

## ASSIGNMENT-2.5

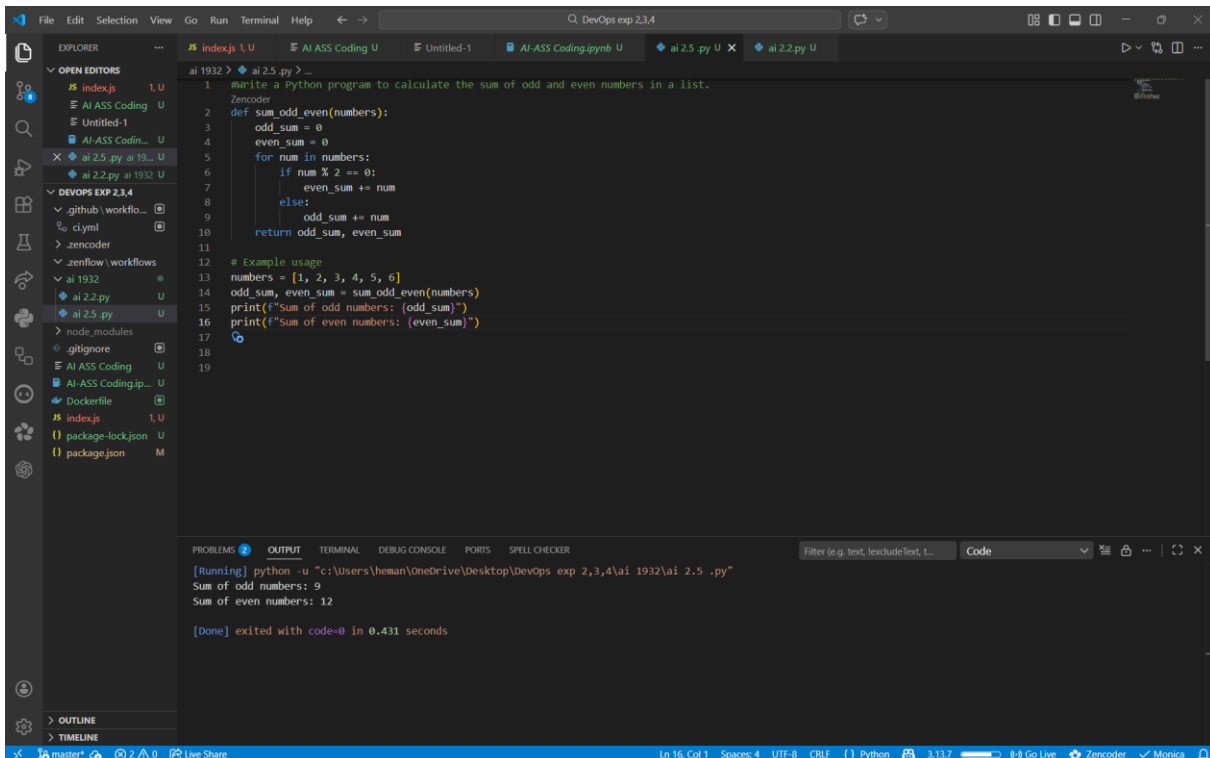
Name-R.Teeneswari

Roll no – 2303A51932

Batch-30 Task-1:

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

Code:



```
1 #write a python program to calculate the sum of odd and even numbers in a list.
2 Zencoder
3 def sum_odd_even(numbers):
4     odd_sum = 0
5     even_sum = 0
6     for num in numbers:
7         if num % 2 == 0:
8             even_sum += num
9         else:
10            odd_sum += num
11    return odd_sum, even_sum
12
13 # Example usage
14 numbers = [1, 2, 3, 4, 5, 6]
15 odd_sum, even_sum = sum_odd_even(numbers)
16 print(f"Sum of odd numbers: {odd_sum}")
17 print(f"Sum of even numbers: {even_sum}")
18
19
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS SPELL CHECKER

[Running] python -u "c:\Users\heman\OneDrive\Desktop\DevOps exp 2,3,4\ai 1932\ai 2.5 .py"

Sum of odd numbers: 9  
Sum of even numbers: 12

[Done] exited with code=0 in 0.431 seconds

Observation:

The **original code** works correctly but is written as a single block, making it harder to reuse and test.

The **refactored (AI-improved) code** separates logic into a function, improving:

- Readability
- Reusability
- Maintainability

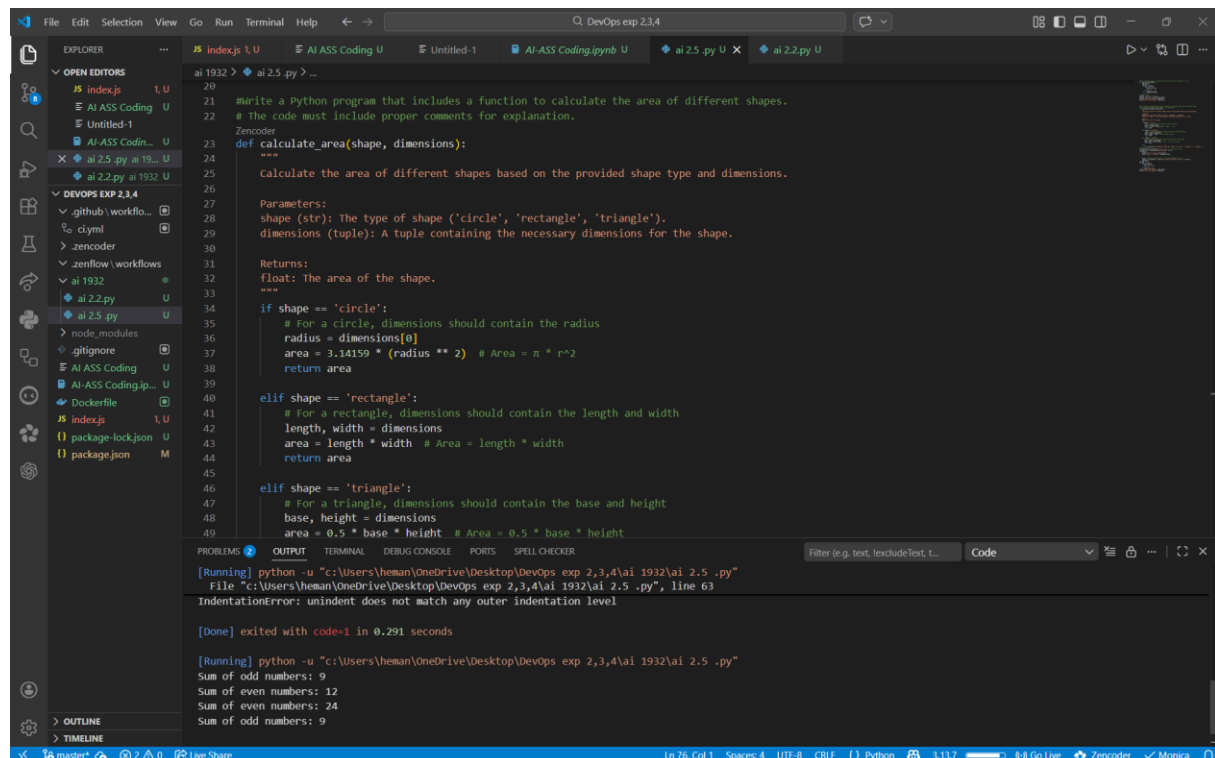
Using a function allows the same logic to be reused with different lists without rewriting code.

## Task-2:

Prompt: write a program explain a function that calculates the area of different shapes.

The code must include proper comments for explanation.

Code:



```
20
21 #Write a Python program that includes a function to calculate the area of different shapes.
22 # The code must include proper comments for explanation.
23
24 def calculate_area(shape, dimensions):
25     """
26     Calculate the area of different shapes based on the provided shape type and dimensions.
27
28     Parameters:
29     shape (str): The type of shape ('circle', 'rectangle', 'triangle').
30     dimensions (tuple): A tuple containing the necessary dimensions for the shape.
31
32     Returns:
33     float: The area of the shape.
34     """
35     if shape == 'circle':
36         # For a circle, dimensions should contain the radius
37         radius = dimensions[0]
38         area = 3.14159 * (radius ** 2) # Area = π * r^2
39         return area
40     elif shape == 'rectangle':
41         # For a rectangle, dimensions should contain the length and width
42         length, width = dimensions
43         area = length * width # Area = length * width
44         return area
45     elif shape == 'triangle':
46         # For a triangle, dimensions should contain the base and height
47         base, height = dimensions
48         area = 0.5 * base * height # Area = 0.5 * base * height
49         return area
```

Observation:

This program uses **one function** to calculate the area of **multiple shapes**, which avoids code duplication.

The shape parameter decides **which formula** to apply.

The function uses **conditional statements** (if / elif) to select the correct formula.

It improves **code clarity**, making onboarding easier and faster.

## Task:3

Prompt: explain a function that calculates the area of different shapes (curser used)

Shapes. Write a program to find the sum of even and odd numbers in a list Code:

```
57 def main():
58     #write a program to find the sum of even and odd numbers in a list
59     numbers = list(map(int, input("Enter numbers separated by space: ").split()))
60     even_sum=0
61     odd_sum = 0
62     for num in numbers:
63         if num % 2 == 0:
64             even_sum += num
65         else:
66             odd_sum += num
67     print(f"Sum of even numbers: {even_sum}")
68     print(f"Sum of odd numbers: {odd_sum}")
69
70 if __name__ == "__main__":
71     main()
```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE PORTS SPELL CHECKER

[Done] exited with code=1 in 0.291 seconds

[Running] python -u "c:\Users\heman\OneDrive\Desktop\DevOps exp 2,3,4\ai 1932\ai 2.5 .py"

Sum of odd numbers: 9  
Sum of even numbers: 12  
Sum of even numbers: 24  
Sum of odd numbers: 9

[Done] exited with code=0 in 0.271 seconds

Observation:

The program demonstrates **how one function can handle multiple use cases**.

Comments clearly explain:

What the function does

Why each condition exists

What each parameter represents

Using comments makes the code **junior-developer friendly**, which is ideal for onboarding.

The main () function separates **user interaction** from **business logic**, improving structure.

This style is considered **clean, readable, and professional** in real-world projects. Task-4:

Prompt: Based on practical usage and experimentation, compare **Gemini**, **GitHub Copilot**, and **Cursor AI** in terms of **usability** and **code quality**.

Observation:

**Gemini** is best suited for **explanations and learning support**. It produces readable, beginner-friendly code and clear step-by-step reasoning, making it ideal for onboarding juniors and understanding concepts.

**GitHub Copilot** excels in **real-time coding assistance** inside IDEs. It is fast, contextaware, and highly productive for experienced developers, but its code may lack explanations.

**Cursor AI** stands out for **prompt sensitivity and refactoring quality**. It responds strongly to detailed prompts, generating cleaner, more structured, and optimized code, making it suitable for improving legacy codebases.

**usability**, Copilot integrates seamlessly into workflows, Gemini is conversational and educational, and Cursor AI offers powerful prompt-driven refactoring.

**code quality**, Cursor AI and Copilot generally produce more professional, productionready code, while Gemini focuses on clarity over optimization