

Assignment – 2.5

Name-Anushka Voorugonda

H.no-2303A51176

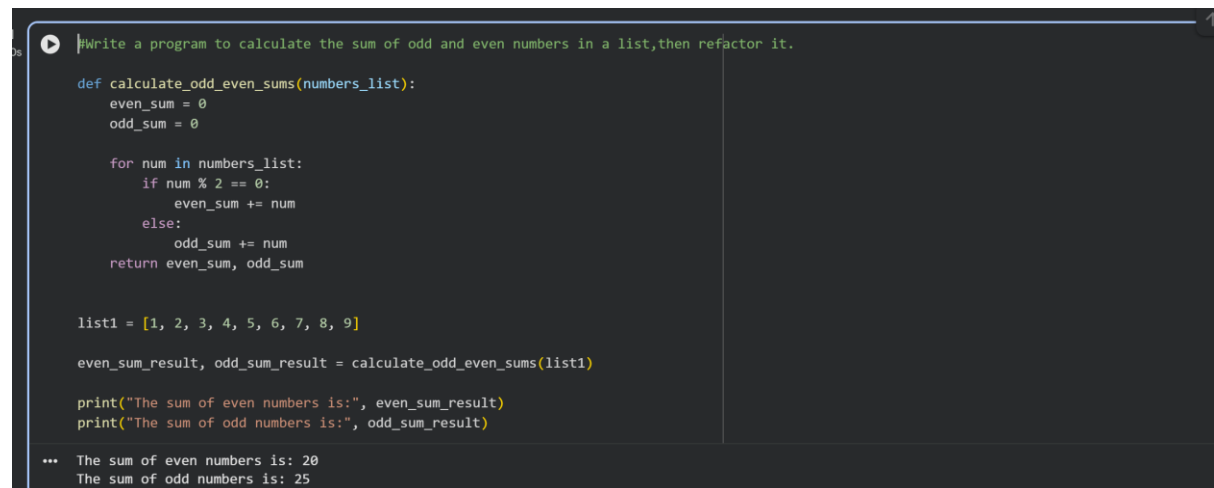
Batch-29

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 :

In this task, I was given a scenario where I had to improve an old or legacy Python program. The program was used to calculate the sum of odd and even numbers from a list. Initially, the program used a for loop with index values and separate variables to store the sums of odd and even numbers. Although the program worked correctly, the code was long and not very clean. With the help of AI tools like Gemini and Cursor AI, I refactored the code to make it shorter and more readable. The improved version used Python's built-in `sum()` function and conditional expressions to calculate the sums directly. This refactored code is easier to understand and maintain. It also follows better coding practices. From this task, I understood how AI tools can help in improving existing code and making it more efficient.

Task 2: Area Calculation Explanation

PROMPT :

Explain a function that calculates the area of different shapes.

CODE and OUTPUT :

```
# 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:
```

```
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 focused on explaining a Python function that calculates the area of different shapes. The scenario was that a junior developer is joining the team and needs to understand how the function works. I wrote a function that calculates the area of a circle, rectangle, and triangle using conditional statements. I then used Google Gemini to explain the code. Gemini explained the purpose of each parameter, how the if and elif conditions work, and how the correct formula is selected based on the shape. The explanation given by Gemini was simple and easy to understand. This made it helpful for beginners. From this task, I learned that Gemini is very useful for learning and teaching purposes because it explains code logic clearly.

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:

```
Triangle Area: 10.5
Sum of even numbers: 60
Sum of odd numbers: 75
Sum of even numbers: 60
Sum of odd numbers: 75
PS D:\AI>
```

Justification :

In this task, I tested how Cursor AI reacts to different prompts for the same programming problem. The problem was to write a function to check whether a number is prime. I gave Cursor AI different prompts such as a basic prompt, an optimized prompt, and a prompt asking for explanation. Each prompt resulted in different versions of the code. When I asked for an optimized solution, the AI generated a faster and more efficient algorithm. When I asked for explanation, the AI added comments and made the code easier to understand. This experiment helped me understand that AI tools are very sensitive to the way prompts are written. Clear and detailed prompts give better results. This task showed me the importance of writing good prompts while using AI tools.

Task 4: Tool Comparison Reflection

Based on my experience with Gemini, GitHub Copilot, and Cursor AI, all three are effective AI tools for programming, but they vary in terms of ease of use and the quality of code they generate. Gemini is particularly strong in explaining concepts and providing guidance. It offers simple, well-organized code along with clear, beginner-friendly explanations, which makes it very suitable for learning and for helping new developers understand programming tasks. GitHub Copilot is mainly focused on real-time development support. Since it works directly inside the code editor, it provides quick, context-based code suggestions. The code it

generates is usually practical and reliable, making it a powerful tool for improving coding speed and overall productivity. Cursor AI is especially useful for experimenting with prompts and improving existing code. It handles detailed instructions well, generates multiple versions of solutions, and is effective for refactoring, optimization, and working with legacy code. In conclusion, Gemini is best for learning and concept clarification, Copilot excels in live coding assistance, and Cursor AI is most suitable for refining code and prompt-driven development.