

AI-Assisted Coding

Assignment 2.5

HTNo:2303A52299

Name:A.Sai Shrehan

Batch:45

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:

❖ improved code

Output:

The screenshot displays a code editor with a Python script named `assignment_2.5.py`. The script defines a `main` function that takes user input, splits it into a list of integers, and calculates the sum of odd and even numbers. The code is well-documented with comments and docstrings. The terminal output shows the program running successfully, displaying the sum of odd numbers (16), the sum of even numbers (20), and the total sum (36). The AI-assisted coding interface is visible on the right side, showing suggestions for refactoring the code and a test result for the input `1 2 3 4 5 6 7 8 9 10`.

```
def main():  
    # Convert input to list of integers  
    numbers = [int(x) for x in user_input.split()]  
  
    # calculate sums  
    sum_odd, sum_even = calculate_odd_even_sums(numbers)  
  
    # Display results  
    print("\n" + "-" * 50)  
    print(f"Numbers entered: {numbers}")  
    print("-" * 50)  
    print(f"Sum of odd numbers: {sum_odd}")  
    print(f"Sum of even numbers: {sum_even}")  
    print(f"Total sum: {sum_odd + sum_even}")  
    print("-" * 50)  
  
except ValueError:  
    print("Error: Please enter valid integers only.")
```

Sum of Odd and Even Numbers Calculator

Enter numbers separated by spaces: 1 2 3 4 5 6 7 8

Numbers entered: [1, 2, 3, 4, 5, 6, 7, 8]

Sum of odd numbers: 16
Sum of even numbers: 20
Total sum: 36

AI-assisted coding suggestions:

- cd "d:\AI assisted coding"; python assignment_2.5.py <<< "1 2 3 4 5 6 7 8 9 10"
- Used echo to pipe data instead of here-string syntax
- cd "d:\AI assisted coding"; echo "1 2 3 4 5 6 7 8 9 10" | python assignment_2.5.py
- Confirmed program functionality and tested results

Perfect! I've created a complete program in `assignment_2.5.py` that calculates the sum of odd and even numbers in a list.

Key features:

- `calculate_odd_even_sums()` function: Separates and sums odd and even numbers
- Input validation:** Handles invalid input gracefully
- Clear output:** Displays the numbers, odd sum, even sum, and total
- Well-documented:** Includes docstrings and comments

Test result: With input `1 2 3 4 5 6 7 8 9 10`:

- Odd sum: 25 (1+3+5+7+9)
- Even sum: 30 (2+4+6+8+10)
- Total: 55 ✓

The program is ready to use!

Task 2: Area Calculation Explanation

❖ Scenario:

You are onboarding a junior developer.

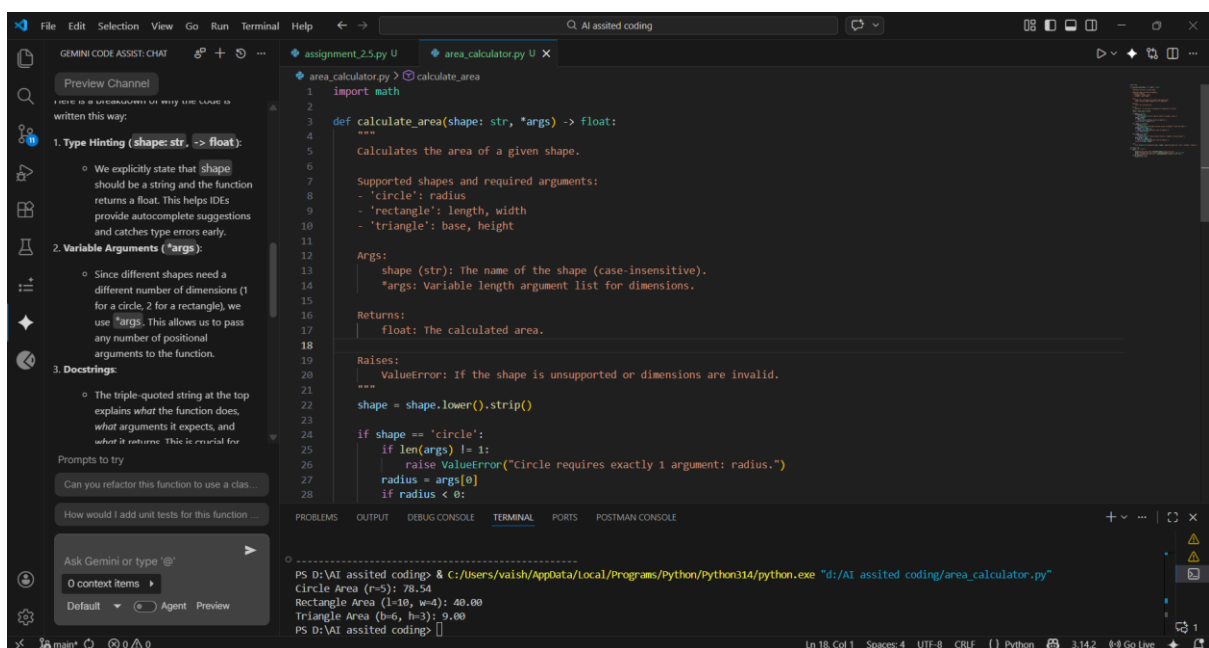
❖ Task:

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

❖ Expected Output:

➤ Code

Output:



The screenshot shows a code editor with a file named `area_calculator.py`. The code defines a function `calculate_area` that takes a shape name and dimensions as input and returns the area. The function uses a dictionary to map shapes to their respective area calculations. The code is as follows:

```
1 import math
2
3 def calculate_area(shape: str, *args) -> float:
4     """
5     Calculates the area of a given shape.
6
7     Supported shapes and required arguments:
8     - 'circle': radius
9     - 'rectangle': length, width
10    - 'triangle': base, height
11
12    Args:
13        shape (str): The name of the shape (case-insensitive).
14        *args: Variable length argument list for dimensions.
15
16    Returns:
17        float: The calculated area.
18
19    Raises:
20        ValueError: If the shape is unsupported or dimensions are invalid.
21    """
22    shape = shape.lower().strip()
23
24    if shape == 'circle':
25        if len(args) != 1:
26            raise ValueError("Circle requires exactly 1 argument: radius.")
27        radius = args[0]
28        if radius < 0:
```

The left sidebar shows a chat window with the following prompts:

- 1. **Type Hinting** (`shape: str, => float`):
 - We explicitly state that `shape` should be a string and the function returns a float. This helps IDEs provide autocomplete suggestions and catches type errors early.
- 2. **Variable Arguments** (`*args`):
 - Since different shapes need a different number of dimensions (1 for a circle, 2 for a rectangle), we use `*args`. This allows us to pass any number of positional arguments to the function.
- 3. **Docstrings**:
 - The triple-quoted string at the top explains what the function does, what arguments it expects, and what it returns. This is essential for

The bottom terminal shows the output of the function:

```
PS D:\AI assisted coding> & C:/Users/Vaish/AppData/Local/Programs/Python/Python314/python.exe "d:/AI assisted coding/area_calculator.py"
Circle Area (r=5): 78.54
Rectangle Area (l=10, w=4): 40.00
Triangle Area (b=6, h=3): 9.00
PS D:\AI assisted coding>
```

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.

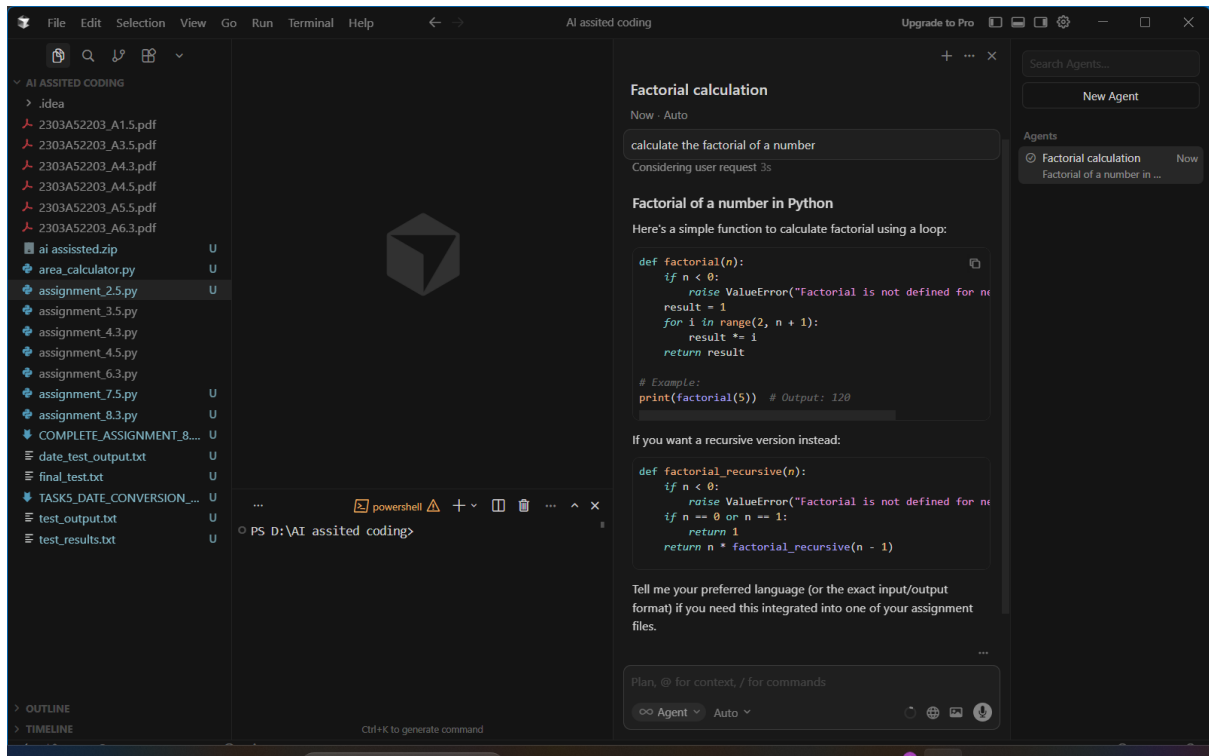
❖ Expected Output:

➤ Prompt list

➤ Code variations

Prompt:-

=> Calculate the factorial of a number



=> write factorial function in python using recursion

