AI ASSISTED CODING

NAME : NAREDDY JASMITHA
ENROLLMENT NO : 2403A51281
BRANCH : CSE/AIML
SEM : 3RD
COURSE : AI CODING
DATE : 24-09-2025

Task Description#1

Basic Docstring Generation

- Write python function to return sum of even and odd numbers in the given list.
- Incorporate manual docstring in code with Google Style
- Use an AI-assisted tool (e.g., Copilot, Cursor AI) to generate a docstring describing the function.

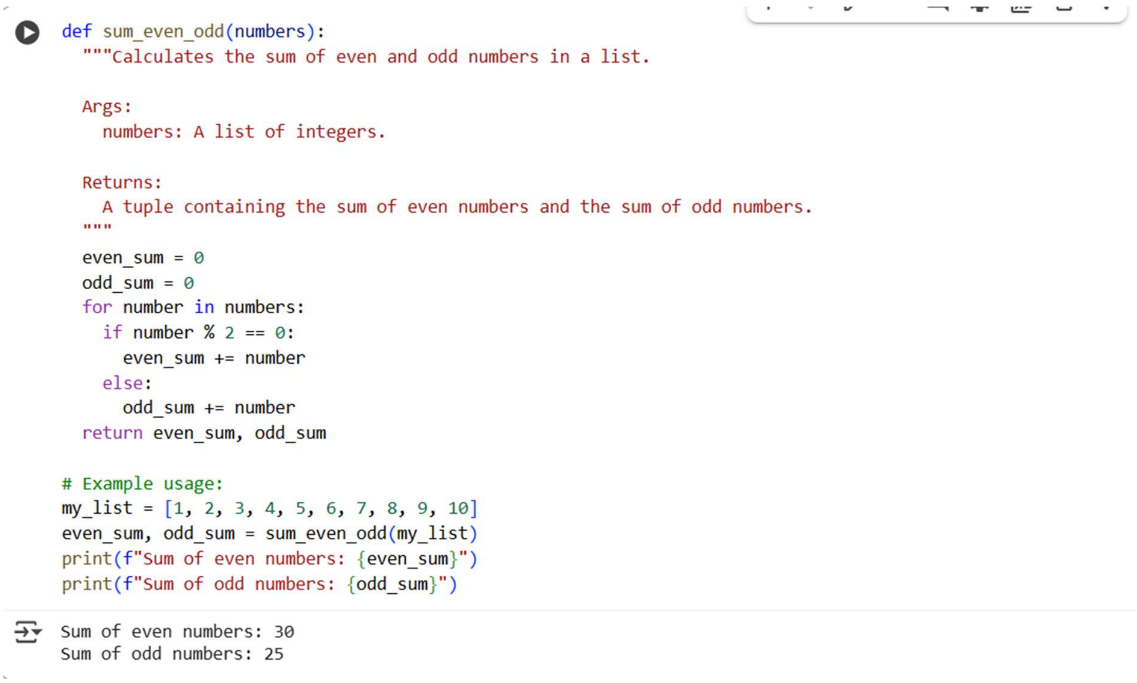
▪ Compare the AI-generated docstring with your manually written one

PROMPT:

Generate a code to develop an Incorporate manual docstring in code with Google Style ▪ Use an AI-assisted tool (e.g., Copilot, Cursor AI) to generate a docstring describing the function. ▪ Compare the AI-generated docstring with your manually written one

Expected Outcome#1:

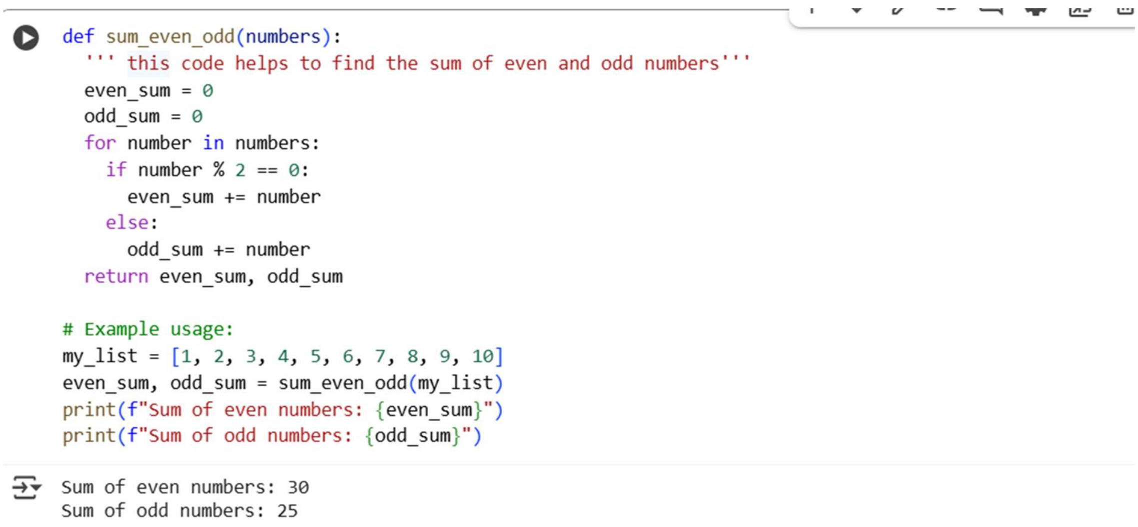
Students understand how AI can produce function-level documentation



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

  Args:
    numbers: A list of integers.

  Returns:
    A tuple containing the sum of even numbers and the sum of odd numbers.
  """
  even_sum = 0
  odd_sum = 0
  for number in numbers:
    if number % 2 == 0:
      even_sum += number
    else:
      odd_sum += number
  return even_sum, odd_sum

# Example usage:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum, odd_sum = sum_even_odd(my_list)
print(f"Sum of even numbers: {even_sum}")
print(f"Sum of odd numbers: {odd_sum}")
```

```
Sum of even numbers: 30
Sum of odd numbers: 25
```

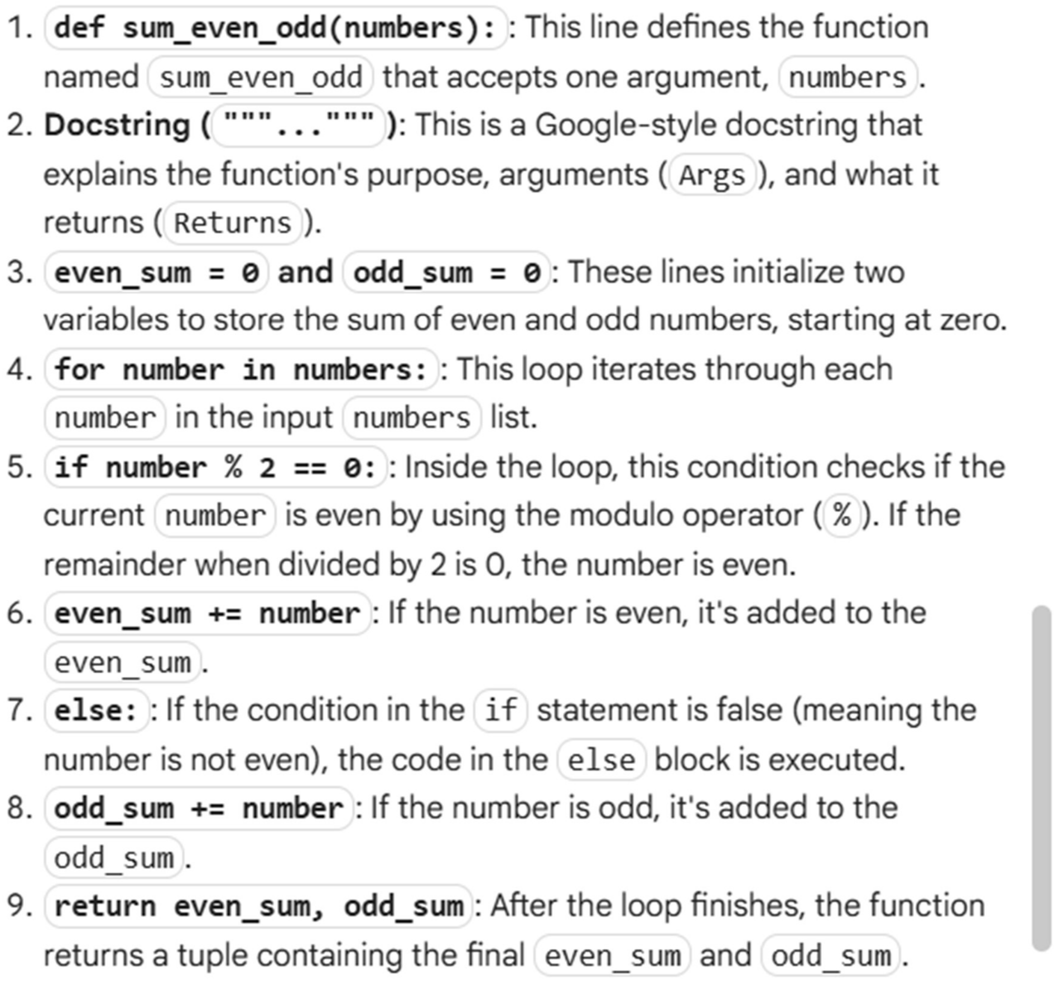
With ,my docstring



```python
def sum_even_odd(numbers):
  ''' this code helps to find the sum of even and odd numbers'''
  even_sum = 0
  odd_sum = 0
  for number in numbers:
    if number % 2 == 0:
      even_sum += number
    else:
      odd_sum += number
  return even_sum, odd_sum

# Example usage:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum, odd_sum = sum_even_odd(my_list)
print(f"Sum of even numbers: {even_sum}")
print(f"Sum of odd numbers: {odd_sum}")
```

```
Sum of even numbers: 30
Sum of odd numbers: 25
```

EXPLANATION :



1. **`def sum_even_odd(numbers):`**: This line defines the function named `sum_even_odd` that accepts one argument, `numbers`.
2. **Docstring (`"""..."""`)**: This is a Google-style docstring that explains the function's purpose, arguments (`Args`), and what it returns (`Returns`).
3. **`even_sum = 0`** and **`odd_sum = 0`**: These lines initialize two variables to store the sum of even and odd numbers, starting at zero.
4. **`for number in numbers:`**: This loop iterates through each `number` in the input `numbers` list.
5. **`if number % 2 == 0:`**: Inside the loop, this condition checks if the current `number` is even by using the modulo operator (`%`). If the remainder when divided by 2 is 0, the number is even.
6. **`even_sum += number`**: If the number is even, it's added to the `even_sum`.
7. **`else:`**: If the condition in the `if` statement is false (meaning the number is not even), the code in the `else` block is executed.
8. **`odd_sum += number`**: If the number is odd, it's added to the `odd_sum`.
9. **`return even_sum, odd_sum`**: After the loop finishes, the function returns a tuple containing the final `even_sum` and `odd_sum`.

Task Description#2

Automatic Inline Comments

- **Write python program for sru_student class with attributes like name, roll no., hostel_status and fee_update method and display_details method.**
- **Write comments manually for each line/code block**

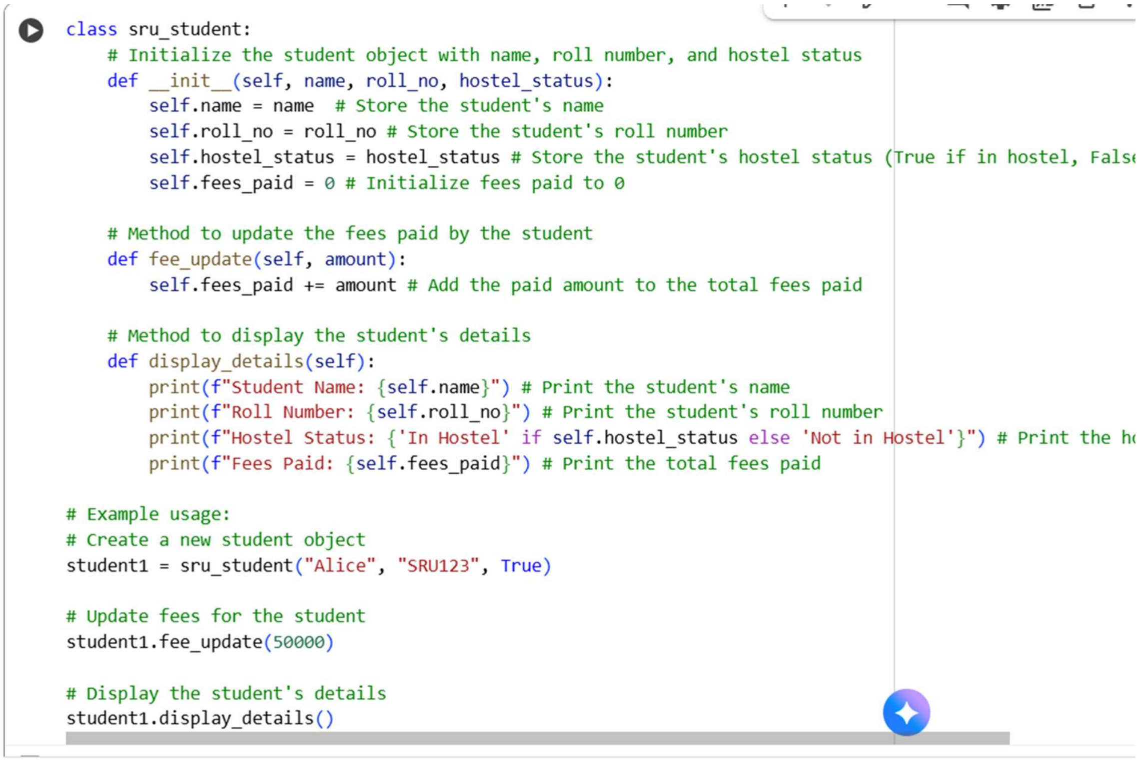
▪ Ask an AI tool to add inline comments explaining each line/step.
▪ Compare the AI-generated comments with your manually written one

PROMPT:

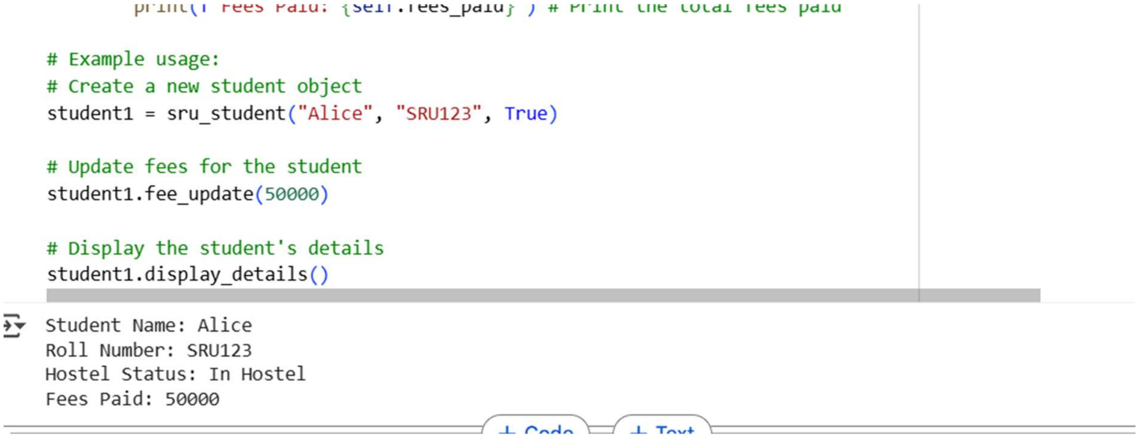
Develop a python code to develop python program for sru_student class with attributes like name, roll no., hostel_status and fee_update method and display_details method. ▪ Write comments manually for each line/code block ▪ Ask an AI tool to add inline comments explaining each line/step. ▪ Compare the AI-generated comments with your manually written one

Expected Output#2:

Students critically analyze AI-generated code comments



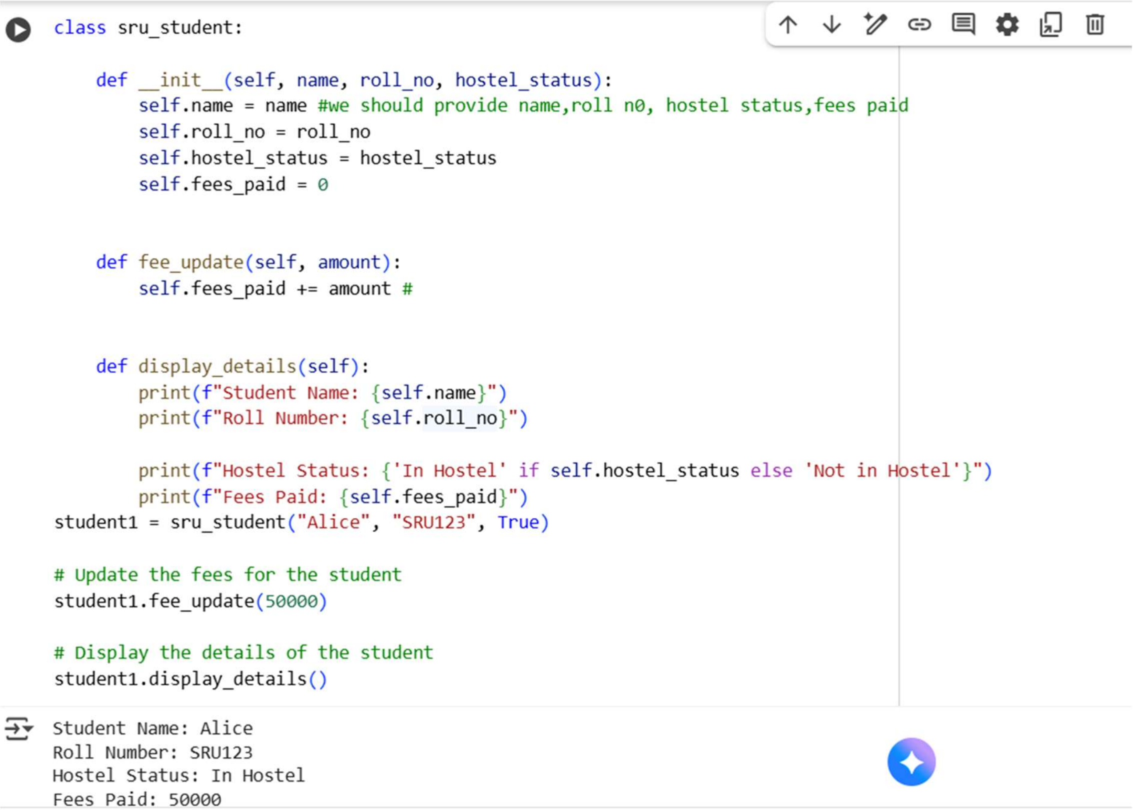
```
class sru_student:
    # Initialize the student object with name, roll number, and hostel status
    def __init__(self, name, roll_no, hostel_status):
        self.name = name  # Store the student's name
        self.roll_no = roll_no # Store the student's roll number
        self.hostel_status = hostel_status # Store the student's hostel status (True if in hostel, Fals
        self.fees_paid = 0 # Initialize fees paid to 0

    # Method to update the fees paid by the student
    def fee_update(self, amount):
        self.fees_paid += amount # Add the paid amount to the total fees paid

    # Method to display the student's details
    def display_details(self):
        print(f"Student Name: {self.name}") # Print the student's name
        print(f"Roll Number: {self.roll_no}") # Print the student's roll number
        print(f"Hostel Status: {'In Hostel' if self.hostel_status else 'Not in Hostel'}") # Print the h
        print(f"Fees Paid: {self.fees_paid}") # Print the total fees paid

# Example usage:
# Create a new student object
student1 = sru_student("Alice", "SRU123", True)

# Update fees for the student
student1.fee_update(50000)

# Display the student's details
student1.display_details()
```



```
        print(f"Fees Paid: {self.fees_paid}") # Print the total fees paid

# Example usage:
# Create a new student object
student1 = sru_student("Alice", "SRU123", True)

# Update fees for the student
student1.fee_update(50000)

# Display the student's details
student1.display_details()
```

```
Student Name: Alice
Roll Number: SRU123
Hostel Status: In Hostel
Fees Paid: 50000
```

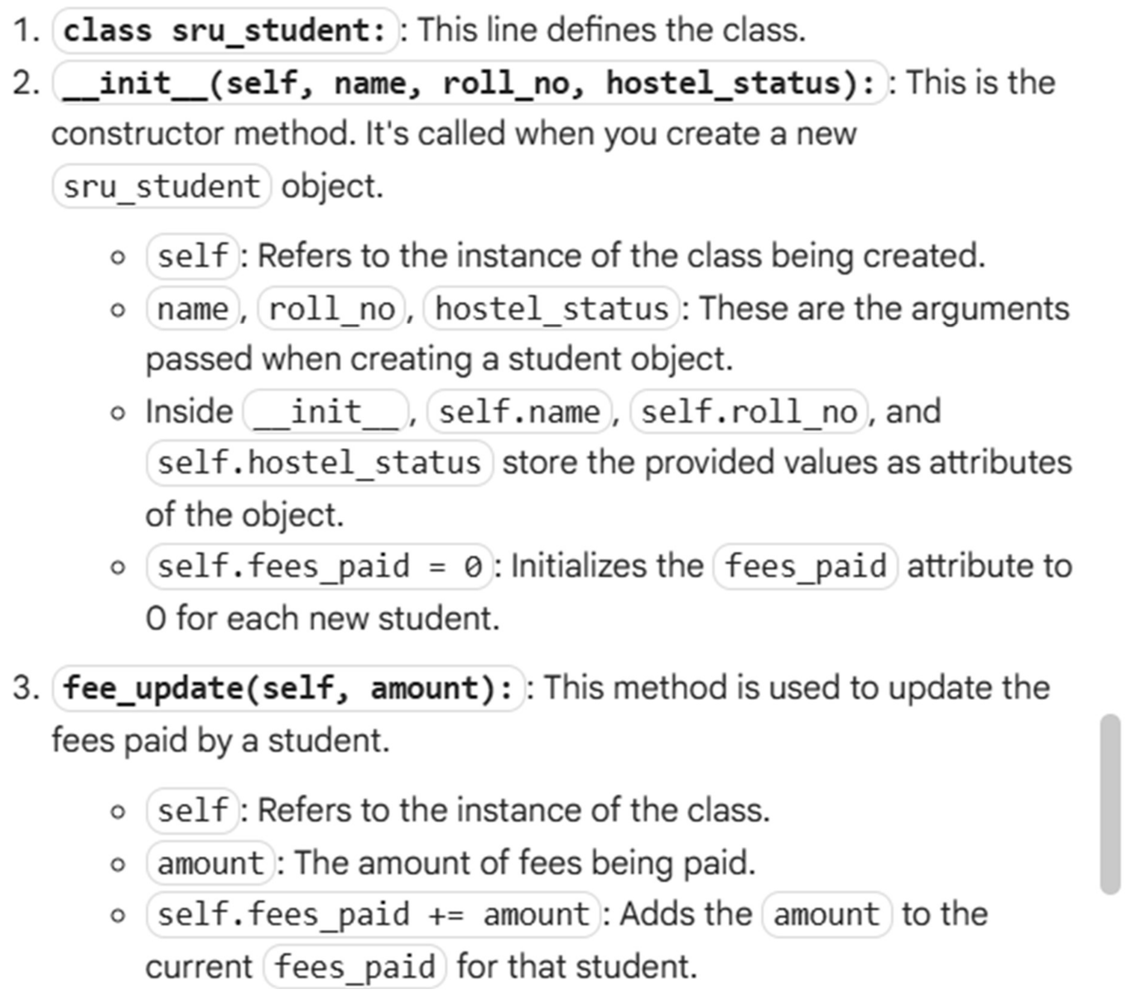
Without comments



```python
class sru_student:

    def __init__(self, name, roll_no, hostel_status):
        self.name = name #we should provide name,roll n0, hostel status,fees paid
        self.roll_no = roll_no
        self.hostel_status = hostel_status
        self.fees_paid = 0


    def fee_update(self, amount):
        self.fees_paid += amount #


    def display_details(self):
        print(f"Student Name: {self.name}")
        print(f"Roll Number: {self.roll_no}")

        print(f"Hostel Status: {'In Hostel' if self.hostel_status else 'Not in Hostel'}")
        print(f"Fees Paid: {self.fees_paid}")
student1 = sru_student("Alice", "SRU123", True)

# Update the fees for the student
student1.fee_update(50000)

# Display the details of the student
student1.display_details()
```

```
Student Name: Alice
Roll Number: SRU123
Hostel Status: In Hostel
Fees Paid: 50000
```

EXPLANATION:



1. **`class sru_student:`**: This line defines the class.
2. **`__init__(self, name, roll_no, hostel_status):`**: This is the constructor method. It's called when you create a new `sru_student` object.
    - `self`: Refers to the instance of the class being created.
    - `name`, `roll_no`, `hostel_status`: These are the arguments passed when creating a student object.
    - Inside `__init__`, `self.name`, `self.roll_no`, and `self.hostel_status` store the provided values as attributes of the object.
    - `self.fees_paid = 0`: Initializes the `fees_paid` attribute to 0 for each new student.
3. **`fee_update(self, amount):`**: This method is used to update the fees paid by a student.
    - `self`: Refers to the instance of the class.
    - `amount`: The amount of fees being paid.
    - `self.fees_paid += amount`: Adds the `amount` to the current `fees_paid` for that student.

Task Description#3

- **Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).**
- **Incorporate manual docstring in code with NumPy Style**
- **Use AI assistance to generate a module-level docstring + individual function**

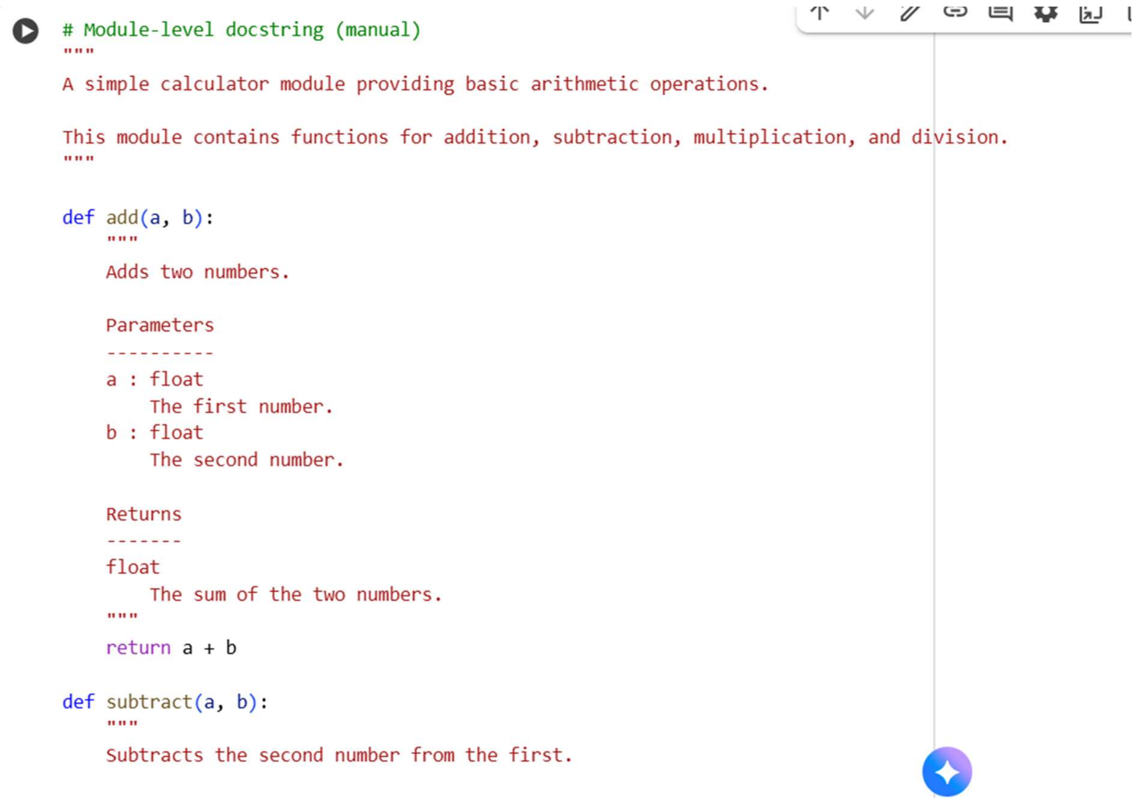
docstrings.
▪ Compare the AI-generated docstring with your manually written one

PROMPT:

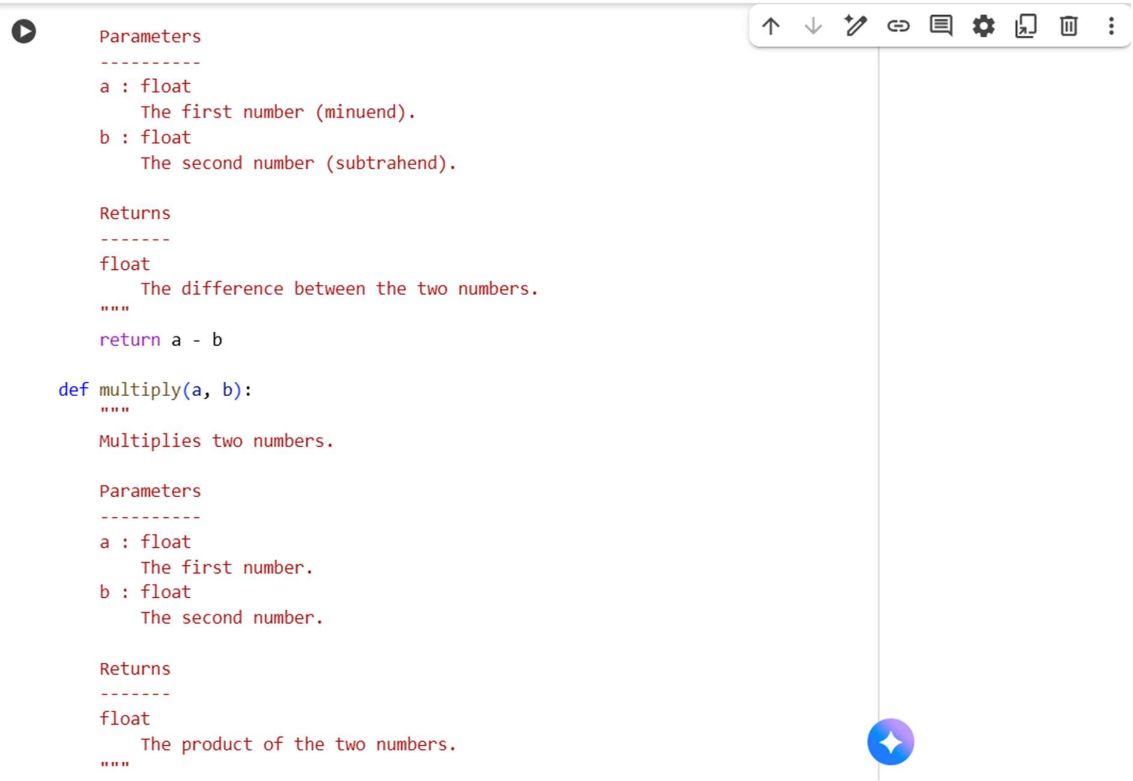
Generate a python code to develop functions (e.g., calculator: add, subtract, multiply, divide). ▪ Incorporate manual docstring in code with NumPy Style ▪ Use AI assistance to generate a module-level docstring + individual function docstrings. ▪ Compare the AI-generated docstring with your manually written one

Expected Output#3:

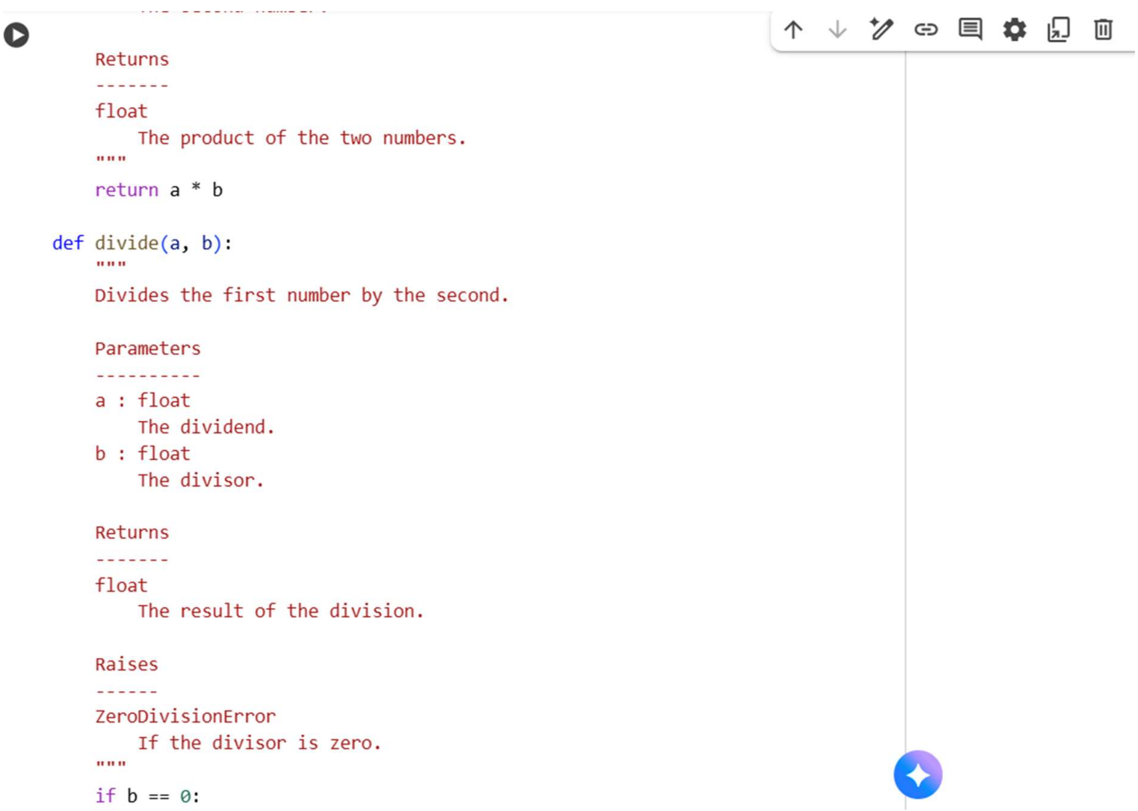
Students learn structured documentation for multi-function scripts
Push documentation whole workspace as .md file in GitHub Repository
Note: Report should be submitted a word document for all tasks in a single document with



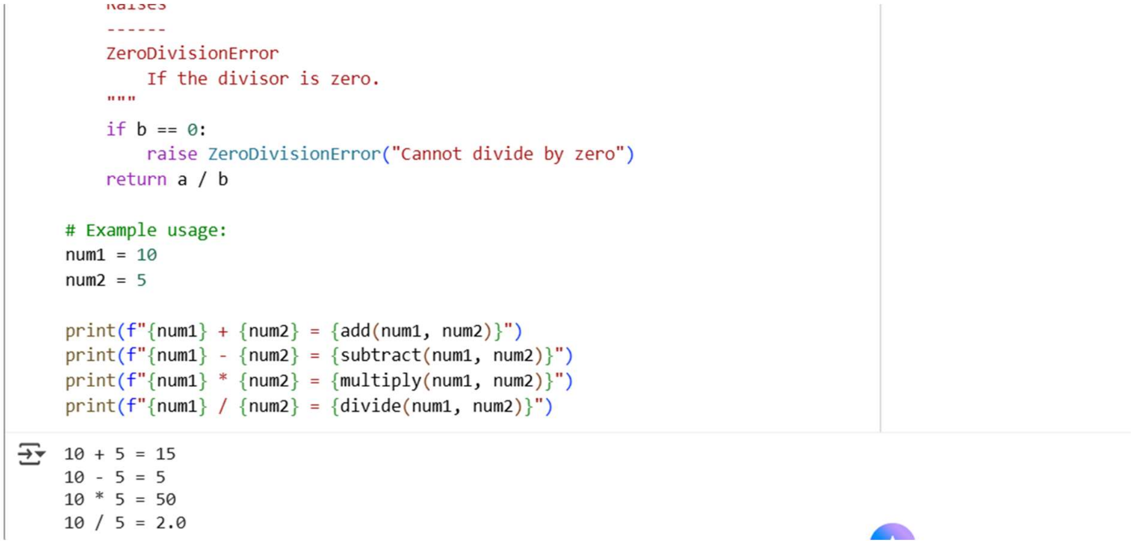
```
# Module-level docstring (manual)
"""
A simple calculator module providing basic arithmetic operations.

This module contains functions for addition, subtraction, multiplication, and division.
"""

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

    Parameters
    ----------
    a : float
        The first number.
    b : float
        The second number.

    Returns
    -------
    float
        The sum of the two numbers.
    """
    return a + b

def subtract(a, b):
    """
    Subtracts the second number from the first.
```



```
    Parameters
    ----------
    a : float
        The first number (minuend).
    b : float
        The second number (subtrahend).

    Returns
    -------
    float
        The difference between the two numbers.
    """
    return a - b

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

    Parameters
    ----------
    a : float
        The first number.
    b : float
        The second number.

    Returns
    -------
    float
        The product of the two numbers.
    """
```



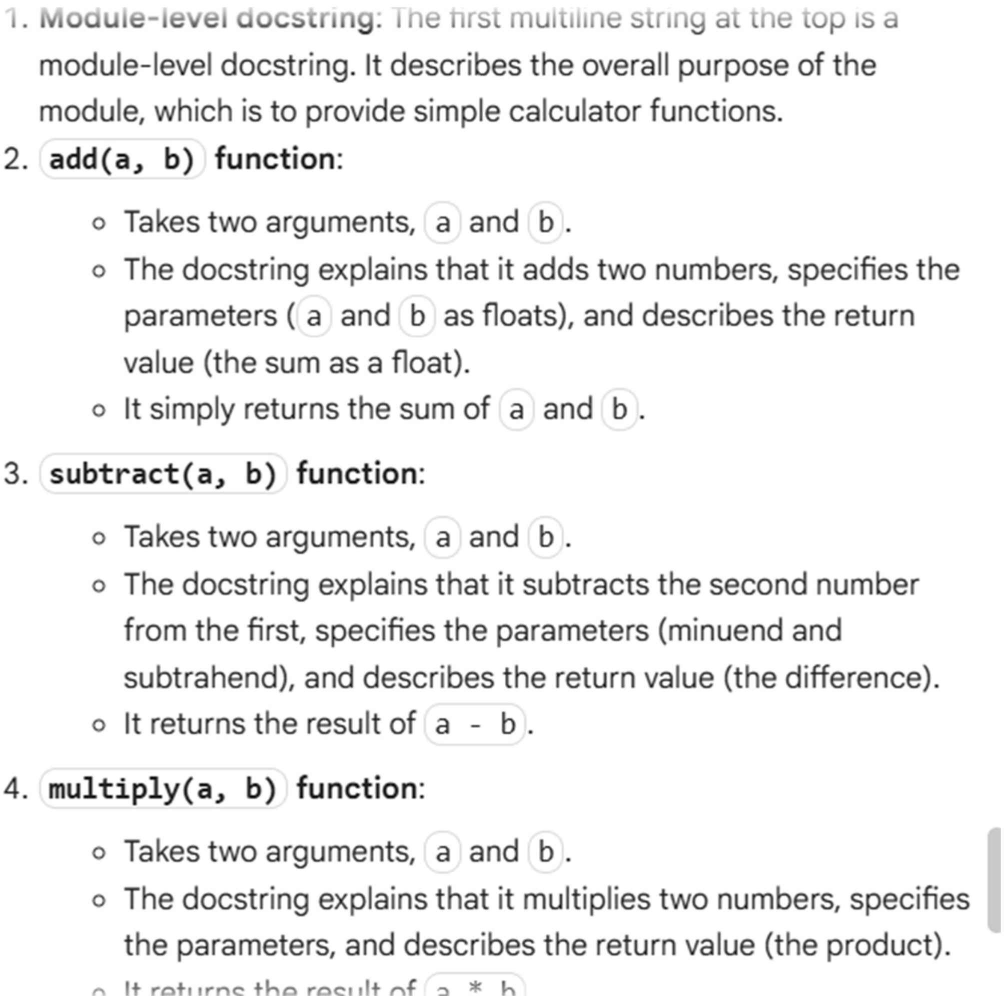
```
    Returns
    -------
    float
        The product of the two numbers.
    """
    return a * b

def divide(a, b):
    """
    Divides the first number by the second.

    Parameters
    ----------
    a : float
        The dividend.
    b : float
        The divisor.

    Returns
    -------
    float
        The result of the division.

    Raises
    ------
    ZeroDivisionError
        If the divisor is zero.
    """
    if b == 0:
```



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

# Example usage:
num1 = 10
num2 = 5

print(f"{num1} + {num2} = {add(num1, num2)}")
print(f"{num1} - {num2} = {subtract(num1, num2)}")
print(f"{num1} * {num2} = {multiply(num1, num2)}")
print(f"{num1} / {num2} = {divide(num1, num2)}")
```

```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.0
```

EXPLANATION:



1. **Module-level docstring:** The first multiline string at the top is a module-level docstring. It describes the overall purpose of the module, which is to provide simple calculator functions.
2. **`add(a, b)` function:**
    - Takes two arguments, `a` and `b`.
    - The docstring explains that it adds two numbers, specifies the parameters (`a` and `b` as floats), and describes the return value (the sum as a float).
    - It simply returns the sum of `a` and `b`.
3. **`subtract(a, b)` function:**
    - Takes two arguments, `a` and `b`.
    - The docstring explains that it subtracts the second number from the first, specifies the parameters (minuend and subtrahend), and describes the return value (the difference).
    - It returns the result of `a - b`.
4. **`multiply(a, b)` function:**
    - Takes two arguments, `a` and `b`.
    - The docstring explains that it multiplies two numbers, specifies the parameters, and describes the return value (the product).
    - It returns the result of `a * b`