Assignment-2.5

Roll.No:2303A51062
Batch-29

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

Code:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

even_sum = 0
odd_sum = 0

for num in numbers:
    if num % 2 == 0:
        even_sum = even_sum + num
    else:
        odd_sum = odd_sum + num

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

```
Sum of even numbers: 30
Sum of odd numbers: 25
```

Justification:

The original program correctly calculates the sum of odd and even numbers using a loop and conditional statements, but it is verbose and requires manual variable updates. This approach increases code length and reduces readability, especially as the program grows.

The refactored version improves the logic by using Python's built-in `sum()` function with generator expressions. This reduces the number of lines of code, eliminates explicit loops, and makes the intent of the program clearer. The improved code is easier to read, maintain, and less error-prone while producing the same output.

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

Output:

```
def calculate_area(shape, value1, value2=0):
    if shape == "circle":
        return 3.14 * value1 * value1      # value1 = radius

    elif shape == "rectangle":
        return value1 * value2             # value1 = length, value2 = width

    elif shape == "triangle":
        return 0.5 * value1 * value2       # value1 = base, value2 = height

    else:
        return "Invalid shape"


# Example usage
print(calculate_area("circle", 5))
print(calculate_area("rectangle", 4, 6))
print(calculate_area("triangle", 3, 8))
```

```
78.5
24
12.0
```

Justification

The function uses a single, well-structured logic to calculate the area of multiple shapes, which makes the code simple and reusable. By identifying the shape through a parameter, it avoids writing separate functions for each shape and reduces code duplication.

The conditional structure ensures that the correct formula is applied based on the shape type, making the program easy to understand for a junior developer. Clear parameter usage (`value1`, `value2`) helps explain how different dimensions are passed to the same function.

Overall, this approach improves readability, maintainability, and scalability, making it ideal for onboarding new developers while ensuring correct area calculations.

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

Code:

```
]
3s
# Take input from the user
num = int(input("Enter a number: "))

# Check if the number is divisible by 2
if num % 2 == 0:
    print("The number is Even")
else:
    print("The number is Odd")
```

```
Enter a number: 5
The number is Odd
```

Justification

This code uses a concise conditional expression to determine whether a number is even or odd. It reduces the number of lines while maintaining clarity and correctness. The logic is easy to understand, efficient, and demonstrates how prompt optimization can lead to cleaner and more compact AI-generated code without changing functionality.

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

Code:

```
# Tool Comparison Summary

# Gemini:
# - Best for explanations and concept clarity
# - Generates readable, beginner-friendly code
# - Useful for learning and onboarding

# GitHub Copilot:
# - Strong IDE integration
# - Fast and practical code suggestions
# - Ideal for daily professional development

# Cursor AI:
# - Highly prompt-sensitive
# - Produces varied code styles for the same problem
# - Good for refactoring and experimentation
```

Justification

Gemini is recommended when clarity and understanding are the priority, as it provides well-explained and readable code suitable for learners. GitHub Copilot is ideal for professional development because it integrates directly into the IDE and delivers fast, context-aware code that improves productivity. Cursor AI stands out for experimentation, as its sensitivity to prompt wording helps developers explore multiple coding styles and refactoring options. Together, these tools serve different purposes based on usability and code quality needs.