

ASSIGNMENT-6.3

NAME : M.TANVI
ROLL NO : 2403A510A4
COURSE : AI ASSISTED CODING
BATCH : 01

TASK-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.

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

MANUAL CODE:

```
def sum_even_odd(numbers):  
    """Manually written docstring (Google Style)  
    Args:  
        numbers (list[int]): A list of integers. Returns:  
        tuple: A tuple containing two integers:  
            - Sum of even numbers  
            - Sum of odd numbers
```

Example:

```
>>> sum_even_odd([1, 2, 3, 4, 5])
```

```
(6, 9)
```

```
even_sum = sum(n for n in numbers if n % 2 == 0)
```

```
odd_sum = sum(n for n in numbers if n % 2 != 0)
```

```
return even_sum, odd_sum
```

PROMPT:

Write a Python function that takes a list of numbers as input and returns the sum of even numbers and the sum of odd numbers separately.

Add a function-level docstring in Google Style format. The docstring should include:

- A short description of the function
- Arguments section
- Returns section
- Example usage

AI CODE:

```
def sum_even_odd(numbers):  
    """Calculates the sum of even and odd numbers in a list.  
  
    Args:  
        numbers: A list of numbers (integers or floats).  
  
    Returns:  
        A tuple containing two elements: the sum of even numbers and the sum of odd numbers.  
        (sum_even, sum_odd)  
  
    Examples:  
        >>> sum_even_odd([1, 2, 3, 4, 5, 6])  
        (12, 9)  
        >>> sum_even_odd([10, 15, 20, 25])  
        (30, 40)  
        >>> sum_even_odd([])  
        (0, 0)  
    """  
    sum_even = 0  
    sum_odd = 0  
    for number in numbers:  
        if number % 2 == 0:  
            sum_even += number  
        else:  
            sum_odd += number  
    return sum_even, sum_odd  
  
# 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}")
```

COMPARISION OF MANUAL CODE AND AI CODE:

Manual docstring

More structured (Google Style format).

Includes an example for clarity.

Clearly separates args, returns, and usage.

AI-generated docstring

Easier wording, straight to the point.

Explains logic in plain English.

Does not always follow strict Google Style, but still understandable.

TASK-2:

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.

Expected Output#2: Students critically analyze AI-generated code comments.

MANUAL CODE:

```
class SRUStudent:
```

```
    """Class to represent an SRU student with basic details and fee update
    functionality."""
```

```
    def __init__(self, name, roll_no, hostel_status):
```

```
        # Initialize student attributes
```

```
        self.name = name
```

```
        self.roll_no = roll_no
```

```

        self.hostel_status = hostel_status

self.fee = 0 # Default fee as 0


def fee_update(self, amount):

    # Update the student's fee by adding the given amount

    self.fee += amount


def display_details(self):

    # Print all details of the student

    print(f"Name: {self.name}")

    print(f"Roll No: {self.roll_no}")

    print(f"Hostel Status: {self.hostel_status}")

    print(f"Fee Paid: {self.fee}")


# Example usage

student1 = SRUStudent("Rahul", 101, "Hosteller") # Create an object

student1.fee_update(50000)                # Update fee

student1.display_details()                # Display student details

```

PROMPT:

Write a Python program for a class named `sru_student` with the following:

- Attributes: name, roll_no, hostel_status

- Methods:

1. fee_update(self, amount): Updates the fee for the student
2. display_details(self): Displays all details of the student. Then, add inline comments explaining each line of the code.

Make sure every line has a short explanation comment.

AI CODE:

```
class sru_student: # Define a class named 'sru_student'
    def __init__(self, name, roll_no, hostel_status): # Constructor to initialize the object with name, roll_no, and hostel_status
        self.name = name # Assign the name attribute
        self.roll_no = roll_no # Assign the roll_no attribute
        self.hostel_status = hostel_status # Assign the hostel_status attribute
        self.fee = 0 # Initialize the fee attribute to 0

    def fee_update(self, amount): # Method to update the student's fee
        self.fee += amount # Add the given amount to the current fee

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

# Example usage (optional - you can remove this if you just need the class definition)
# Create an instance of the sru_student class
student1 = sru_student("John Doe", "12345", "Resident")

# Update the fee for student1
student1.fee_update(5000)

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

COMPARISION OF MANUAL CODE AND AI CODE:

Manual comments :

Short and simple.

Explain code in blocks (not every single line).

Focused on clarity.

AI-generated comments (from gemini):

More detailed, often explains each line separately.

Sometimes repetitive (e.g., "store roll number").

Good for beginners, but can feel lengthy.

TASK-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.

Expected Output#3: Students learn structured documentation for multi-function scripts, prompt for this description

MANUAL CODE:

```
.....
```

This module provides basic calculator functions such as addition, subtraction, multiplication, and division.

```
....
```

```
def add(a, b):
```

```
    """Add two numbers.
```

```
    Parameters
```

```
    a : int or float
```

```
        First number.
```

```
    b : int or float
```

```
        Second number.
```

```
    Returns
```

```
    int or float
```

```
        Sum of a and b.
```

```
    """
```

```
    return a + b
```

```
def subtract(a, b):
```

```
    """Subtract two numbers.
```

```
    Parameters
```

```
    a : int or float
```

```
        First number.
```

```
    b : int or float
```

```
        Second number
```

```
    Returns
```

int or float

Difference between a and 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

Product of a and b.

.....

return a * b

def divide(a, b):

"""Divide two numbers.

Parameter

a : int or float

Numerator.

b : int or float

Denominator.

Returns

float

Quotient of a divided by b.

Raise

ZeroDivisionError

If denominator is zero.

```
"""
```

```
if b == 0:
```

```
    raise ZeroDivisionError("Division by zero is not allowed.")
```

```
return a / b
```

PROMPT:

Write a Python script that defines a simple calculator with 4 functions:

- add(a, b)
- subtract(a, b)
- multiply(a, b)
- divide(a, b)

Add a module-level docstring at the top and NumPy-style docstrings for each function.

Make sure each docstring explains parameters, return values, and includes examples.

AI CODE:

```
"""
This module provides a simple calculator with basic arithmetic operations.
"""

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 a and b.

    Examples
    -----
    >>> add(2, 3)
    5
    >>> add(1.5, 2.5)
    4.0
    """
    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 a and b.

Examples
-----
>>> subtract(5, 2)
3
>>> subtract(10.0, 3.5)
6.5
"""
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 a and b.
```



```
Examples
-----
>>> multiply(4, 5)
20
>>> multiply(2.5, 3.0)
7.5
"""
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.

    Raises
    -----
    ZeroDivisionError
        If the divisor b is zero.

    Examples
    -----
    >>> divide(10, 2)
    5.0
    >>> divide(7.5, 2.5)
```

```

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

Raises
-----
ZeroDivisionError
    If the divisor b is zero.

Examples
-----
>>> divide(10, 2)
5.0
>>> divide(7.5, 2.5)
3.0
"""
if b == 0:
    raise ZeroDivisionError("Division by zero is not allowed.")
return a / b

# Example usage (optional)
# print(add(5, 3))
# print(subtract(10, 4))
# print(multiply(6, 7))
# print(divide(20, 5))
# print(divide(10, 0)) # This will raise a ZeroDivisionError

```

COMPARISION OF MANUAL CODE AND AI CODE:

Manual Docstrings (NumPy Style):

Structured and detailed.

Includes Parameters, Returns, and Raises sections.

Best for professional or academic use.

AI-generated Docstrings:

Short and simple, one-liners.

Easier for beginners to read.

Missing NumPy-style structure and exceptions documentation.