2)
print(f"\nOriginal list: {list_2}")
print(f"Sum of even numbers: {even_sum_2}") print(f"Sum
of odd numbers: {odd_sum_2}") Output:
```

```
•••  Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
     Sum of even numbers: 30
     Sum of odd numbers: 25

     Original list: [11, 22, 33, 44, 55]
     Sum of even numbers: 66
     Sum of odd numbers: 99
```

Cursor Ai:

Prompt: # generate a program to calculate the sum of odd and even numbers in a list

```
def sum_odd_even(numbers):
    """
    Calculate the sum of odd and even numbers in a list.

    Args:       numbers: List of
integers

    Returns:
        tuple: (sum_of_odd, sum_of_even)
    """
    sum_odd = 0
sum_even = 0

    for num in numbers:
    if num % 2 == 0:
            sum_even += num
        else:
            sum_odd += num

    return sum_odd, sum_even
```

```
# Example usage if
__name__ == "__main__":
    # Example list     numbers = [1, 2, 3,
4, 5, 6, 7, 8, 9, 10]

    # Calculate sums
    odd_sum, even_sum = sum_odd_even(numbers)

    # Display results     print(f"List: {numbers}")
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")

    # Interactive version - uncomment to use
    # user_input = input("Enter numbers separated by spaces: ")
    # numbers = [int(x) for x in user_input.split()]
    # odd_sum, even_sum = sum_odd_even(numbers)
    # print(f"\nSum of odd numbers: {odd_sum}")
    # print(f"Sum of even numbers: {even_sum}")
```

Output:

```
==========================================================
(.venv) PS D:\3-2\AI Assitant Coding> & "D:\3-2\AI Assitant Coding\.venv\Scripts\python.exe" "d:/3-2/AI Assitant Coding/sum_odd_even.py"
List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of even numbers: 30
(.venv) PS D:\3-2\AI Assitant Coding>
```

Task 2: Area Calculation Explanation

✞ **Scenario:**
You are onboarding a junior developer. ✞

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

✞ **Expected Output:** ❍ Code
❍ Explanation

Gemini:

Prompt:

"""Scenario: You are onboarding a junior developer. explain a function that calculates the area of different shapes. □ Expected Output: □ Code □ Explanation"""

Code:

```
import math


def calculate_area(shape, **kwargs):
    """
    Calculates the area of different geometric shapes.


    Args:
        shape (str): The type of the shape ('circle', 'square', 'rectangle').
        **kwargs: Keyword arguments specific to the shape:
- For 'circle': radius (float or int)
- For 'square': side (float or int)
- For 'rectangle': length (float or int), width (float or int)


    Returns:
        float: The calculated area of the shape.
        str: An error message if the shape is not recognized or required arguments are missing.
    """     if shape.lower() ==
'circle':
        if 'radius' in kwargs:
            return math.pi * (kwargs['radius'] ** 2)
        else:
            return "Error: Radius is required for a circle."
    elif shape.lower() == 'square':

        if 'side' in kwargs:
            return kwargs['side'] ** 2
    else:
            return "Error: Side length is required for a square."
    elif shape.lower() == 'rectangle':
```

```
        if 'length' in kwargs and 'width' in kwargs:
return kwargs['length'] * kwargs['width']
        else:
            return "Error: Both length and width are required for a rectangle."
    else:
        return "Error: Unknown shape type. Supported shapes are 'circle', 'square', 'rectangle'."


# Example usage:
print(f"Area of circle with radius 5: {calculate_area('circle', radius=5)}") print(f"Area
of square with side 4: {calculate_area('square', side=4)}")
print(f"Area of rectangle with length 6 and width 3: {calculate_area('rectangle', length=6,
width=3)}") print(f"Area of unknown shape: {calculate_area('triangle', base=10,
height=5)}") print(f"Area of circle without radius: {calculate_area('circle')}")
```

output:

```
•••  Area of circle with radius 5: 78.53981633974483
     Area of square with side 4: 16
     Area of rectangle with length 6 and width 3: 18
     Area of unknown shape: Error: Unknown shape type. Supported shapes are 'circle', 'square', 'rectangle'.
     Area of circle without radius: Error: Radius is required for a circle.
```

Cursor AI:

Prompt: Scenario: You are onboarding a junior developer. explain a function that calculates the area of different shapes. □ Expected Output: □ Code □ Explanation

Code:

```
"""
Area Calculator for Different Shapes
This program demonstrates how to calculate the area of various geometric shapes.
"""


import math


def calculate_circle_area(radius):
    """
    Calculate the area of a circle.
```

Formula: Area = π × r² where π (pi) ≈ 3.14159 and r is the radius

Args: radius (float): The radius of the circle

Returns: float: The area of the circle

""" *if radius* < 0:

raise ValueError("Radius cannot be negative")

return math.pi * *radius* ** 2

def calculate_rectangle_area(*length*, *width*):

"""

Calculate the area of a rectangle.

Formula: Area = length × width

Args:

length (float): The length of the rectangle

width (float): The width of the rectangle Returns:

float: The area of the rectangle

""" *if length* < 0 or *width* < 0:

raise ValueError("Length and width cannot be negative")

return length * *width*

def calculate_triangle_area(*base*, *height*):

"""

Calculate the area of a triangle.

Formula: Area = (1/2) × base × height

Args:

base (float): The base of the triangle height (float): The height of the triangle (perpendicular to base)

Returns:

float: The area of the triangle

"""

if base < 0 or *height* < 0: *raise* ValueError("Base and height cannot be negative") *return* 0.5 * *base* * *height*

def calculate_square_area(*side*):

"""

Calculate the area of a square.

Formula: Area = side²

(A square is a special rectangle where length = width)

Args: side (float): The length of one side of the square

Returns:

float: The area of the square

""" *if side* < 0:

raise ValueError("Side cannot be negative")

return side ** 2

def calculate_trapezoid_area(*base1*, *base2*, *height*):

"""

Calculate the area of a trapezoid.

Formula: Area = (1/2) × (base1 + base2) × height

Args:

base1 (float): The length of the first parallel side

base2 (float): The length of the second parallel side height

(float): The height (distance between parallel sides)

Returns:

float: The area of the trapezoid

"""

if base1 < 0 or *base2* < 0 or *height* < 0: *raise*

ValueError("Bases and height cannot be negative") *return*

0.5 * (*base1* + *base2*) * *height*

def main():

"""

Main function to demonstrate area calculations for different shapes. """ print("=" * 60)

print("AREA CALCULATOR FOR DIFFERENT SHAPES")

print("=" * 60)

print()

 # Circle example print("1. CIRCLE")

print("-" * 60) radius = 5.0 circle_area =

calculate_circle_area(radius) print(f"

Radius: {radius} units") print(f" Formula:

Area = pi * r^2")

```
    print(f"   Calculation: pi * {radius}^2 = {math.pi:.2f} * {radius**2} = {circle_area:.2f} square
units")     print(f"   Result: {circle_area:.2f} square units")
    print()

    # Rectangle example     print("2. RECTANGLE")     print("-" * 60)     length
= 8.0     width = 6.0     rectangle_area = calculate_rectangle_area(length,
width)     print(f"   Length: {length} units, Width: {width} units")     print(f"
Formula: Area = length * width")     print(f"   Calculation: {length} * {width}
= {rectangle_area} square units")     print(f"   Result: {rectangle_area}
square units")
    print()

    # Triangle example
    print("3. TRIANGLE")
    print("-" * 60) base = 10.0 height = 7.0 triangle_area =
calculate_triangle_area(base, height)     print(f"   Base: {base} units, Height:
{height} units")     print(f"   Formula: Area = (1/2) * base * height")     print(f"
Calculation: 0.5 * {base} * {height} = {triangle_area} square units")     print(f"
Result: {triangle_area} square units")     print()

    # Square example
print("4. SQUARE")
    print("-" * 60)
side = 4.0
    square_area = calculate_square_area(side)     print(f"   Side:
{side} units")     print(f"   Formula: Area = side^2")     print(f"
Calculation: {side}^2 = {square_area} square units")     print(f"
Result: {square_area} square units")
    print()
```

```
    # Trapezoid example
print("5. TRAPEZOID")
    print("-" * 60)     base1 = 6.0     base2 = 10.0     height_trap = 5.0     trapezoid_area
= calculate_trapezoid_area(base1, base2, height_trap) print(f"   Base 1: {base1}
units, Base 2: {base2} units, Height: {height_trap} units") print(f"   Formula: Area =
(1/2) * (base1 + base2) * height")

    print(f"   Calculation: 0.5 * ({base1} + {base2}) * {height_trap} = {trapezoid_area} square units")
    print(f"   Result: {trapezoid_area} square units") print()


    print("=" * 60)


if __name__ == "__main__":
    main()
```

Output:

```
(.venv) PS D:\3-2\AI Assitant Coding> & "D:\3-2\AI Assitant Coding\.venv\Scripts\python.exe" "d:/3-2/AI Assitant Coding/area_calculator.py
"
============================================================
AREA CALCULATOR FOR DIFFERENT SHAPES
============================================================

1. CIRCLE
------------------------------------------------------------
   Radius: 5.0 units
   Formula: Area = pi * r^2
   Calculation: pi * 5.0^2 = 3.14 * 25.0 = 78.54 square units
   Result: 78.54 square units

2. RECTANGLE
------------------------------------------------------------
   Length: 8.0 units, Width: 6.0 units
   Formula: Area = length * width
   Calculation: 8.0 * 6.0 = 48.0 square units
   Result: 48.0 square units

3. TRIANGLE
------------------------------------------------------------
   Base: 10.0 units, Height: 7.0 units
   Formula: Area = (1/2) * base * height
   Calculation: 0.5 * 10.0 * 7.0 = 35.0 square units
   Result: 35.0 square units

4. SQUARE
------------------------------------------------------------
   Side: 4.0 units
   Formula: Area = side^2
   Calculation: 4.0^2 = 16.0 square units
   Result: 16.0 square units
```

Task 3: Prompt Sensitivity Experiment

✞ **Scenario:**
You are testing how AI responds to different prompts.

✞ **Task:**

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

✞ **Expected Output: ❍ Prompt list**

❍ Code variations

Code Variations:

Below is a clear explanation of the code variations you observed, written in a student-friendly, exam-oriented way. This directly justifies the comparison between Gemini and Cursor AI.

---

1. Code Style Variation

Gemini

- **Uses single functions with conditional logic (if-elif) to handle multiple cases.**
- **Focuses on compact code and minimal structure.**
- **Easy to read for beginners because everything is in one place.**

Example concept:

```
def calculate_area(shape, **kwargs):

if shape == "circle":

    return math.pi * kwargs["radius"] ** 2
```

✓ Simple
✓ Beginner-friendly
✗ Less modular

Cursor AI

- **Uses separate functions for each task.**
- **Follows modular programming principles.**
- **Each function has a single responsibility.**

Example concept:

```
def calculate_circle_area(radius):

return math.pi * radius ** 2
```

✓ Clean design
✓ Scalable
✓ Industry standard

---

2. Error Handling Difference

Gemini

- Handles errors by returning error messages as strings.
- Does not stop program execution.

return "Error: Radius is required"

✓ Easy to understand
✗ Not ideal for real applications

Cursor AI

- Uses exceptions (ValueError) for invalid inputs.
- This is a professional Python practice. if radius < 0: raise

ValueError("Radius cannot be negative")

✓ Robust
✓ Safer
✓ Preferred in production

---

3. Code Output & Demonstration

Gemini

- Provides basic example usage.
- Prints direct results.
- Focuses on correctness over presentation.

✓ Quick testing
✓ Less explanation

Cursor AI

- Provides step-by-step demonstration.
- Prints formulas, calculations, and results.
- Includes a structured main() function.

✓ Better for teaching
✓ Better readability
✓ Real-world style

---

4. Extensibility (Adding New Features)

Gemini

- Adding a new shape means modifying the same function.
- Can become complex as features increase.

✗ Harder to scale

Cursor AI

- Adding a new shape = add a new function.
- Existing code remains untouched.

✓ Easy to extend
✓ Follows Open–Closed Principle

---

5. Overall Comparison Summary

Aspect	Gemini	Cursor AI
Ease of use	★★★★★★	
Beginner friendly	☑ Yes	⚠ Moderate
Code structure	Basic	Professional
Error handling	Simple strings	Exceptions
Scalability	Low	High
Industry readiness	Low	High

Final Explanation (Ready to Write in Exam / Assignment)

Gemini generates concise and beginner-friendly code that is easy to understand and suitable for small tasks and learning purposes. Cursor AI, however, produces more structured, modular, and robust code with proper error handling and extensibility, making it more suitable for real-world and production-level applications.

If you want, I can also:

- Convert this into 5–6 exam points
- Rewrite it as a reflection paragraph
- Shorten it for internal assessment answers

Task 4: Tool Comparison Reflection

✞ Scenario:

You must recommend an AI coding tool.

✞

Task:

Based on your work in this topic, compare Gemini, Copilot, and Cursor AI for usability and code quality.

✞ **Expected Output:**

Short written reflection

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.

Code:

```
def print_tool_comparison_reflection():
    reflection = """
Tool Comparison Reflection: Gemini vs Cursor AI


Based on the given outputs, Gemini is more beginner-friendly in terms of usability.
Its code is concise, readable, and includes clear docstrings and simple examples,
making it suitable for quick learning and straightforward tasks. However, its error
handling is basic and the solutions are limited to the exact problem scope.


On the other hand, Cursor AI produces more structured and production-level code. It
separates logic into well-defined functions, includes proper validation using
exceptions, and supports extensibility by handling more cases and shapes. Although
the code is more verbose, the overall code quality is higher and better suited for
real-world applications.


Recommendation:
Gemini is ideal for beginners and rapid understanding, while Cursor AI is better for
writing robust, maintainable, and scalable code.
"""     print(reflection) # Call the
function
print_tool_comparison_reflection(
) output:
```

```
Tool Comparison Reflection: Gemini vs Cursor AI

Based on the given outputs, Gemini is more beginner-friendly in terms of usability.
Its code is concise, readable, and includes clear docstrings and simple examples,
making it suitable for quick learning and straightforward tasks. However, its
error handling is basic and the solutions are limited to the exact problem scope.

On the other hand, Cursor AI produces more structured and production-level code.
It separates logic into well-defined functions, includes proper validation using
exceptions, and supports extensibility by handling more cases and shapes. Although
the code is more verbose, the overall code quality is higher and better suited for
real-world applications.

Recommendation:
Gemini is ideal for beginners and rapid understanding, while Cursor AI is better
for writing robust, maintainable, and scalable code.
```