AI ASSISTED CODING

NAME: G.SANJANA

ENROLL NUMBER: 2403A52380

BATCH NUMBER :14

Lab assignment-9.3

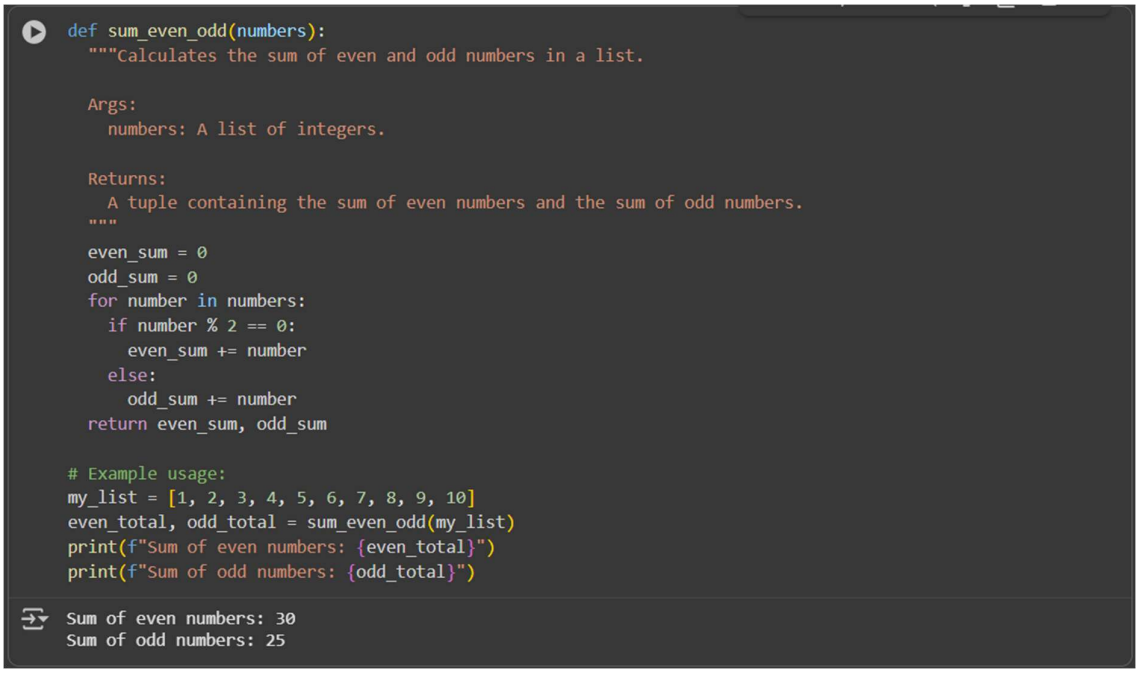
Task Description#1

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:



```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_total, odd_total = sum_even_odd(my_list)
print(f"Sum of even numbers: {even_total}")
print(f"Sum of odd numbers: {odd_total}")
```

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

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:

```python
class sru_student:
    # Constructor to initialize the sru_student object
    def __init__(self, name, roll_no, hostel_status):
        # Initialize the name attribute
        self.name = name
        # Initialize the roll_no attribute
        self.roll_no = roll_no
        # Initialize the hostel_status attribute
        self.hostel_status = hostel_status
        # Initialize the fee_status attribute (assuming initial status is 'Not Paid')
        self.fee_status = 'Not Paid'

    # Method to update the fee status
    def fee_update(self, status):
        # Update the fee_status attribute with the provided status
        self.fee_status = status
        # Print a confirmation message
        print(f"Fee status for {self.name} (Roll No: {self.roll_no}) updated to: {self.fee_status}")

    # Method to display the student's details
    def display_details(self):
        # Print the student's name
        print(f"Name: {self.name}")
        # Print the student's roll number
        print(f"Roll No: {self.roll_no}")
        # Print the student's hostel status
        print(f"Hostel Status: {self.hostel_status}")
        # Print the student's fee status
        print(f"Fee Status: {self.fee_status}")

# Example usage:
# Create a new sru_student object
student1 = sru_student("Alice", "SRU123", "Resident")
# Display the initial details of the student
student1.display_details()
```

```python
# Update the fee status of the student
student1.fee_update("Paid")
# Display the updated details of the student
student1.display_details()
```

```
Name: Alice
Roll No: SRU123
Hostel Status: Resident
Fee Status: Not Paid
Fee status for Alice (Roll No: SRU123) updated to: Paid
Name: Alice
Roll No: SRU123
Hostel Status: Resident
Fee Status: Paid
```

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.

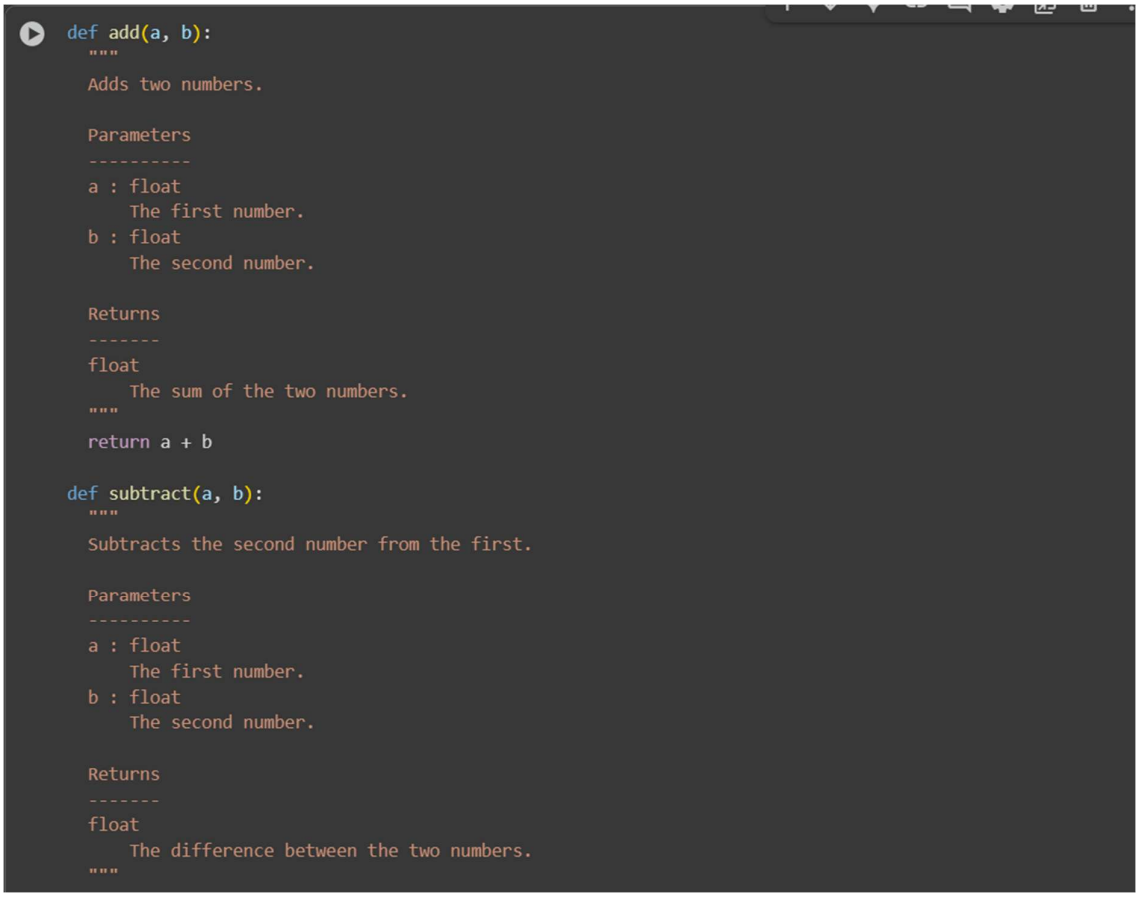
Task Description#3

- 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:



```python
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.
    b : float
        The second number.

    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.
    """
    return a * b

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

    Parameters
    ----------
    a : float
        The numerator.
    b : float
        The denominator.

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

```
    Raises
    ------
    ZeroDivisionError
        If the denominator 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)}")

# Example of division by zero
# try:
#     print(f"{num1} / 0 = {divide(num1, 0)}")
# except ZeroDivisionError as e:
#     print(e)
```

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

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.