AI Assisted Coding

Assignment – 9.3

M.Sathwik || 2303A51483 || Batch:- 08

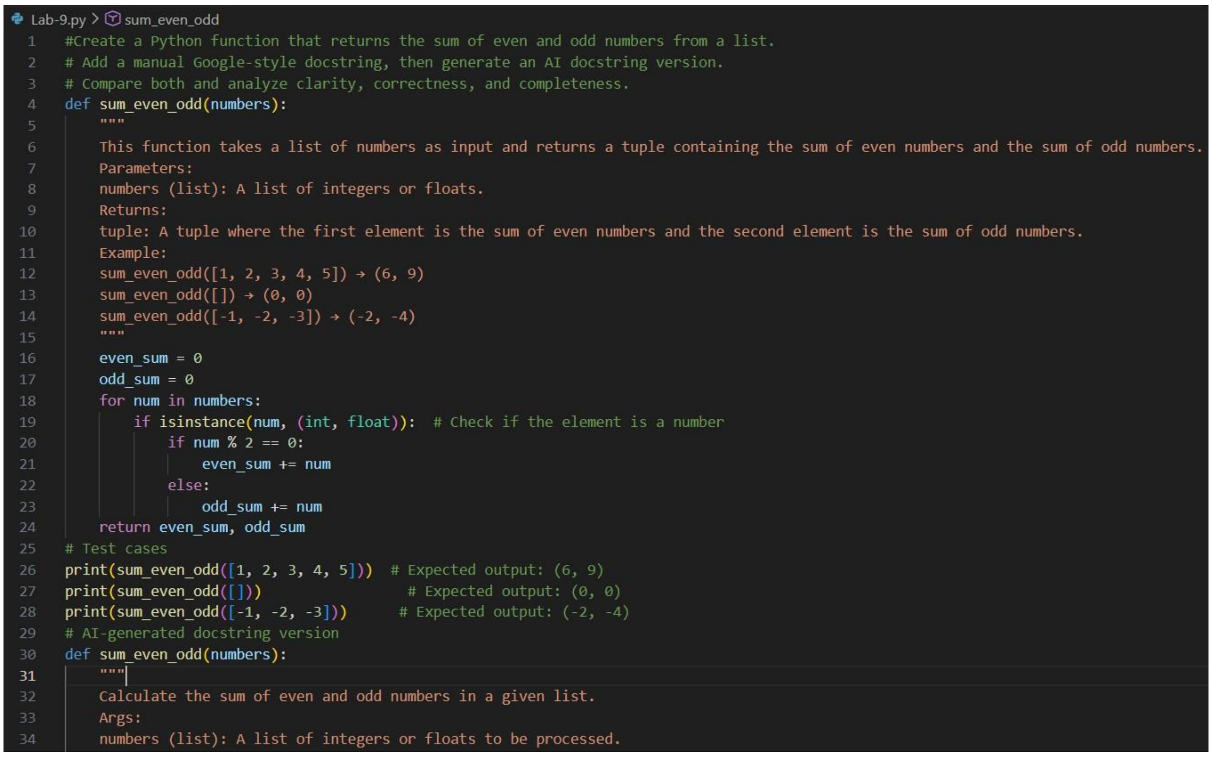
Task 1: Basic Docstring Generation

Scenario

You are developing a utility function that processes numerical lists and must be properly documented for future maintenance.

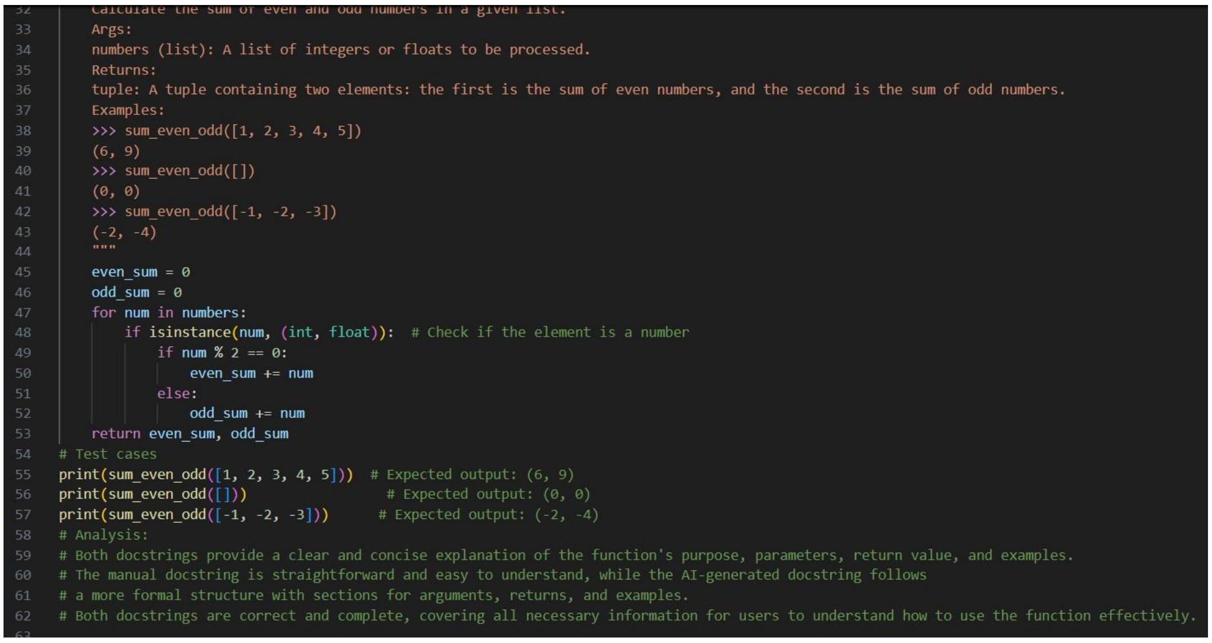
Requirements

- Write a Python function to return the sum of even numbers and sum of odd numbers in a given list
- Manually add a Google Style docstring to the function
- Use an AI-assisted tool (Copilot / Cursor AI) to generate a function-level docstring
- Compare the AI-generated docstring with the manually written docstring
- Analyze clarity, correctness, and completeness



Lab-9.py > sum_even_odd

```python
#Create a Python function that returns the sum of even and odd numbers from a list.
# Add a manual Google-style docstring, then generate an AI docstring version.
# Compare both and analyze clarity, correctness, and completeness.
def sum_even_odd(numbers):
    """
    This function takes a list of numbers as input and returns a tuple containing the sum of even numbers and the sum of odd numbers.
    Parameters:
    numbers (list): A list of integers or floats.
    Returns:
    tuple: A tuple where the first element is the sum of even numbers and the second element is the sum of odd numbers.
    Example:
    sum_even_odd([1, 2, 3, 4, 5]) → (6, 9)
    sum_even_odd([]) → (0, 0)
    sum_even_odd([-1, -2, -3]) → (-2, -4)
    """
    even_sum = 0
    odd_sum = 0
    for num in numbers:
        if isinstance(num, (int, float)):  # Check if the element is a number
            if num % 2 == 0:
                even_sum += num
            else:
                odd_sum += num
    return even_sum, odd_sum
# Test cases
print(sum_even_odd([1, 2, 3, 4, 5]))  # Expected output: (6, 9)
print(sum_even_odd([]))                 # Expected output: (0, 0)
print(sum_even_odd([-1, -2, -3]))      # Expected output: (-2, -4)
# AI-generated docstring version
def sum_even_odd(numbers):
    """
    Calculate the sum of even and odd numbers in a given list.
    Args:
    numbers (list): A list of integers or floats to be processed.
```



```python
    Calculate the sum of even and odd numbers in a given list.
    Args:
    numbers (list): A list of integers or floats to be processed.
    Returns:
    tuple: A tuple containing two elements: the first is the sum of even numbers, and the second is the sum of odd numbers.
    Examples:
    >>> sum_even_odd([1, 2, 3, 4, 5])
    (6, 9)
    >>> sum_even_odd([])
    (0, 0)
    >>> sum_even_odd([-1, -2, -3])
    (-2, -4)
    """
    even_sum = 0
    odd_sum = 0
    for num in numbers:
        if isinstance(num, (int, float)):  # Check if the element is a number
            if num % 2 == 0:
                even_sum += num
            else:
                odd_sum += num
    return even_sum, odd_sum
# Test cases
print(sum_even_odd([1, 2, 3, 4, 5]))  # Expected output: (6, 9)
print(sum_even_odd([]))                 # Expected output: (0, 0)
print(sum_even_odd([-1, -2, -3]))      # Expected output: (-2, -4)
# Analysis:
# Both docstrings provide a clear and concise explanation of the function's purpose, parameters, return value, and examples.
# The manual docstring is straightforward and easy to understand, while the AI-generated docstring follows
# a more formal structure with sections for arguments, returns, and examples.
# Both docstrings are correct and complete, covering all necessary information for users to understand how to use the function effectively.
```

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

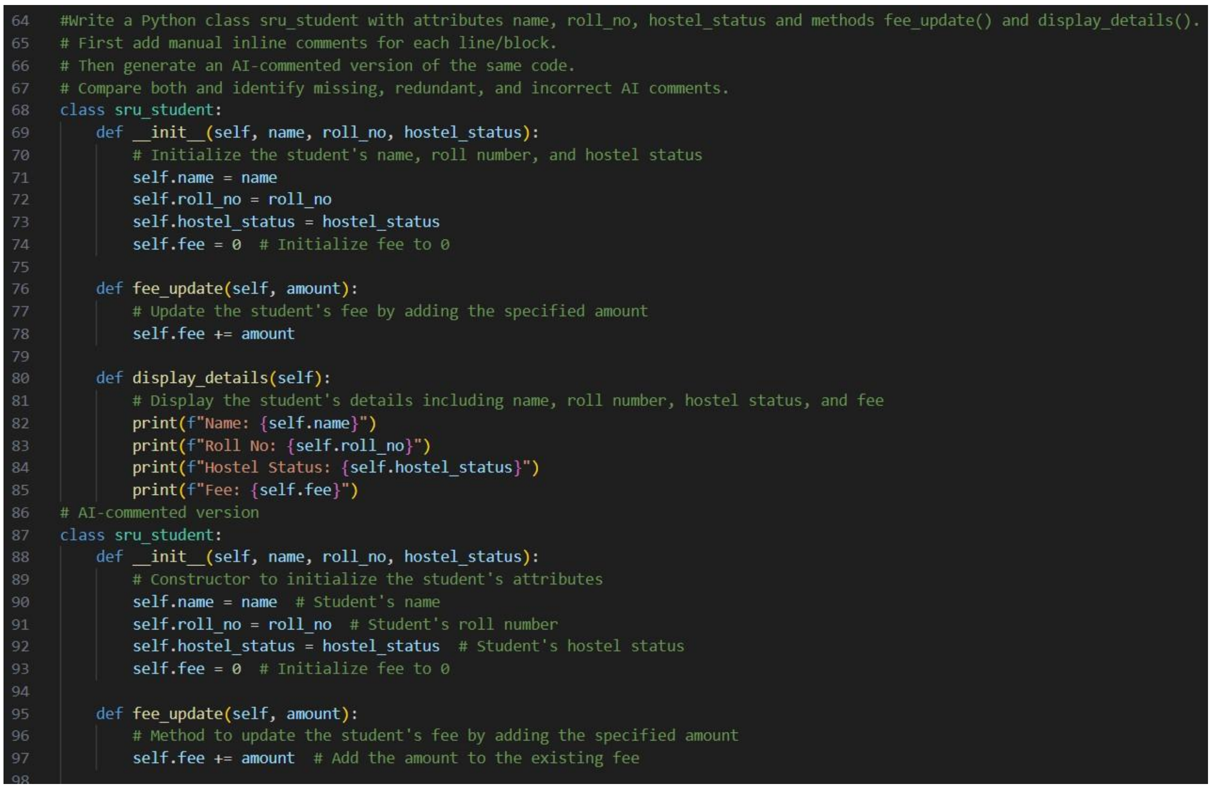
– Methods: fee_update() and display_details()

• Manually write inline comments for each line or logical block

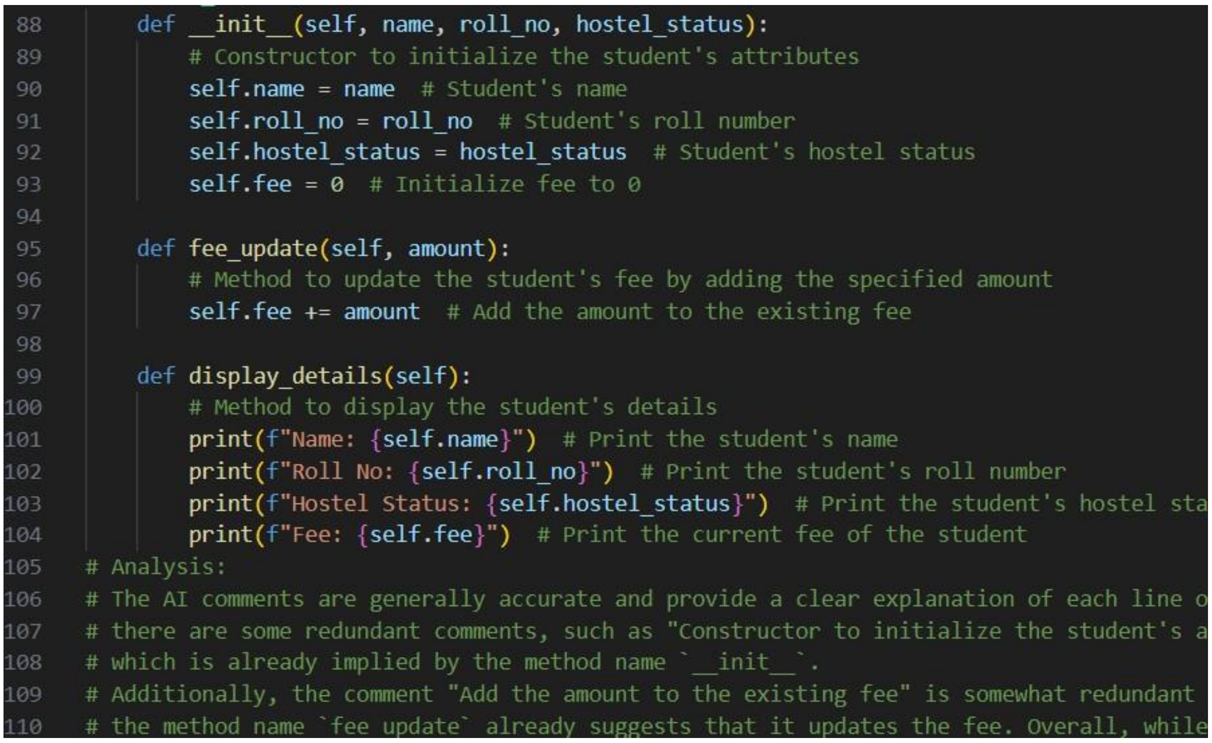
• Use an AI-assisted tool to automatically add inline comments

• Compare manual comments with AI-generated comments

• Identify missing, redundant, or incorrect AI comments



```python
#Write a Python class sru_student with attributes name, roll_no, hostel_status and methods fee_update() and display_details().
# First add manual inline comments for each line/block.
# Then generate an AI-commented version of the same code.
# Compare both and identify missing, redundant, and incorrect AI comments.
class sru_student:
    def __init__(self, name, roll_no, hostel_status):
        # Initialize the student's name, roll number, and hostel status
        self.name = name
        self.roll_no = roll_no
        self.hostel_status = hostel_status
        self.fee = 0  # Initialize fee to 0

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

    def display_details(self):
        # Display the student's details including name, roll number, hostel status, and fee
        print(f"Name: {self.name}")
        print(f"Roll No: {self.roll_no}")
        print(f"Hostel Status: {self.hostel_status}")
        print(f"Fee: {self.fee}")
# AI-commented version
class sru_student:
    def __init__(self, name, roll_no, hostel_status):
        # Constructor to initialize the student's attributes
        self.name = name  # Student's name
        self.roll_no = roll_no  # Student's roll number
        self.hostel_status = hostel_status  # Student's hostel status
        self.fee = 0  # Initialize fee to 0

    def fee_update(self, amount):
        # Method to update the student's fee by adding the specified amount
        self.fee += amount  # Add the amount to the existing fee
```



```python
    def __init__(self, name, roll_no, hostel_status):
        # Constructor to initialize the student's attributes
        self.name = name  # Student's name
        self.roll_no = roll_no  # Student's roll number
        self.hostel_status = hostel_status  # Student's hostel status
        self.fee = 0  # Initialize fee to 0

    def fee_update(self, amount):
        # Method to update the student's fee by adding the specified amount
        self.fee += amount  # Add the amount to the existing fee

    def display_details(self):
        # Method to display the student's details
        print(f"Name: {self.name}")  # Print the student's name
        print(f"Roll No: {self.roll_no}")  # Print the student's roll number
        print(f"Hostel Status: {self.hostel_status}")  # Print the student's hostel sta
        print(f"Fee: {self.fee}")  # Print the current fee of the student
# Analysis:
# The AI comments are generally accurate and provide a clear explanation of each line o
# there are some redundant comments, such as "Constructor to initialize the student's a
# which is already implied by the method name `__init__`.
# Additionally, the comment "Add the amount to the existing fee" is somewhat redundant
# the method name `fee_update` already suggests that it updates the fee. Overall, while
```

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

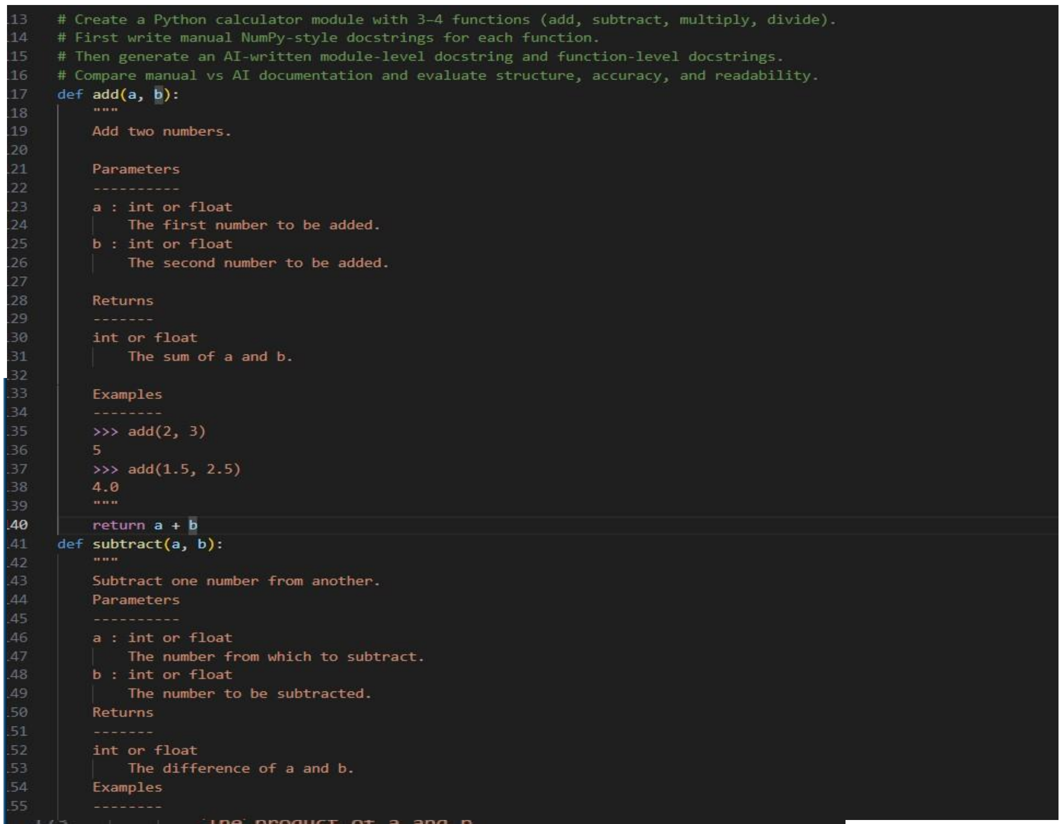
• Use AI assistance to generate:

– A module-level docstring

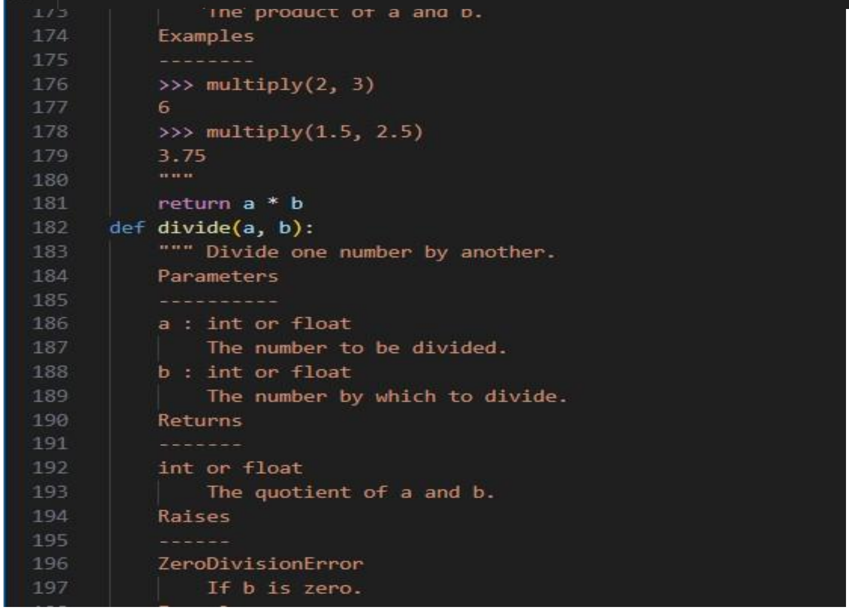
– Individual function-level docstrings

• Compare AI-generated docstrings with manually written ones

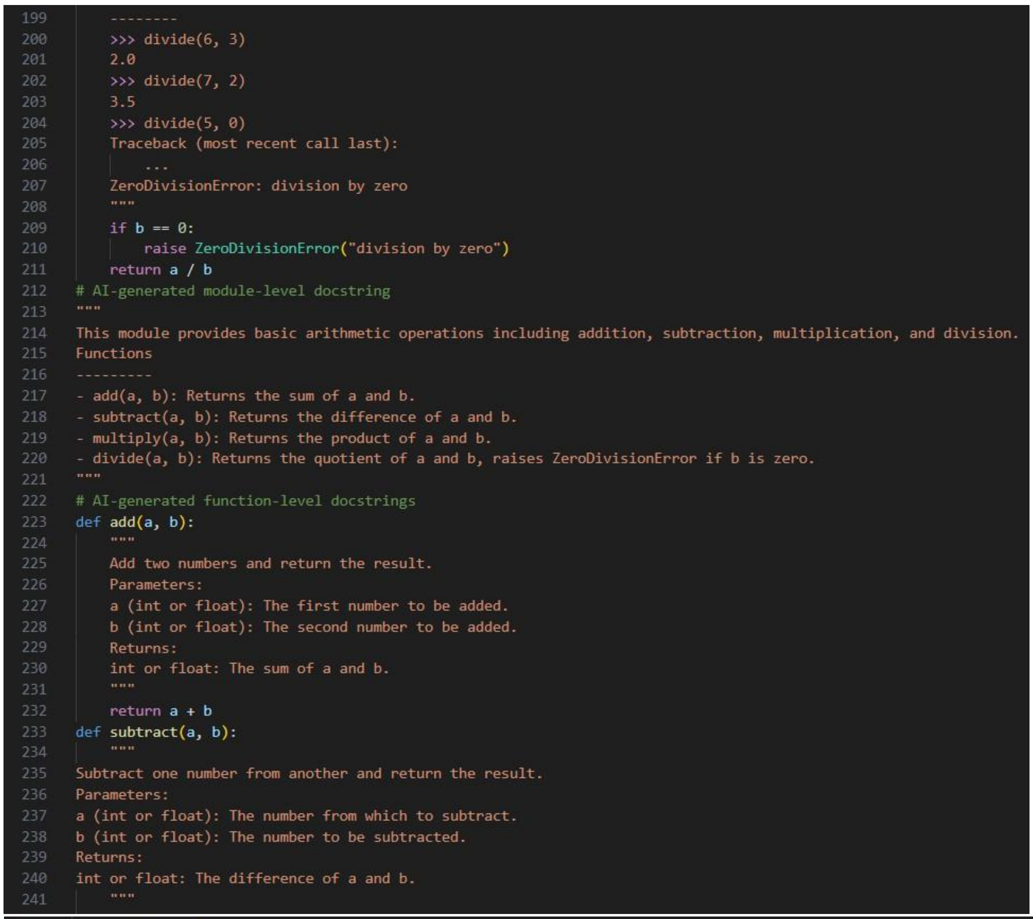
• Evaluate documentation structure, accuracy, and readability



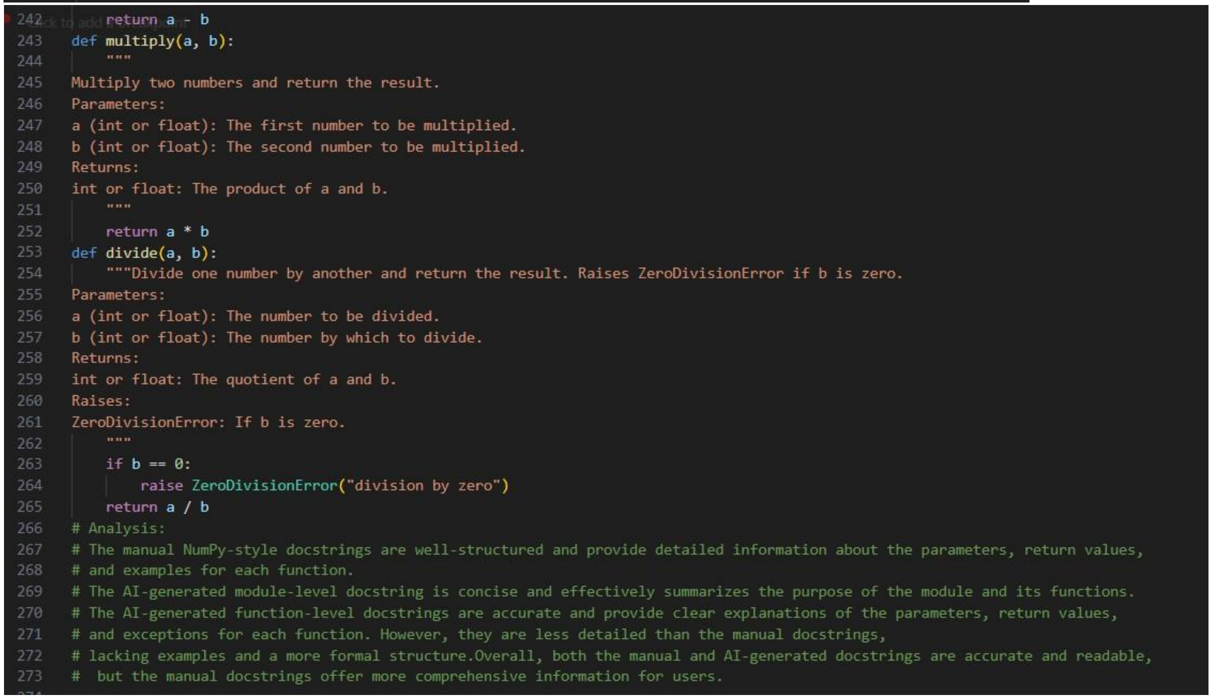
```python
# Create a Python calculator module with 3-4 functions (add, subtract, multiply, divide).
# First write manual NumPy-style docstrings for each function.
# Then generate an AI-written module-level docstring and function-level docstrings.
# Compare manual vs AI documentation and evaluate structure, accuracy, and readability.
def add(a, b):
    """
    Add two numbers.

    Parameters
    ----------
    a : int or float
        The first number to be added.
    b : int or float
        The second number to be added.

    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 from which to subtract.
    b : int or float
        The number to be subtracted.
    Returns
    -------
    int or float
        The difference of a and b.
    Examples
    --------
```



```python
        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 number to be divided.
    b : int or float
        The number by which to divide.
    Returns
    -------
    int or float
        The quotient of a and b.
    Raises
    ------
    ZeroDivisionError
        If b is zero.
```



```python
    --------
    >>> divide(6, 3)
    2.0
    >>> divide(7, 2)
    3.5
    >>> divide(5, 0)
    Traceback (most recent call last):
        ...
    ZeroDivisionError: division by zero
    """
    if b == 0:
        raise ZeroDivisionError("division by zero")
    return a / b
# AI-generated module-level docstring
"""
This module provides basic arithmetic operations including addition, subtraction, multiplication, and division.
Functions
---------
- add(a, b): Returns the sum of a and b.
- subtract(a, b): Returns the difference of a and b.
- multiply(a, b): Returns the product of a and b.
- divide(a, b): Returns the quotient of a and b, raises ZeroDivisionError if b is zero.
"""
# AI-generated function-level docstrings
def add(a, b):
    """
    Add two numbers and return the result.
    Parameters:
    a (int or float): The first number to be added.
    b (int or float): The second number to be added.
    Returns:
    int or float: The sum of a and b.
    """
    return a + b
def subtract(a, b):
    """
Subtract one number from another and return the result.
Parameters:
a (int or float): The number from which to subtract.
b (int or float): The number to be subtracted.
Returns:
int or float: The difference of a and b.
    """
```



```python
    return a - b
def multiply(a, b):
    """
Multiply two numbers and return the result.
Parameters:
a (int or float): The first number to be multiplied.
b (int or float): The second number to be multiplied.
Returns:
int or float: The product of a and b.
    """
    return a * b
def divide(a, b):
    """Divide one number by another and return the result. Raises ZeroDivisionError if b is zero.
Parameters:
a (int or float): The number to be divided.
b (int or float): The number by which to divide.
Returns:
int or float: The quotient of a and b.
Raises:
ZeroDivisionError: If b is zero.
    """
    if b == 0:
        raise ZeroDivisionError("division by zero")
    return a / b
# Analysis:
# The manual NumPy-style docstrings are well-structured and provide detailed information about the parameters, return values,
# and examples for each function.
# The AI-generated module-level docstring is concise and effectively summarizes the purpose of the module and its functions.
# The AI-generated function-level docstrings are accurate and provide clear explanations of the parameters, return values,
# and exceptions for each function. However, they are less detailed than the manual docstrings,
# lacking examples and a more formal structure.Overall, both the manual and AI-generated docstrings are accurate and readable,
#  but the manual docstrings offer more comprehensive information for users.
```