

# 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)**

```
[5] 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-ZZ7MMVZ6) that calculates the sum of odd and even numbers in a list. Here's how it works:

- sum\_odd\_even(numbers: list[int]) -> tuple[int, int]:** This function takes a list of integers as input and returns a tuple containing the sum of even numbers and the sum of odd numbers.
- 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

**Variables** **Terminal**

colab.research.google.com/drive/1oACUp0JQk1inhXS-RB81kyQbOEMMdQak7hl=en-GB#scrollTo=PAR-zZ7MMVZ6

---

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

```
[5] 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

**Variables** **Terminal**

colab.research.google.com/drive/1oACUp0JQk1inhXS-RB81kyQbOEMMdQak7hl=en-GB#scrollTo=TCW418QUW5zb

## 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

colab.research.google.com/drive/1oACUp0JQk1nhXS-RB81kyQbOEMMdQak?hl=en-GB#scrollTo=00c1ce0d

Untitled17.ipynb

File Edit View Insert Runtime Tools Help

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 2.5 Flash

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

Variables Terminal

10:58 Python 3

Untitled17.ipynb - Colab

colab.research.google.com/drive/1oACUp0JQk1nhXS-RB81kyQbOEMMdQak?hl=en-GB#scrollTo=00c1ce0d

Untitled17.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

def calculate\_area(shape: str, \*\*kwargs):

"""

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

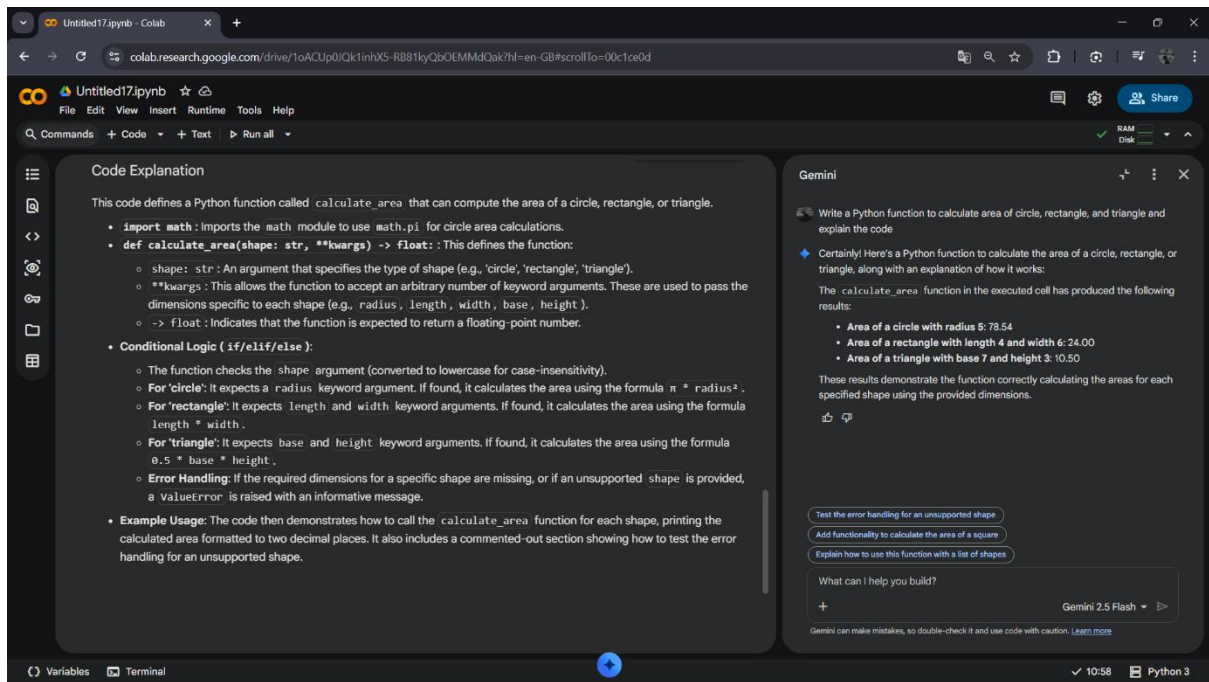
What can I help you build?

Gemini 2.5 Flash

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

Variables Terminal

10:58 Python 3



## 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 Cursor AI code editor interface. On the left, there's a sidebar with options like 'Open Folder', 'Clone Repository', and 'To learn more about how to use Git and source control in VS Code read our docs.' The main editor area displays three different Python code snippets for calculating a factorial, each generated by AI based on a different prompt. The first snippet is a simple recursive function. The second snippet is an optimized version with comments and test cases. The third snippet is a beginner-friendly version with error handling. The status bar at the bottom indicates the current file is 'Evenodd.py' and the cursor is at line 43, column 1.

```
1 #1. Write a Python program to calculate factorial of a number
2 def factorial(n):
3     if n == 0:
4         return 1
5     else:
6         return n * factorial(n-1)
7 print(factorial(5))
8 #test case 1:-
9 print(factorial(5))
10 #Output:
11 120
12 #test case 2:-
13 print(factorial(0))
14 #Output:
15 1
16
17
18
19 #2. Write an optimized Python function to calculate factorial with comments and test cases
20 def factorial(n):
21     if n == 0:
22         return 1
23     else:
24         return n * factorial(n-1)
25 print(factorial(5))
26 #test case 1:-
27 print(factorial(5))
28 #Output:
29 120
30
31
32 #3. Generate a beginner-friendly factorial program with error handling
33 def factorial(n):
34     if n == 0:
35         return 1
36     else:
37         return n * factorial(n-1)
38 print(factorial(5))
39 #test case 1:-
40 print(factorial(5))
41 #Output:
42 120
43 # Click to Chat, Click to generate.
```

## 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.