# AI Assisted Coding

# Assignment – 9.3

# K.VAMSHIDHAR || 2303A510H7 || Batch:- 8

## Task 1: Basic Docstring Generation

Scenario

You are developing a utility function that processes numerical lists and must be properly

documented for future maintenance.

Requirements

• Write a Python function to return the sum of even numbers and sum of odd numbers in a

given list

• Manually add a Google Style docstring to the function

• Use an AI-assisted tool (Copilot / Cursor AI) to generate a function-level docstring

• Compare the AI-generated docstring with the manually written docstring

• Analyze clarity, correctness, and completeness

lab 9.3.py

lab 9.3.py > sum_even_odd_ai

```python
#Create a Python function that returns the sum of even and odd numbers from a list.
#Add a manual Google-style docstring, then generate an AI docstring version.
#Compare both and analyze clarity, correctness, and completeness.
def sum_even_odd(numbers):
    """
    This function takes a list of numbers as input and returns a tuple containing the sum of even numbers and the sum of odd numbers.
    The first element of the tuple is the sum of even numbers, and the second element is the sum of odd numbers.

    Parameters:
    numbers (list): A list of integers.

    Returns:
    tuple: A tuple where the first element is the sum of even numbers and the second element is the sum of odd 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
def sum_even_odd_ai(numbers):

    even_sum = sum(num for num in numbers if num % 2 == 0)
    odd_sum = sum(num for num in numbers if num % 2 != 0)

    return even_sum, odd_sum
# Comparison of docstrings:
# Clarity: Both docstrings are clear in explaining the purpose of the function and its parameters. The manual docstring is more detailed, while the AI-generated docstring is concise
# Correctness: Both docstrings correctly describe the functionality of the function and the expected input and output.
# Completeness: The manual docstring provides a more comprehensive explanation, including the structure of the returned tuple. The AI-generated docstring is complete but less detail
# Example usage:
numbers = [1, 2, 3, 4, 5, 6]
print(sum_even_odd(numbers))  # Output: (12, 9)
print(sum_even_odd_ai(numbers))  # Output: (12, 9)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
(12, 9)
(12, 9)
PS C:\Users\komma\Desktop\3rd YEAR\AI-AC>
```

## Task 2: Automatic Inline Comments

Scenario

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

developers.

Requirements

- Write a Python program for an sru_student class with the following:

– Attributes: name, roll_no, hostel_status

– Methods: fee_update() and display_details()

- Manually write inline comments for each line or logical block
- Use an AI-assisted tool to automatically add inline comments
- Compare manual comments with AI-generated comments
- Identify missing, redundant, or incorrect AI comments

lab 9.3.py

lab 9.3.py > ...

```python
# Write a Python class sru_student with attributes name, roll_no, hostel_status and methods fee_update() and display_details().
# First add manual inline comments for each line/block.
# Then generate an AI-commented version of the same code.
# Compare both and identify missing, redundant, and incorrect AI comments
class sru_student:
    # Initialize the class with name, roll_no, and hostel_status attributes
    def __init__(self, name, roll_no, hostel_status):
        self.name = name  # Store the student's name
        self.roll_no = roll_no  # Store the student's roll number
        self.hostel_status = hostel_status  # Store the student's hostel status
    # Method to update the fee based on hostel status
    def fee_update(self):
        if self.hostel_status:  # Check if the student is in the hostel
            return 10000  # Return fee for hostel students
        else:
            return 5000  # Return fee for non-hostel students
    # Method to display the student's details
    def display_details(self):
        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: {'Yes' if self.hostel_status else 'No'}")  # Print the student's hostel status
# AI-commented version of the same code

class sru_student:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name  # Student's name
        self.roll_no = roll_no  # Student's roll number
        self.hostel_status = hostel_status  # Student's hostel status (True/False)

    def fee_update(self):
        if self.hostel_status:  # Check if the student is a hostel resident
            return 10000  # Fee for hostel students
        else:
            return 5000  # Fee for non-hostel students

    def display_details(self):
        print(f"Name: {self.name}")  # Display student's name
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
Name: Bob
Roll No: SRU124
Roll No: SRU124
Hostel Status: No
Fee for Bob: 5000
```

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.

Requirements

• Write a Python script containing 3–4 functions (e.g., add, subtract, multiply, divide)

• Manually write NumPy Style docstrings for each function

• Use AI assistance to generate:

– A module-level docstring

– Individual function-level docstrings

• Compare AI-generated docstrings with manually written ones

• Evaluate documentation structure, accuracy, and readability

lab 9.3.py

lab 9.3.py > ...

```python

# Create a Python calculator module with 3-4 functions (add, subtract, multiply, divide).
# First write manual NumPy-style docstrings for each function.
# Then generate an AI-written module-level docstring and function-level docstrings.
# Compare manual vs AI documentation and evaluate structure, accuracy, and readability.
# Manual NumPy-style docstrings for each function
def add(a, b):
    """
    Add two numbers.

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

    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
        The number to be subtracted from.
    b : int or float

    Returns
    -------
    int or float
        The difference of a and b.
```

lab 9.3.py ●

lab 9.3.py > ...

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

    Parameters
    ----------
    a : int or float
        The numerator.
    b : int or float

    Returns
    -------
    int or float
        The quotient of a and b.
    """
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b
# AI-written module-level docstring and function-level docstrings
"""
Calculator Module
This module provides basic arithmetic functions: add, subtract, multiply, and divide.
Functions
---------
add(a, b): Returns the sum of a and b.
subtract(a, b): Returns the difference of a and b.
multiply(a, b): Returns the product of a and b.
divide(a, b): Returns the quotient of a and b. Raises ValueError if b is zero.
"""
# Comparison of manual vs AI documentation:
# Structure: The manual docstrings follow the NumPy-style format, which is structured and detailed. The AI docstring is more concise and lacks the structured format.
# Accuracy: Both the manual and AI docstrings accurately describe the functionality of the functions. However, the manual docstrings provide more detailed information about paramete
# Readability: The manual docstrings are more comprehensive and may be easier to understand for users who are familiar with the NumPy-style documentation. The AI docstring is concis
# Example usage:
print(add(5, 3))        # Output: 8
print(subtract(5, 3))   # Output: 2
print(multiply(5, 3))   # Output: 15
print(divide(5, 3))     # Output: 1.6666666666666667
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
8
2
15
1.6666666666666667
PS C:\Users\komma\Desktop\3rd YEAR\AI-AC>
```