ASSIGNMENT-2.5

Name: Praneeth banda

HT.No: 2303A51711

Batch: 20

Task 1:

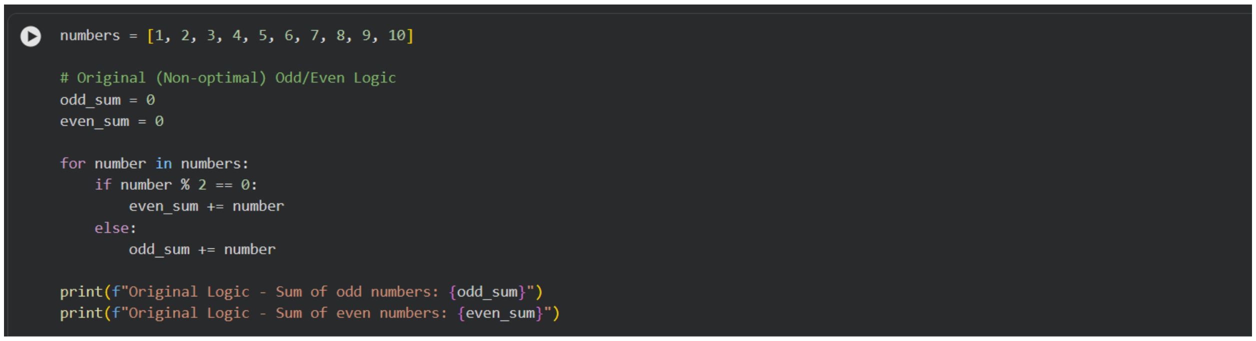
Refactoring Odd/Even Logic (List Version) Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI.

Scenario

You are improving legacy code.

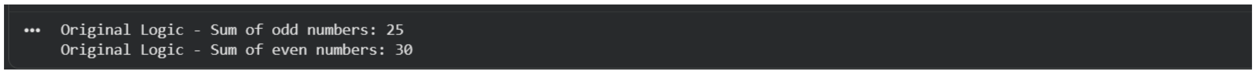
Prompt: calculate the sum of odd and even numbers and refactor it using AI.

Code:



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

# Original (Non-optimal) Odd/Even Logic
odd_sum = 0
even_sum = 0

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

print(f"Original Logic - Sum of odd numbers: {odd_sum}")
print(f"Original Logic - Sum of even numbers: {even_sum}")
```

Result:



```
Original Logic - Sum of odd numbers: 25
Original Logic - Sum of even numbers: 30
```

Observation:

The refactored version eliminates the need for manual loops and step-by-step conditional accumulation, which simplifies the overall logic. As a result, the code becomes more concise, easier to read, and more efficient in execution. Despite these improvements, it still produces the same correct output, demonstrating how cleaner approaches can enhance code quality without changing functionality.

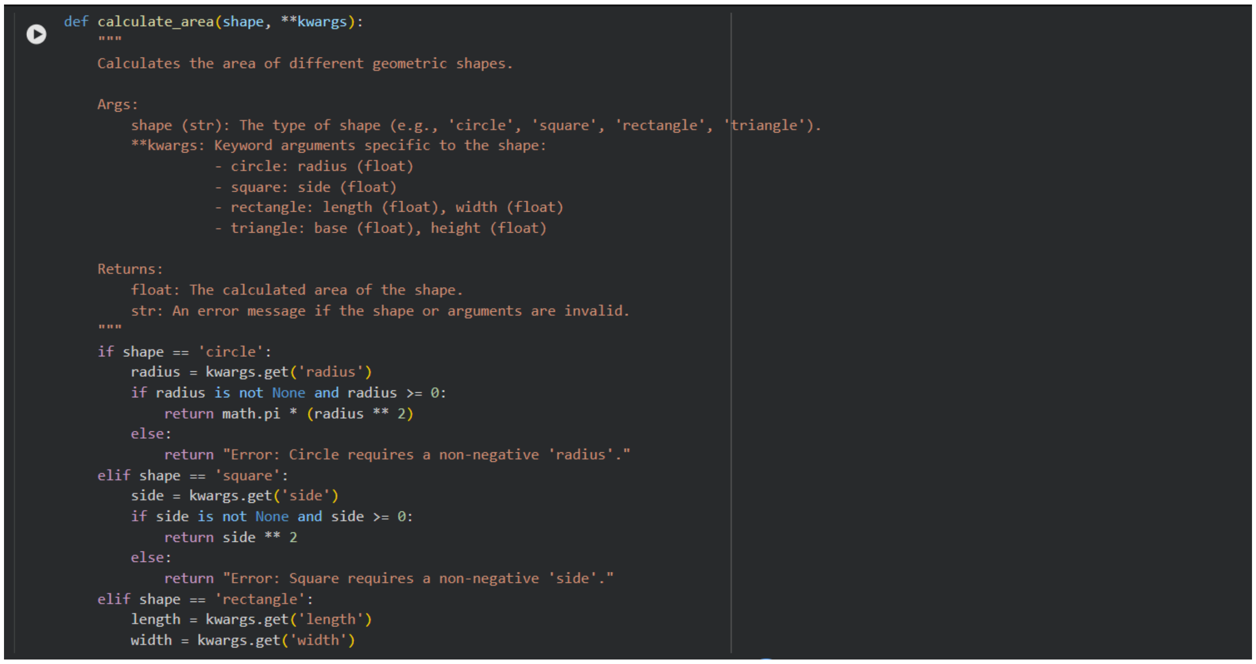
Task 2: Area Calculation Explanation. Ask Gemini to explain a function that calculates the area of different shapes.

Scenario

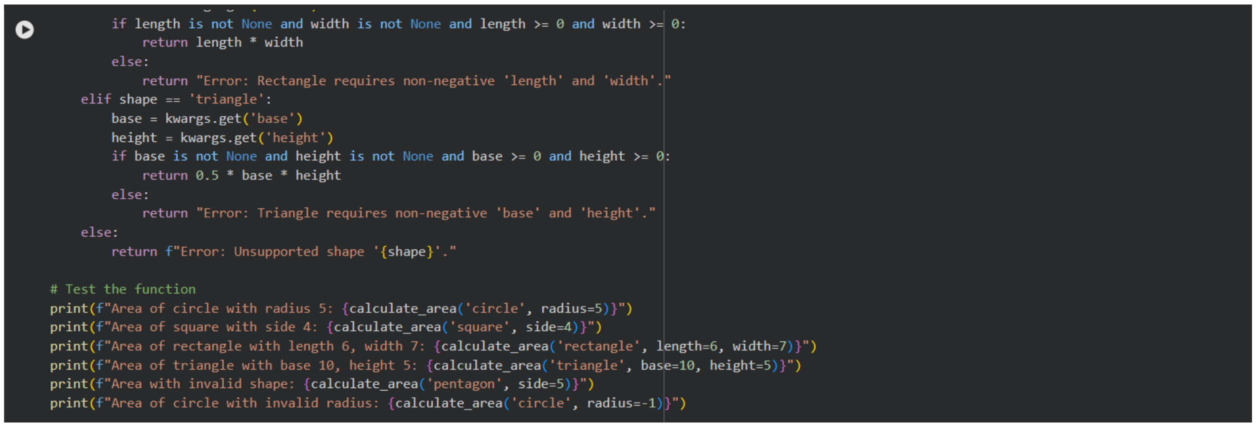
You are onboarding a junior developer.

Prompt: give a function that calculates the area of different shapes

Code:

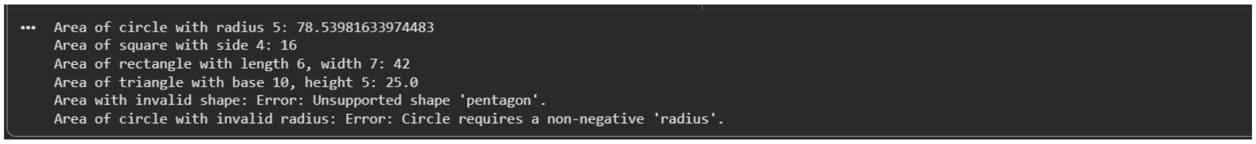


```python
def calculate_area(shape, **kwargs):
    """
    Calculates the area of different geometric shapes.

    Args:
        shape (str): The type of shape (e.g., 'circle', 'square', 'rectangle', 'triangle').
        **kwargs: Keyword arguments specific to the shape:
                  - circle: radius (float)
                  - square: side (float)
                  - rectangle: length (float), width (float)
                  - triangle: base (float), height (float)

    Returns:
        float: The calculated area of the shape.
        str: An error message if the shape or arguments are invalid.
    """
    if shape == 'circle':
        radius = kwargs.get('radius')
        if radius is not None and radius >= 0:
            return math.pi * (radius ** 2)
        else:
            return "Error: Circle requires a non-negative 'radius'."
    elif shape == 'square':
        side = kwargs.get('side')
        if side is not None and side >= 0:
            return side ** 2
        else:
            return "Error: Square requires a non-negative 'side'."
    elif shape == 'rectangle':
        length = kwargs.get('length')
        width = kwargs.get('width')
```



```python
        if length is not None and width is not None and length >= 0 and width >= 0:
            return length * width
        else:
            return "Error: Rectangle requires non-negative 'length' and 'width'."
    elif shape == 'triangle':
        base = kwargs.get('base')
        height = kwargs.get('height')
        if base is not None and height is not None and base >= 0 and height >= 0:
            return 0.5 * base * height
        else:
            return "Error: Triangle requires non-negative 'base' and 'height'."
    else:
        return f"Error: Unsupported shape '{shape}'."

# Test the function
print(f"Area of circle with radius 5: {calculate_area('circle', radius=5)}")
print(f"Area of square with side 4: {calculate_area('square', side=4)}")
print(f"Area of rectangle with length 6, width 7: {calculate_area('rectangle', length=6, width=7)}")
print(f"Area of triangle with base 10, height 5: {calculate_area('triangle', base=10, height=5)}")
print(f"Area with invalid shape: {calculate_area('pentagon', side=5)}")
print(f"Area of circle with invalid radius: {calculate_area('circle', radius=-1)}")
```

Result:



```
Area of circle with radius 5: 78.53981633974483
Area of square with side 4: 16
Area of rectangle with length 6, width 7: 42
Area of triangle with base 10, height 5: 25.0
Area with invalid shape: Error: Unsupported shape 'pentagon'.
Area of circle with invalid radius: Error: Circle requires a non-negative 'radius'.
```

Observation:

Gemini clearly explains both the programming logic and the underlying mathematical reasoning in a well-organized manner, which makes complex ideas easier to understand. Its step-by-step explanations help learners follow the flow of the solution without confusion. This approach is especially helpful for beginners and junior developers who are still developing their problem-solving skills and need strong conceptual clarity.

Task 3: Prompt Sensitivity Experiment Use Cursor AI with different prompts for the same problem and observe code changes.

Scenario

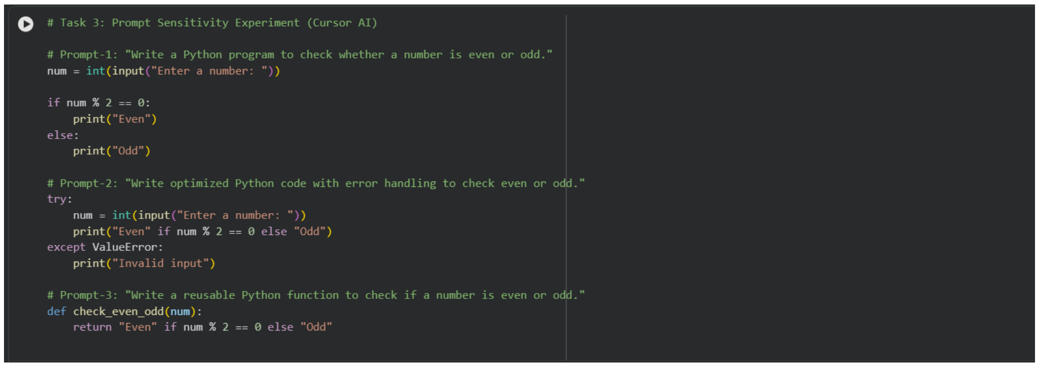
You are testing how AI responds to different prompts.

Prompt-1: Write a Python program to check whether a number is even or odd

Prompt-2: Write optimized Python code with error handling to check even or odd

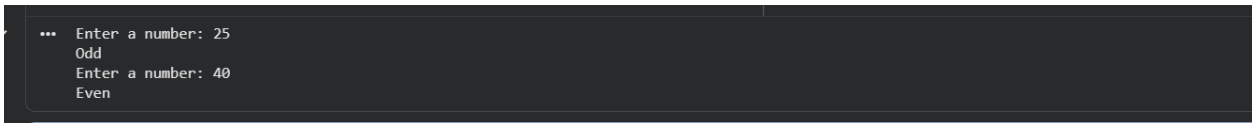
Prompt-3: Write a reusable Python function to check if a number is even or odd

Code:



```python
# Task 3: Prompt Sensitivity Experiment (Cursor AI)

# Prompt-1: "Write a Python program to check whether a number is even or odd."
num = int(input("Enter a number: "))

if num % 2 == 0:
    print("Even")
else:
    print("Odd")

# Prompt-2: "Write optimized Python code with error handling to check even or odd."
try:
    num = int(input("Enter a number: "))
    print("Even" if num % 2 == 0 else "Odd")
except ValueError:
    print("Invalid input")

# Prompt-3: "Write a reusable Python function to check if a number is even or odd."
def check_even_odd(num):
    return "Even" if num % 2 == 0 else "Odd"
```

Result:



```
Enter a number: 25
Odd
Enter a number: 40
Even
```

Observation:

Modifying the prompt had a direct impact on the organization and quality of the generated code. Basic prompts produced simple and straightforward logic. When optimization was requested, the code included error handling and more compact syntax. Prompts focused on reusability resulted in a function-based and modular design.

Task 4: Tool Comparison Reflection. Based on your work in this topic, compare Gemini, Copilot, and Cursor AI for usability and code quality.

Scenario

You must recommend an AI coding tool.

Prompt: Compare Google Gemini, GitHub Copilot, and Cursor AI in terms of usability and code quality. Highlight their strengths and limitations, and recommend the most suitable tool for regular software development

Observation:

Tool Comparison

Google Gemini (Google Colab)
Google Gemini is well suited for explaining programming concepts and offering detailed code explanations. It is highly beneficial for beginners and learners who need conceptual understanding. Gemini is commonly used in educational environments, tutorials, and documentation. However, it does not provide direct integration with code editors, which limits its effectiveness for real-time software development.

GitHub Copilot
GitHub Copilot integrates directly with popular IDEs such as Visual Studio Code and provides intelligent, real-time code suggestions while developers write code. It significantly increases productivity by reducing repetitive coding tasks and maintaining consistent code quality. Copilot

supports modern programming languages and frameworks, making it ideal for continuous and professional software development.

Cursor AI
Cursor AI is mainly focused on code refactoring, optimization, and prompt-based experimentation. It is effective for improving existing code by enhancing structure and readability. Cursor AI is particularly useful during code reviews and when working with large or complex codebases. However, it may be less beginner-friendly compared to Gemini and is more dependent on clear prompts.

Recommendation
Based on usability, integration, and overall code quality, GitHub Copilot is recommended as the most suitable AI coding tool for regular software development. Its seamless IDE integration and real-time assistance make it the most practical and efficient option for everyday coding tasks.

Reflection
This comparison highlighted how different AI tools support developers at various stages of the development process. Gemini proved to be effective for learning and understanding programming logic, while Cursor AI demonstrated strong capabilities in improving and refining existing code. GitHub Copilot stood out for enhancing productivity through real-time coding support. This exercise emphasized the importance of choosing the right tool based on specific development needs.

Conclusion
Each AI tool serves a different purpose in software development. Google Gemini is best for learning and understanding programming concepts, Cursor AI is useful for refactoring and code improvement, and GitHub Copilot is ideal for regular development due to its productivity benefits and IDE integration.