

AI ASSISTANT CODING

LAB-02

Name : R. Anirudh

Roll No : 2303A52103

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}")
```

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

```
[S] 0s
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}")

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

Gemini

- Write a program to calculate the sum of odd and even numbers in a list.
- I've already provided a Python program in your notebook (cell PAR-2Z7MVZ6) that calculates the sum of odd and even numbers in a list.

Here's how it works:

- Initialization: `even_sum` and `odd_sum` are initialized to 0.
- Iteration: It iterates through each `num` in the `numbers` list.
- Conditional Summation:
 - If `num % 2 == 0`, the number is even, so it's added to `even_sum`.

Explain the 'sum_odd_even' function in more detail
Refactor 'sum_odd_even' using a list comprehension
Add error handling for non-integer inputs to the function

What can I help you build?

Gemini 2.5 Flash ▶

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

Variables Terminal

10:51 Python 3

Untitled17.ipynb

```
[E] 0s
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}")

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

Gemini

5, 6, 7, 8, 9, 10], calls `sum_odd_even(my_list)`, and prints the results.

Output from the executed cell: List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] Sum of even numbers: 30 Sum of odd numbers: 25

Refactor this Python code to improve readability and efficiency.

I'll refactor the `sum_odd_even` function to use a more Pythonic and efficient approach with list comprehensions for summing odd and even numbers. This will make the code more concise and readable.

Show me the refactored code Explain the new refactored code Test the refactored code with a new list

What can I help you build?

Gemini 2.5 Flash ▶

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

Variables Terminal

10:54 Python 3

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}")

```

Untitled17.ipynb - Colab

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text Run all

```

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}")

... 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

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, or triangle, along with an explanation of how it works:

The `calculate_area` function in the executed cell has produced the following results:

- Area of a circle with radius 5: 78.54
- Area of a rectangle with length 4 and width 6: 24.00
- Area of a triangle with base 7 and height 3: 10.50

These results demonstrate the function correctly calculating the areas for each specified shape using the provided dimensions.

+ Test the error handling for an unsupported shape
+ Add functionality to calculate the area of a square
+ Explain how to use this function with a list of shapes

What can I help you build?

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

✓ 10:58 Python 3

The screenshot shows a Google Colab notebook titled "Untitled17.ipynb". On the left, there's a "Code Explanation" panel that provides a detailed breakdown of a Python function named `calculate_area`. This function calculates the area of a circle, rectangle, or triangle based on the provided shape and dimensions. The explanation covers imports, function definition, conditional logic for different shapes, and example usage. On the right, there's a "Gemini" AI interface that generates a Python function to calculate the area of a circle, rectangle, and triangle. It also provides three specific results: the area of a circle with radius 5 (78.54), the area of a rectangle with length 4 and width 6 (24.00), and the area of a triangle with base 7 and height 3 (10.50). The AI interface includes a sidebar with various prompts like "Test the error handling for an unsupported shape" and "Add functionality to calculate the area of a square".

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 the Microsoft Visual Studio Code interface. The left sidebar has 'NO FOLDER OPENED' and 'Clone Repository' buttons. The main editor area contains three blocks of Python code, each with a different prompt:

```
#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
```

The status bar at the bottom shows 'Cursor Tab' and 'Ln 43, Col 1'. The bottom right corner indicates 'Python 3.13.0 64-bit (Microsoft Store)'. The code editor has a dark theme.

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.