SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju	
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week1 – Wednesday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number:1.3(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*

1

Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI

Lab Objectives:

- To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab.
- To understand and use Cursor AI for code generation, explanation, and refactoring.
- To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI.
- To perform code optimization and documentation using AI tools.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Generate Python code using Google Gemini in Google Colab.
- Analyze the effectiveness of code explanations and suggestions by Gemini.
- Set up and use Cursor AI for AI-powered coding assistance.
- Evaluate and refactor code using Cursor AI features.
- Compare AI tool behavior and code quality across different platforms.

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.

- **Expected Output:**
- Original and improved code

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.

- **Expected Output:**
  ⍰ Code
  ⍰ Explanation

Task 3: Prompt Sensitivity Experiment

- **Scenario:**

Week1 - Monday

Lab Assignment - 2.5

Name: Ch. Sushruta Reddy

Roll Number: 2303A51219

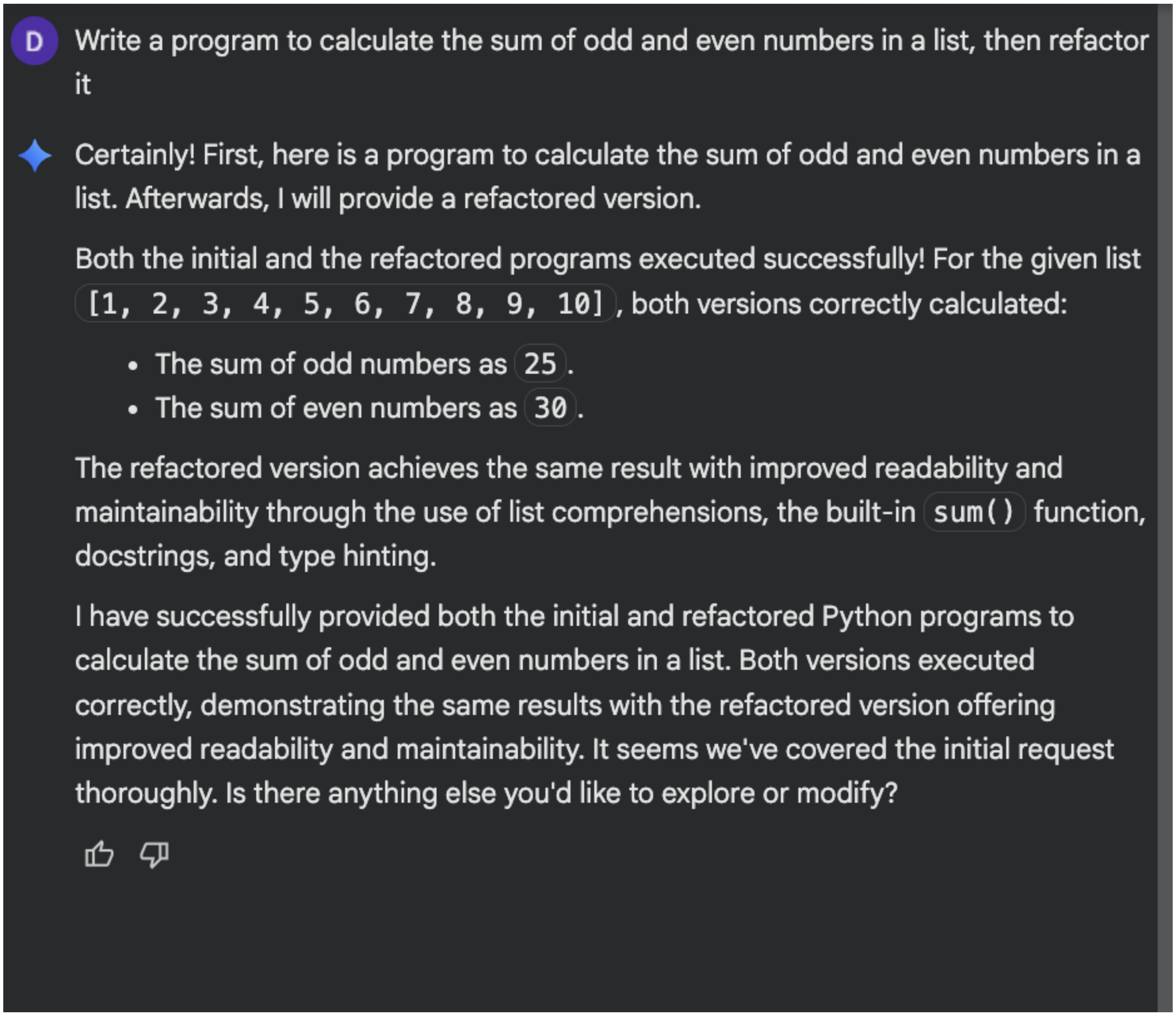
Batch - 04

AI Assisted Coding

09-01-2026

Task 1:

Prompt:



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

Certainly! First, here is a program to calculate the sum of odd and even numbers in a list. Afterwards, I will provide a refactored version.

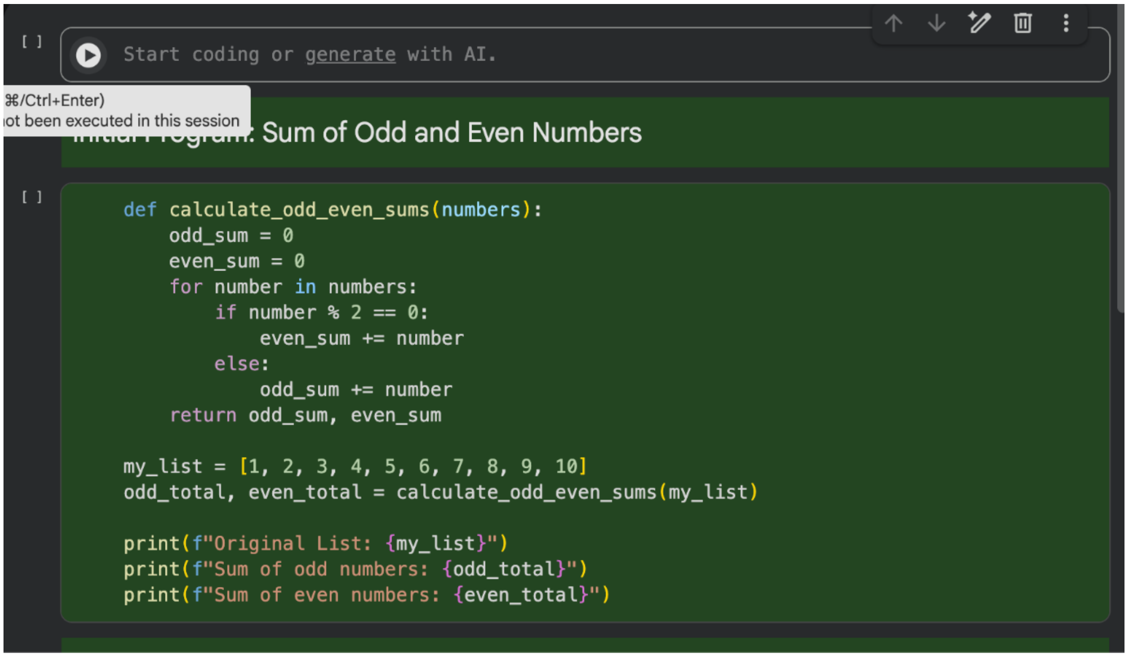
Both the initial and the refactored programs executed successfully! For the given list `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`, both versions correctly calculated:

- The sum of odd numbers as `25`.
- The sum of even numbers as `30`.

The refactored version achieves the same result with improved readability and maintainability through the use of list comprehensions, the built-in `sum()` function, docstrings, and type hinting.

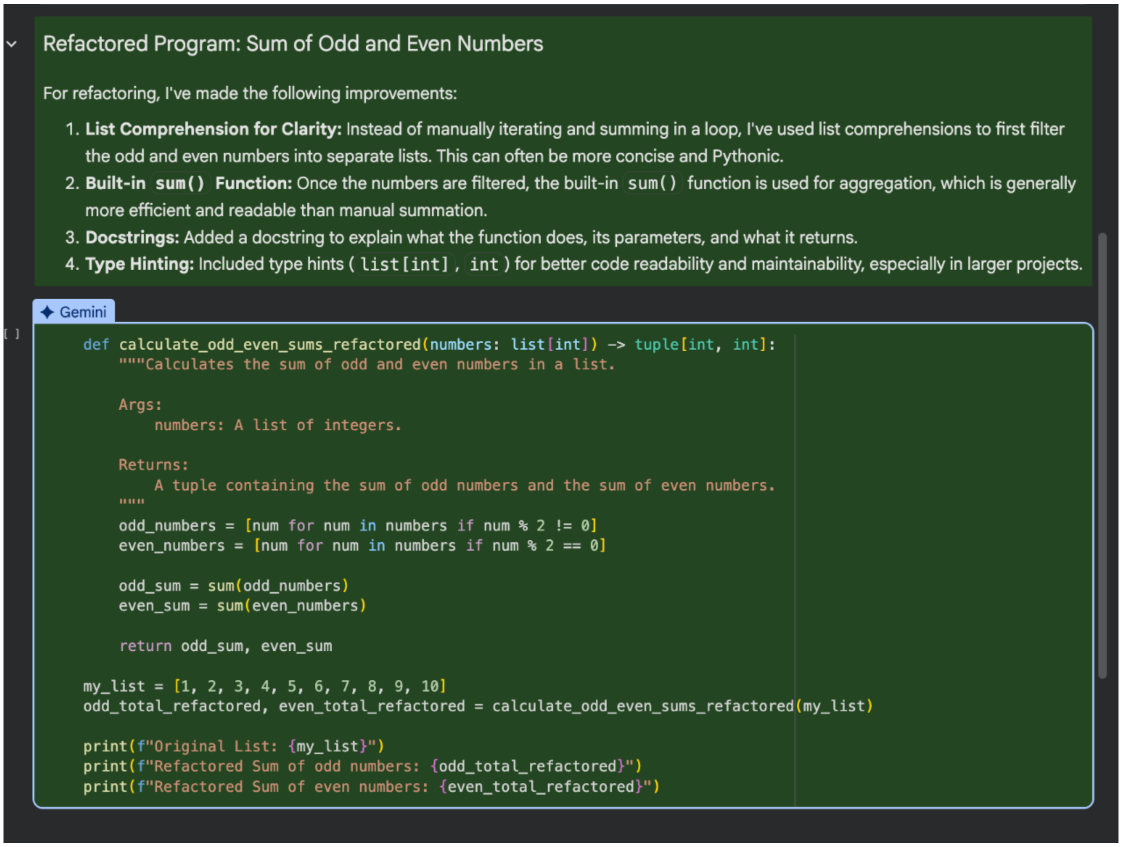
I have successfully provided both the initial and refactored Python programs to calculate the sum of odd and even numbers in a list. Both versions executed correctly, demonstrating the same results with the refactored version offering improved readability and maintainability. It seems we've covered the initial request thoroughly. Is there anything else you'd like to explore or modify?

Code:


[ ]
Start coding or generate with AI.
⌘/Ctrl+Enter)
ot been executed in this session
[ ]

Initial Program: Sum of Odd and Even Numbers

```python
def calculate_odd_even_sums(numbers):
    odd_sum = 0
    even_sum = 0
    for number in numbers:
        if number % 2 == 0:
            even_sum += number
        else:
            odd_sum += number
    return odd_sum, even_sum

my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
odd_total, even_total = calculate_odd_even_sums(my_list)

print(f"Original List: {my_list}")
print(f"Sum of odd numbers: {odd_total}")
print(f"Sum of even numbers: {even_total}")
```



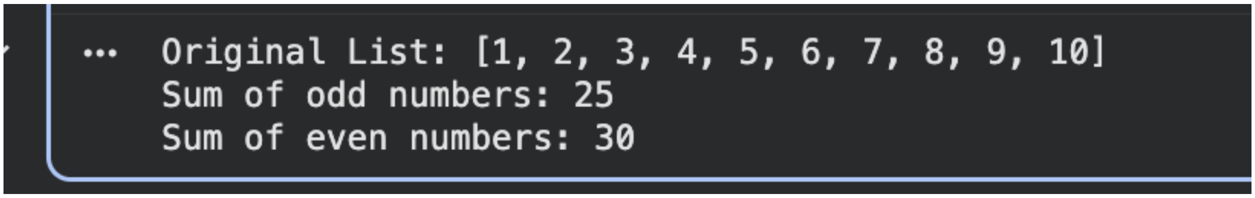
Refactored Program: Sum of Odd and Even Numbers

For refactoring, I've made the following improvements:

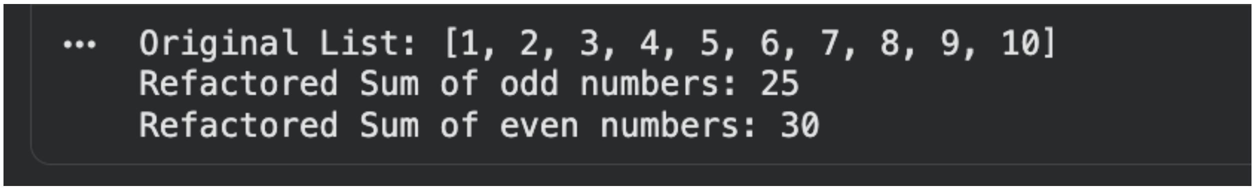
1. **List Comprehension for Clarity:** Instead of manually iterating and summing in a loop, I've used list comprehensions to first filter the odd and even numbers into separate lists. This can often be more concise and Pythonic.
2. **Built-in `sum()` Function:** Once the numbers are filtered, the built-in `sum()` function is used for aggregation, which is generally more efficient and readable than manual summation.
3. **Docstrings:** Added a docstring to explain what the function does, its parameters, and what it returns.
4. **Type Hinting:** Included type hints (`list[int]` , `int`) for better code readability and maintainability, especially in larger projects.

```python
def calculate_odd_even_sums_refactored(numbers: list[int]) -> tuple[int, int]:
    """Calculates the sum of odd and even numbers in a list.

    Args:
        numbers: A list of integers.

    Returns:
        A tuple containing the sum of odd numbers and the sum of even numbers.
    """
    odd_numbers = [num for num in numbers if num % 2 != 0]
    even_numbers = [num for num in numbers if num % 2 == 0]

    odd_sum = sum(odd_numbers)
    even_sum = sum(even_numbers)

    return odd_sum, even_sum

my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
odd_total_refactored, even_total_refactored = calculate_odd_even_sums_refactored(my_list)

print(f"Original List: {my_list}")
print(f"Refactored Sum of odd numbers: {odd_total_refactored}")
print(f"Refactored Sum of even numbers: {even_total_refactored}")
```

Output:



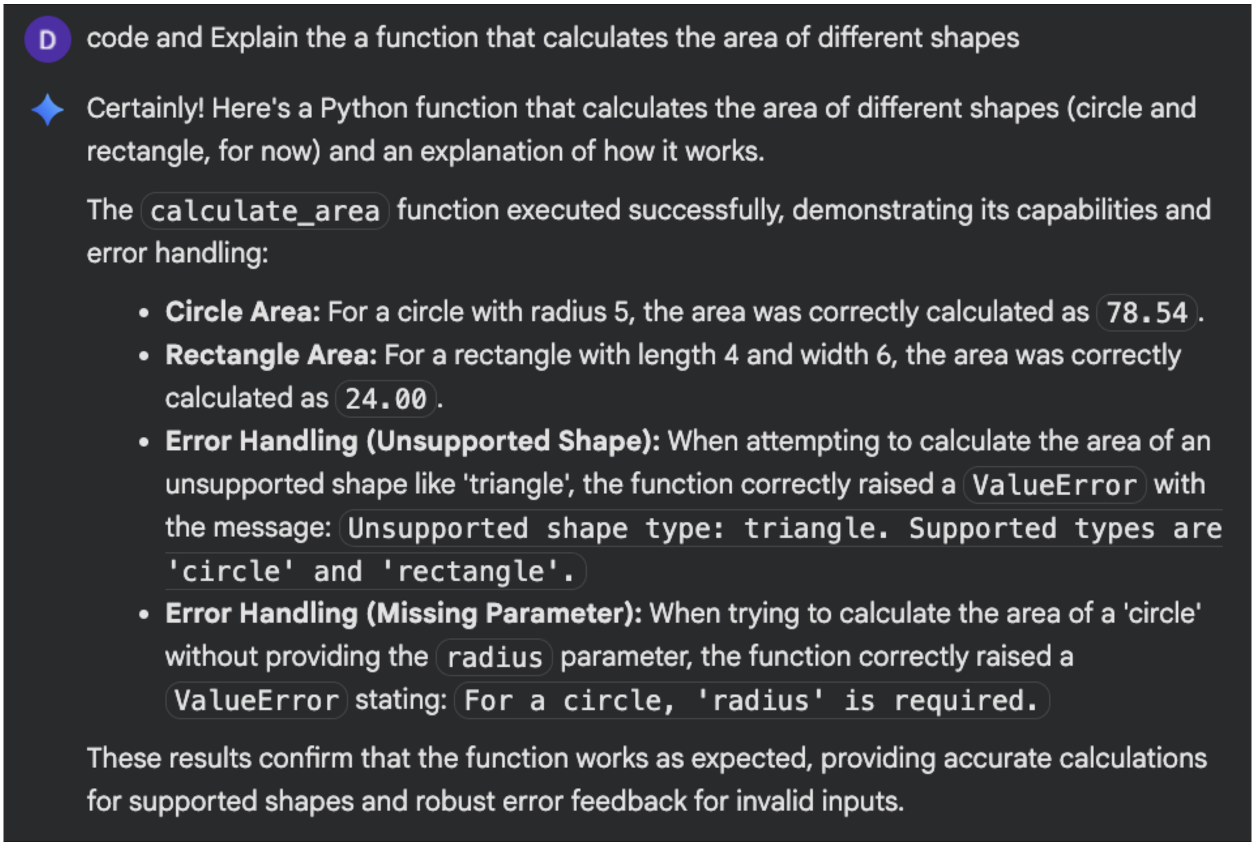
```
Original List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of even numbers: 30
```



```
Original List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Refactored Sum of odd numbers: 25
Refactored Sum of even numbers: 30
```

Task 2:

Prompt:



code and Explain the a function that calculates the area of different shapes

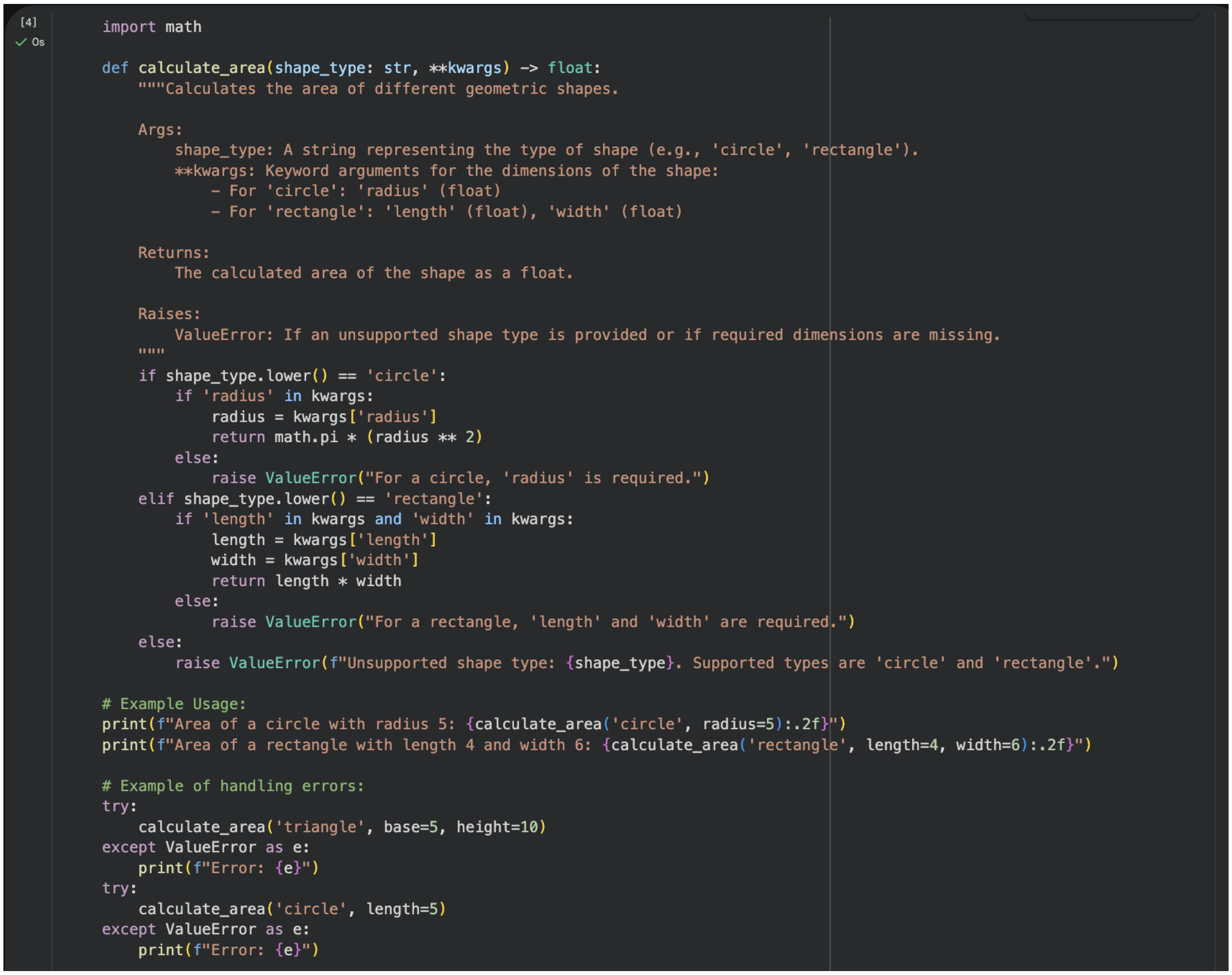
Certainly! Here's a Python function that calculates the area of different shapes (circle and rectangle, for now) and an explanation of how it works.

The `calculate_area` function executed successfully, demonstrating its capabilities and error handling:

- **Circle Area:** For a circle with radius 5, the area was correctly calculated as `78.54`.
- **Rectangle Area:** For a rectangle with length 4 and width 6, the area was correctly calculated as `24.00`.
- **Error Handling (Unsupported Shape):** When attempting to calculate the area of an unsupported shape like 'triangle', the function correctly raised a `ValueError` with the message: `Unsupported shape type: triangle. Supported types are 'circle' and 'rectangle'.`
- **Error Handling (Missing Parameter):** When trying to calculate the area of a 'circle' without providing the `radius` parameter, the function correctly raised a `ValueError` stating: `For a circle, 'radius' is required.`

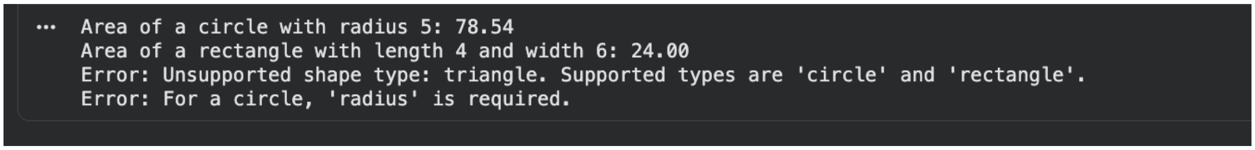
These results confirm that the function works as expected, providing accurate calculations for supported shapes and robust error feedback for invalid inputs.

Code:



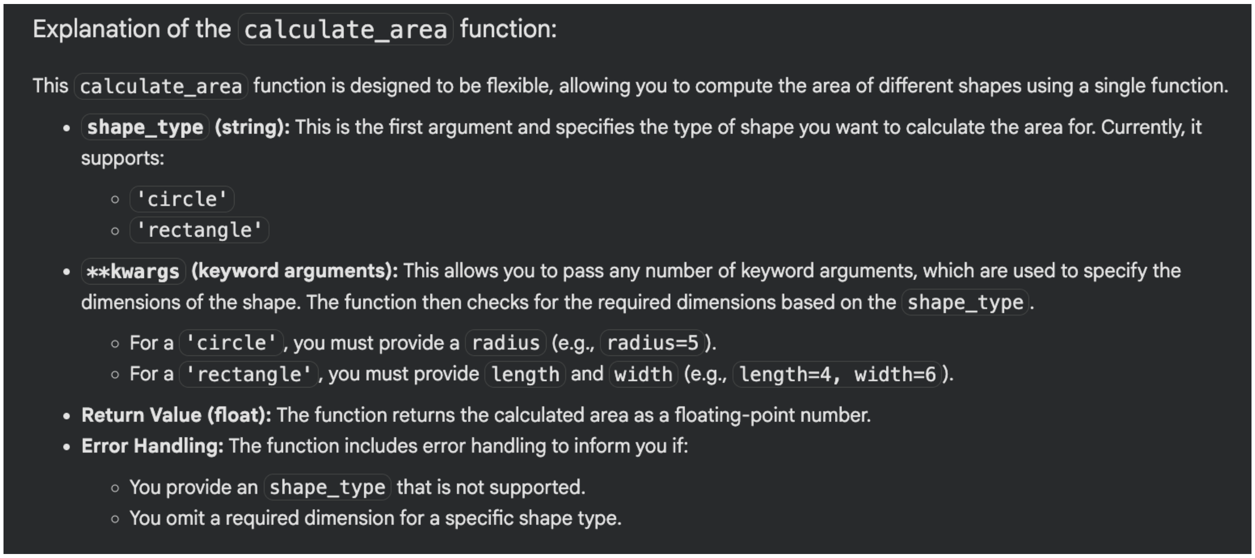
```python
import math

def calculate_area(shape_type: str, **kwargs) -> float:
    """Calculates the area of different geometric shapes.

    Args:
        shape_type: A string representing the type of shape (e.g., 'circle', 'rectangle').
        **kwargs: Keyword arguments for the dimensions of the shape:
            - For 'circle': 'radius' (float)
            - For 'rectangle': 'length' (float), 'width' (float)

    Returns:
        The calculated area of the shape as a float.

    Raises:
        ValueError: If an unsupported shape type is provided or if required dimensions are missing.
    """
    if shape_type.lower() == 'circle':
        if 'radius' in kwargs:
            radius = kwargs['radius']
            return math.pi * (radius ** 2)
        else:
            raise ValueError("For a circle, 'radius' is required.")
    elif shape_type.lower() == 'rectangle':
        if 'length' in kwargs and 'width' in kwargs:
            length = kwargs['length']
            width = kwargs['width']
            return length * width
        else:
            raise ValueError("For a rectangle, 'length' and 'width' are required.")
    else:
        raise ValueError(f"Unsupported shape type: {shape_type}. Supported types are 'circle' and 'rectangle'.")

# Example Usage:
print(f"Area of a circle with radius 5: {calculate_area('circle', radius=5):.2f}")
print(f"Area of a rectangle with length 4 and width 6: {calculate_area('rectangle', length=4, width=6):.2f}")

# Example of handling errors:
try:
    calculate_area('triangle', base=5, height=10)
except ValueError as e:
    print(f"Error: {e}")
try:
    calculate_area('circle', length=5)
except ValueError as e:
    print(f"Error: {e}")
```

Output:



```
Area of a circle with radius 5: 78.54
Area of a rectangle with length 4 and width 6: 24.00
Error: Unsupported shape type: triangle. Supported types are 'circle' and 'rectangle'.
Error: For a circle, 'radius' is required.
```

Explanation:



Explanation of the `calculate_area` function:

This `calculate_area` function is designed to be flexible, allowing you to compute the area of different shapes using a single function.

- **`shape_type` (string):** This is the first argument and specifies the type of shape you want to calculate the area for. Currently, it supports:
  - `'circle'`
  - `'rectangle'`
- **`**kwargs` (keyword arguments):** This allows you to pass any number of keyword arguments, which are used to specify the dimensions of the shape. The function then checks for the required dimensions based on the `shape_type`.
  - For a `'circle'`, you must provide a `radius` (e.g., `radius=5`).
  - For a `'rectangle'`, you must provide `length` and `width` (e.g., `length=4, width=6`).
- **Return Value (float):** The function returns the calculated area as a floating-point number.
- **Error Handling:** The function includes error handling to inform you if:
  - You provide an `shape_type` that is not supported.
  - You omit a required dimension for a specific shape type.

Task 3:

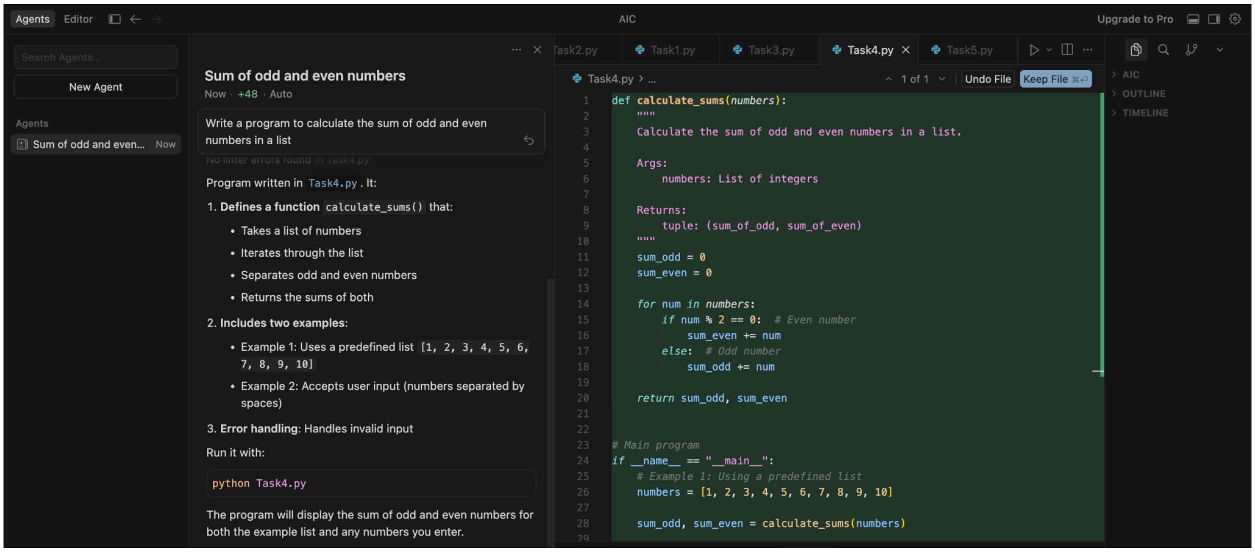
The Problem : To calculate the sum of odd and even numbers in a list

Prompt List:

1. Write a program to calculate the sum of odd and even numbers in a list
2. Write a program that takes a list of integers as input and calculates the sum of all even numbers and the sum of all odd numbers separately.

3. Create a program to find the sum of odd numbers and the sum of even numbers in a given list.
4. Develop a program that computes the sum of odd and even numbers from a list of integers.
5. You are given a list of numbers. Write a program that goes through each number and adds even numbers to one total and odd numbers to another total.

Codes:



Sum of odd and even numbers

Now · +48 · Auto

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

No linter errors found in Task4.py

Program written in `Task4.py`. It:

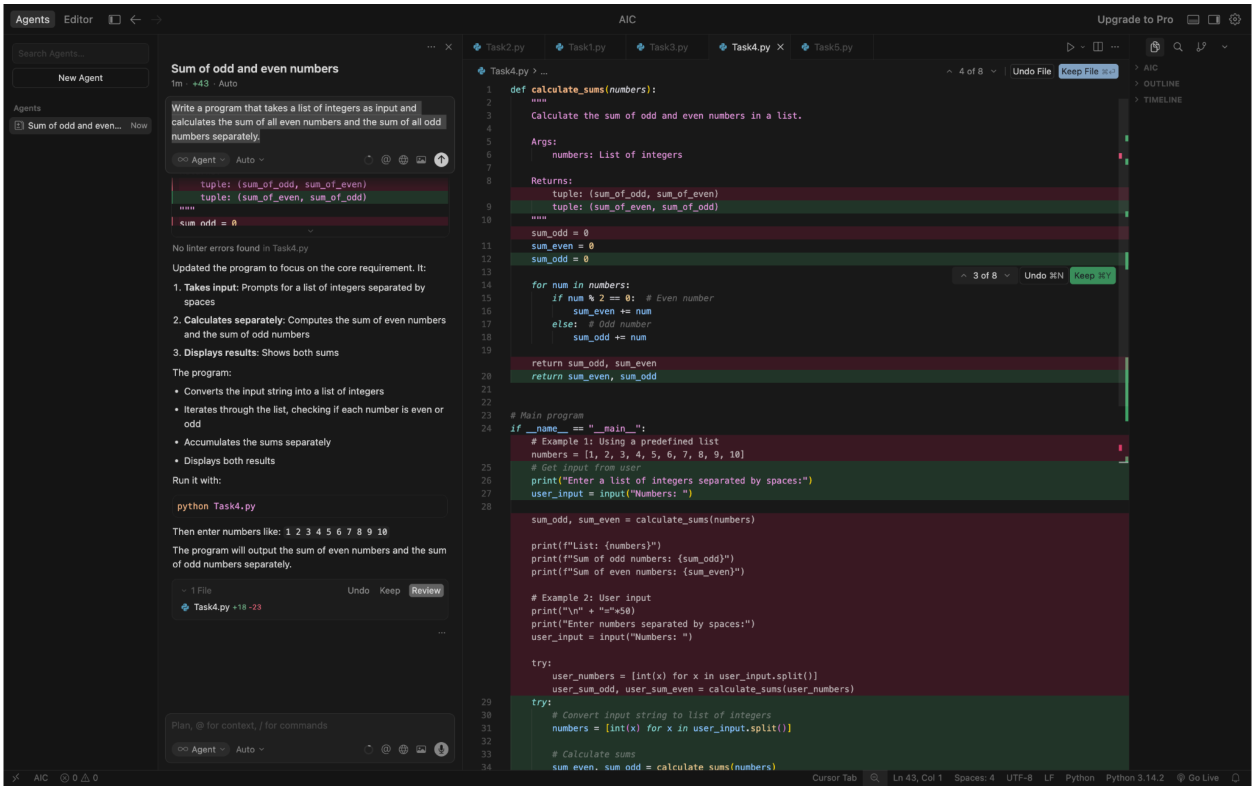
1. **Defines a function** `calculate_sums()` that:
    - Takes a list of numbers
    - Iterates through the list
    - Separates odd and even numbers
    - Returns the sums of both
2. **Includes two examples**:
    - Example 1: Uses a predefined list `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
    - Example 2: Accepts user input (numbers separated by spaces)
3. **Error handling**: Handles invalid input

Run it with:

```
python Task4.py
```

The program will display the sum of odd and even numbers for both the example list and any numbers you enter.

```python
def calculate_sums(numbers):
    """
    Calculate the sum of odd and even numbers in a list.

    Args:
        numbers: List of integers

    Returns:
        tuple: (sum_of_odd, sum_of_even)
    """
    sum_odd = 0
    sum_even = 0

    for num in numbers:
        if num % 2 == 0:  # Even number
            sum_even += num
        else:  # Odd number
            sum_odd += num

    return sum_odd, sum_even


# Main program
if __name__ == "__main__":
    # Example 1: Using a predefined list
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    sum_odd, sum_even = calculate_sums(numbers)
```



Sum of odd and even numbers

1m · +43 · Auto

Write a program that takes a list of integers as input and calculates the sum of all even numbers and the sum of all odd numbers separately.

```
        tuple: (sum_of_odd, sum_of_even)
        tuple: (sum_of_even, sum_of_odd)
    """
    sum_odd = 0
```

No linter errors found in Task4.py

Updated the program to focus on the core requirement. It:

1. **Takes input:** Prompts for a list of integers separated by spaces
2. **Calculates separately:** Computes the sum of even numbers and the sum of odd numbers
3. **Displays results:** Shows both sums

The program:

- Converts the input string into a list of integers
- Iterates through the list, checking if each number is even or odd
- Accumulates the sums separately
- Displays both results

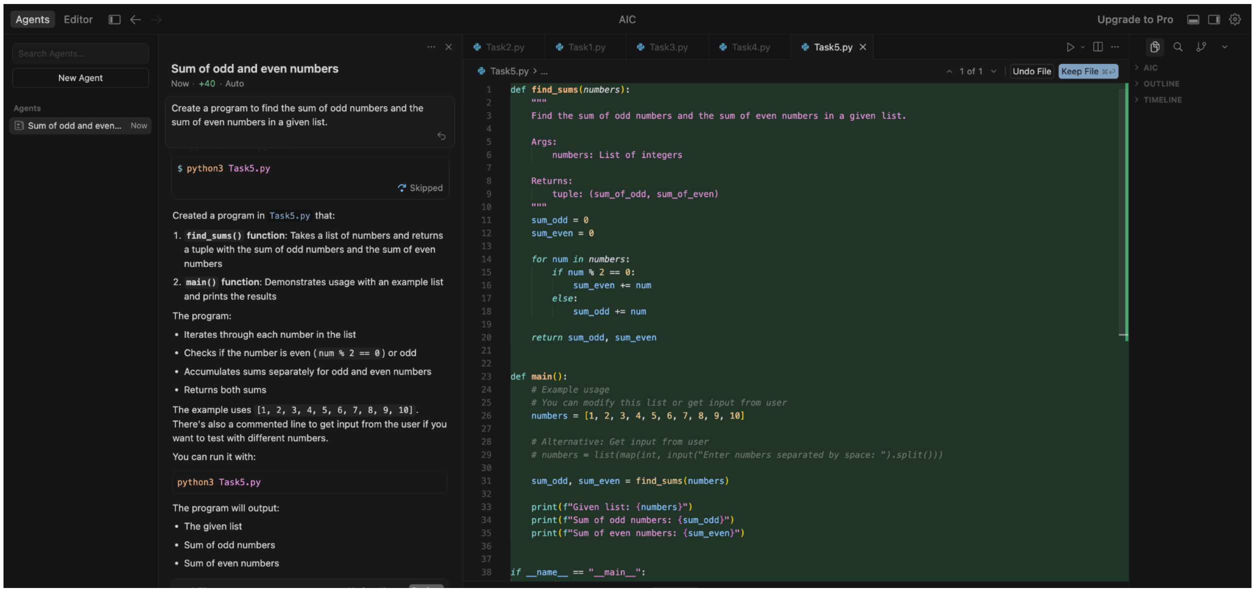
Run it with:

```
python Task4.py
```

Then enter numbers like: `1 2 3 4 5 6 7 8 9 10`

The program will output the sum of even numbers and the sum of odd numbers separately.

```python
def calculate_sums(numbers):
    """
    Calculate the sum of odd and even numbers in a list.

    Args:
        numbers: List of integers

    Returns:
        tuple: (sum_of_odd, sum_of_even)
        tuple: (sum_of_even, sum_of_odd)
    """
    sum_odd = 0
    sum_even = 0
    sum_odd = 0

    for num in numbers:
        if num % 2 == 0:  # Even number
            sum_even += num
        else:  # Odd number
            sum_odd += num

    return sum_odd, sum_even
    return sum_even, sum_odd


# Main program
if __name__ == "__main__":
    # Example 1: Using a predefined list
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    # Get input from user
    print("Enter a list of integers separated by spaces:")
    user_input = input("Numbers: ")

    sum_odd, sum_even = calculate_sums(numbers)

    print(f"List: {numbers}")
    print(f"Sum of odd numbers: {sum_odd}")
    print(f"Sum of even numbers: {sum_even}")

    # Example 2: User input
    print("\n" + "="*50)
    print("Enter numbers separated by spaces:")
    user_input = input("Numbers: ")

    try:
        user_numbers = [int(x) for x in user_input.split()]
        user_sum_odd, user_sum_even = calculate_sums(user_numbers)
    try:
        # Convert input string to list of integers
        numbers = [int(x) for x in user_input.split()]

        # Calculate sums
        sum_even, sum_odd = calculate_sums(numbers)
```



Sum of odd and even numbers

Now · +40 · Auto

Create a program to find the sum of odd numbers and the sum of even numbers in a given list.

```
$ python3 Task5.py
```

Created a program in `Task5.py` that:

1. `find_sums()` **function**: Takes a list of numbers and returns a tuple with the sum of odd numbers and the sum of even numbers
2. `main()` **function**: Demonstrates usage with an example list and prints the results

The program:

- Iterates through each number in the list
- Checks if the number is even (`num % 2 == 0`) or odd
- Accumulates sums separately for odd and even numbers
- Returns both sums

The example uses `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`. There's also a commented line to get input from the user if you want to test with different numbers.

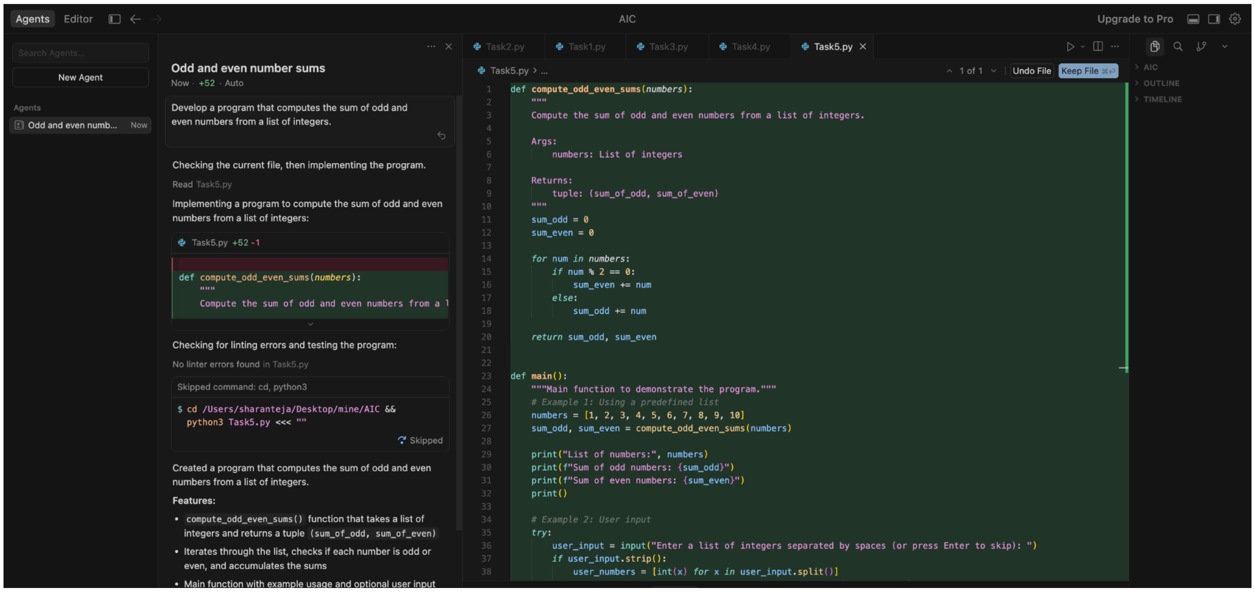
You can run it with:

```
python3 Task5.py
```

The program will output:

- The given list
- Sum of odd numbers
- Sum of even numbers

```python
def find_sums(numbers):
    """
    Find the sum of odd numbers and the sum of even numbers in a given list.

    Args:
        numbers: List of integers

    Returns:
        tuple: (sum_of_odd, sum_of_even)
    """
    sum_odd = 0
    sum_even = 0

    for num in numbers:
        if num % 2 == 0:
            sum_even += num
        else:
            sum_odd += num

    return sum_odd, sum_even


def main():
    # Example usage
    # You can modify this list or get input from user
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    # Alternative: Get input from user
    # numbers = list(map(int, input("Enter numbers separated by space: ").split()))

    sum_odd, sum_even = find_sums(numbers)

    print(f"Given list: {numbers}")
    print(f"Sum of odd numbers: {sum_odd}")
    print(f"Sum of even numbers: {sum_even}")


if __name__ == "__main__":
```



Odd and even number sums

Now · +52 · Auto

Develop a program that computes the sum of odd and even numbers from a list of integers.

Checking the current file, then implementing the program.

Read Task5.py

Implementing a program to compute the sum of odd and even numbers from a list of integers:

Task5.py +52 -1

```
def compute_odd_even_sums(numbers):
    """
    Compute the sum of odd and even numbers from a l
```

Checking for linting errors and testing the program:

No linter errors found in Task5.py

Skipped command: cd, python3

```
$ cd /Users/sharanteja/Desktop/mine/AIC &&
  python3 Task5.py <<< ""
```

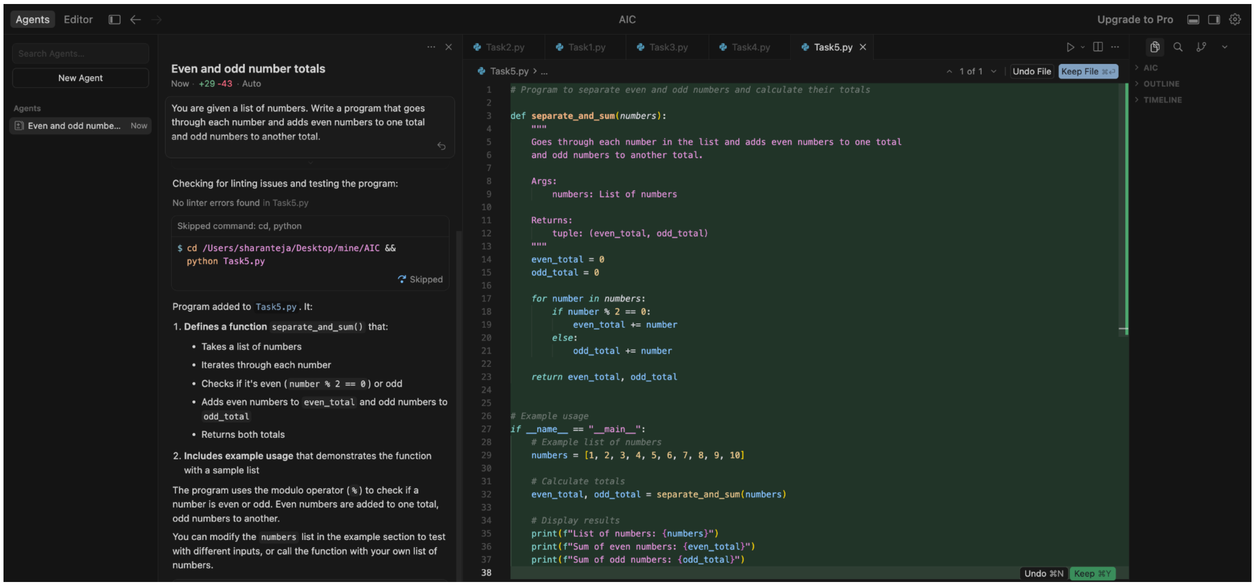
Skipped

Created a program that computes the sum of odd and even numbers from a list of integers.

Features:

- `compute_odd_even_sums()` function that takes a list of integers and returns a tuple `(sum_of_odd, sum_of_even)`
- Iterates through the list, checks if each number is odd or even, and accumulates the sums
- Main function with example usage and optional user input

```python
def compute_odd_even_sums(numbers):
    """
    Compute the sum of odd and even numbers from a list of integers.

    Args:
        numbers: List of integers

    Returns:
        tuple: (sum_of_odd, sum_of_even)
    """
    sum_odd = 0
    sum_even = 0

    for num in numbers:
        if num % 2 == 0:
            sum_even += num
        else:
            sum_odd += num

    return sum_odd, sum_even


def main():
    """Main function to demonstrate the program."""
    # Example 1: Using a predefined list
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    sum_odd, sum_even = compute_odd_even_sums(numbers)

    print("List of numbers:", numbers)
    print(f"Sum of odd numbers: {sum_odd}")
    print(f"Sum of even numbers: {sum_even}")
    print()

    # Example 2: User input
    try:
        user_input = input("Enter a list of integers separated by spaces (or press Enter to skip): ")
        if user_input.strip():
            user_numbers = [int(x) for x in user_input.split()]
```



Even and odd number totals

Now · +29 -43 · Auto

You are given a list of numbers. Write a program that goes through each number and adds even numbers to one total and odd numbers to another total.

Checking for linting issues and testing the program:

No linter errors found in Task5.py

Skipped command: cd, python

```
$ cd /Users/sharanteja/Desktop/mine/AIC &&
  python Task5.py
```

Skipped

Program added to `Task5.py`. It:

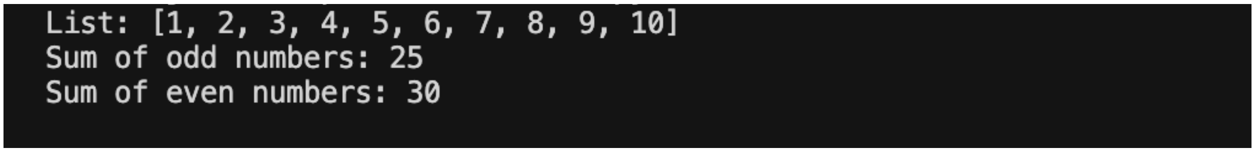
1. **Defines a function** `separate_and_sum()` that:
   - Takes a list of numbers
   - Iterates through each number
   - Checks if it's even (`number % 2 == 0`) or odd
   - Adds even numbers to `even_total` and odd numbers to `odd_total`
   - Returns both totals
2. **Includes example usage** that demonstrates the function with a sample list

The program uses the modulo operator (`%`) to check if a number is even or odd. Even numbers are added to one total, odd numbers to another.

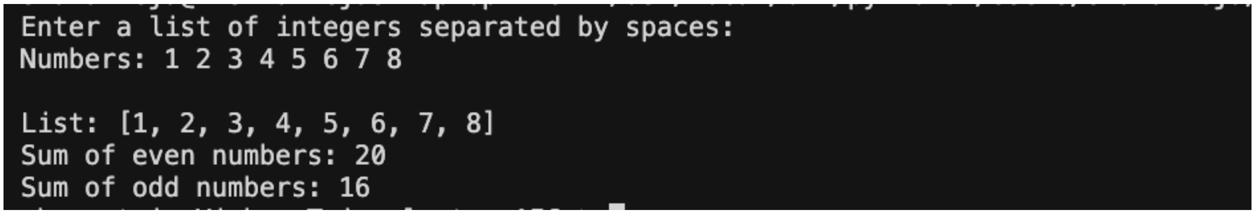
You can modify the `numbers` list in the example section to test with different inputs, or call the function with your own list of numbers.

```python
# Program to separate even and odd numbers and calculate their totals

def separate_and_sum(numbers):
    """
    Goes through each number in the list and adds even numbers to one total
    and odd numbers to another total.

    Args:
        numbers: List of numbers

    Returns:
        tuple: (even_total, odd_total)
    """
    even_total = 0
    odd_total = 0

    for number in numbers:
        if number % 2 == 0:
            even_total += number
        else:
            odd_total += number

    return even_total, odd_total


# Example usage
if __name__ == "__main__":
    # Example list of numbers
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    # Calculate totals
    even_total, odd_total = separate_and_sum(numbers)

    # Display results
    print(f"List of numbers: {numbers}")
    print(f"Sum of even numbers: {even_total}")
    print(f"Sum of odd numbers: {odd_total}")

```

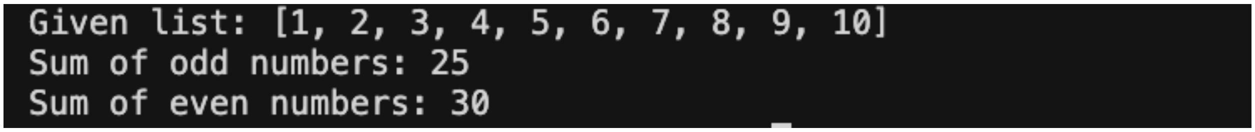
Outputs:



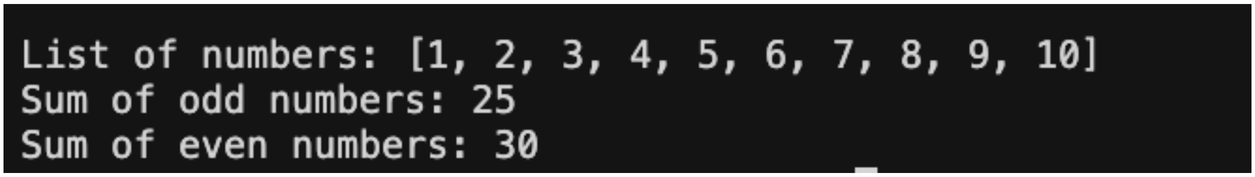
```
List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of even numbers: 30
```



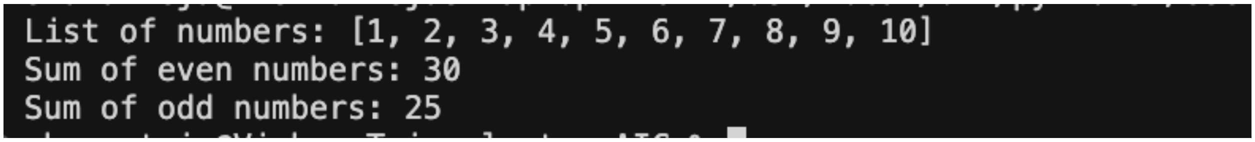
```
Enter a list of integers separated by spaces:
Numbers: 1 2 3 4 5 6 7 8

List: [1, 2, 3, 4, 5, 6, 7, 8]
Sum of even numbers: 20
Sum of odd numbers: 16
```



```
Given list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of even numbers: 30
```

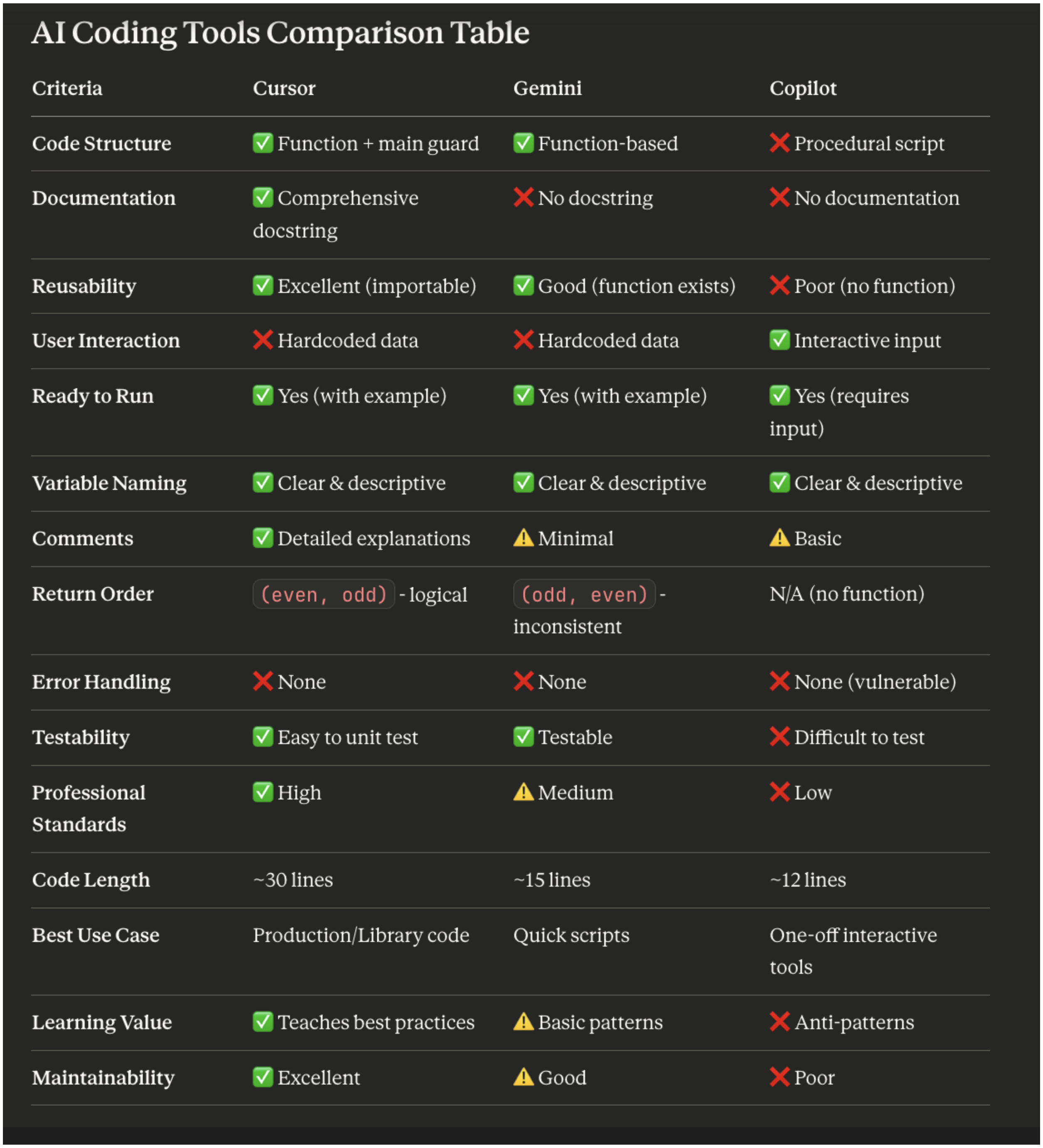


```
List of numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of even numbers: 30
```



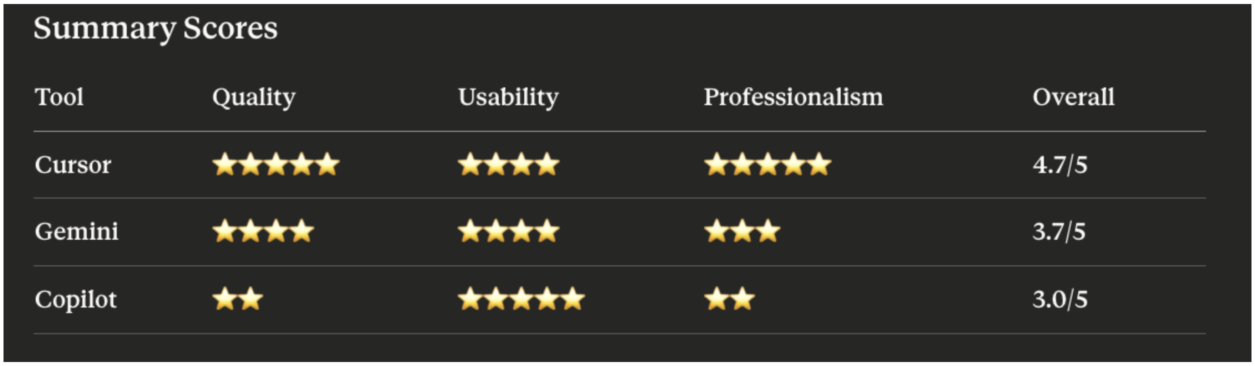
```
List of numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of even numbers: 30
Sum of odd numbers: 25
```

Task 4:



AI Coding Tools Comparison Table

Criteria	Cursor	Gemini	Copilot
Code Structure	✅ Function + main guard	✅ Function-based	❌ Procedural script
Documentation	✅ Comprehensive docstring	❌ No docstring	❌ No documentation
Reusability	✅ Excellent (importable)	✅ Good (function exists)	❌ Poor (no function)
User Interaction	❌ Hardcoded data	❌ Hardcoded data	✅ Interactive input
Ready to Run	✅ Yes (with example)	✅ Yes (with example)	✅ Yes (requires input)
Variable Naming	✅ Clear & descriptive	✅ Clear & descriptive	✅ Clear & descriptive
Comments	✅ Detailed explanations	⚠️ Minimal	⚠️ Basic
Return Order	`(even, odd)` - logical	`(odd, even)` - inconsistent	N/A (no function)
Error Handling	❌ None	❌ None	❌ None (vulnerable)
Testability	✅ Easy to unit test	✅ Testable	❌ Difficult to test
Professional Standards	✅ High	⚠️ Medium	❌ Low
Code Length	~30 lines	~15 lines	~12 lines
Best Use Case	Production/Library code	Quick scripts	One-off interactive tools
Learning Value	✅ Teaches best practices	⚠️ Basic patterns	❌ Anti-patterns
Maintainability	✅ Excellent	⚠️ Good	❌ Poor



Summary Scores

Tool	Quality	Usability	Professionalism	Overall
Cursor	★★★★★	★★★★	★★★★★	4.7/5
Gemini	★★★★	★★★★	★★★	3.7/5
Copilot	★★	★★★★★	★★	3.0/5

Short Reflection:

Each AI tool demonstrated different priorities:

- **Cursor** focused on **professional software engineering practices** - treating this as production code with documentation, modularity, and maintainability
- **Gemini** balanced **simplicity and functionality** - providing clean code without over-engineering
- **Copilot** prioritized **immediate usability** - generating a script ready to execute right away, but sacrificing long-term quality

The differences reveal their design philosophies: Cursor seems optimized for professional development workflows, Gemini for quick functional solutions, and Copilot for rapid prototyping and scripting.