

AI ASSISTANT CODING

LAB-02

Name : V . Shiva Harsha

Roll No : 2303A52375

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.ipynb". In the code cell, there is a Python function named `sum_odd_even` which calculates the sum of odd and even numbers in a list. The function uses two variables, `even_sum` and `odd_sum`, to keep track of the sums of even and odd numbers respectively. The function iterates through each number in the input list and adds it to the appropriate sum based on its parity. The code then prints the lists and their respective sums.

```
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("List: ", my_list)
print("Sum of even numbers: ", even_sum)
print("Sum of odd numbers: ", odd_sum)
```

The output of the code cell shows the list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and the sums 30 and 25 respectively. To the right of the code cell is a Gemini AI interface window. The user has asked Gemini to write a program to calculate the sum of odd and even numbers in a list. Gemini has responded with the same Python code, indicating that it executed successfully and calculated the correct sums.

The screenshot shows the same Google Colab notebook "LAB 2.ipynb". The code cell now contains a refactored version of the `sum_odd_even` function. The function includes a detailed docstring that describes its purpose, arguments, and return value. It also includes a section for example usage. The logic remains the same, using generator expressions to sum the even and odd numbers directly from the list.

```
[S] 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("List: ", my_list)
print("Sum of even numbers: ", even_sum)
print("Sum of odd numbers: ", odd_sum)
```

The output of the code cell is identical to the previous one, showing the list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and the sums 30 and 25. To the right of the code cell is a Gemini AI interface window. The user has asked Gemini to refactor the code to improve readability and efficiency. Gemini has responded with the refactored code, indicating that it executed successfully and calculated the correct sums.

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}")

```

LAB 2.ipynb - Colab

```

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'.")
```

Gemini

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 a circle, rectangle, and triangle, along with an explanation:

The `calculate_area` function in cell 59128c0 executed successfully, providing the following results:

- The area of a circle with a radius of 5 is 78.54.
- The area of a rectangle with a length of 4 and width of 6 is 24.00.
- The area of a triangle with a base of 7 and height of 3 is 10.50.

LAB 2.ipynb - Colab

```

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("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

Code Explanation:

Gemini

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 a circle, rectangle, and triangle, along with an explanation:

The `calculate_area` function in cell 59128c0 executed successfully, providing the following results:

- The area of a circle with a radius of 5 is 78.54.
- The area of a rectangle with a length of 4 and width of 6 is 24.00.
- The area of a triangle with a base of 7 and height of 3 is 10.50.

The screenshot shows a Google Colab notebook titled "LAB 2.ipynb - Colab". The main code cell contains a function definition:

```

def calculate_area(shape: str, **kwargs) -> float:
    ...

```

The "Code Explanation" sidebar provides a detailed breakdown of the code, including:

- Import:** Imports the `math` module.
- Function Definition:** `calculate_area` takes a string `shape` and keyword arguments `**kwargs`.
- Docstring:** Describes the function's purpose and parameters.
- Conditional Logic:** Checks if `shape` is 'circle' (calculates area from radius), 'rectangle' (calculates area from length and width), or 'triangle' (calculates area from base and height).
- Example Usage:** Shows how to call the function for different shapes.

The "Gemini" sidebar shows generated explanations and examples for the `calculate_area` function.

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

```
#1. Write a Python program to calculate factorial of a number
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

#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
```

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.