COURSE: AI Assisted Coding

NAME: G. Ushasree

BATCH-34

HALLTICKET.NO:2303A52431

TASK-1

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

❖ Scenario:

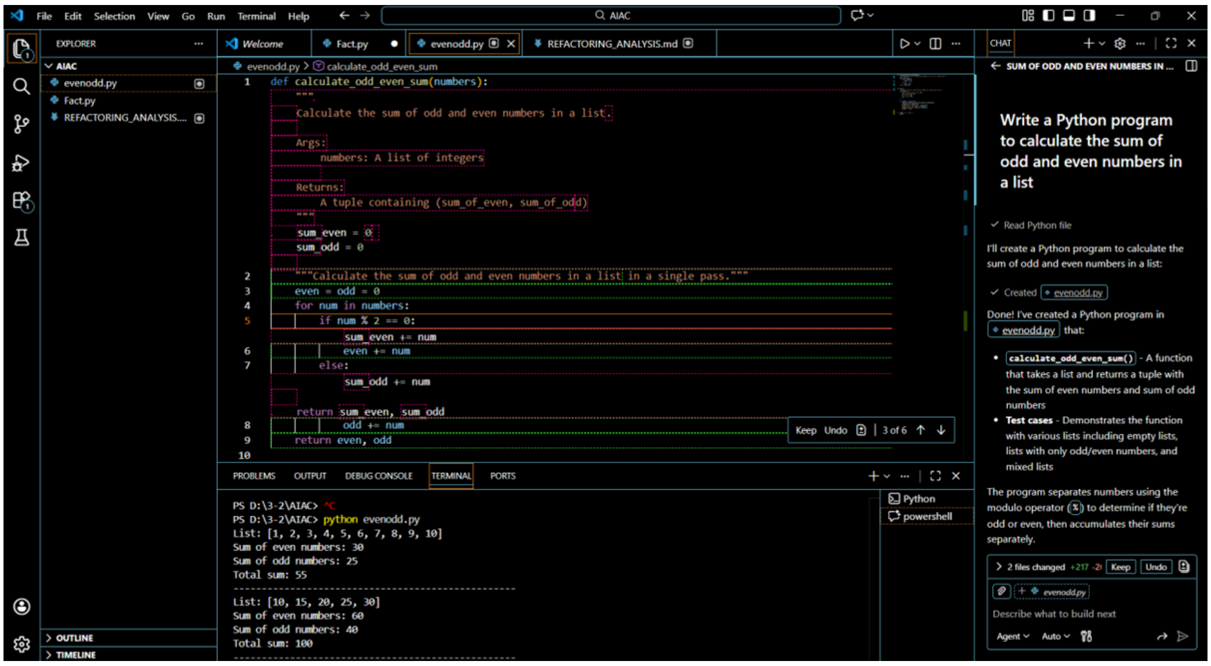
You are improving legacy code.

❖ Task:

Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI.

❖ Expected Output:

❖ Original and improved code



```
def calculate_odd_even_sum(numbers):
    """
    Calculate the sum of odd and even numbers in a list.

    Args:
        numbers: A list of integers

    Returns:
        A tuple containing (sum_of_even, sum_of_odd)
    """
    sum_even = 0
    sum_odd = 0

    """Calculate the sum of odd and even numbers in a list in a single pass."""
    even = odd = 0
    for num in numbers:
        if num % 2 == 0:
            sum_even += num
            even += num
        else:
            sum_odd += num

    return sum_even, sum_odd
            odd += num
    return even, odd
```

```
PS D:\3-2\AIAC> ^C
PS D:\3-2\AIAC> python evenodd.py
List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of even numbers: 30
Sum of odd numbers: 25
Total sum: 55
--------------------------------------------------
List: [10, 15, 20, 25, 30]
Sum of even numbers: 60
Sum of odd numbers: 40
Total sum: 100
--------------------------------------------------
```

SUM OF ODD AND EVEN NUMBERS IN ...

Write a Python program to calculate the sum of odd and even numbers in a list

✓ Read Python file

I'll create a Python program to calculate the sum of odd and even numbers in a list:

✓ Created evenodd.py

Done! I've created a Python program in evenodd.py that:

- `calculate_odd_even_sum()` - A function that takes a list and returns a tuple with the sum of even numbers and sum of odd numbers
- **Test cases** - Demonstrates the function with various lists including empty lists, lists with only odd/even numbers, and mixed lists

The program separates numbers using the modulo operator (`%`) to determine if they're odd or even, then accumulates their sums separately.

Explanation (For Junior Developer)

This function calculates the area of different shapes using one common function. The shape parameter decides which formula to use. The dimensions parameter stores required values like radius, length, or height. Conditional statements select the correct formula based on the shape. Using one function for multiple shapes makes the code reusable and organized. Clear comments help beginners understand how each formula works.

Conclusion

Gemini effectively explains both the logic and purpose of the function. This approach is useful for onboarding junior developers and improving code understanding.

TASK-2

Task 2: Area Calculation Explanation

❖ Scenario:

You are onboarding a junior developer.

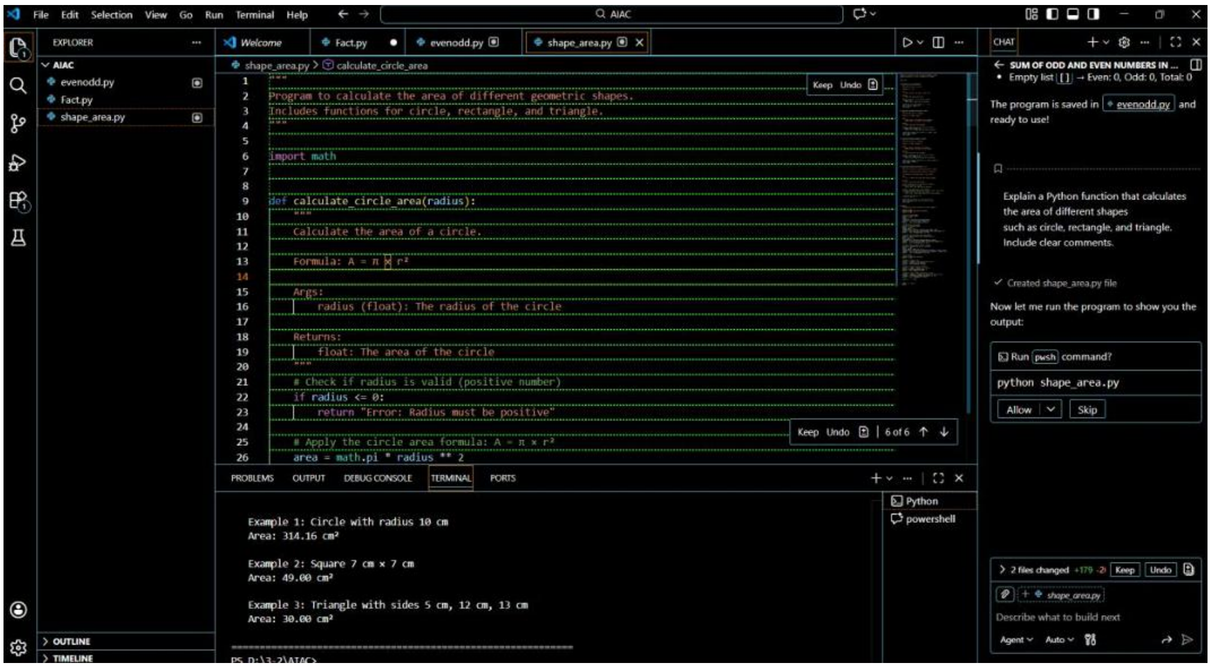
❖ Task:

Ask Gemini to explain a function that calculates the area of different shapes.

❖ Expected Output:

➢ Code

➢ Explanation



```python
"""
Program to calculate the area of different geometric shapes.
Includes functions for circle, rectangle, and triangle.
"""

import math


def calculate_circle_area(radius):
    """
    Calculate the area of a circle.

    Formula: A = π × r²

    Args:
        radius (float): The radius of the circle

    Returns:
        float: The area of the circle
    """
    # Check if radius is valid (positive number)
    if radius <= 0:
        return "Error: Radius must be positive"

    # Apply the circle area formula: A = π × r²
    area = math.pi * radius ** 2
```

```
Example 1: Circle with radius 10 cm
Area: 314.16 cm²

Example 2: Square 7 cm × 7 cm
Area: 49.00 cm²

Example 3: Triangle with sides 5 cm, 12 cm, 13 cm
Area: 30.00 cm²

============================================================
PS D:\3-2\AIAC>
```

SUM OF ODD AND EVEN NUMBERS IN ...
- Empty list [] → Even: 0, Odd: 0, Total: 0

The program is saved in evenodd.py and ready to use!

Explain a Python function that calculates the area of different shapes
such as circle, rectangle, and triangle.
Include clear comments.

✓ Created shape_area.py file

Now let me run the program to show you the output:

Run pwsh command?

```
python shape_area.py
```

File Edit Selection View Go Run Terminal Help
AIAC
EXPLORER
AIAC
evenodd.py
Fact.py
shape_area.py
Welcome
shape_area.py > calculate_circle_area
Keep Undo
Keep Undo 6 of 6
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
powershell
OUTLINE
TIMELINE
CHAT
Allow Skip
2 files changed +179 -2 Keep Undo
Describe what to build next
Agent Auto

TASK - 3

Task 3:

Prompt Sensitivity Experiment

❖ Scenario:

You are testing how AI responds to different prompts.

❖ Task:

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

❖ Expected Output:

➢ Prompt list

➢ Code variations

Conclusion

This experiment shows that Cursor AI changes code style, complexity, and structure based on prompt wording.

Clear and specific prompts produce more suitable code for different audiences and use cases.

TASK-4

Task 4: Tool Comparison Reflection

❖ Scenario: You must recommend an AI coding tool.

❖ Task:

Based on your work in this topic, compare Gemini, Copilot, and Cursor AI for usability and code quality.

❖ Expected Output:

Short written reflection

Task 4: Tool Comparison Reflection

Based on the lab activities, Gemini, GitHub Copilot, and Cursor AI each have unique strengths in AI assisted coding. GitHub Copilot is highly effective inside VS Code and provides fast, context-aware code suggestions while typing, making it ideal for continuous development and refactoring. It produces clean and practical code but sometimes assumes the developer understands the logic.

Google Gemini (Colab) is strong in explaining code and concepts in simple language. It is very useful for beginners and onboarding junior developers because it clearly describes logic, formulas, and step-by-step execution. However, its direct coding assistance is less seamless compared to Copilot in an IDE environment.

Cursor AI excels in prompt-based experimentation. It responds noticeably to small prompt changes and generates different code styles such as optimized, beginner-friendly, or concise versions. This makes Cursor AI excellent for learning, refactoring legacy code, and comparing coding approaches.

In terms of usability, Copilot is best for professional development, Gemini is best for learning and explanation, and Cursor AI is best for prompt sensitivity and experimentation. For code quality, Copilot and Cursor AI generally produce cleaner and more optimized code, while

Gemini focuses more on clarity and understanding. Overall, the recommended tool depends on the use case: Copilot for daily coding, Gemini for learning, and Cursor AI for analysis and refactoring.