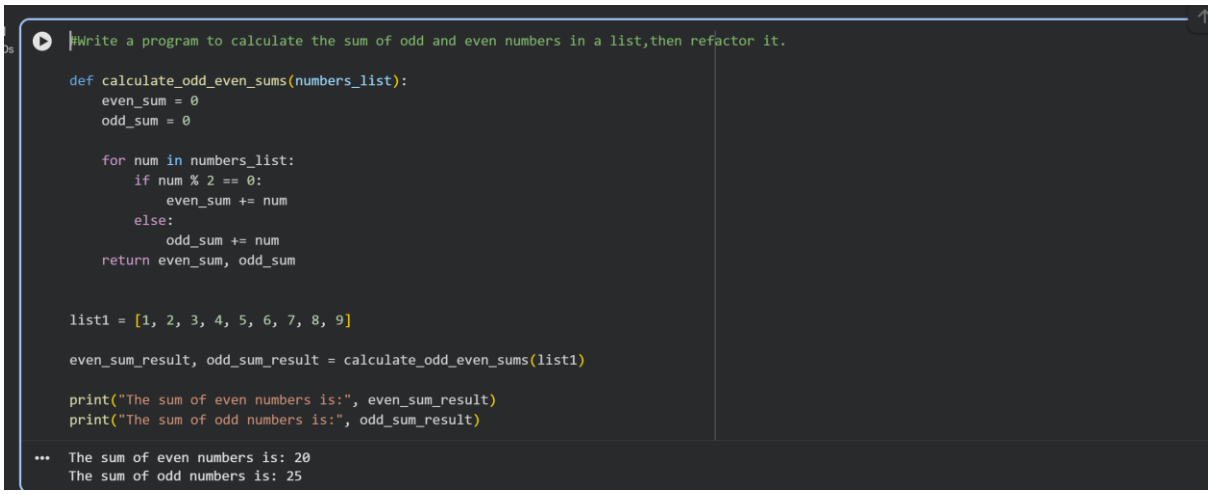
Assignment – 02
2303A51927

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

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

CODE and OUTPUT:



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

def calculate_odd_even_sums(numbers_list):
    even_sum = 0
    odd_sum = 0

    for num in numbers_list:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num
    return even_sum, odd_sum


list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]

even_sum_result, odd_sum_result = calculate_odd_even_sums(list1)

print("The sum of even numbers is:", even_sum_result)
print("The sum of odd numbers is:", odd_sum_result)
```

```
The sum of even numbers is: 20
The sum of odd numbers is: 25
```

Justification :
I was given a scenario that involved enhancing an old or legacy Python program designed to calculate the sum of odd and even numbers from a list. The original code used a loop with index-based iteration and separate variables to store the sums of odd and even values. Although the logic was correct, the code was lengthy and not very clean. By using AI tools such as Gemini and Cursor AI, I refactored the program to make it more concise and readable. The updated version utilizes Python's built-in function along with conditional expressions to directly compute the required sums. This reduced the complexity of the code and improved its structure. The refactored code is easier to understand, maintain, and modify. It also follows better programming practices and modern Python standards. Through this experience, I gained a clear understanding of how AI tools can assist in improving existing code and making it more efficient and maintainable.

Task 2: Area Calculation Explanation

PROMPT :
Explain a function that calculates the area of different shapes.

CODE and OUTPUT :

```python
#Explain a function that calculates the area of different shapes.
def calculate_area(shape, dimensions):
    """
    Calculate the area of different shapes based on the provided dimensions.

    Parameters:
    shape (str): The type of shape ('circle', 'rectangle', 'triangle').
    dimensions (tuple): The dimensions required to calculate the area.
                        For 'circle', provide (radius,).
                        For 'rectangle', provide (length, width).
                        For 'triangle', provide (base, height).

    Returns:
    float: The area of the shape.
    """
    if shape == 'circle':
        radius = dimensions[0]
        area = 3.14159 * (radius ** 2)
    elif shape == 'rectangle':
        length, width = dimensions
        area = length * width
    elif shape == 'triangle':
        base, height = dimensions
        area = 0.5 * base * height
    else:
        raise ValueError("Unsupported shape type. Supported shapes are: circle, rectangle, triangle.")

    return area
# Example usage:
```

```python
circle_area = calculate_area('circle', (5,))
rectangle_area = calculate_area('rectangle', (4, 6))
triangle_area = calculate_area('triangle', (3, 7))
print(f"Circle Area: {circle_area}")
print(f"Rectangle Area: {rectangle_area}")
print(f"Triangle Area: {triangle_area}")
# This function `calculate_area` takes in two parameters:
# 1. `shape`: a string that specifies the type of shape for which we want
#    to calculate the area. It can be 'circle', 'rectangle', or 'triangle'.
# 2. `dimensions`: a tuple that contains the necessary dimensions to compute
#    the area of the specified shape.
# The function checks the type of shape and uses the appropriate formula to
# calculate the area based on the provided dimensions. It then returns the
# calculated area as a float. If an unsupported shape is provided, it raises
# a ValueError with a message indicating the supported shapes.
# The example usage demonstrates how to call the function for each shape
# and prints the calculated areas.
```

```
Circle Area: 78.53975
Rectangle Area: 24
Triangle Area: 10.5
```

Justification :

The second task involved providing a clear explanation of a Python function designed to calculate the area of different geometric shapes. The given scenario required that the explanation be suitable for a junior developer who had recently joined the team and needed to understand the functionality of the code. The function was implemented using conditional statements to calculate the area of a circle, rectangle, and triangle. To support the explanation process, Google Gemini was used to analyze and describe the code. Gemini clearly explained the role of each parameter, the working of the conditional statements, and how the appropriate formula is selected based on the specified shape. The explanation was presented in a simple and structured manner, making it easy to understand. This task demonstrated that Gemini is an effective tool for learning and teaching, as it can clearly communicate code logic and make complex concepts more accessible, especially for beginners.

Task 3: Prompt Sensitivity Experiment

PROMPT :

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

CODE :

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

def sum_odd_even(numbers):
    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
numbers_list = [10, 15, 20, 25, 30, 35]
even_sum, odd_sum = sum_odd_even(numbers_list)
print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)
#Write a program to calculate the sum of odd and even numbers in a list

def calculate_sums(numbers):
    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
numbers_list = [10, 15, 20, 25, 30, 35]
even_sum, odd_sum = calculate_sums(numbers_list)
print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)
```

OUTPUT:

```
Sum of even numbers: 60
Sum of odd numbers: 75
Sum of even numbers: 60
Sum of odd numbers: 75
PS D:\AI>
```

Justification :

An experiment was conducted to observe how Cursor AI responds to different prompts for the same programming problem. The problem involved writing a Python function to determine whether a given number is prime. Multiple prompts were provided to Cursor AI, including a basic prompt, an optimized prompt, and a prompt requesting an explanation of the code. Each prompt produced a different version of the solution. When an optimized prompt was used, the AI generated a more efficient algorithm with improved performance. When an explanation was requested, the AI included detailed comments and structured the code in a way that was easier to understand. These variations highlighted how the nature of the prompt directly influences the output generated by the AI. This exercise demonstrated that AI tools are highly sensitive to prompt formulation. Well-structured and detailed prompts lead to more accurate, efficient, and useful results. The experience emphasized the importance of effective prompt writing when working with AI-based development tools.

Task 4: Tool Comparison Reflection

Based on practical experience with **Gemini**, **GitHub Copilot**, and **Cursor AI**, it can be observed that all three tools are effective for programming tasks, though they differ in usability and the nature of the support they provide. Each tool serves a distinct purpose depending on the developer's needs. Gemini is particularly effective for concept explanation and guided learning. It provides clear, well-structured code along with simple, beginner-friendly explanations, making it especially suitable for learning environments and for assisting new developers in understanding programming concepts and logic.

GitHub Copilot is primarily designed to support real-time software development. By operating directly within the code editor, it offers fast, context-aware code suggestions. The generated code is generally practical and reliable, which makes Copilot a powerful tool for increasing development speed and enhancing overall productivity. Cursor AI is especially useful for prompt-based experimentation and code improvement. It responds well to detailed instructions, produces multiple variations of solutions, and is highly effective for refactoring, optimization, and working with legacy codebases.

In conclusion, Gemini is most suitable for learning and conceptual understanding, GitHub Copilot excels in live coding assistance, and Cursor AI is best suited for refining code and prompt-driven development.