ASSIGNMENT:02

HALLTICKET:2303A51598

BATCH:29

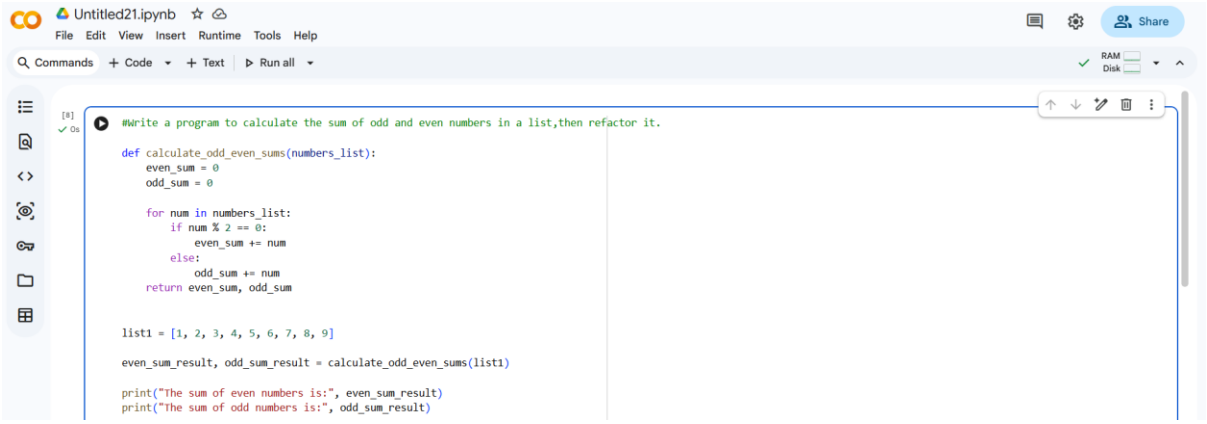
SHAIK NEHA FARIYAL

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 using AI.



OUTPUT:

```
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.

```
#explain a function that calculates the area of different shapes.
def calculate_area(shape_type, **kwargs):
    if shape_type == 'circle':
        if 'radius' in kwargs:
            import math
            return math.pi * kwargs['radius']**2
        else:
            return "Error: Radius not provided for circle."
    elif shape_type == 'rectangle':
        if 'length' in kwargs and 'width' in kwargs:
            return kwargs['length'] * kwargs['width']
        else:
            return "Error: Length or width not provided for rectangle."
    elif shape_type == 'triangle':
        if 'base' in kwargs and 'height' in kwargs:
            return 0.5 * kwargs['base'] * kwargs['height']
        else:
            return "Error: Base or height not provided for triangle."
    else:
        return "Error: Unknown shape type."
```

```
print(f"Area of circle with radius 5: {calculate_area('circle', radius=5)}")
print(f"Area of rectangle with length 4 and width 6: {calculate_area('rectangle', length=4, width=6)}")
print(f"Area of triangle with base 10 and height 5: {calculate_area('triangle', base=10, height=5)}")
print(f"Error for rectangle with missing width: {calculate_area('rectangle', length=4)}")
print(f"Error for unknown shape: {calculate_area('square', side=5)}")
```

OUTPUT:

```
Area of circle with radius 5: 78.53981633974483
Area of rectangle with length 4 and width 6: 24
Area of triangle with base 10 and height 5: 25.0
Error for rectangle with missing width: Error: Length or width not provided for rectangle.
Error for unknown shape: Error: Unknown shape type.
```

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.

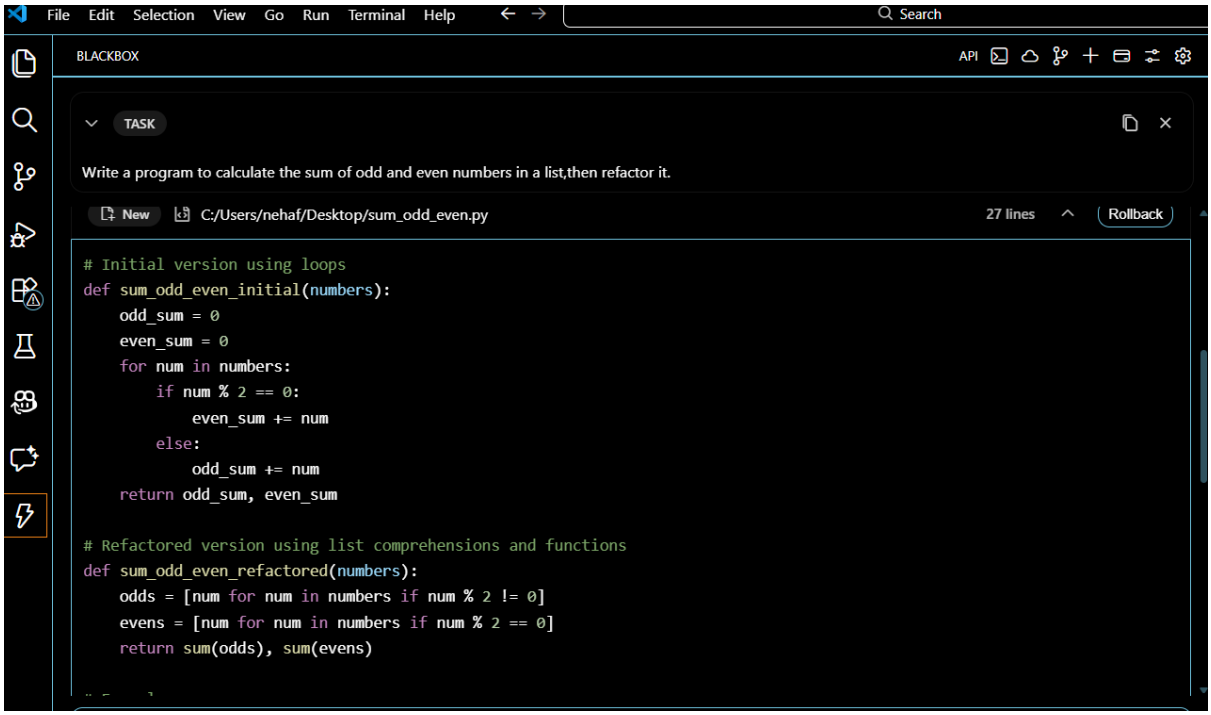
I wrote a function that calculates the area of a circle, rectangle, and triangle using conditional statements. I then used Google Gemini to explain the code. Gemini explained the purpose of each parameter, how the if and elif conditions work, and how the correct formula is selected based on the shape.

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:

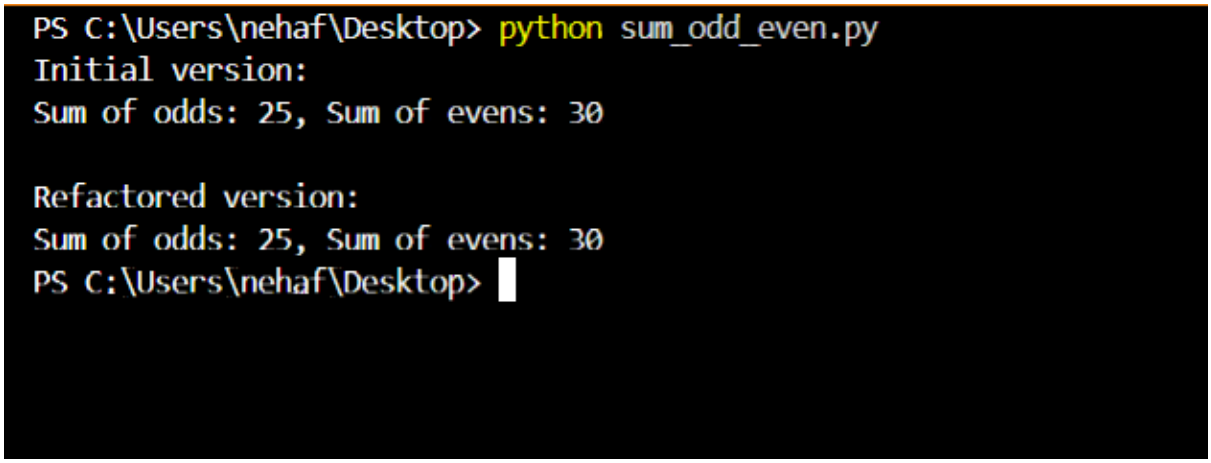
Write a program to calculate the sum of odd and even numbers in a list, then refactor it.




```
# Example usage
if __name__ == "__main__":
    sample_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    print("Initial version:")
    odd_sum, even_sum = sum_odd_even_initial(sample_list)
    print(f"Sum of odds: {odd_sum}, Sum of evens: {even_sum}")

    print("\nRefactored version:")
    odd_sum, even_sum = sum_odd_even_refactored(sample_list)
    print(f"Sum of odds: {odd_sum}, Sum of evens: {even_sum}")
```

OUTPUT:



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

PROMPT:

compare Gemini, Copilot, and Cursor AI for usability and code quality.

CODE:

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.

JUSTIFICATION:

In the final task, I compared Google Gemini, GitHub Copilot, and Cursor AI based on my experience in this lab. Google Gemini is very good at explaining code and concepts, which makes it suitable for students and beginners. GitHub Copilot is useful for fast coding and real-time suggestions but does not explain the logic much.

Cursor AI combines both features. It helps in writing code, improving existing code, and explaining logic when asked. Because of this, I found Cursor AI to be the most useful tool for learning as well as development.