

# AI ASSISTANT CODING

## LAB-02

Name : CH.RAMCHARAN

Roll No : 2303A52095

Batch No : 33

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

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

Scenario: You are improving legacy code.

The given legacy program calculates the sum of odd and even numbers in a list. The code needs to be improved for better readability and efficiency using AI tools.

#### **Prompt Used :**

Write a program to calculate the sum of odd and even numbers in a list,  
Refactor this Python code to improve readability and efficiency.

#### **Original Code :**

```
def sum_odd_even(numbers: list[int]) -> tuple[int, int]:  
    even_sum = 0  
    odd_sum = 0  
    for num in numbers:  
        if num % 2 == 0:  
            even_sum += num  
        else:  
            odd_sum += num  
    return even_sum, odd_sum  
  
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
even_sum, odd_sum = sum_odd_even(my_list)  
print(f"List: {my_list}")
```

```
print(f"Sum of even numbers: {even_sum}")  
print(f"Sum of odd numbers: {odd_sum}")
```

## AI-Refactored Code :

```
def sum_odd_even(numbers: list[int]) -> tuple[int, int]:
```

```
    """
```

Calculates the sum of even and odd numbers in a list.

Args:

numbers: A list of integers.

Returns:

A tuple containing (even\_sum, odd\_sum).

```
    """
```

```
even_sum = sum(num for num in numbers if num % 2 == 0)
```

```
odd_sum = sum(num for num in numbers if num % 2 != 0)
```

```
return even_sum, odd_sum
```

# 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"List: {my_list}")
```

```
print(f"Sum of even numbers: {even_sum}")
```

```
print(f"Sum of odd numbers: {odd_sum}")
```

The screenshot shows a Google Colab notebook titled "LAB\_(2)\_2095.ipynb". In the code editor, there is a cell containing Python code for calculating the sum of odd and even numbers in a list. The code uses two separate loops to calculate the sums of odd and even numbers. The Gemini sidebar on the right provides a response to a user query about writing a program to calculate the sum of odd and even numbers.

```
def sum_odd_even(numbers: list[int]) -> tuple[int, int]:  
    """  
    Calculates the sum of even and odd numbers in a list.  
  
    Args:  
        numbers: A list of integers.  
  
    Returns:  
        A tuple containing (even_sum, odd_sum).  
    """  
    even_sum = sum(num for num in numbers if num % 2 == 0)  
    odd_sum = sum(num for num in numbers if num % 2 != 0)  
    return even_sum, odd_sum  
  
# 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"List: {my_list}")  
print(f"Sum of even numbers: {even_sum}")  
print(f"Sum of odd numbers: {odd_sum}")
```

Gemini response:

- R Write a program to calculate the sum of odd and even numbers in a list,
- ◆ I can help with that! Here's a Python program that defines a function to calculate the sum of even and odd numbers in a list, along with an example of how to use it:

The code executed successfully and calculated the sum of even and odd numbers in the list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

  - The sum of even numbers is 30.
  - The sum of odd numbers is 25.

The screenshot shows the same Google Colab notebook after refactoring. The code now uses Python's built-in `sum()` function with a generator expression to calculate the sums of odd and even numbers in a single loop. The Gemini sidebar provides feedback on the refactoring.

```
def sum_odd_even(numbers: list[int]) -> tuple[int, int]:  
    """  
    Calculates the sum of even and odd numbers in a list.  
  
    Args:  
        numbers: A list of integers.  
  
    Returns:  
        A tuple containing (even_sum, odd_sum).  
    """  
    even_sum = sum(num for num in numbers if num % 2 == 0)  
    odd_sum = sum(num for num in numbers if num % 2 != 0)  
    return even_sum, odd_sum  
  
# 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"List: {my_list}")  
print(f"Sum of even numbers: {even_sum}")  
print(f"Sum of odd numbers: {odd_sum}")
```

Gemini response:

- R Write a program to calculate the sum of odd and even numbers in a list, Refactor this Python code to improve readability and efficiency.
- ◆ Certainly! I've refactored the `sum_odd_even` function to iterate through the list only once, which improves efficiency for larger lists while maintaining readability.

The refactored code for `sum_odd_even` has executed successfully. As you can see, the list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] has its even numbers summing to 30 and odd numbers summing to 25. This refactoring improved efficiency by using a single loop for the calculation.

## Explanation :

The refactored code reduces the number of variables and uses Python's built-in `sum()` function. It improves readability, reduces code length, and enhances maintainability. The logic is clearer and easier to modify.

## Task 2: Area Calculation Explanation (Gemini)

### Scenario

This task helps onboard a junior developer by explaining a function that calculates areas of different shapes.

### Prompt Used :

Write a Python function to calculate area of circle, rectangle, and triangle and explain the code

```
import math
```

```
def calculate_area(shape: str, **kwargs) -> float:  
    """  
    Calculates the area of different geometric shapes.  
    """
```

Args:

shape (str): The type of shape ('circle', 'rectangle', 'triangle').

\*\*kwargs: Keyword arguments for the dimensions of the shape:

- For 'circle': 'radius' (float)
- For 'rectangle': 'length' (float), 'width' (float)
- For 'triangle': 'base' (float), 'height' (float)

Returns:

float: The calculated area of the shape.

Raises:

ValueError: If an unsupported shape is provided or required dimensions are missing.

```
"""
```

```
if shape.lower() == 'circle':  
    if 'radius' in kwargs:  
        radius = kwargs['radius']  
        return math.pi * radius**2  
    else:  
        raise ValueError("For 'circle', 'radius' is required.")
```

```

elif shape.lower() == 'rectangle':
    if 'length' in kwargs and 'width' in kwargs:
        length = kwargs['length']
        width = kwargs['width']
        return length * width
    else:
        raise ValueError("For 'rectangle', 'length' and 'width' are required.")
elif shape.lower() == 'triangle':
    if 'base' in kwargs and 'height' in kwargs:
        base = kwargs['base']
        height = kwargs['height']
        return 0.5 * base * height
    else:
        raise ValueError("For 'triangle', 'base' and 'height' are required.")
    else:
        raise ValueError("Unsupported shape. Choose from 'circle', 'rectangle', or 'triangle'.")

# Example usage:
print(f"Area of circle (radius=5): {calculate_area('circle', radius=5):.2f}")
print(f"Area of rectangle (length=4, width=6): {calculate_area('rectangle', length=4, width=6):.2f}")
print(f"Area of triangle (base=7, height=3): {calculate_area('triangle', base=7, height=3):.2f}")

# Example of handling an error (uncomment to test):
# try:
#     print(calculate_area('square', side=5))
# except ValueError as e:
#     print(f"Error: {e}")

```

The screenshot shows a Google Colab interface with a dark theme. On the left, there's a sidebar with icons for file operations like Open, Save, and Share. The main area has a code cell containing Python code for calculating the area of geometric shapes. To the right of the code cell is a Gemini AI panel. The Gemini panel has a title 'Gemini' and a section 'explain the code'. It contains a text block explaining the purpose of the code and its results. Below this, there are three bullet points: 'Area of circle (radius=5): 78.54', 'Area of rectangle (length=4, width=6): 24.00', and 'Area of triangle (base=7, height=3): 10.50'. At the bottom of the Gemini panel, there are several buttons: 'Add support for calculating the area of a square', 'Test the error handling for unsupported shapes', 'Explain the sum\_odd\_even function in the first cell', and a 'What can I help you build?' input field.

```
import math

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

    Args:
        shape (str): The type of shape ('circle', 'rectangle', 'triangle').

    **kwargs: Keyword arguments for the dimensions of the shape:
        - For 'circle': 'radius' (float)
        - For 'rectangle': 'length' (float), 'width' (float)
        - For 'triangle': 'base' (float), 'height' (float)

    Returns:
        float: The calculated area of the shape.

    Raises:
        ValueError: If an unsupported shape is provided or required dimensions are missing.
    """
    if shape.lower() == 'circle':
        if 'radius' in kwargs:
            radius = kwargs['radius']
            return math.pi * radius**2
        else:
            raise ValueError("For 'circle', 'radius' is required.")
    elif shape.lower() == 'rectangle':
        if 'length' in kwargs and 'width' in kwargs:
            length = kwargs['length']
            width = kwargs['width']
            return length * width
        else:
            raise ValueError("For 'rectangle', 'length' and 'width' are required.")
    elif shape.lower() == 'triangle':
        if 'base' in kwargs and 'height' in kwargs:
            base = kwargs['base']
            height = kwargs['height']
            return 0.5 * base * height
        else:
            raise ValueError("For 'triangle', 'base' and 'height' are required.")

    else:
        raise ValueError("Unsupported shape. Choose from 'circle', 'rectangle', or 'triangle'.")
```

This screenshot shows the same Google Colab interface as the previous one, but with more code added to the cell. The new code includes examples of how to use the function and handle errors. The Gemini AI panel remains the same, providing the same explanation and results for the added code. The desktop taskbar at the bottom shows various open applications like a browser, file explorer, and system tray.

```
# Example usage:
print("Area of circle (radius=5): (calculate_area('circle', radius=5):.2f)")
print("Area of rectangle (length=4, width=6): (calculate_area('rectangle', length=4, width=6):.2f)")
print("Area of triangle (base=7, height=3): (calculate_area('triangle', base=7, height=3):.2f)")

# Example of handling an error (uncomment to test):
# try:
#     print(calculate_area('square', side=5))
# except ValueError as e:
#     print(f"Error: {e}")

... Area of circle (radius=5): 78.54
Area of rectangle (length=4, width=6): 24.00
Area of triangle (base=7, height=3): 10.50
```

The screenshot shows a Google Colab interface with two main panes. The left pane displays a Python function `calculate_area` and its documentation. The right pane shows an AI interface named "Gemini" responding to prompts about the function's purpose and results.

```

 1. import math: Imports the math module, which is necessary for accessing math.pi (the value of Pi) for circle area calculation.
 2. def calculate_area(shape: str, **kwargs) -> float:
    This defines the function calculate_area.
    ○ shape: str : This parameter takes a string indicating the type of shape (e.g., 'circle', 'rectangle', 'triangle').
    ○ **kwargs : This allows the function to accept an arbitrary number of keyword arguments. This is used to pass the specific dimensions required for each shape (e.g., radius=5, length=4, width=6).
    ○ -> float : This is a type hint indicating that the function is expected to return a floating-point number (the area).

 3. Docstring ( """...""" ):
    ○ Explains what the function does, its arguments, what it returns, and potential errors it might raise. This is good practice for code documentation.

 4. Conditional Logic ( if-elif-else ):
    ○ The function uses if-elif-else statements to determine which shape's area needs to be calculated.
    ○ 'circle': If shape is 'circle', it checks for the radius keyword argument. If found, it calculates the area using the formula pi * radius**2. If radius is missing, it raises a ValueError.
    ○ 'rectangle': If shape is rectangle, it checks for both length and width. If both are present, it calculates the area as length * width. Otherwise, it raises a ValueError.
    ○ 'triangle': If shape is 'triangle', it checks for both base and height. If both are present, it calculates the area as 0.5 * base * height. Otherwise, it raises a ValueError.
    ○ Unsupported Shape: If the shape provided does not match 'circle', 'rectangle', or 'triangle', a ValueError is raised.

 5. Example Usage:
    ○ The code demonstrates how to call the calculate_area function for each supported shape, passing the required dimensions as keyword arguments.
    ○ {:.2f} is used in the f-string to format the output area to two decimal places, making it cleaner to read.
    ○ An example of error handling is commented out, showing how the ValueError would be caught if an unsupported shape or missing dimensions were provided.

```

**Gemini**

- R Write a Python function to calculate area of circle, rectangle, and triangle and explain the code
 

Certainly! Here's a Python function to calculate the area of circles, rectangles, and triangles, along with an explanation of the code.
- The code to calculate the area of various shapes has executed successfully. Here are the results:
  - Area of circle (radius=5): 78.54
  - Area of rectangle (length=4, width=6): 24.00
  - Area of triangle (base=7, height=3): 10.50

## Explanation :

The function accepts the shape name and required dimensions as input. Conditional statements decide which formula to apply. For a circle, radius is used, while rectangle and triangle require two values. The function returns the calculated area based on the selected shape.

## Task 3: Prompt Sensitivity Experiment (Cursor AI)

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

The factorial problem was used to test how different prompts affect AI-generated code.

### Prompts Used :

1. Write a Python program to calculate factorial
2. Write an optimized Python function to calculate factorial with comments
3. Generate a beginner-friendly factorial program with error handling

The screenshot shows a Microsoft Visual Studio Code interface. The title bar says "E:\ E:\ Evenodd.py • Extension: Python". The left sidebar has "OUTLINE" and "TIMELINE" buttons. The main editor area contains the following Python code:

```
#1. Write a Python program to calculate factorial of a number
def factorial():
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
print(factorial(5))
#test case 1:
print(factorial(5))
#Output:
120
#2. Write an optimized Python function to calculate factorial with comments and test cases
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
print(factorial(5))
#test case 1:
print(factorial(5))
#Output:
120
#3. Generate a beginner-friendly factorial program with error handling
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
print(factorial(5))
#test case 1:
print(factorial(5))
#Output:
120
```

The status bar at the bottom shows "Cursor Tab" with a search icon, "Ln 43, Col 1", "Spaces: 4", "UTF-8", "CRLF", "Python 3.13.9 64-bit (Microsoft Store)", and a "Copilot" icon.

## Observation

The AI produced different code styles depending on the prompt. Optimized prompts improved performance and readability, while beginner-friendly prompts added validation and safety checks.

## Task 4: Tool Comparison Reflection

Comparison of Gemini, Copilot, and Cursor AI

### Reflection :

Google Gemini is best suited for explanations and learning support. GitHub Copilot provides real-time inline suggestions, improving developer productivity. Cursor AI excels in experimentation, refactoring, and prompt-based exploration. Each tool serves a different purpose, and choosing the right one depends on whether the goal is learning, development speed, or code analysis.