AI-ASSISTANT-COADING

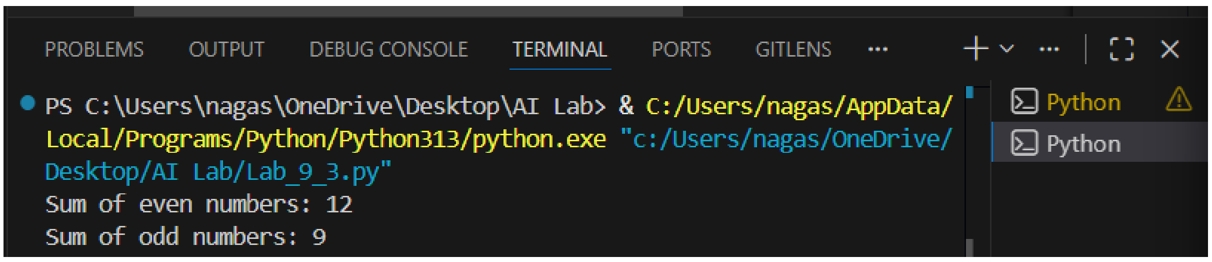
Assignment-9_3
2303A51247
Batch-18

Task 1: Basic Docstring Generation
Scenario
You are developing a utility function that processes numerical lists and must be properly documented for future maintenance.
Requirements
• Write a Python function to return the sum of even numbers and sum of odd numbers in a given list
Expected Output
• Python function with manual Google-style docstring
• AI-generated docstring for the same function
• Comparison explaining differences between manual and AI-generated documentation
• Improved understanding of AI-generated function-level documentation

Code:

```python
#Write a Python function to return the sum of even numbers and sum of odd numbers in a given list of integers.
def sum_even_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
# Test cases
numbers = [1, 2, 3, 4, 5, 6]
even_sum, odd_sum = sum_even_odd(numbers)
print(f"Sum of even numbers: {even_sum}")  # Output: 12
print(f"Sum of odd numbers: {odd_sum}")    # Output: 9
```

Output:

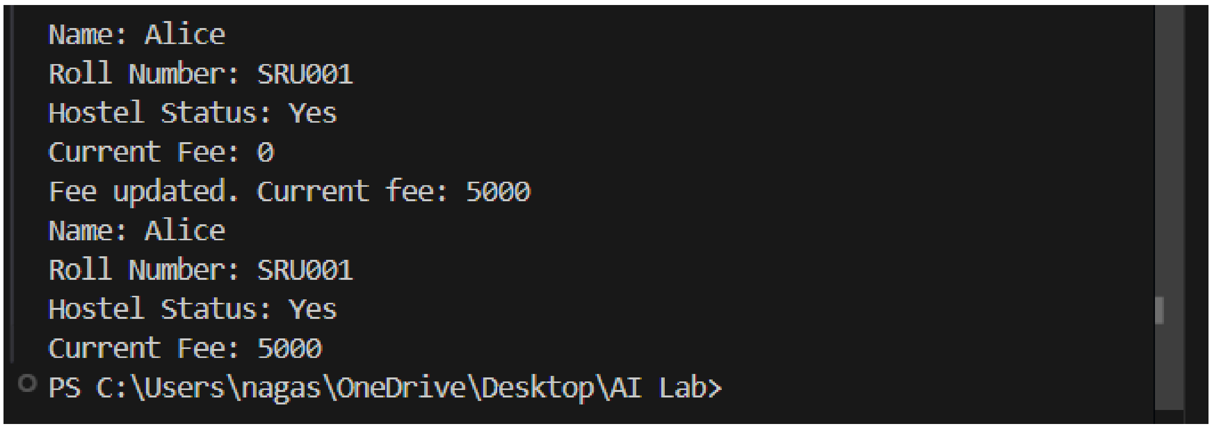


```
PS C:\Users\nagas\OneDrive\Desktop\AI Lab> & C:/Users/nagas/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nagas/OneDrive/Desktop/AI Lab/Lab_9_3.py"
Sum of even numbers: 12
Sum of odd numbers: 9
```

Task 2: Automatic Inline Comments
Scenario
You are developing a student management module that must be easy to understand for new developers.
Requirements
• Write a Python program for an sru_student class with the following:
– Attributes: name, roll_no, hostel_status
– Methods: fee_update() and display_details()
• Manually write inline comments for each line or logical block
• Use an AI-assisted tool to automatically add inline comments
• Compare manual comments with AI-generated comments
• Identify missing, redundant, or incorrect AI comments
Expected Output
• Python class with manually written inline comments
• AI-generated inline comments added to the same code
• Comparative analysis of manual vs AI comments
• Critical discussion on strengths and limitations of AI-generated comments

Code:
```python
#write a python code for an sru_student classwith attributes name,roll number,hostel_status and a method to display the details of the fee_update() and display_details()
class SRUStudent:
    def __init__(self, name, roll_number, hostel_status):
        self.name = name
        self.roll_number = roll_number
        self.hostel_status = hostel_status
        self.fee = 0

    def fee_update(self, amount):
        self.fee += amount
        print(f"Fee updated. Current fee: {self.fee}")

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Hostel Status: {self.hostel_status}")
        print(f"Current Fee: {self.fee}")
# Test cases
student1 = SRUStudent("Alice", "SRU001", "Yes")
student1.display_details()
student1.fee_update(5000)
student1.display_details()
```

Output:



```
Name: Alice
Roll Number: SRU001
Hostel Status: Yes
Current Fee: 0
Fee updated. Current fee: 5000
Name: Alice
Roll Number: SRU001
Hostel Status: Yes
Current Fee: 5000
PS C:\Users\nagas\OneDrive\Desktop\AI Lab>
```

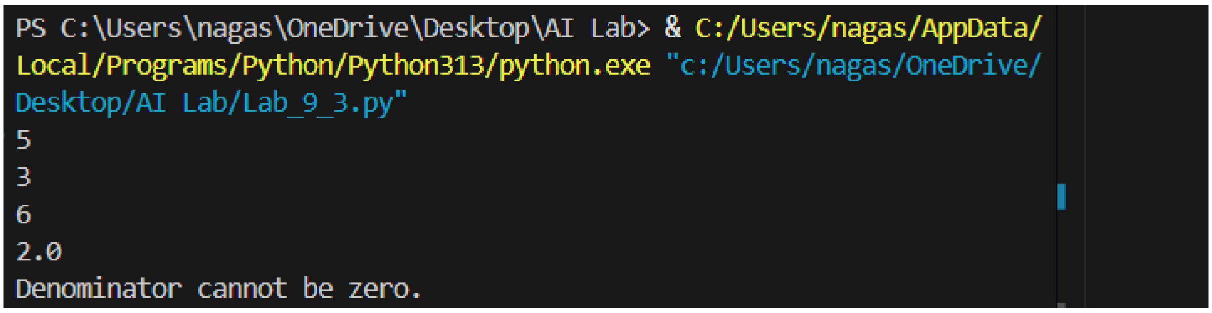
Task 3: Module-Level and Function-Level Documentation
Scenario
You are building a small calculator module that will be shared across multiple projects and requires structured documentation.
Requirements
• Write a Python script containing 3–4 functions (e.g., add, subtract, multiply, divide)
• Manually write NumPy Style docstrings for each function
• Use AI assistance to generate:
– A module-level docstring
– Individual function-level docstrings
• Compare AI-generated docstrings with manually written ones
• Evaluate documentation structure, accuracy, and readability
Expected Output
• Python script with manual NumPy-style docstrings
• AI-generated module-level and function-level documentation
• Comparison between AI-generated and manual documentation
• Clear understanding of structured documentation for multi-function scripts

Code:

```
#Write a Python script and code containing 3-4 functions (e.g., add,
subtract, multiply, divide)Manually write NumPy Style docstrings for
each function

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

    Parameters
    ----------
    a : int or float
```

```
        The first number to add.
    b : int or float
        The second number to add.

    Returns
    -------
    int or float
        The sum of a and b.

    Examples
    --------
    >>> add(2, 3)
    5
    >>> add(1.5, 2.5)
    4.0
    """
    return a + b
def subtract(a, b):
    """
    Subtract one number from another.
    Parameters
    ----------
    a : int or float
        The number to be subtracted from.
    b : int or float



        The number to subtract.
    Returns
    -------
    int or float
        The difference of a and b.
    Examples
    --------
    >>> subtract(5, 2)
    3
    >>> subtract(2.5, 1.5)
    1.0
    """
    return a - b
def multiply(a, b):
    """
    Multiply two numbers.
    Parameters
    ----------
    a : int or float
        The first number to multiply.
    b : int or float
        The second number to multiply.
    Returns
    -------
    int or float
        The product of a and b.
    Examples
    --------
    >>> multiply(2, 3)
    6
    >>> multiply(1.5, 2.5)
    3.75
    """
    return a * b
def divide(a, b):
    """
    Divide one number by another.
    Parameters
    ----------
    a : int or float
        The numerator.
    b : int or float
        The denominator. Must not be zero.
    Returns
    -------
    int or float
        The quotient of a and b.
    Raises
    ------
    ValueError
        If b is zero.
    Examples
    --------
    >>> divide(6, 3)
    2.0
    >>> divide(3.75, 1.5)
    2.5
    >>> divide(5, 0)
    ValueError: Denominator cannot be zero.
    """
    if b == 0:
        raise ValueError("Denominator cannot be zero.")
    return a / b
#continue the code
# Test cases
print(add(2, 3))          # Output: 5
print(subtract(5, 2))     # Output: 3
print(multiply(2, 3))     # Output: 6
print(divide(6, 3))       # Output: 2.0
try:
    print(divide(5, 0))   # This will raise an error
except ValueError as e:
    print(e)              # Output: Denominator cannot be zero.
```

Code:



```
PS C:\Users\nagas\OneDrive\Desktop\AI Lab> & C:/Users/nagas/AppData/
Local/Programs/Python/Python313/python.exe "c:/Users/nagas/OneDrive/
Desktop/AI Lab/Lab_9_3.py"
5
3
6
2.0
Denominator cannot be zero.
```