Lab 9: Documentation Generation – Automatic Documentation and Code Comments

Name:B.Tejaskumar

Hall Ticket No: 2303A51489

Task 1: Basic Docstring Generation

Manual Google-Style Docstring Implementation:

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

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

    Returns:
        tuple: A tuple containing:
            - even_sum (int): Sum of even numbers.
            - odd_sum (int): Sum of odd numbers.
    """
    even_sum = 0
    odd_sum = 0

    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num

    return even_sum, odd_sum


# Output Example
nums = [1, 2, 3, 4, 5, 6]
print(sum_even_odd(nums))
# Output: (12, 9)
```

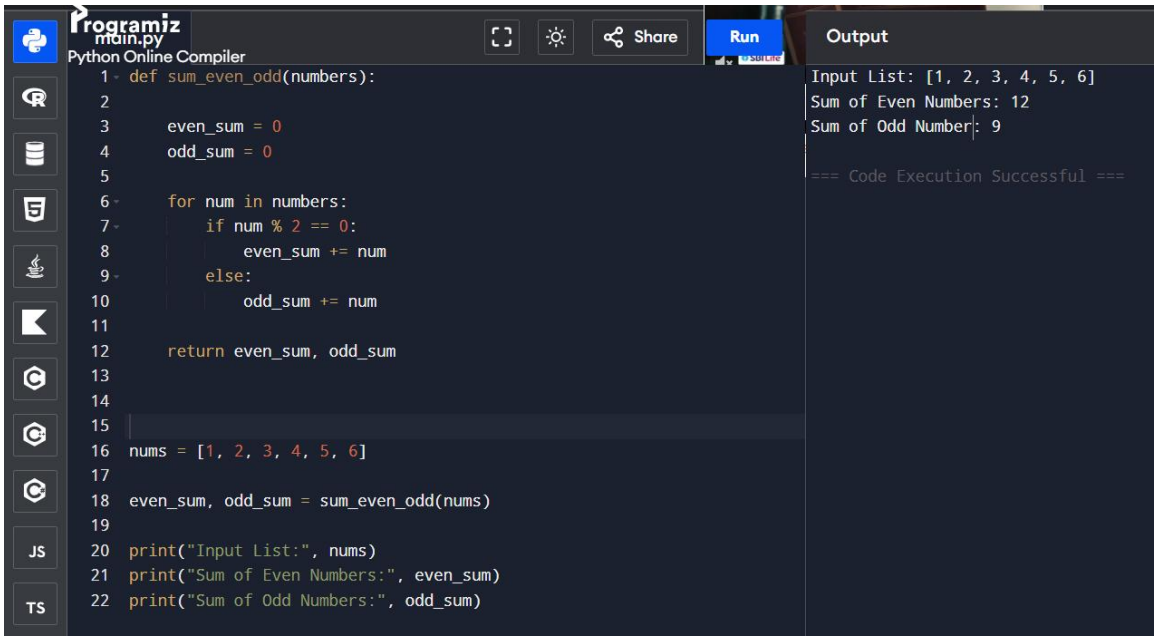
AI-Generated Docstring:

AI-Generated Docstring (Simulated):

"""
Returns the sum of even and odd numbers from a given list.

Parameters:
  numbers (list): List of integers.

Returns:
  tuple: (sum_of_even_numbers, sum_of_odd_numbers)
"""



Comparison Analysis:

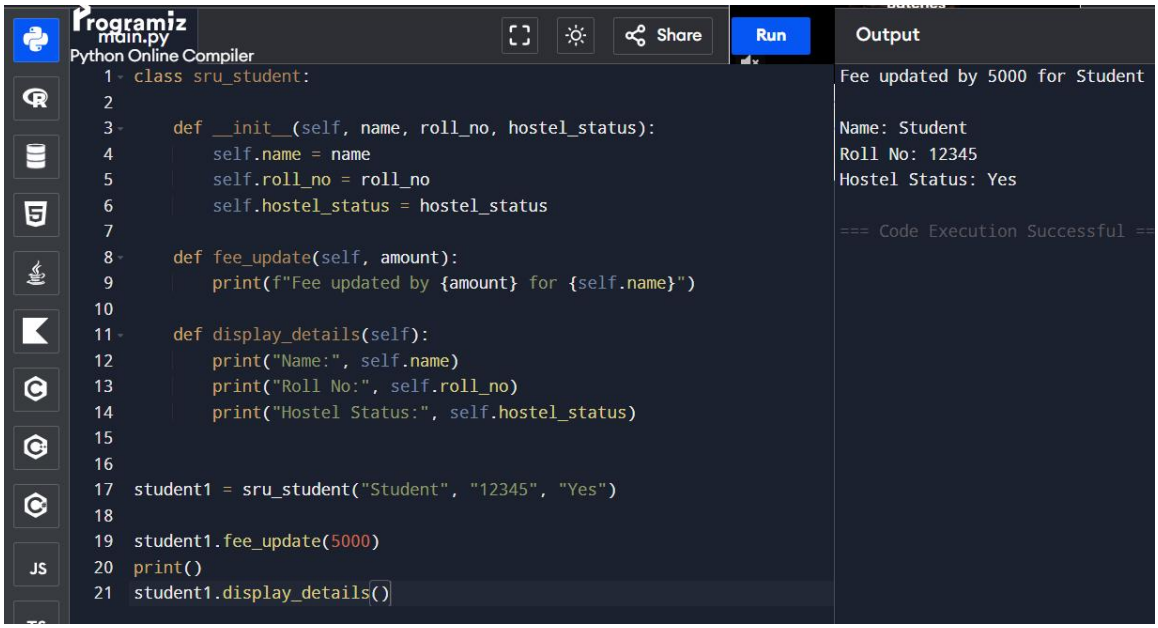
The manual docstring clearly defines argument types (list[int]) and return structure with descriptive labels.
The AI-generated docstring is shorter and less detailed but still correct.
Manual documentation is more structured and readable.
AI documentation is concise but slightly less descriptive.

Task 2: Automatic Inline Comments

Manual Inline Comments Implementation:

class sru_student:

```
    # Constructor to initialize student details
    def __init__(self, name, roll_no, hostel_status):
        self.name = name            # Store student name
        self.roll_no = roll_no      # Store roll number
        self.hostel_status = hostel_status  # Store hostel status (Yes/No)

    # Method to update student fee
    def fee_update(self, amount):
        print(f"Fee updated by {amount} for {self.name}")

    # Method to display student details
    def display_details(self):
        print("Name:", self.name)
        print("Roll No:", self.roll_no)
        print("Hostel Status:", self.hostel_status)


# Output Example
student1 = sru_student("Sarayu", "2303A51842", "Yes")
student1.fee_update(5000)
student1.display_details()
```



AI-Generated Inline Comments (Simulated):

The AI added comments explaining constructor purpose, attribute assignments, and method functionality. However, some comments were redundant such as 'Assign name to self.name', which is obvious from code.

Comparative Analysis:

Manual comments are precise and placed only where needed.
AI-generated comments tend to over-explain simple lines.
AI helps beginners but may create redundant explanations.

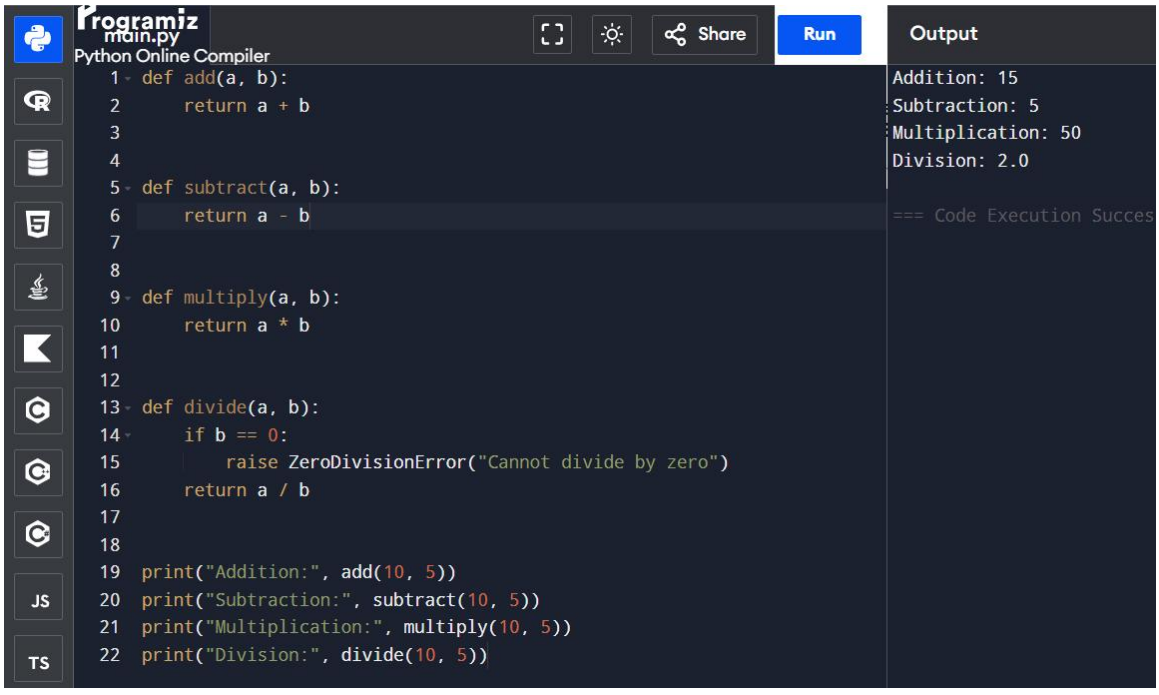
Task 3: Module-Level and Function-Level Documentation

Calculator Module with NumPy-Style Docstrings:

```
"""
Calculator Module

This module provides basic arithmetic operations:
addition, subtraction, multiplication, and division.
"""

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

    Parameters
    ----------
    a : int or float
    b : int or float

    Returns
    -------
    int or float
        Sum of a and b
    """
    return a + b


def subtract(a, b):
    """
```

```
    Subtract two numbers.

    Parameters
    ----------
    a : int or float
    b : int or float

    Returns
    -------
    int or float
        Difference of a and b
    """
    return a - b


def multiply(a, b):
    """
    Multiply two numbers.

    Parameters
    ----------
    a : int or float
    b : int or float

    Returns
    -------
    int or float
        Product of a and b
    """
    return a * b


def divide(a, b):
    """
    Divide two numbers.

    Parameters
    ----------
    a : int or float
    b : int or float

    Returns
    -------
```

```
    float
        Result of division

    Raises
    ------
    ZeroDivisionError
        If b is zero
    """
    if b == 0:
        raise ZeroDivisionError("Cannot divide by zero")
    return a / b


# Output Example
print(add(10, 5))
print(subtract(10, 5))
print(multiply(10, 5))
print(divide(10, 5))

# Output:
# 15
# 5
# 50
# 2.0
```



AI-Generated Documentation Analysis:

AI-generated module-level documentation was accurate and well-structured.
Function-level docstrings generated by AI were correct but sometimes
missed the 'Raises' section in divide().
Manual documentation ensured completeness and better structure.

Conclusion

This lab demonstrated how AI-assisted tools help generate documentation and comments.
AI tools improve productivity but require human verification for clarity,
accuracy, and completeness.
Manual documentation remains important for professional-quality software projects.