ASSIGNMENT-9.3

TASK-1

Prompt: 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.

Expected Outcome#1: Students understand how AI can produce function-level documentation.

Code with output:

```
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"Original list: {my_list}")
print(f"Sum of even numbers: {even_sum}")
print(f"Sum of odd numbers: {odd_sum}")
```

```
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of even numbers: 30
Sum of odd numbers: 25
```

Explanation:

The code in the last cell defines a Python function called `sum_even_odd` that takes a list of numbers as input and calculates the sum of the even and odd numbers within that list.

Here's how it works:

1. **Initialization:**
   - `even_sum = 0`: A variable `even_sum` is initialized to 0 to store the sum of even numbers.
   - `odd_sum = 0`: A variable `odd_sum` is initialized to 0 to store the sum of odd numbers.
2. **Iteration:**
   - `for number in numbers:`: The code iterates through each `number` in the input `numbers` list.
3. **Checking for Even or Odd:**
   - `if number % 2 == 0:`: Inside the loop, it 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.
   - `even_sum += number`: If the number is even, it is added to the `even_sum`.
   - `else:`: If the number is not even (i.e., it's odd).
   - `odd_sum += number`: The number is added to the `odd_sum`.
4. **Return Value:**
   - `return even_sum, odd_sum`: After iterating through all the numbers in the list, the function returns a tuple containing the final `even_sum` and `odd_sum`.
5. **Example Usage:**
   - `my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`: A sample list `my_list` is created.
   - `even_sum, odd_sum = sum_even_odd(my_list)`: The `sum_even_odd` function is called with `my_list`, and the returned tuple is unpacked into the `even_sum` and `odd_sum` variables.
   - The original list, the calculated sum of even numbers, and the calculated sum of odd numbers are then printed.

TASK-2

Prompt: 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 with ouput:

```python
class sru_student:
    # Constructor to initialize the 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 fees attribute to 0
        self.fees = 0

    # Method to update the student's fees
    def fee_update(self, amount):
        # Add the specified amount to the current fees
        self.fees += amount
        # Print a confirmation message
        print(f"Fees updated for {self.name}. Current fees: {self.fees}")

    # Method to display the student's details
    def display_details(self):
        # Print the student's name
        print(f"Student Name: {self.name}")
        # Print the student's roll number
        print(f"Roll Number: {self.roll_no}")
        # Print the student's hostel status
        print(f"Hostel Status: {self.hostel_status}")
        # Print the student's current fees
        print(f"Current Fees: {self.fees}")

# Example Usage:
# Create an instance of the sru_student class
student1 = sru_student("nandhu", "SRU123", "Resident")
# Display the initial details of the student
student1.display_details()
```

```python
# Display the initial details of the student
student1.display_details()
# Update the student's fees
student1.fee_update(50000)
# Display the updated details of the student
student1.display_details()

# Create another instance of the sru_student class
student2 = sru_student("Bob", "SRU456", "Day Scholar")
# Display the initial details of the second student
student2.display_details()
# Update the second student's fees
student2.fee_update(10000)
# Display the updated details of the second student
student2.display_details()
```

```
Student Name: nandhu
Roll Number: SRU123
Hostel Status: Resident
Current Fees: 0
Fees updated for nandhu. Current fees: 50000
Student Name: nandhu
Roll Number: SRU123
Hostel Status: Resident
Current Fees: 50000
Student Name: Bob
Roll Number: SRU456
Hostel Status: Day Scholar
Current Fees: 0
Fees updated for Bob. Current fees: 10000
Student Name: Bob
Roll Number: SRU456
Hostel Status: Day Scholar
Current Fees: 10000
```

Explanation:

This code defines a Python class called `sru_student` to represent student information and manage their fees.

Here's a breakdown of the class and its components:

- `class sru_student:` : This line defines a new class named `sru_student`.
- `__init__(self, name, roll_no, hostel_status)` : This is the constructor method. It's called when you create a new object (instance) of the `sru_student` class.
  - `self` : Refers to the instance of the class itself.
  - `name` , `roll_no` , `hostel_status` : These are parameters that you pass when creating a student object to initialize the student's name, roll number, and hostel status.
  - `self.name = name` : Assigns the value of the `name` parameter to the `name` attribute of the object.
  - `self.roll_no = roll_no` : Assigns the value of the `roll_no` parameter to the `roll_no` attribute of the object.
  - `self.hostel_status = hostel_status` : Assigns the value of the `hostel_status` parameter to the `hostel_status` attribute of the object.
  - `self.fees = 0` : Initializes a `fees` attribute for the student object and sets it to 0.
- `fee_update(self, amount)` : This method is used to update the student's fees.
  - `self` : Refers to the instance of the class.
  - `amount` : The amount to add to the student's fees.
  - `self.fees += amount` : Adds the specified `amount` to the current `fees` of the student object.
  - `print(...)` : Prints a confirmation message showing the updated fees.
- `display_details(self)` : This method is used to display the student's details.
  - `self` : Refers to the instance of the class.
  - `print(...)` : These lines print the student's name, roll number, hostel status, and current fees.
- **Example Usage:** The code then demonstrates how to create instances of the `sru_student` class (`student1` and `student2`), call the `display_details` method to show their initial information, call the `fee_update` method to modify their fees, and then call `display_details` again to show the updated information.

TASK-3

Prompt: 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 with output:

```python
# This is a placeholder for a module-level docstring in NumPy Style.
# An AI tool can be used to generate a more comprehensive one.

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

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

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

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

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

    Returns
    -------
    float or int
        The difference between the two numbers.
```

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

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

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

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

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

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

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

```
        The result of the division.

    Raises
    ------
    ZeroDivisionError
        If the divisor is zero.
    """
    if b == 0:
        raise ZeroDivisionError("Division by zero is not allowed.")
    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 ZeroDivisionError
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
Division by zero is not allowed.
```

Explanation:

This code defines a Python script with four basic arithmetic functions: `add`, `subtract`, `multiply`, and `divide`. Each function includes a docstring in NumPy style, explaining its purpose, arguments (Parameters), return value (Returns), and potential errors (Raises).

Here's a breakdown of each function:

- `add(a, b)`:
  - Takes two numbers, `a` and `b`, as input.
  - Returns their sum.
- `subtract(a, b)`:
  - Takes two numbers, `a` and `b`, as input.
  - Returns the result of subtracting `b` from `a`.
- `multiply(a, b)`:
  - Takes two numbers, `a` and `b`, as input.
  - Returns their product.
- `divide(a, b)`:
  - Takes two numbers, `a` (dividend) and `b` (divisor), as input.
  - Returns the result of dividing `a` by `b`.
  - Includes error handling to raise a `ZeroDivisionError` if the divisor (`b`) is zero.

The example usage at the end demonstrates how to call these functions with sample numbers and prints the results. It also includes a `try-except` block to show how the `ZeroDivisionError` is handled when attempting to divide by zero.