# Lab Report 9.2

Name: V. Vishnu Vardhan

Roll Number: 2503A51L26

Course Code: 24CS002PC215

Course Title: AI Assisted Coding

Assignment Number: 9.2

Academic Year: 2025-2026

Branch :cse

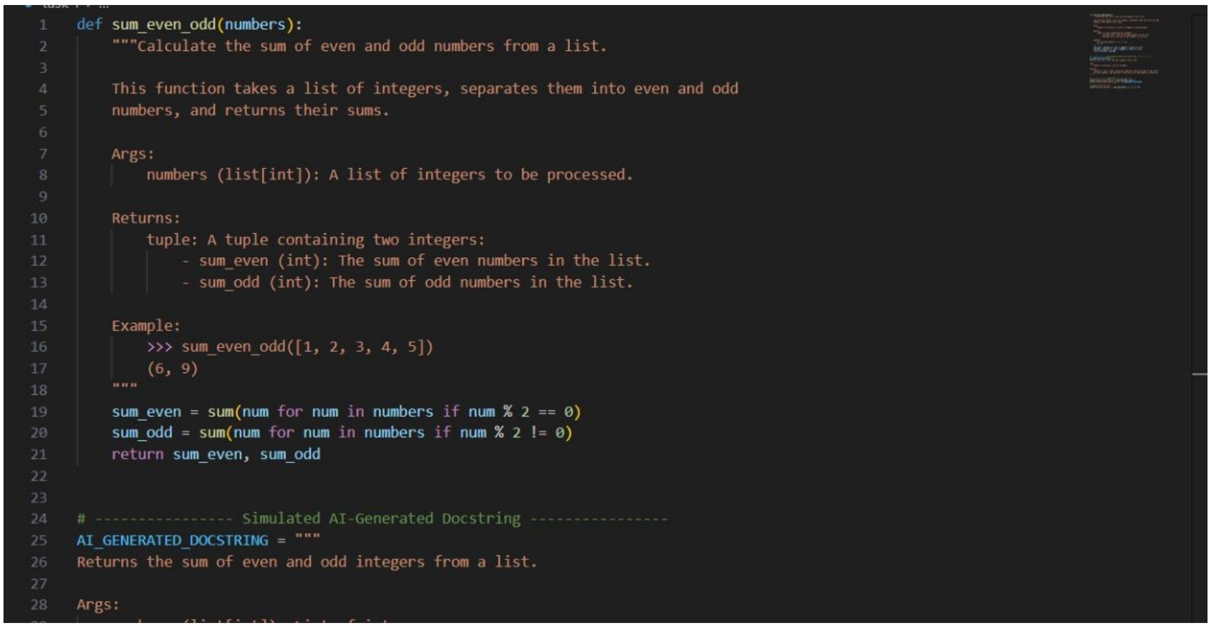
## Lab Objectives

- To explore AI-powered auto-completion features for core Python constructs.
- To analyze how AI suggests logic for class definitions, loops, and conditionals.
- To evaluate the completeness and correctness of code generated by AI assistants.

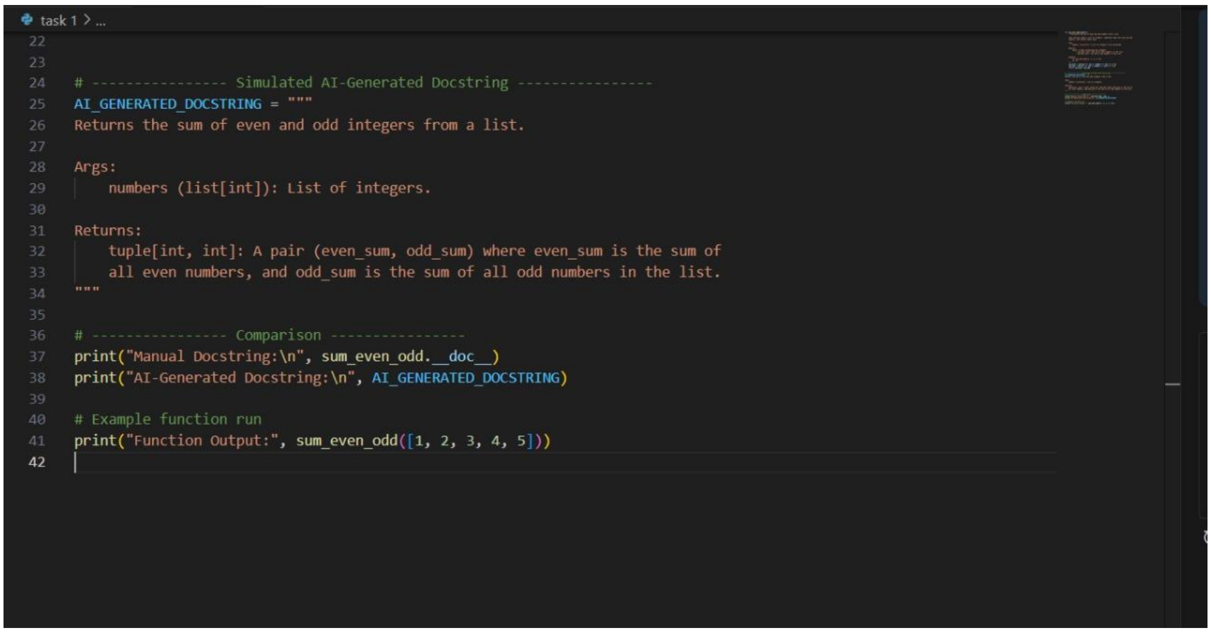
## Lab Outcomes

- Use AI tools to generate and complete class definitions and methods.
- Understand and assess AI-suggested loops for iterative tasks.
- Generate conditional statements through prompt-driven suggestions.
- Critically evaluate AI-assisted code for correctness and clarity.

## TASK :-1



```
def sum_even_odd(numbers):
    """Calculate the sum of even and odd numbers from a list.

    This function takes a list of integers, separates them into even and odd
    numbers, and returns their sums.

    Args:
        numbers (list[int]): A list of integers to be processed.

    Returns:
        tuple: A tuple containing two integers:
            - sum_even (int): The sum of even numbers in the list.
            - sum_odd (int): The sum of odd numbers in the list.

    Example:
        >>> sum_even_odd([1, 2, 3, 4, 5])
        (6, 9)
    """
    sum_even = sum(num for num in numbers if num % 2 == 0)
    sum_odd = sum(num for num in numbers if num % 2 != 0)
    return sum_even, sum_odd


# --------------- Simulated AI-Generated Docstring ---------------
AI_GENERATED_DOCSTRING = """
Returns the sum of even and odd integers from a list.

Args:
```



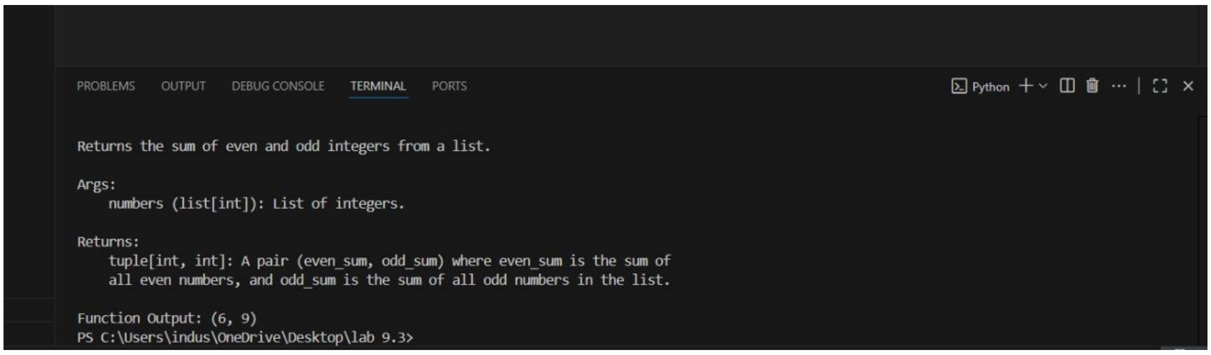
```
# --------------- Simulated AI-Generated Docstring ---------------
AI_GENERATED_DOCSTRING = """
Returns the sum of even and odd integers from a list.

Args:
    numbers (list[int]): List of integers.

Returns:
    tuple[int, int]: A pair (even_sum, odd_sum) where even_sum is the sum of
    all even numbers, and odd_sum is the sum of all odd numbers in the list.
"""

# --------------- Comparison ---------------
print("Manual Docstring:\n", sum_even_odd.__doc__)
print("AI-Generated Docstring:\n", AI_GENERATED_DOCSTRING)

# Example function run
print("Function Output:", sum_even_odd([1, 2, 3, 4, 5]))
```

## Observation:-

### Manual (Google Style):

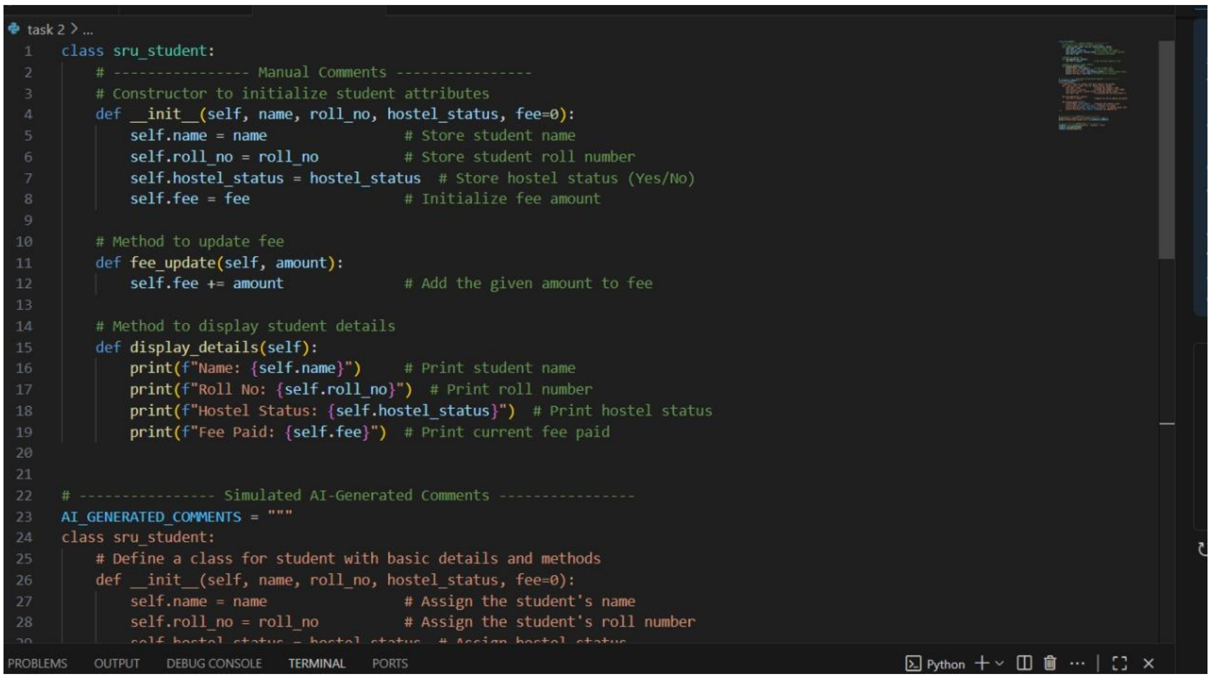
- More **structured and detailed**.
- Includes **Examples** section (useful for quick understanding).
- Explicitly states return values with explanation.

## Output:-

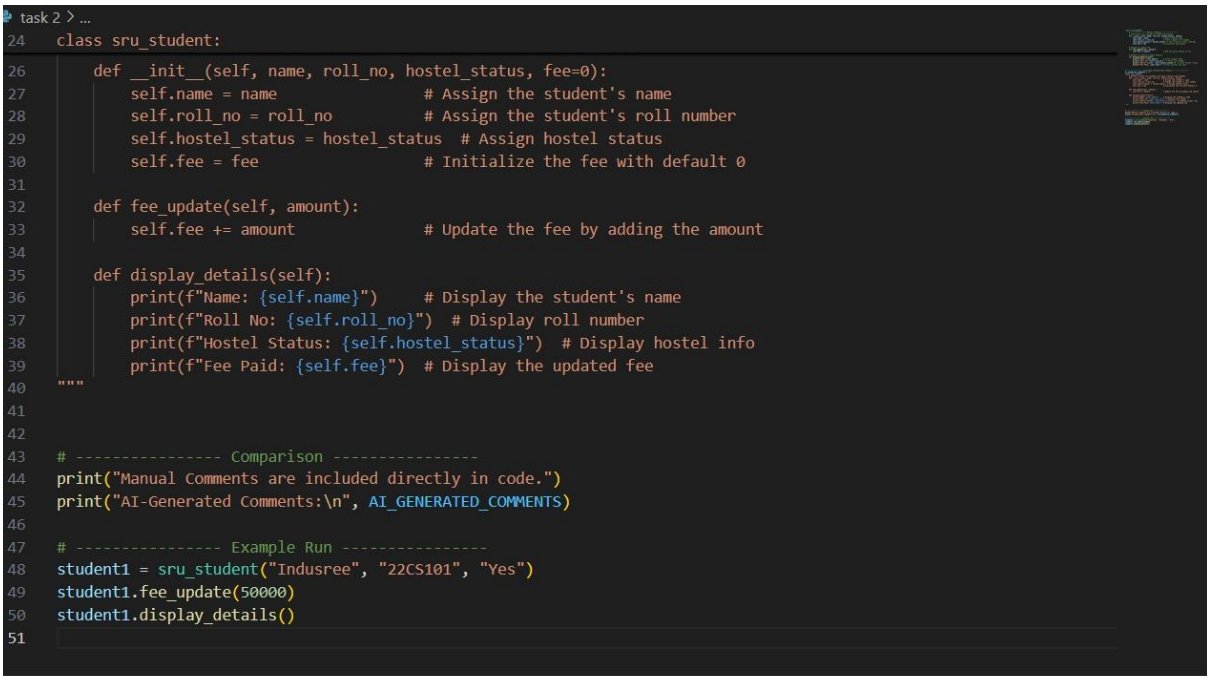


```
Returns the sum of even and odd integers from a list.

Args:
    numbers (list[int]): List of integers.

Returns:
    tuple[int, int]: A pair (even_sum, odd_sum) where even_sum is the sum of
    all even numbers, and odd_sum is the sum of all odd numbers in the list.

Function Output: (6, 9)
PS C:\Users\indus\OneDrive\Desktop\lab 9.3>
```

## TASK:-2



```python
class sru_student:
    # ---------------- Manual Comments ----------------
    # Constructor to initialize student attributes
    def __init__(self, name, roll_no, hostel_status, fee=0):
        self.name = name                # Store student name
        self.roll_no = roll_no          # Store student roll number
        self.hostel_status = hostel_status  # Store hostel status (Yes/No)
        self.fee = fee                  # Initialize fee amount

    # Method to update fee
    def fee_update(self, amount):
        self.fee += amount              # Add the given amount to fee

    # Method to display student details
    def display_details(self):
        print(f"Name: {self.name}")      # Print student name
        print(f"Roll No: {self.roll_no}")  # Print roll number
        print(f"Hostel Status: {self.hostel_status}")  # Print hostel status
        print(f"Fee Paid: {self.fee}")  # Print current fee paid


# ---------------- Simulated AI-Generated Comments ----------------
AI_GENERATED_COMMENTS = """
class sru_student:
    # Define a class for student with basic details and methods
    def __init__(self, name, roll_no, hostel_status, fee=0):
        self.name = name                # Assign the student's name
        self.roll_no = roll_no          # Assign the student's roll number
```


task 2 > ...

```python
class sru_student:

    def __init__(self, name, roll_no, hostel_status, fee=0):
        self.name = name                # Assign the student's name
        self.roll_no = roll_no          # Assign the student's roll number
        self.hostel_status = hostel_status  # Assign hostel status
        self.fee = fee                  # Initialize the fee with default 0

    def fee_update(self, amount):
        self.fee += amount              # Update the fee by adding the amount

    def display_details(self):
        print(f"Name: {self.name}")     # Display the student's name
        print(f"Roll No: {self.roll_no}")  # Display roll number
        print(f"Hostel Status: {self.hostel_status}")  # Display hostel info
        print(f"Fee Paid: {self.fee}")  # Display the updated fee
"""


# --------------- Comparison ---------------
print("Manual Comments are included directly in code.")
print("AI-Generated Comments:\n", AI_GENERATED_COMMENTS)

# --------------- Example Run ---------------
student1 = sru_student("Indusree", "22CS101", "Yes")
student1.fee_update(50000)
student1.display_details()
```

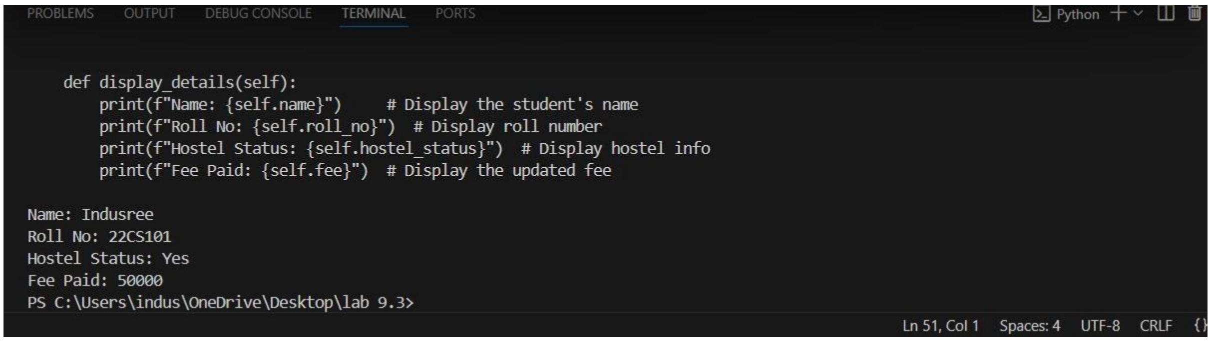
## Observation:-

1. **Manual Comments:**
   - **Concise and to the point.**
   - **Written in human-friendly language (like a teacher explaining).**
   - **Sometimes slightly less formal but quicker to read.**
2. **AI-Generated Comments:**
   - **More verbose and descriptive, explaining each line in detail.**
   - **Uses technical phrasing (e.g., *"assign the provided value"*) instead of simpler wording.**
   - **Good for beginners, but may feel repetitive for experienced programmers.**

**Conclusion:**

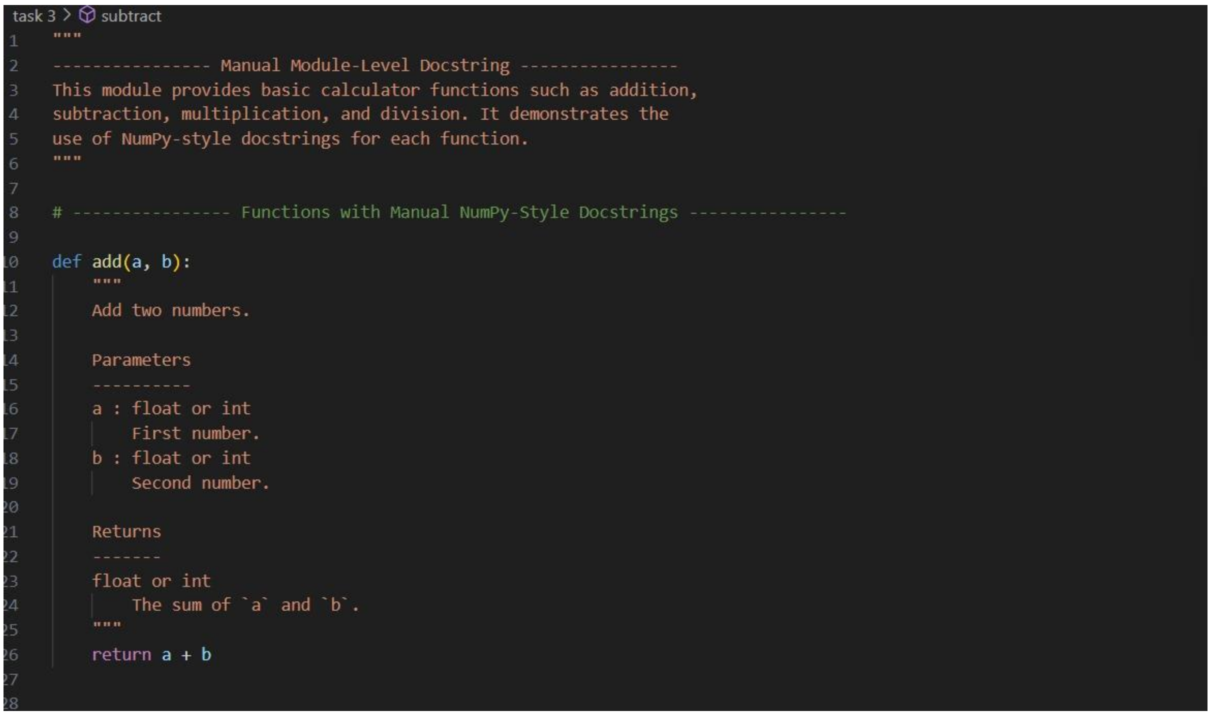
- **Manual comments are better for readability and give a summary explanation.**
- **AI-generated comments are thorough and line-by-line, which can be helpful for learning or code reviews, but may be overly detailed in larger programs.**

**Output:-**



```
    def display_details(self):
        print(f"Name: {self.name}")     # Display the student's name
        print(f"Roll No: {self.roll_no}")  # Display roll number
        print(f"Hostel Status: {self.hostel_status}")  # Display hostel info
        print(f"Fee Paid: {self.fee}")  # Display the updated fee

Name: Indusree
Roll No: 22CS101
Hostel Status: Yes
Fee Paid: 50000
PS C:\Users\indus\OneDrive\Desktop\lab 9.3>
```

## TASK:-3

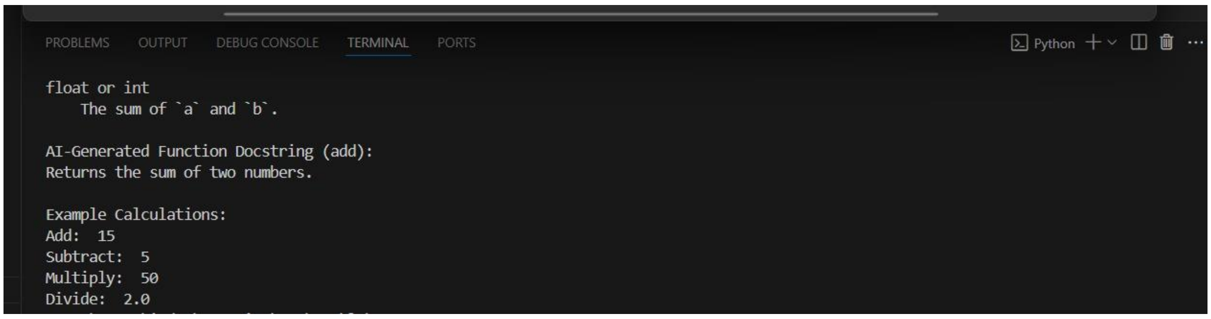


```
"""
---------------- Manual Module-Level Docstring ----------------
This module provides basic calculator functions such as addition,
subtraction, multiplication, and division. It demonstrates the
use of NumPy-style docstrings for each function.
"""

# ---------------- Functions with Manual NumPy-Style Docstrings ----------------

def add(a, b):
    """
    Add two numbers.

    Parameters
    ----------
    a : float or int
        First number.
    b : float or int
        Second number.

    Returns
    -------
    float or int
        The sum of `a` and `b`.
    """
    return a + b

```

## Observation:-

- **Well-structured with Parameters, Returns, Raises, Examples.**
- **Very useful for both developers and users of the code.**
- **Consistent across all functions.**
- **Slightly verbose, but excellent for documentation standards.**

## Output:-



```
float or int
    The sum of `a` and `b`.

AI-Generated Function Docstring (add):
Returns the sum of two numbers.

Example Calculations:
Add:  15
Subtract:  5
Multiply:  50
Divide:  2.0
```