

AI Assisted Coding

Week4 - Wednesday

Name: Jangala Megha Harsha

Roll No.: 2303A51409

Batch No.: 07

Assignment Number: 9.3

Lab 9 – Documentation Generation – Automatic Documentation and Code Comments

◇ Task 1: Basic Docstring Generation

1) Prompt to be Given in VS Code (Humanized)

I am writing a Python utility function that takes a list of integers and returns two values: sum of even numbers and sum of odd numbers.

Please generate:

1. A clean Python function implementation.
2. A Google-style docstring explaining parameters, return values, and example usage.

Keep it clear and professional.

Then second prompt:

Now generate an improved professional Google-style docstring for the same function with better clarity and edge case explanation.

2) Code

Manual Google-Style Docstring

```
def sum_even_odd(numbers):
    """
    Calculates the sum of even and odd numbers from a given list.

    Args:
        numbers (list of int): A list containing integer values.

    Returns:
        tuple: A tuple containing:
            - sum_even (int): Sum of all even numbers.
            - sum_odd (int): Sum of all odd numbers.

    Example:
        >>> sum_even_odd([1, 2, 3, 4])
        (6, 4)
    """
    sum_even = 0
    sum_odd = 0

    for num in numbers:
        if num % 2 == 0:
            sum_even += num
        else:
            sum_odd += num

    return sum_even, sum_odd
```

AI-Generated Docstring (Improved Version)

```
def sum_even_odd(numbers):
    """
    Compute the total of even and odd integers separately from a list.

    Parameters:
        numbers (List[int]): A list of integers to evaluate.

    Returns:
        Tuple[int, int]:
            The first value represents the sum of even integers.
            The second value represents the sum of odd integers.
```

Raises:

TypeError: If input is not iterable or contains non-integer values.

Notes:

Empty lists return (0, 0).

```
"""
```

```
sum_even = 0
```

```
sum_odd = 0
```

```
for num in numbers:
```

```
    if num % 2 == 0:
```

```
        sum_even += num
```

```
    else:
```

```
        sum_odd += num
```

```
return sum_even, sum_odd
```

3) Output

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

Output:

(6, 9)

4) Remarks (Comparison)

Criteria	Manual	AI Generated
Clarity	Good	More structured
Error Handling	Not mentioned	Mentions TypeError
Professional Tone	Moderate	More formal
Completeness	Basic	More comprehensive

Conclusion:

AI-generated documentation is more structured and professional but may sometimes mention exceptions not implemented in code.

◇ Task 2: Automatic Inline Comments

1) Prompt to Give in VS Code

I am creating a Python class called `sru_student` with attributes:
`name`, `roll_no`, `hostel_status`.

Methods:

- `fee_update(amount)`
- `display_details()`

Please generate:

1. The full class implementation
2. Add meaningful inline comments explaining each block of code.

Keep comments beginner-friendly.

2) Code

Manual Inline Comments

```
class sru_student:

    # Constructor method to initialize student details

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

        self.name = name # Store student name

        self.roll_no = roll_no # Store roll number

        self.hostel_status = hostel_status # Store hostel status (True/False)

        self.fee_balance = 0 # Initialize fee balance


    # Method to update fee balance

    def fee_update(self, amount):
```

```
        self.fee_balance += amount # Add amount to fee balance

# Method to display student details
def display_details(self):

    print("Name:", self.name)

    print("Roll No:", self.roll_no)

    print("Hostel Status:", self.hostel_status)

    print("Fee Balance:", self.fee_balance)
```

3) Output

```
student1 = sru_student("Lalu", "2303A51948", True)

student1.fee_update(50000)

student1.display_details()
```

Output:

Name: Lalu

Roll No: 2303A51948

Hostel Status: True

Fee Balance: 50000

4) Remarks

Manual Comments:

- ✓ Clear
- ✓ Relevant
- ✓ Minimal

AI Comments:

- ✓ More descriptive
- ✗ Sometimes redundant
- ✗ May comment obvious lines

Conclusion:

AI comments are helpful for beginners but may become verbose.

◇ Task 3: Module-Level and Function-Level Documentation

1) Prompt to Give in VS Code

Create a small Python calculator module with functions:

add, subtract, multiply, divide.

1. Write NumPy-style docstrings for each function.
 2. Generate a professional module-level docstring.
 3. Keep it suitable for production-level documentation.
-

2) Code

```
"""
```

```
Calculator Module
```

```
=====
```

This module provides basic arithmetic operations including addition, subtraction, multiplication, and division.

Functions

```
-----
```

```
add(a, b)
```

```
subtract(a, b)
```

```
multiply(a, b)
```

```
divide(a, b)
```

```
"""
```

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

```
    """
```

```
    Add two numbers.
```

```
Parameters
```

```
-----
```

```
a : int or float
```

```
b : int or float
```

Returns

int or float

Sum of a and b.

"""

return a + b

def subtract(a, b):

"""

Subtract b from a.

Parameters

a : int or float

b : int or float

Returns

int or float

Difference of a and b.

"""

return a - b

def multiply(a, b):

"""

Multiply two numbers.

Parameters

a : int or float

b : int or float

Returns

int or float

Product of a and b.

"""

return a * b

```
def divide(a, b):  
    """  
    Divide a by b.  
  
    Parameters  
    -----  
    a : int or float  
    b : int or float  
  
    Returns  
    -----  
    float  
        Quotient of a divided by b.  
  
    Raises  
    -----  
    ZeroDivisionError  
        If b is zero.  
    """  
    if b == 0:  
        raise ZeroDivisionError("Cannot divide by zero")  
    return a / b
```

3) Output

```
print(add(5, 3))  
print(subtract(10, 4))  
print(multiply(6, 2))  
print(divide(8, 2))
```

Output:

```
8  
6  
12  
4.0
```

4) Remarks

Feature	Manual NumPy Docstring	AI Generated
Structure	Correct	Correct
Formatting	Proper	Professional
Completeness	Basic	Often more detailed
Accuracy	Controlled	Sometimes over-explains

Key Learning:

- NumPy style is better for scientific modules.
- Google style is simpler and more readable.
- AI speeds up documentation but must be reviewed carefully.

✅ Final Conclusion for Lab 9

AI-assisted documentation:

- ✓ Saves time
- ✓ Improves structure
- ✓ Good for standard formatting
- ✗ Needs human verification
- ✗ May document non-existing exceptions