

## ASSIGNMENT-2.5

Name-A. Sravani

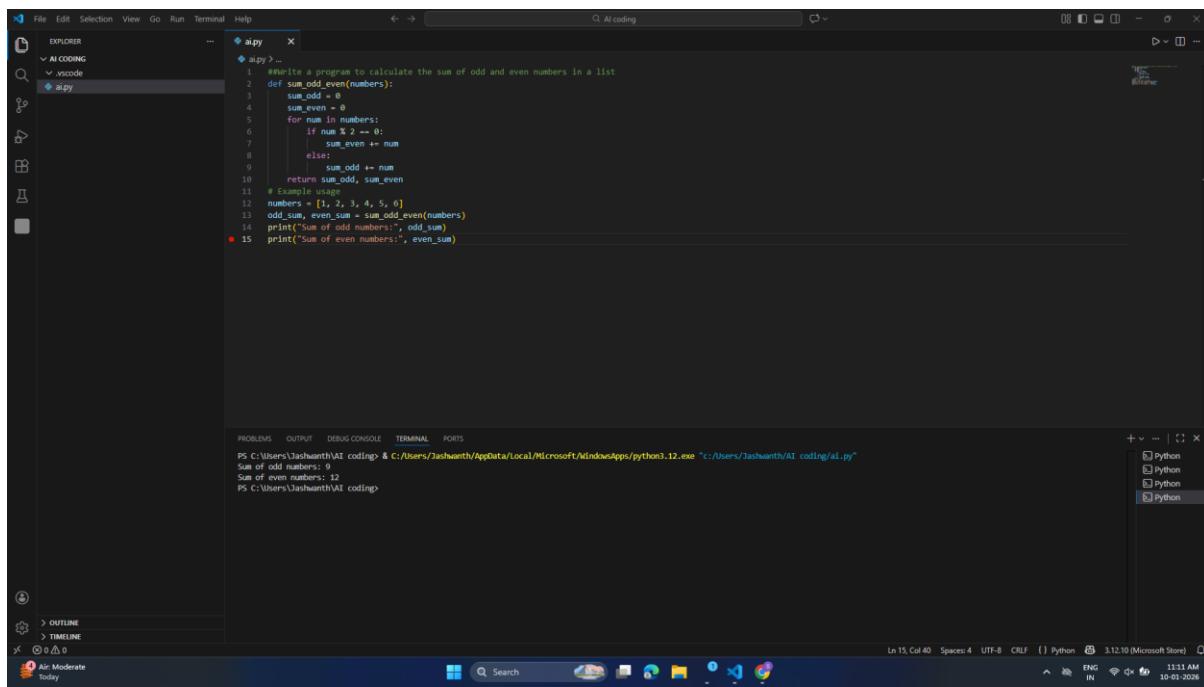
Roll no – 2303A510G7

Batch-30

Task-1:

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

Code:



```
ai.py
# Write a program to calculate the sum of odd and even numbers in a list
def sum_odd_even(numbers):
    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
# Example usage
numbers = [1, 2, 3, 4, 5, 6]
odd_sum, even_sum = sum_odd_even(numbers)
print("Sum of odd numbers:", odd_sum)
print("Sum of even numbers:", even_sum)
```

The screenshot shows the VS Code interface with the file `ai.py` open. The code defines a function `sum_odd_even` that takes a list of numbers and returns the sum of odd and even numbers. The function uses a for loop to iterate through the list, checking if each number is odd or even using the modulo operator (%). The script then prints the results. The terminal tab shows the output of running the script with the provided data, resulting in a sum of 9 for odd numbers and 12 for even numbers.

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:

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows the file structure with 'ai.py' selected. The main editor tab contains the following Python code:

```
File Edit Selection View Go Run Terminal Help
ai.py x
ai.py > calculate_area
1 # write a program explain a function that calculates the area of different shapes.
2 # The code must include proper comments for explanation.
3
4 def calculate_area(shape, value1, value2=0):
5     """
6         This function calculates the area of different shapes.
7         shape -> type of shape (circle, rectangle, triangle)
8         value1 -> radius (circle) OR length/width
9         value2 -> width/height (default is 0 for circle)
10    """
11
12
13    # Check if the shape is a circle
14    if shape == "circle":
15        return 3.14 * value1 * value1 # π * r * r
16
17    # Check if the shape is a rectangle
18    elif shape == "rectangle":
19        return value1 * value2 # length * width
20
21    # Check if the shape is a triangle
22    elif shape == "triangle":
23        return 0.5 * value1 * value2 # ½ * base * height
24
25    # If shape is not supported
26    else:
27        return "Invalid shape"
28
29
30 def main():
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Jashwanth\AI coding & C:/Users/Jashwanth/AppData/Local/Microsoft/WindowsApps/python3.12.exe "C:/Users/Jashwanth/AI coding/ai.py"
Enter shape (circle/rectangle/triangle): circle
Enter radius: 10
Area: 314.1592653589793
PS C:\Users\Jashwanth\AI coding>
```

The terminal at the bottom shows the execution of the script and the output for a circle with radius 10.

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 (cursor used)

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

Code:

The screenshot shows a Microsoft Windows desktop with the Visual Studio Code (VS Code) application open. The code editor displays a Python file named 'ai.py' with the following content:

```

1  #Write a program to find the sum of even and odd numbers in a list
2  numbers = list(map(int, input("Enter numbers: ").split()))
3
4  even_sum = 0
5  odd_sum = 0
6
7  for num in numbers:
8      if num % 2 == 0:
9          even_sum += num
10     else:
11         odd_sum += num
12
13 print("Even Sum:", even_sum)
14 print("Odd Sum:", odd_sum)

```

The terminal below the code editor shows the execution of the script and its output:

```

PS C:\Users\Jashwanth\AI coding> & C:/Users/Jashwanth/AppData/Local/Microsoft/WindowsApps/python3.12.exe "C:/Users/Jashwanth/AI coding/ai.py"
Enter numbers: 12
Even Sum: 12
Odd Sum: 0
PS C:\Users\Jashwanth\AI coding>

```

The status bar at the bottom right indicates the system is running at 17°C, the date is 10-01-2026, and the time is 11:22 AM.

**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, context-aware, 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, production-ready code, while Gemini focuses on clarity over optimization