AI ASSISTED CODING

NAME:B.NANDINI

ENROLL NUMBER: 2403A52390

BATCH NUMBER :14

Lab assignment-9.3

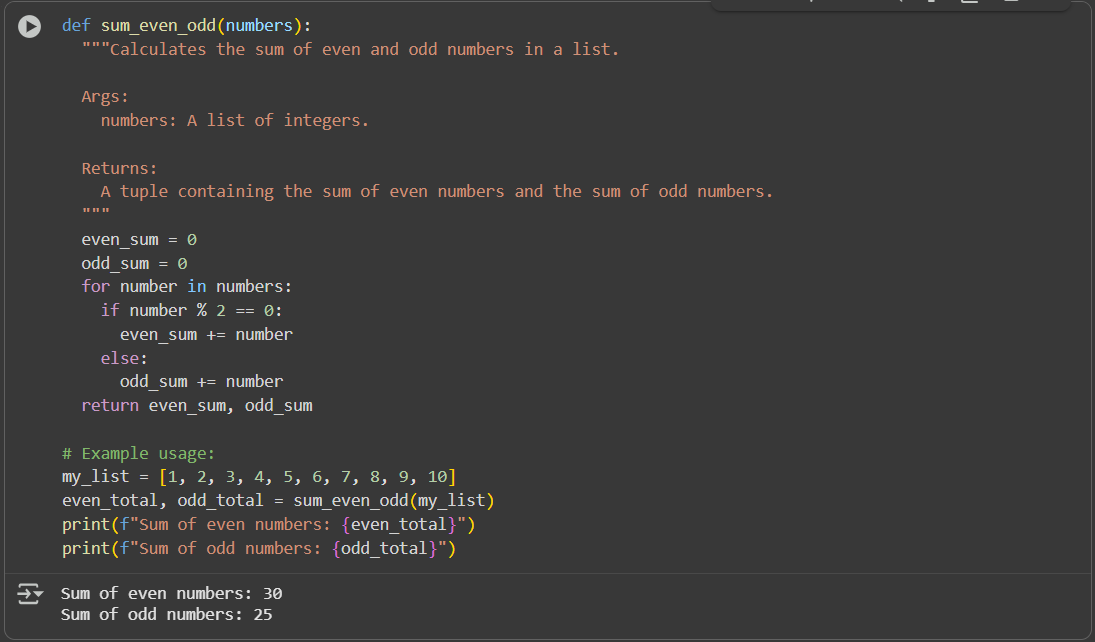
**Task Description#1**

**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., Gemini, Copilot, Cursor AI) to generate a docstring describing the function.

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

Code:



Code explanation:

1. **Function Definition:** def sum\_even\_odd(numbers): defines the function sum\_even\_odd that accepts one argument, numbers.
2. **Docstring:** The triple-quoted string is a docstring, explaining what the function does, its arguments (Args), and what it returns (Returns). This is written in Google Style for documentation.
3. **Initialization:** even\_sum = 0 and odd\_sum = 0 initialize two variables to store the sums of even and odd numbers, starting at zero.
4. **Iteration:** The for number in numbers: loop iterates through each number in the input numbers list.
5. **Even/Odd Check:** if number % 2 == 0: 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. **Summation:**
   * If the number is even, it's added to even\_sum (even\_sum += number).
   * If the number is odd (the else block), it's added to odd\_sum (odd\_sum += number).
7. **Return Value:** return even\_sum, odd\_sum returns a tuple containing the final even\_sum and odd\_sum.
8. **Example Usage:**
   * my\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] creates a sample list.
   * even\_total, odd\_total = sum\_even\_odd(my\_list) calls the function with my\_list and unpacks the returned tuple into even\_total and odd\_total.
   * print(f"Sum of even numbers: {even\_total}") and print(f"Sum of odd numbers: {odd\_total}") print the calculated sums.

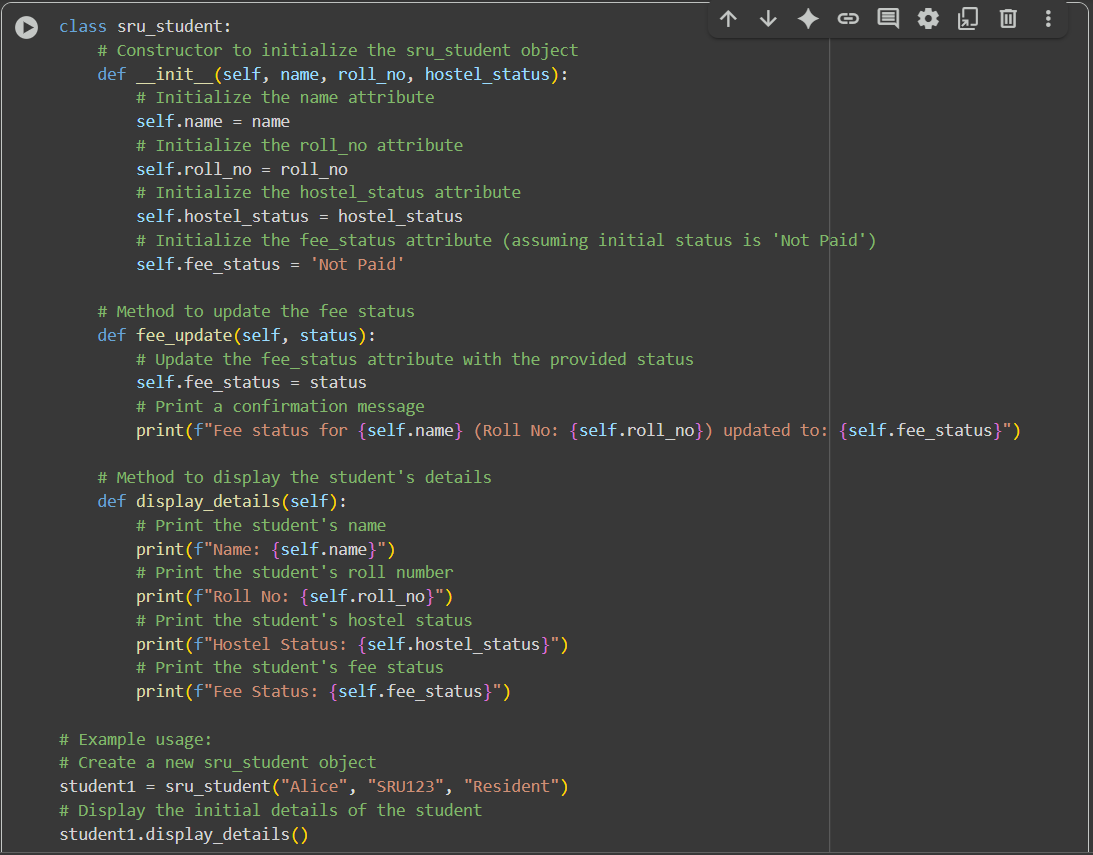
**Task Description#2**

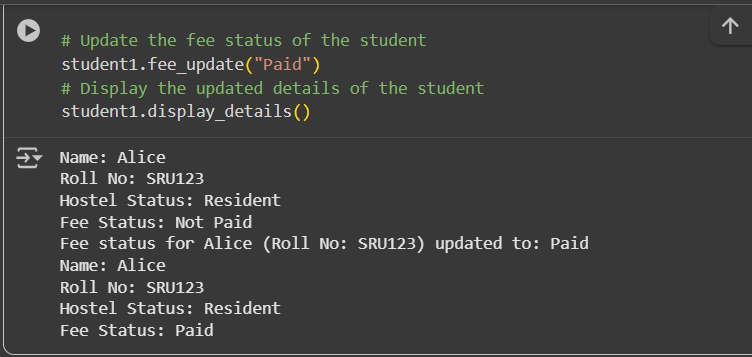
**Automatic Inline Comments**

* 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

Code:





Code explanation:

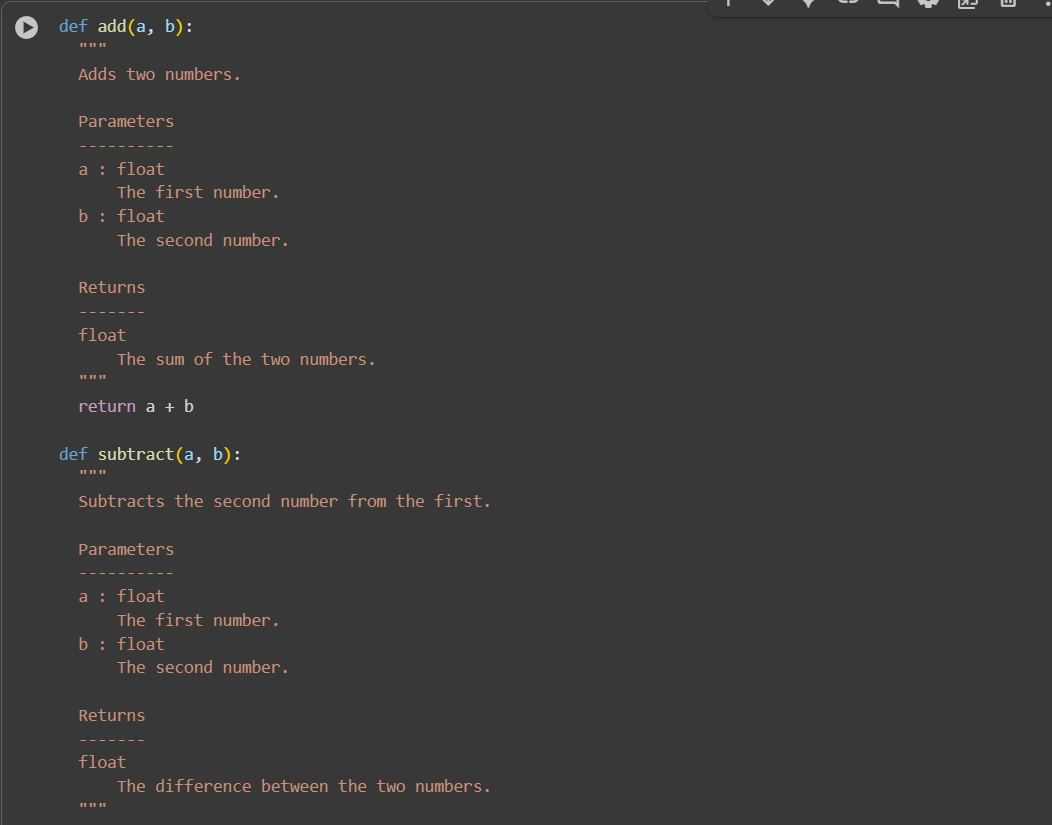
1. **Class Definition:** class sru\_student: defines a new class named sru\_student. This is a blueprint for creating student objects.
2. **Constructor (\_\_init\_\_)**:
   * def \_\_init\_\_(self, name, roll\_no, hostel\_status): is the constructor method. It's called when you create a new sru\_student object.
   * self refers to the instance of the class being created.
   * name, roll\_no, and hostel\_status are parameters passed when creating an object.
   * Inside \_\_init\_\_, self.name = name, self.roll\_no = roll\_no, and self.hostel\_status = hostel\_status assign the values passed in to the object's attributes (characteristics).
   * self.fee\_status = 'Not Paid' initializes the fee\_status attribute to 'Not Paid' by default when a new student object is created.
3. **fee\_update Method:**
   * def fee\_update(self, status): defines a method to update the student's fee status.
   * self refers to the object the method is called on.
   * status is the new fee status you want to set.
   * self.fee\_status = status updates the fee\_status attribute of the object.
   * print(f"Fee status for {self.name} (Roll No: {self.roll\_no}) updated to: {self.fee\_status}") prints a confirmation message showing the student's name, roll number, and the updated fee status.
4. **display\_details Method:**
   * def display\_details(self): defines a method to print the details of the student.
   * self refers to the object the method is called on.
   * The print() statements access the object's attributes (self.name, self.roll\_no, self.hostel\_status, self.fee\_status) and display them in a formatted string.
5. **Example Usage:**
   * student1 = sru\_student("Alice", "SRU123", "Resident") creates a new instance of the sru\_student class named student1 with the provided details.
   * student1.display\_details() calls the display\_details method on the student1 object to show its initial information.
   * student1.fee\_update("Paid") calls the fee\_update method on student1 to change the fee status to 'Paid'.
   * student1.display\_details() calls display\_details again to show the updated information for student1.

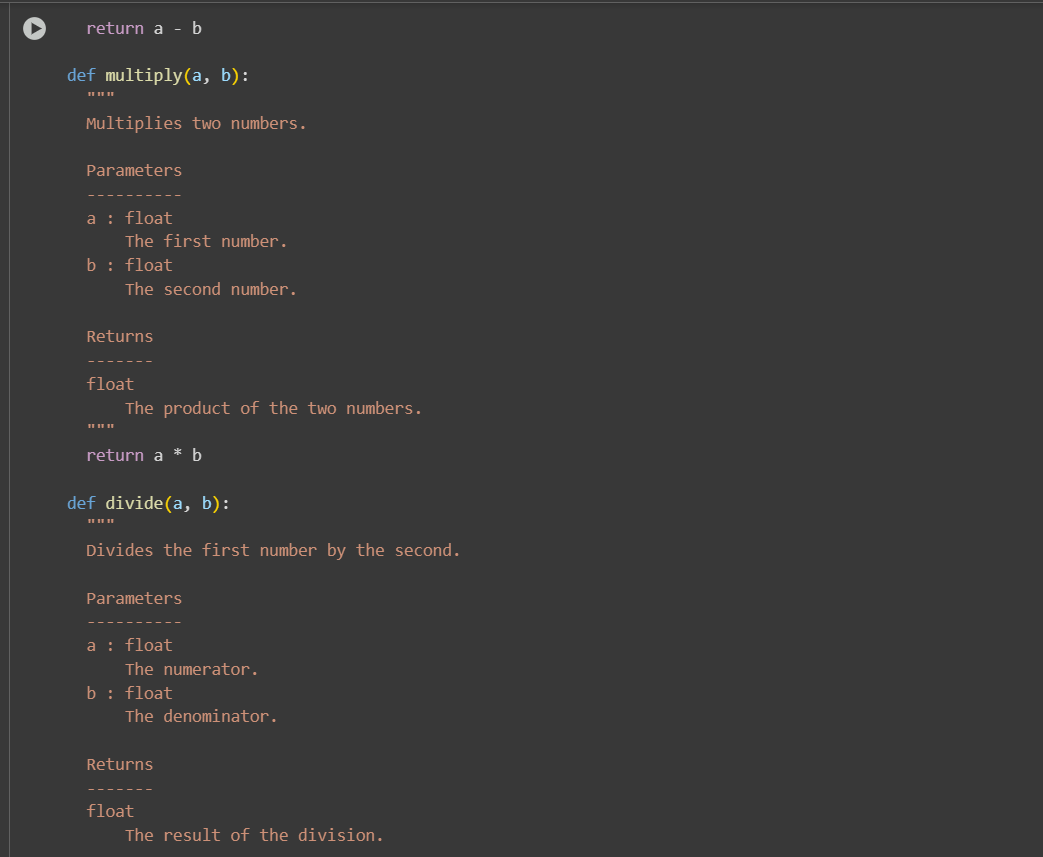
**Task Description#3**

* 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

Code:







Code explanation:

1. **add(a, b):**
   * Takes two arguments, a and b.
   * The docstring explains that it adds two numbers and describes the parameters (a and b) and what it returns (their sum). This docstring follows the NumPy style.
   * return a + b returns the sum of a and b.
2. **subtract(a, b):**
   * Takes two arguments, a and b.
   * The docstring explains that it subtracts b from a and describes the parameters and return value in NumPy style.
   * return a - b returns the difference between a and b.
3. **multiply(a, b):**
   * Takes two arguments, a and b.
   * The docstring explains that it multiplies two numbers and describes the parameters and return value in NumPy style.
   * return a \* b returns the product of a and b.
4. **divide(a, b):**
   * Takes two arguments, a and b.
   * The docstring explains that it divides a by b, describes the parameters and return value, and importantly, notes that it Raises a ZeroDivisionError if the denominator (b) is zero. This is a good practice to document potential errors.
   * if b == 0: checks if the denominator is zero.
   * raise ZeroDivisionError("Cannot divide by zero") raises an error if division by zero is attempted.
   * return a / b returns the result of the division if b is not zero.
5. **Example Usage:**
   * num1 = 10 and num2 = 5 assign values to variables.
   * The print() statements demonstrate how to call each function with num1 and num2 and display the results using f-strings for formatted output.
   * The commented-out try...except block shows how you would handle a potential ZeroDivisionError when attempting to divide by zero.