

AI ASSISTANT CODING

LAB-02

Name : D.Manideep

Roll No : 2303A5202127

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 the VS Code interface with the following details:

- File:** lab-1.py
- Code Content:**

```
D: > sru > 3> aiac > lab-1.py > ...
1 def sum_odd_even(numbers: list[int]) -> tuple[int, int]:
2     even_sum = 0
3     odd_sum = 0
4     for num in numbers:
5         if num % 2 == 0:
6             even_sum += num
7         else:
8             odd_sum += num
9     return even_sum, odd_sum
10 my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
11 even_sum, odd_sum = sum_odd_even(my_list)
12 print(f"List: {my_list}")
13 print(f"Sum of even numbers: {even_sum}")
14 print(f"Sum of odd numbers: {odd_sum}")
15
```
- Terminal Output:**

```
-2/aiac/lab-1.py
List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of even numbers: 30
Sum of odd numbers: 25
PS C:\Users\De...>
```
- Bottom Status Bar:** Ln 15, Col 1 | Spaces: 4 | UTF-8 | CRLF | Python | 3.11.9 (Microsoft Store)

The screenshot shows the VS Code interface with the following details:

- File:** lab-1.py
- Code Content:**

```
D: > sru > 3> aiac > lab-1.py > ...
16 def sum_odd_even(numbers: list[int]) -> tuple[int, int]:
17     """
18         Calculates the sum of even and odd numbers in a list.
19
20     Args:
21         numbers: A list of integers.
22
23     Returns:
24         A tuple containing (even_sum, odd_sum).
25     """
26     even_sum = sum(num for num in numbers if num % 2 == 0)
27     odd_sum = sum(num for num in numbers if num % 2 != 0)
28     return even_sum, odd_sum
29
30     # Example usage:
31 my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
32 even_sum, odd_sum = sum_odd_even(my_list)
33
34 print(f"List: {my_list}")
```
- Terminal Output:**

```
Sum of odd numbers: 25
PS C:\Users\De...> & C:/Users/De.../Local/Microsoft/WindowsApps/python3.11.exe d:/sr.../3
-2/aiac/lab-1.py
List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of even numbers: 30
Sum of odd numbers: 25
```
- Bottom Status Bar:** + ... | powershell | Python | 3.11.9 (Microsoft Store)

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}")
```

```

File Edit Selection View ... ← → ⌂ Search
lab-1.py 2
D:> sru > 3-2 > aiac > lab-1.py > calculate_area
39 import math
40
41 def calculate_area(shape: str, **kwargs) -> float:
42     """
43         Calculates the area of different geometric shapes.
44
45     Args:
46         shape (str): The type of shape ('circle', 'rectangle', 'triangle').
47         **kwargs: Keyword arguments for the dimensions of the shape:
48             - For 'circle': 'radius' (float)
49             - For 'rectangle': 'length' (float), 'width' (float)
50             - For 'triangle': 'base' (float), 'height' (float)
51
52     Returns:
53         float: The calculated area of the shape.
54
55     Raises:
56         ValueError: If an unsupported shape is provided or required dimensions are missing.
57     """
58     if shape.lower() == 'circle':
59         if 'radius' in kwargs:
60             radius = kwargs['radius']
61             return math.pi * radius**2
62         else:
63             raise ValueError("For 'circle', 'radius' is required.")
64     elif shape.lower() == 'rectangle':
65         if 'length' in kwargs and 'width' in kwargs:
66             length = kwargs['length']
67             width = kwargs['width']
68             return length * width
69         else:
70             raise ValueError("For 'rectangle', 'length' and 'width' are required.")
71     elif shape.lower() == 'triangle':
72         if 'base' in kwargs and 'height' in kwargs:
73             base = kwargs['base']
74             height = kwargs['height']
75             return 0.5 * base * height
76         else:
77             raise ValueError("For 'triangle', 'base' and 'height' are required.")
78     else:
79         raise ValueError("Unsupported shape. Choose from 'circle', 'rectangle', or 'triangle'.")
80
81 # Example usage:
82 print(f"Area of circle (radius=5): {calculate_area('circle', radius=5):.2f}")
83 print(f"Area of rectangle (length=4, width=6): {calculate_area('rectangle', length=4, width=6):.2f}")
84 print(f"Area of triangle (base=7, height=3): {calculate_area('triangle', base=7, height=3):.2f}")
85
86 # Example of handling an error (uncomment to test):
87 # try:
88 #     print(calculate_area('square', side=5))
89 # except ValueError as e:
90 #     print(f"Error: {e}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\Del1> & C:/Users/Deli/AppData/Local/Microsoft/WindowsApps/python3.11.exe d:/sru/3-2/aiac/lab-1.py
File "d:/sru/3-2/aiac/lab-1.py", line 17
    """
IndentationError: unexpected indent
PS C:\Users\Del1> 78.25
PS C:\Users\Del1> & C:/Users/Deli/AppData/Local/Microsoft/WindowsApps/python3.11.exe d:/sru/3-2/aiac/lab-1.py
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

```

CHAT + ⌂ RECENT SESSIONS

- Updating GitHub Copilot instructions for c... Completed Local • 1 day
- Greeting and assistance inquiry Completed Local • 3 days

Show More

Build with Agent

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

+ ⌂ lab-1.py

Describe what to build next

Agent Auto ⌂

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 with three sections of Python code side-by-side, each representing a different prompt response:

```
E:\ > Eevenodd.py ->
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 print(factorial(5))
9 print(factorial(5))
10 #output:
11
12 #test case 1:
13 print(factorial(0))
14 #output:
15
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
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
43
```

The code is annotated with comments (#1, #2, #3) corresponding to the prompts listed above. The first section is a simple recursive function. The second section is more optimized with comments explaining the steps. The third section is a beginner-friendly version with validation for non-negative integers.

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.