ASSIGNMENT_9.4

Name : Chirra Jyothika

Ht.no:2303A51280

Batch:05

Task 1: Basic Docstring Generation

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

PROMPT:

Generate a Python function that takes a list of integers and returns two values: the sum of even numbers,the sum of odd numbers. Add a Google-style docstring to the function explaining: what the function does its parameters return values The docstring should be clear, correct, and complete.

CODE: (Using AI Tools)

```python
def sum_even_odd(numbers: list[int]) -> tuple[int, int]:
    """Calculates the sum of even and odd numbers in a list of integers.

    This function iterates through a given list of integers and computes
    the total sum of all even numbers and the total sum of all odd numbers
    present in the list.

    Args:
        numbers: A list of integers.

    Returns:
        A tuple containing two integers:
        - The sum of all even numbers in the list.
        - The sum of all odd numbers in the list.
    """
    even_sum = 0
    odd_sum = 0
    for num in numbers
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num
    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"Original list: {my_list}")
print(f"Sum of even numbers: {even_total}")
print(f"Sum of odd numbers: {odd_total}")

my_empty_list = []
even_empty, odd_empty = sum_even_odd(my_empty_list)
print(f"\nOriginal list: {my_empty_list}")
print(f"Sum of even numbers (empty list): {even_empty}")
print(f"Sum of odd numbers (empty list): {odd_empty}")
```

OUTPUT:

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

Original list: []
Sum of even numbers (empty list): 0
Sum of odd numbers (empty list): 0
```

CODE: (Using Manual)

```
def sum_even_odd(numbers):
    """Generates list of numbers for sum of odd numbers and sum of even numbers"""
    even_sum = 0
    odd_sum = 0
    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num

    return even_sum, odd_sum
n = int(input("Enter number of elements: "))
numbers = []
print("Enter the numbers:")
for _ in range(n):
    numbers.append(int(input()))
even, odd = sum_even_odd(numbers)
print("Sum of even numbers:", even)
print("Sum of odd numbers:", odd)
```

OUTPUT:

```
Enter number of elements: 6
Enter the numbers:
3
45
32
12
55
34
Sum of even numbers: 78
Sum of odd numbers: 103
```

EXPLANATION:

Aspect	AI-Generated Docstring	Manual Docstring
Clarity	Very clear and well-explained	Brief and unclear
Correctness	Correctly describes input and output	Partially incorrect description
Completeness	Fully complete with details	Incomplete, missing details

Task 2: Automatic Inline Comments

Scenario:

You are developing a student management module that must be easy to understand for new developers.

PROMPT:

Write a Python program to create an sru_student class with attributes name, roll_no, and hostel_status, and methods fee_update() and display_details() with comments

CODE: (Using Tools)

```
class sru_student:
    # Represents a student at SRU with basic information and fee management.

    def __init__(self, name: str, roll_no: str, hostel_status: bool):
        # Initializes a new sru_student object.
        # Assigning input parameters to instance attributes
        self.name = name
        self.roll_no = roll_no
        self.hostel_status = hostel_status
        # Initialize fees_paid to 0 for a new student
        self.fees_paid = 0

    def fee_update(self, amount: float):
        # Updates the amount of fees paid by the student.
        # Ensure the amount to update is positive
        if amount > 0:
            # Add the amount to the total fees paid
            self.fees_paid += amount
            print(f"Fees updated for {self.name}. New fees paid: {self.fees_paid}")
        else:
            # Inform the user if the amount is not valid
            print("Amount must be positive to update fees.")

    def display_details(self):
        # Displays the details of the student.
        # Print a header for student details
        print("\n--- Student Details ---")
        # Display each attribute of the student
        print(f"Name: {self.name}")
        print(f"Roll No: {self.roll_no}")
        # Use a conditional expression to display 'Hosteler' or 'Day Scholar'
        print(f"Hostel Status: {'Hosteler' if self.hostel_status else 'Day Scholar'}")
        print(f"Fees Paid: {self.fees_paid}")
        # Print a footer
        print("-----------------------")

# Example Usage:
# --- Create student objects ---
# Create the first student, 'Alice Smith', with roll number 'SRU101' and as a hosteler (True)
student1 = sru_student("Alice Smith", "SRU101", True)
# Create the second student, 'Bob Johnson', with roll number 'SRU102' and as a day scholar (False)
student2 = sru_student("Bob Johnson", "SRU102", False)

# --- Display initial details of students ---
student1.display_details()
student2.display_details()

# --- Update fees for students ---
# Update fees for student1 by 5000.0
student1.fee_update(5000.0)
# Update fees for student2 by 2500.0
student2.fee_update(2500.0)
# Further update fees for student1 by 3000.0
student1.fee_update(3000.0)

# --- Display updated details of students ---
student1.display_details()
student2.display_details()
```

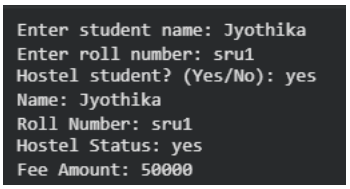
OUTPUT:

```
--- Student Details ---
Name: Alice Smith
Roll No: SRU101
Hostel Status: Hosteler
Fees Paid: 0
----------------------

--- Student Details ---
Name: Bob Johnson
Roll No: SRU102
Hostel Status: Day Scholar
Fees Paid: 0
----------------------
Fees updated for Alice Smith. New fees paid: 5000.0
Fees updated for Bob Johnson. New fees paid: 2500.0
Fees updated for Alice Smith. New fees paid: 8000.0

--- Student Details ---
Name: Alice Smith
Roll No: SRU101
Hostel Status: Hosteler
Fees Paid: 8000.0
----------------------

--- Student Details ---
Name: Bob Johnson
Roll No: SRU102
Hostel Status: Day Scholar
Fees Paid: 2500.0
```

CODE: (Using manual comments)

```python
# This class stores student details
class sru_student:

    # This function runs when a new student object is created
    def __init__(self, name, roll_no, hostel_status):
        self.name = name            # saving the student name
        self.roll_no = roll_no      # saving the roll number
        self.hostel_status = hostel_status  # saving hostel information

    # This function decides the fee based on hostel status
    def fee_update(self):
        # If the student stays in hostel, fee is higher
        if self.hostel_status.lower() == "yes":
            self.fee = 50000
        else:
            # Otherwise, normal fee is applied
            self.fee = 30000

        # Display the final fee
        print("Fee Amount:", self.fee)

    # This function prints all student details
    def display_details(self):
        print("Name:", self.name)
        print("Roll Number:", self.roll_no)
        print("Hostel Status:", self.hostel_status)


# -------- Program starts here --------

# Taking input from the user
name = input("Enter student name: ")
roll_no = input("Enter roll number: ")
hostel_status = input("Hostel student? (Yes/No): ")

# Creating an object for the student
student = sru_student(name, roll_no, hostel_status)

# Showing student information
student.display_details()

# Calculating and showing fee
student.fee_update()
```

OUTPUT:



EXPLANATION

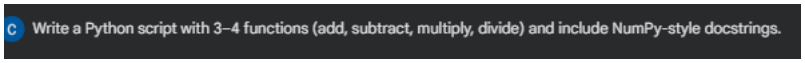
Aspect	Manual Comments	AI-Generated Comments	Issue Identified
Fee Logic	Correctly explains hostel-based fee	Talks about "fees paid"	Incorrect AI comments
Hostel Status	Uses Yes/No input	Uses Boolean True/False	Incorrect AI assumption
User Input	Clearly explained	Not explained	Missing AI comment
Code Lines	Comments only where needed	Comments on obvious lines	Redundant AI comments
Requirements Match	Fully aligned	Partially mismatched	Incorrect / missing explanations

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.

PROMPT:



CODE: (Using Tools)

```python
def add(a: float, b: float) -> float:
    """Adds two numbers.

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

    Returns
    -------
    float
        The sum of `a` and `b`.

    Examples
    --------
    >>> add(5, 3)
    8.0
    >>> add(-1, 10)
    9.0
    """
    return a + b

def subtract(a: float, b: float) -> float:
    """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 `a` and `b`.

    Examples
    --------
    >>> subtract(10, 4)
    6.0
    >>> subtract(5, 7)
    -2.0
    """
    return a - b

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

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

        Returns
        -------
        float
            The product of `a` and `b`.

        Examples
        --------
        >>> multiply(6, 2)
        12.0
        >>> multiply(-3, 4)
        -12.0
        """
        return a * b

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

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

        Returns
        -------
        float
            The quotient of `a` divided by `b`.

        Raises
        ------
        ZeroDivisionError
            If `b` is zero.

        Examples
        --------
        >>> divide(10, 2)
        5.0
        >>> divide(7, 0.5)
        14.0
        """
        if b == 0:
            raise ZeroDivisionError("Cannot divide by zero!")
        return a / b

    # Example Usage:
    print(f"Addition: 10 + 5 = {add(10, 5)}")
    print(f"Subtraction: 10 - 5 = {subtract(10, 5)}")
    print(f"Multiplication: 10 * 5 = {multiply(10, 5)}")
    print(f"Division: 10 / 5 = {divide(10, 5)}")

    try:
        print(f"Division by zero: 10 / 0 = {divide(10, 0)}")
    except ZeroDivisionError as e:
        print(f"Error: {e}")
```

OUTPUT:

```
Addition: 10 + 5 = 15
Subtraction: 10 - 5 = 5
Multiplication: 10 * 5 = 50
Division: 10 / 5 = 2.0
Error: Cannot divide by zero!
```

CODE: (Using Manual)

```
"""
This module provides basic arithmetic operations.
It contains simple functions to add, subtract, multiply,
and divide two numbers.
"""

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

    Parameters
    ----------
    a : int or float
        First number.
    b : int or float
        Second number.

    Returns
    -------
    int or float
        The sum of a and b.
    """
    return a + b

def subtract(a, b):
    """
    Subtract one number from another.

    Parameters
    ----------
    a : int or float
        Number from which subtraction is done.
    b : int or float
        Number to be subtracted.

    Returns
    -------
    int or float
        The result of a minus b.
    """
    return a - b

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

    Parameters
    ----------
    a : int or float
        First number.
    b : int or float
        Second number.

    Returns
    -------
    int or float
        The product of a and b.
    """
    return a * b

def divide(a, b):
    """
    Divide one number by another.

    Parameters
    ----------
    a : int or float
        Dividend.
    b : int or float
        Divisor.

    Returns
    -------
    float
        The result of a divided by b.

    Notes
    -----
    This function returns None if division by zero is attempted.
    """
    if b == 0:
        return None
    return a / b
```

EXPLANATION:

Criteria	AI-Generated Docstrings	Manual Docstrings
Structure	Very well structured with Parameters, Returns, Examples, and Raises	Basic structure with Parameters and Returns
Accuracy	Highly accurate, includes error handling details	Accurate for basic operations but misses errors
Readability	Detailed but slightly verbose	Simple, clear, and easy to read