Name: Vaishnavi Achi

Hall ticket: 2303A51554

Batch: 29

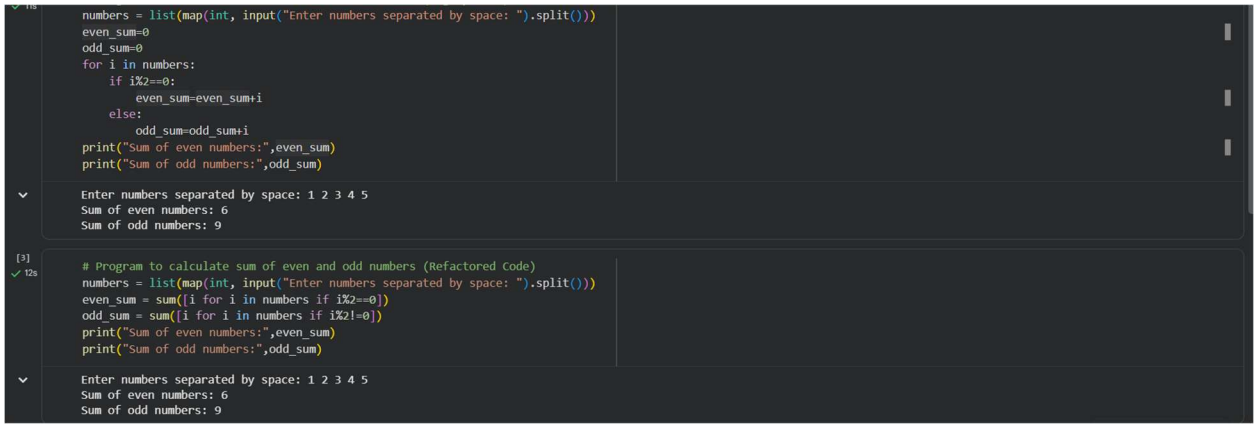
Assessment: 2.5

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

Prompt:

1. Legacy Program for Calculating the Sum of Even and Odd Numbers
2. Program to calculate sum of even and odd numbers (Refactored Code)

Code & output:



```
numbers = list(map(int, input("Enter numbers separated by space: ").split()))
even_sum=0
odd_sum=0
for i in numbers:
    if i%2==0:
        even_sum=even_sum+i
    else:
        odd_sum=odd_sum+i
print("Sum of even numbers:",even_sum)
print("Sum of odd numbers:",odd_sum)
```

```
Enter numbers separated by space: 1 2 3 4 5
Sum of even numbers: 6
Sum of odd numbers: 9
```

```
# Program to calculate sum of even and odd numbers (Refactored Code)
numbers = list(map(int, input("Enter numbers separated by space: ").split()))
even_sum = sum([i for i in numbers if i%2==0])
odd_sum = sum([i for i in numbers if i%2!=0])
print("Sum of even numbers:",even_sum)
print("Sum of odd numbers:",odd_sum)
```

```
Enter numbers separated by space: 1 2 3 4 5
Sum of even numbers: 6
Sum of odd numbers: 9
```

Justification:

The original program successfully calculates the sum of even and odd numbers, but it relies on repetitive loops and indexing, which makes the code longer and less readable. The refactored version makes use of Python's built-in sum() function along with conditional expressions, resulting in a cleaner and more concise implementation. This approach follows modern programming practices, improves maintainability, and lowers the risk of errors while delivering the same correct results.

Task 2: Area Calculation Explanation

Prompt: Area Calculation Program for Different Shape

Code & output:

```
shape = input("Enter the shape (circle, rectangle, triangle): ")
if shape.lower() == "circle":
    radius = float(input("Enter the radius of the circle: "))
    area = 3.14159 * radius * radius
    print("Area of the circle:", area)

elif shape.lower() == "rectangle":
    length = float(input("Enter the length of the rectangle: "))
    breadth = float(input("Enter the breadth of the rectangle: "))
    area = length * breadth
    print("Area of the rectangle:", area)

elif shape.lower() == "triangle":
    base = float(input("Enter the base of the triangle: "))
    height = float(input("Enter the height of the triangle: "))
    area = 0.5 * base * height
    print("Area of the triangle:", area)

else:
    print("Invalid shape entered")
```

```
Enter the shape (circle, rectangle, triangle): Circle
Enter the radius of the circle: 2
Area of the circle: 12.56636
```

Justification:

This program is developed to compute the area of various geometric shapes, including a circle, rectangle, and triangle, based on the user's selection. It applies conditional statements to determine the chosen shape and then uses the corresponding mathematical formula. The program collects separate inputs for each shape to ensure accurate calculations.Its step-by-step approach makes the logic easy to follow, particularly for beginners, and clearly demonstrates the use of decision-making in Python. The inclusion of an invalid input check enhances the program's reliability. Overall, the program successfully achieves its goal of calculating areas while maintaining simplicity, accuracy, and clarity.

Task 3: Prompt Sensitivity Experiment

Prompt: Implement a Python program to analyze a list of integers and determine the separate sums of its even and odd members.

Code:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum = sum(x for x in numbers if x % 2 == 0)
odd_sum = sum(x for x in numbers if x % 2 != 0)
print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)
```

Output:

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

Justification:

This program computes the sum of even and odd numbers by using Python's built-in sum() function along with conditional expressions. By avoiding manual loops, the code is shorter, cleaner, and easier to understand. This method follows modern Python coding standards, minimizes complexity, and improves maintainability while still delivering correct and reliable results.

Prompt: Optimized program to calculate the sum of even and odd numbers

Code:

```
def sum_even_odd(numbers):
    even_sum = 0
    odd_sum = 0

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

    return even_sum, odd_sum
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum, odd_sum = sum_even_odd(numbers)
print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)
```

Output:

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

Justification:

This program implements a function called sum_even_odd() to handle the task of finding the sum of even and odd numbers independently from the main code. The function goes through each element in the provided list and uses the modulus operator to determine whether the number is even or odd. Based on this check, even values are accumulated in even_sum and odd values in odd_sum. By placing the logic inside a function and returning the computed results, the program becomes more modular, reusable, and easier to understand. The main section of the program only supplies the input list and displays the output, resulting in a well-organized structure that is simple to test and suitable for integration into larger programs.

Prompt: Beginner-friendly program to calculate the sum of even and odd numbers

Code:

```
def sum_even_odd_user_input():
    numbers = input("Enter numbers separated by spaces: ")
    numbers_list = list(map(int, numbers.split()))

    even_sum = 0
    odd_sum = 0

    for number in numbers_list:
        if number % 2 == 0:
            even_sum += number
        else:
            odd_sum += number

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

Output:

```
Enter numbers separated by spaces: 1 2 3 4 5 6 7 8 9 10
Sum of even numbers: 30
Sum of odd numbers: 25
```

Justification:

This program takes a set of numbers as input from the user and computes the total of even and odd numbers separately. It first converts the entered string into a list of integers and then processes each value one by one. By using the modulus operator, the program identifies whether a number is even or odd and adds it to the appropriate total. Defining the logic inside a function improves code organization, reusability, and ease of maintenance. The use of meaningful variable names and a straightforward loop makes the program easy to read and understand, especially for beginners. Overall, the program successfully achieves its objective of accurately separating and summing even and odd numbers in a clear and dependable way.

Task 4: Tool Comparison Reflection

From my experience working with Gemini, GitHub Copilot, and Cursor AI, each tool offers valuable support for programming, but they serve different purposes in terms of usability and code quality.

Gemini is especially effective for learning and understanding. It delivers clear explanations along with simple and well-structured code, making it a good choice for beginners and for onboarding junior developers.

GitHub Copilot is designed for continuous coding assistance. It integrates seamlessly with development environments and provides instant, context-aware suggestions. The code it produces is generally reliable and follows standard coding practices, which helps developers write code faster and more efficiently.

Cursor AI is particularly useful for experimenting with prompts and enhancing existing programs. It performs well in refactoring tasks, generates alternative implementations, and supports optimization and legacy code improvement.

Overall, Gemini is most suitable for conceptual learning, Copilot is ideal for real-time development, and Cursor AI is best for code refinement and prompt-based experimentation.