Lab Assignment 2.5 – AI ASSISTED CODING

Name: Syed Murtaza

H.T No: 2303A51259

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

❖ **Scenario:**

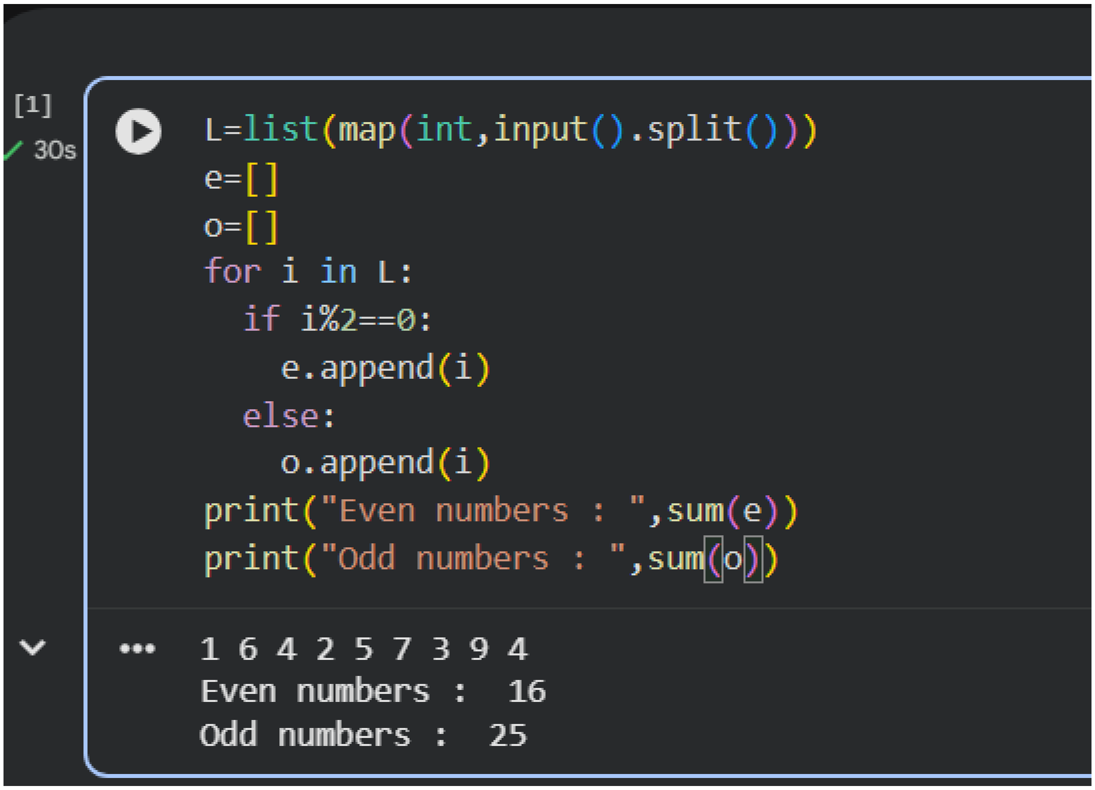
You are improving legacy code.

❖ **Task:**

Write a program to calculate the sum of odd and even numbers in a list,

then refactor it using AI.

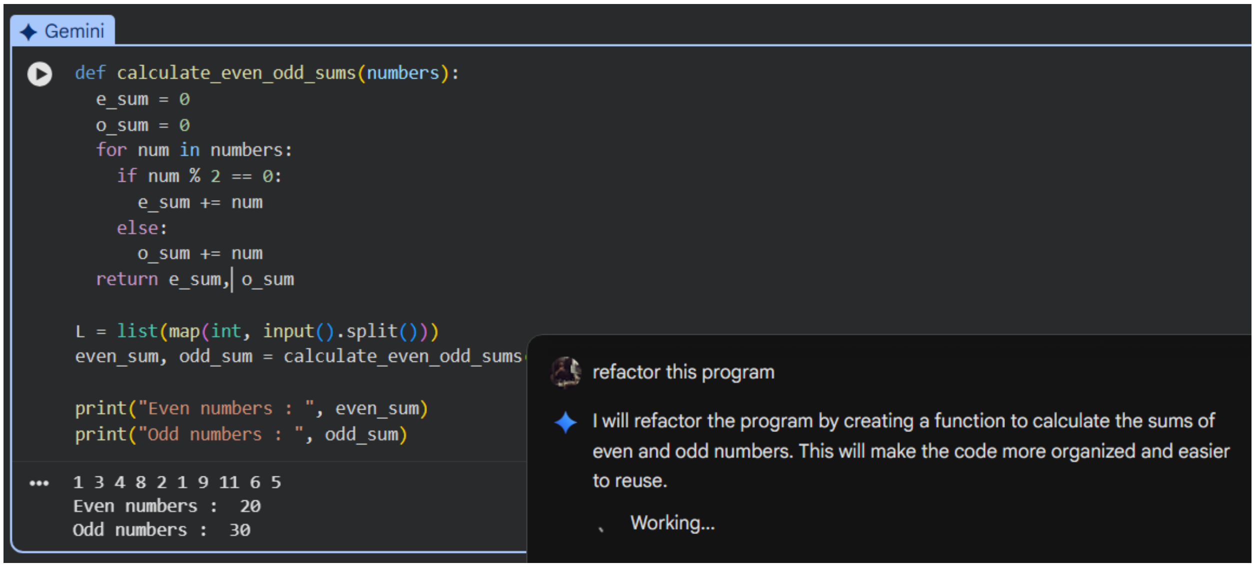
Original Code:



```
L=list(map(int,input().split()))
e=[]
o=[]
for i in L:
  if i%2==0:
    e.append(i)
  else:
    o.append(i)
print("Even numbers : ",sum(e))
print("Odd numbers : ",sum(o))
```

```
1 6 4 2 5 7 3 9 4
Even numbers :  16
Odd numbers :  25
```

Prompted Code:



```python
def calculate_even_odd_sums(numbers):
  e_sum = 0
  o_sum = 0
  for num in numbers:
    if num % 2 == 0:
      e_sum += num
    else:
      o_sum += num
  return e_sum, o_sum

L = list(map(int, input().split()))
even_sum, odd_sum = calculate_even_odd_sums

print("Even numbers : ", even_sum)
print("Odd numbers : ", odd_sum)
```

```
1 3 4 8 2 1 9 11 6 5
Even numbers :  20
Odd numbers :  30
```

refactor this program

I will refactor the program by creating a function to calculate the sums of even and odd numbers. This will make the code more organized and easier to reuse.

Working...

Task 2: Area Calculation Explanation

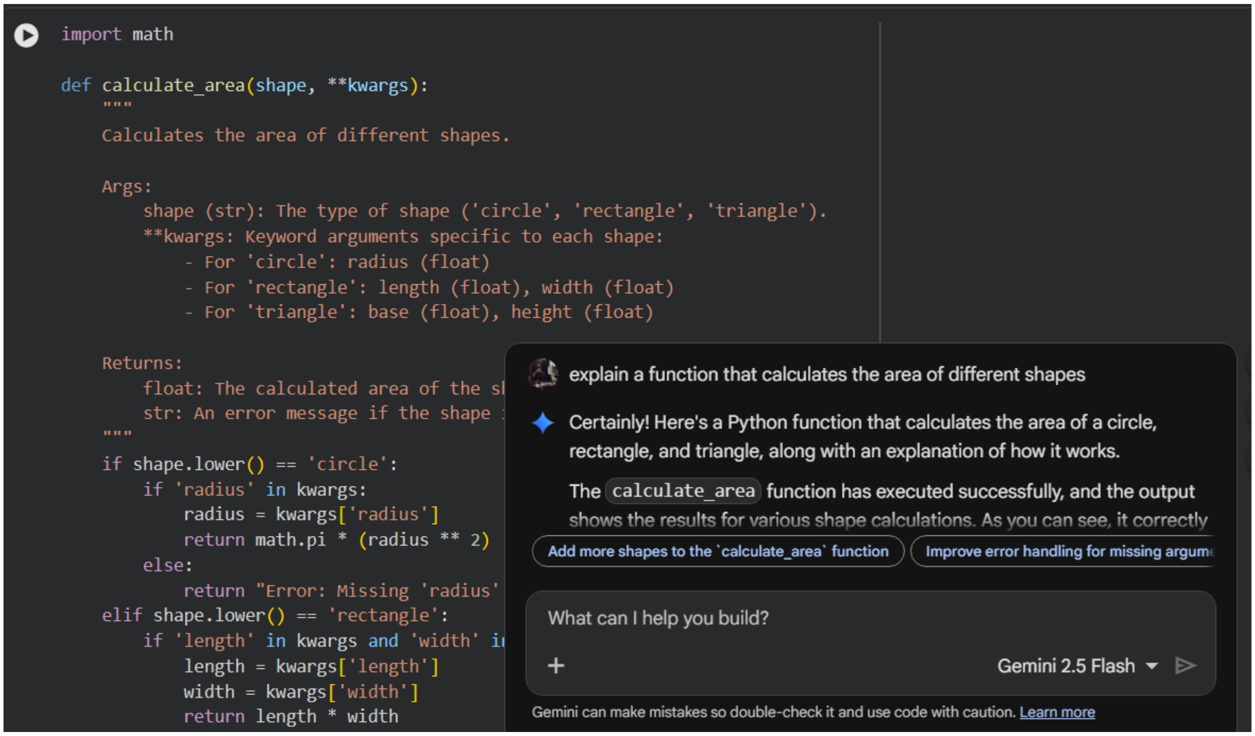
❖ Scenario:

You are onboarding a junior developer.

❖ Task:

Ask Gemini to explain a function that calculates the area of different shapes.

Prompted Code:



```
import math

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

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

    Returns:
        float: The calculated area of the sl
        str: An error message if the shape i
    """
    if shape.lower() == 'circle':
        if 'radius' in kwargs:
            radius = kwargs['radius']
            return math.pi * (radius ** 2)
        else:
            return "Error: Missing 'radius'
    elif shape.lower() == 'rectangle':
        if 'length' in kwargs and 'width' i
            length = kwargs['length']
            width = kwargs['width']
            return length * width
```

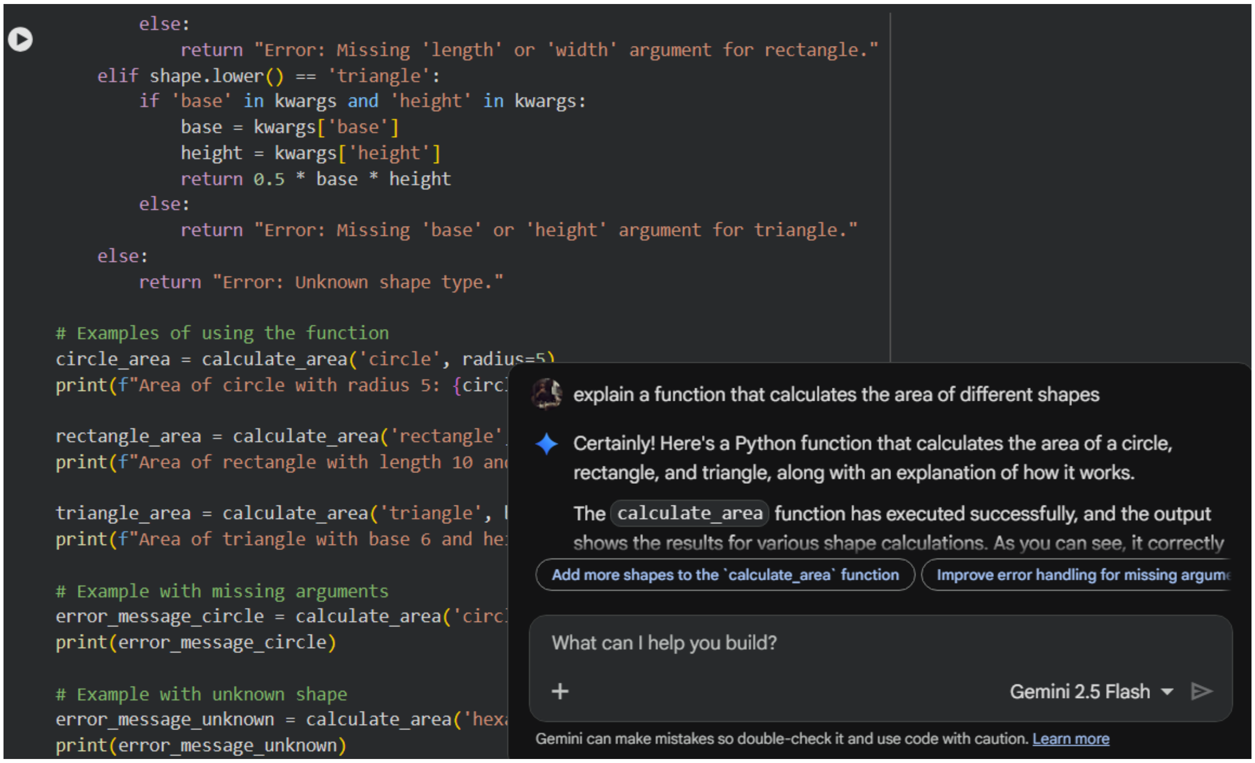
explain a function that calculates the area of different shapes

Certainly! Here's a Python function that calculates the area of a circle, rectangle, and triangle, along with an explanation of how it works.

The `calculate_area` function has executed successfully, and the output shows the results for various shape calculations. As you can see, it correctly

Add more shapes to the `calculate_area` function
Improve error handling for missing argume
What can I help you build?
+
Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. Learn more



```
        else:
            return "Error: Missing 'length' or 'width' argument for rectangle."
    elif shape.lower() == 'triangle':
        if 'base' in kwargs and 'height' in kwargs:
            base = kwargs['base']
            height = kwargs['height']
            return 0.5 * base * height
        else:
            return "Error: Missing 'base' or 'height' argument for triangle."
    else:
        return "Error: Unknown shape type."

# Examples of using the function
circle_area = calculate_area('circle', radius=5)
print(f"Area of circle with radius 5: {circ

rectangle_area = calculate_area('rectangle'
print(f"Area of rectangle with length 10 an

triangle_area = calculate_area('triangle', 
print(f"Area of triangle with base 6 and he

# Example with missing arguments
error_message_circle = calculate_area('circ
print(error_message_circle)

# Example with unknown shape
error_message_unknown = calculate_area('hexa
print(error_message_unknown)
```

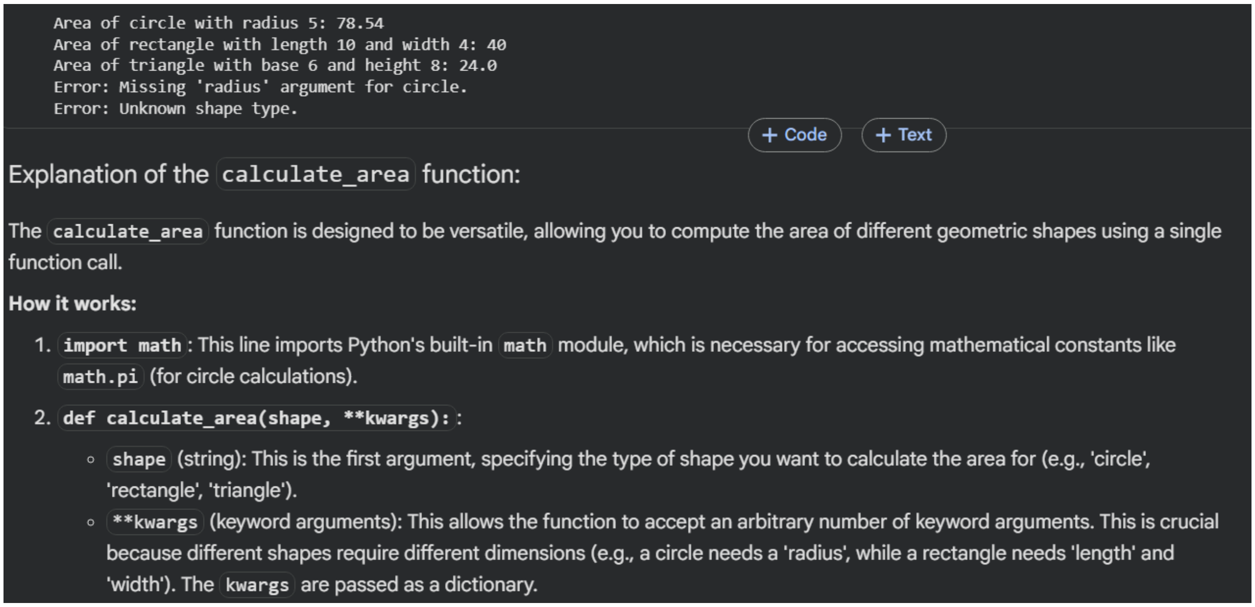
explain a function that calculates the area of different shapes

Certainly! Here's a Python function that calculates the area of a circle, rectangle, and triangle, along with an explanation of how it works.

The `calculate_area` function has executed successfully, and the output shows the results for various shape calculations. As you can see, it correctly

Add more shapes to the `calculate_area` function
Improve error handling for missing argume
What can I help you build?
+
Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. Learn more



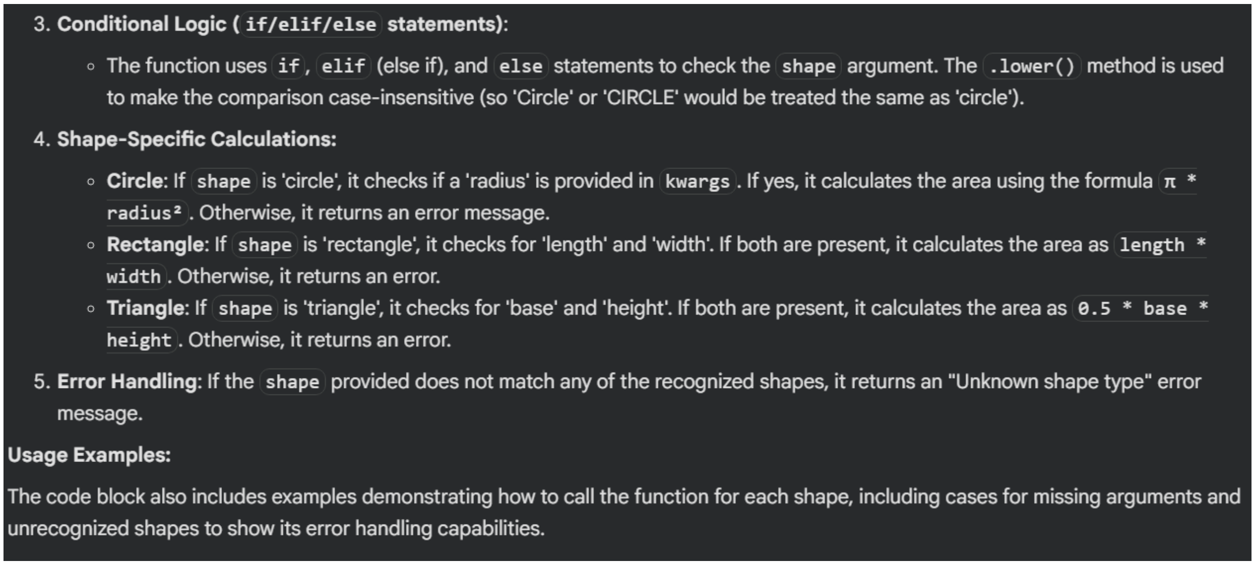
```
Area of circle with radius 5: 78.54
Area of rectangle with length 10 and width 4: 40
Area of triangle with base 6 and height 8: 24.0
Error: Missing 'radius' argument for circle.
Error: Unknown shape type.
```

Explanation of the `calculate_area` function:

The `calculate_area` function is designed to be versatile, allowing you to compute the area of different geometric shapes using a single function call.

How it works:

1. `import math`: This line imports Python's built-in `math` module, which is necessary for accessing mathematical constants like `math.pi` (for circle calculations).
2. `def calculate_area(shape, **kwargs):`:
   - `shape` (string): This is the first argument, specifying the type of shape you want to calculate the area for (e.g., 'circle', 'rectangle', 'triangle').
   - `**kwargs` (keyword arguments): This allows the function to accept an arbitrary number of keyword arguments. This is crucial because different shapes require different dimensions (e.g., a circle needs a 'radius', while a rectangle needs 'length' and 'width'). The `kwargs` are passed as a dictionary.



3. **Conditional Logic (`if/elif/else` statements):**
   - The function uses `if`, `elif` (else if), and `else` statements to check the `shape` argument. The `.lower()` method is used to make the comparison case-insensitive (so 'Circle' or 'CIRCLE' would be treated the same as 'circle').
4. **Shape-Specific Calculations:**
   - **Circle**: If `shape` is 'circle', it checks if a 'radius' is provided in `kwargs`. If yes, it calculates the area using the formula `π * radius²`. Otherwise, it returns an error message.
   - **Rectangle**: If `shape` is 'rectangle', it checks for 'length' and 'width'. If both are present, it calculates the area as `length * width`. Otherwise, it returns an error.
   - **Triangle**: If `shape` is 'triangle', it checks for 'base' and 'height'. If both are present, it calculates the area as `0.5 * base * height`. Otherwise, it returns an error.
5. **Error Handling**: If the `shape` provided does not match any of the recognized shapes, it returns an "Unknown shape type" error message.

Usage Examples:

The code block also includes examples demonstrating how to call the function for each shape, including cases for missing arguments and unrecognized shapes to show its error handling capabilities.

Task 3: Prompt Sensitivity Experiment

❖ Scenario:

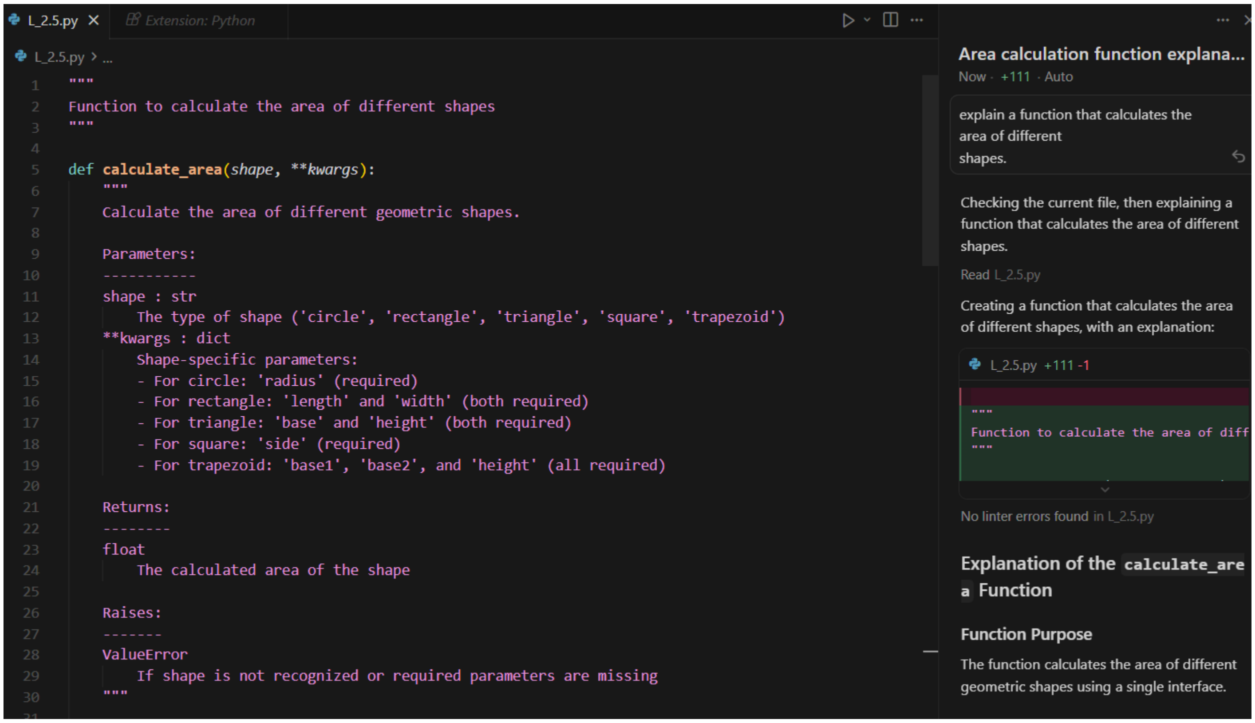
You are testing how AI responds to different prompts.

❖ Task:

Use Cursor AI with different prompts for the same problem and observe

code changes.

Prompt And Code:



```python
"""
Function to calculate the area of different shapes
"""

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

    Parameters:
    -----------
    shape : str
        The type of shape ('circle', 'rectangle', 'triangle', 'square', 'trapezoid')
    **kwargs : dict
        Shape-specific parameters:
        - For circle: 'radius' (required)
        - For rectangle: 'length' and 'width' (both required)
        - For triangle: 'base' and 'height' (both required)
        - For square: 'side' (required)
        - For trapezoid: 'base1', 'base2', and 'height' (all required)

    Returns:
    --------
    float
        The calculated area of the shape

    Raises:
    -------
    ValueError
        If shape is not recognized or required parameters are missing
    """
```

Area calculation function explana...

Now · +111 · Auto

explain a function that calculates the
area of different
shapes.

Checking the current file, then explaining a function that calculates the area of different shapes.

Read L_2.5.py

Creating a function that calculates the area of different shapes, with an explanation:

L_2.5.py +111 -1

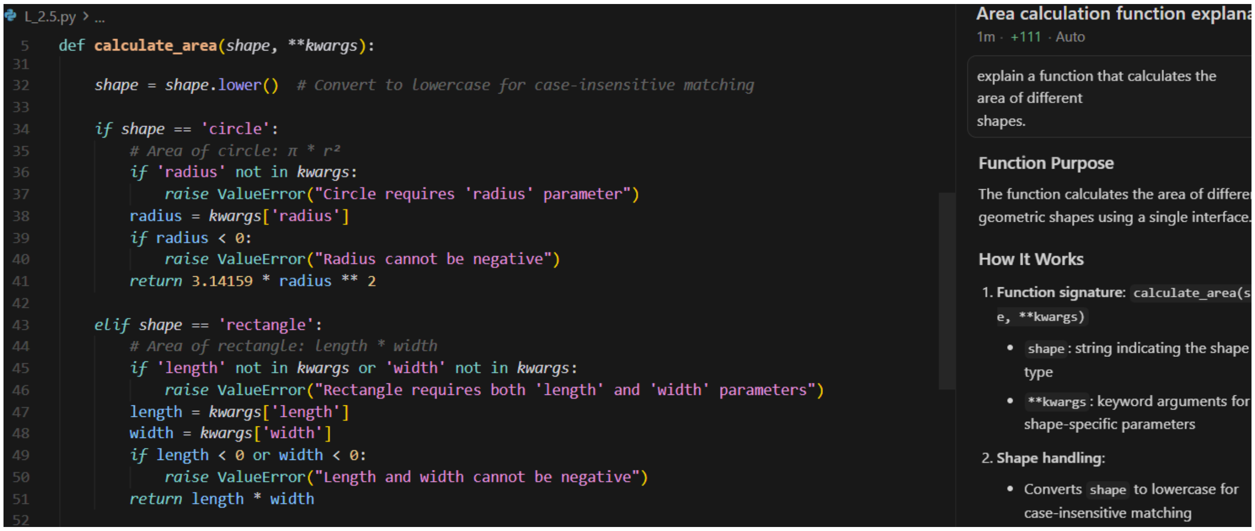
```
"""
Function to calculate the area of diff
"""
```

No linter errors found in L_2.5.py

Explanation of the `calculate_area` Function

Function Purpose

The function calculates the area of different geometric shapes using a single interface.



```python
def calculate_area(shape, **kwargs):

    shape = shape.lower()  # Convert to lowercase for case-insensitive matching

    if shape == 'circle':
        # Area of circle: π * r²
        if 'radius' not in kwargs:
            raise ValueError("Circle requires 'radius' parameter")
        radius = kwargs['radius']
        if radius < 0:
            raise ValueError("Radius cannot be negative")
        return 3.14159 * radius ** 2

    elif shape == 'rectangle':
        # Area of rectangle: length * width
        if 'length' not in kwargs or 'width' not in kwargs:
            raise ValueError("Rectangle requires both 'length' and 'width' parameters")
        length = kwargs['length']
        width = kwargs['width']
        if length < 0 or width < 0:
            raise ValueError("Length and width cannot be negative")
        return length * width

```

Area calculation function explana

1m · +111 · Auto

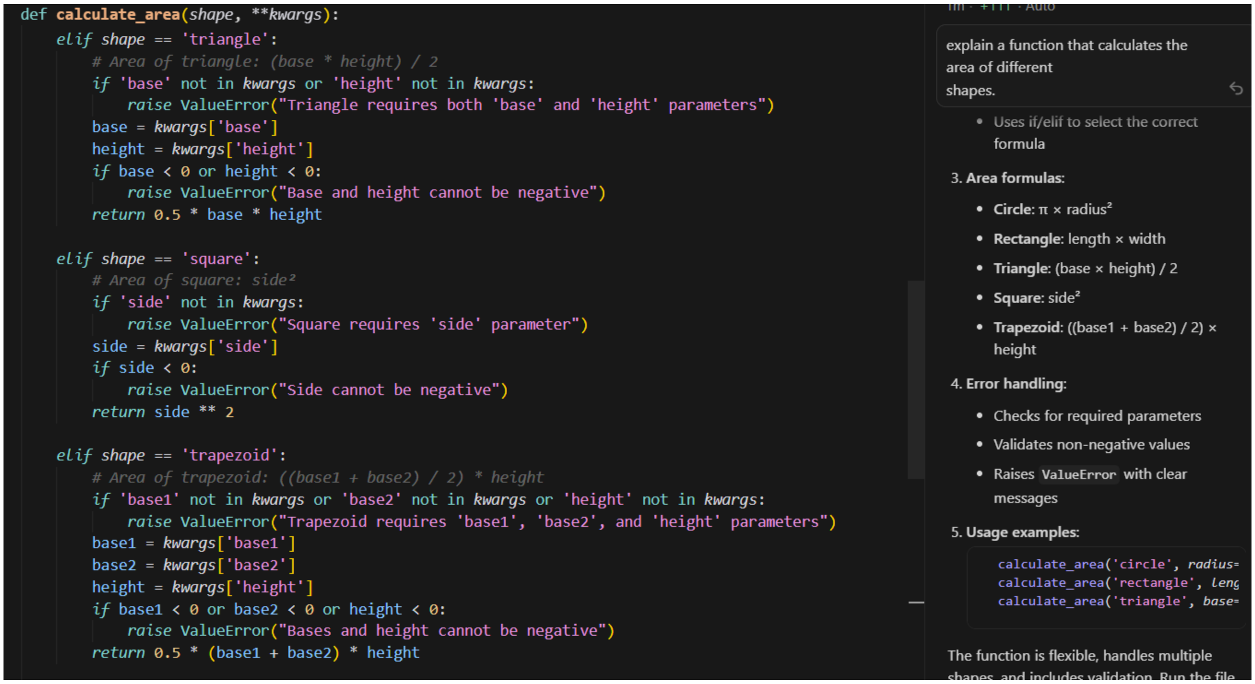
explain a function that calculates the
area of different
shapes.

Function Purpose

The function calculates the area of differe
geometric shapes using a single interface.

How It Works

1. **Function signature:** `calculate_area(s e, **kwargs)`
    - `shape`: string indicating the shape type
    - `**kwargs`: keyword arguments for shape-specific parameters
2. **Shape handling:**
    - Converts `shape` to lowercase for case-insensitive matching



```python
def calculate_area(shape, **kwargs):
    elif shape == 'triangle':
        # Area of triangle: (base * height) / 2
        if 'base' not in kwargs or 'height' not in kwargs:
            raise ValueError("Triangle requires both 'base' and 'height' parameters")
        base = kwargs['base']
        height = kwargs['height']
        if base < 0 or height < 0:
            raise ValueError("Base and height cannot be negative")
        return 0.5 * base * height

    elif shape == 'square':
        # Area of square: side²
        if 'side' not in kwargs:
            raise ValueError("Square requires 'side' parameter")
        side = kwargs['side']
        if side < 0:
            raise ValueError("Side cannot be negative")
        return side ** 2

    elif shape == 'trapezoid':
        # Area of trapezoid: ((base1 + base2) / 2) * height
        if 'base1' not in kwargs or 'base2' not in kwargs or 'height' not in kwargs:
            raise ValueError("Trapezoid requires 'base1', 'base2', and 'height' parameters")
        base1 = kwargs['base1']
        base2 = kwargs['base2']
        height = kwargs['height']
        if base1 < 0 or base2 < 0 or height < 0:
            raise ValueError("Bases and height cannot be negative")
        return 0.5 * (base1 + base2) * height
```

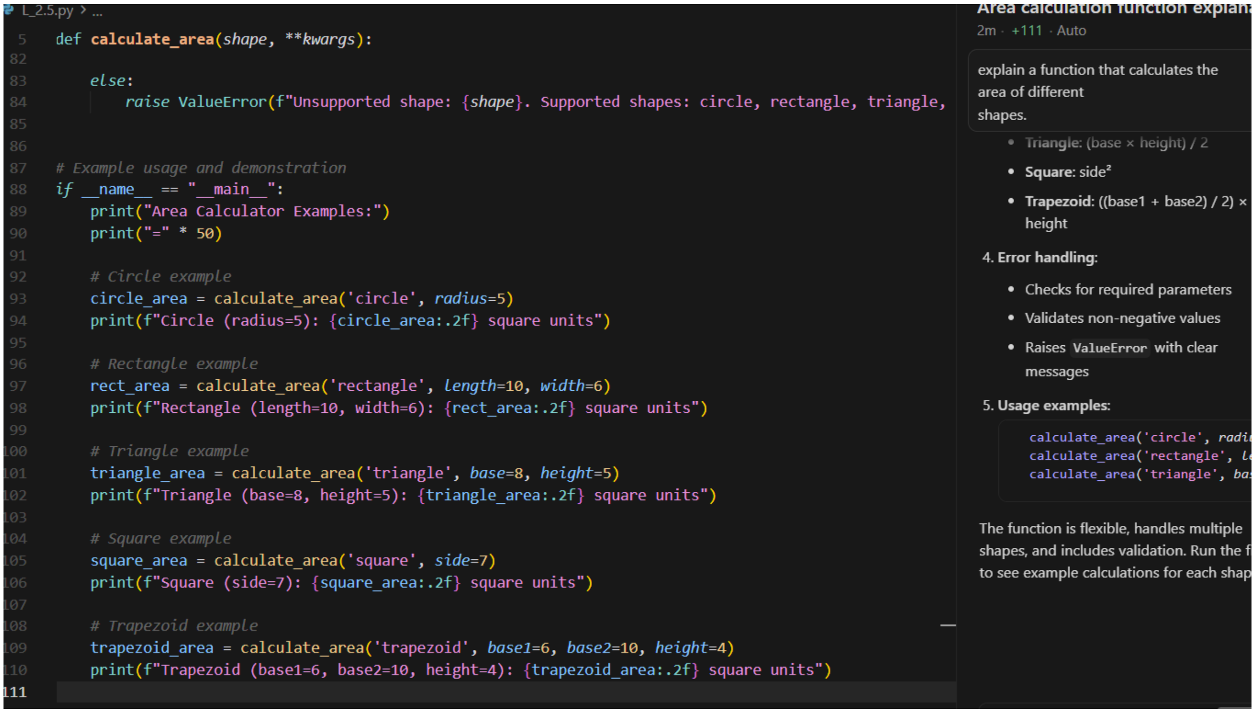
explain a function that calculates the
area of different
shapes.

- Uses if/elif to select the correct formula

3. **Area formulas:**
   - **Circle:** π × radius²
   - **Rectangle:** length × width
   - **Triangle:** (base × height) / 2
   - **Square:** side²
   - **Trapezoid:** ((base1 + base2) / 2) × height
4. **Error handling:**
   - Checks for required parameters
   - Validates non-negative values
   - Raises `ValueError` with clear messages
5. **Usage examples:**

```
calculate_area('circle', radius=
calculate_area('rectangle', leng
calculate_area('triangle', base=
```

The function is flexible, handles multiple shapes, and includes validation. Run the file



```python
def calculate_area(shape, **kwargs):

    else:
        raise ValueError(f"Unsupported shape: {shape}. Supported shapes: circle, rectangle, triangle,


# Example usage and demonstration
if __name__ == "__main__":
    print("Area Calculator Examples:")
    print("=" * 50)

    # Circle example
    circle_area = calculate_area('circle', radius=5)
    print(f"Circle (radius=5): {circle_area:.2f} square units")

    # Rectangle example
    rect_area = calculate_area('rectangle', length=10, width=6)
    print(f"Rectangle (length=10, width=6): {rect_area:.2f} square units")

    # Triangle example
    triangle_area = calculate_area('triangle', base=8, height=5)
    print(f"Triangle (base=8, height=5): {triangle_area:.2f} square units")

    # Square example
    square_area = calculate_area('square', side=7)
    print(f"Square (side=7): {square_area:.2f} square units")

    # Trapezoid example
    trapezoid_area = calculate_area('trapezoid', base1=6, base2=10, height=4)
    print(f"Trapezoid (base1=6, base2=10, height=4): {trapezoid_area:.2f} square units")
```

Area calculation function explana

2m · +111 · Auto

explain a function that calculates the
area of different
shapes.

- **Triangle:** (base × height) / 2
- **Square:** side²
- **Trapezoid:** ((base1 + base2) / 2) × height

4. **Error handling:**
   - Checks for required parameters
   - Validates non-negative values
   - Raises `ValueError` with clear messages
5. **Usage examples:**

```
calculate_area('circle', radi
calculate_area('rectangle', l
calculate_area('triangle', ba
```

The function is flexible, handles multiple shapes, and includes validation. Run the f
to see example calculations for each shap

Task 4: Tool Comparison Reflection

❖ Scenario:

You must recommend an AI coding tool.

❖ Task:

Based on your work in this topic, compare Gemini, Copilot, and Cursor AI

for usability and code quality.

Tool Comparison Reflection – Gemini vs GitHub Copilot vs Cursor AI

Based on the code generated and explanations provided by **Google Gemini**, **GitHub Copilot**, and **Cursor AI**, clear differences can be observed in usability, robustness, and code quality.

Google Gemini generated clean and readable code with a strong focus on simplicity and beginner-friendly explanations. It used standard libraries (such as math) and returned user-friendly error messages as strings. This approach makes Gemini well-suited for teaching concepts and onboarding junior developers. However, its implementation supports fewer shapes and relies on returning error messages instead of raising exceptions, which limits robustness in large-scale applications.

Cursor AI produced a more comprehensive and production-oriented solution. The generated code supports multiple shapes, includes detailed docstrings, input validation, and proper exception handling using ValueError. Cursor AI also demonstrated better software engineering practices such as defensive programming and clear documentation. This makes Cursor AI highly suitable for refactoring, optimization, and real-world application development.

GitHub Copilot, based on prior observations, excels at rapid inline code generation within the IDE. While it is extremely efficient for writing working code quickly, it provides minimal explanation and documentation by default. This makes it ideal for experienced developers but less effective for learning or code review scenarios.

Conclusion

Gemini is best suited for learning and explanation-focused tasks, Copilot is ideal for fast development, and Cursor AI offers the most balanced solution with high-quality code, strong validation, and clear documentation. Therefore, **Cursor AI is the most suitable recommendation for professional and scalable development**, while Gemini is preferable for educational use.