

Assignment 9.3

2303A51146

K.Suchira

Batch – 17

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

Expected Output

- Python function with manual Google-style docstring
- AI-generated docstring for the same function
- Comparison explaining differences between manual and AI-generated documentation
- Improved understanding of AI-generated function-level documentation

Prompt :

PROMPT 1 (Use in Copilot / Cursor AI)

Write a Python function that takes a list of numbers and

returns the sum of even numbers and sum of odd numbers.

Add proper input validation and return both sums.

Input validation

Loop through list and calculate sums

Generate a detailed Google Style docstring for this Python

function that calculates the sum of even and odd numbers

from a list.

Compare manual docstring and AI-generated docstring for

clarity, correctness, and completeness. Explain differences.

CODE:

PROMPT 1 (Use in Copilot / Cursor AI)

Write a Python function that takes a list of numbers and

returns the sum of even numbers and sum of odd numbers.

Add proper input validation and return both sums.

```
def sum_even_odd(numbers):
```

```
    # Input validation
```

```
    if not isinstance(numbers, list):
```

```
        raise ValueError("Input must be a list")
```

```
    even_sum = 0
```

```
    odd_sum = 0
```

```
    # Loop through list and calculate sums
```

```
    for num in numbers:
```

```
        if not isinstance(num, (int, float)):
```

```
            raise ValueError("List must contain only numbers")
```

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

```
            even_sum += num
```

```
        else:
```

```
            odd_sum += num
```

```
    return even_sum, odd_sum
```

Generate a detailed Google Style docstring for this Python

function that calculates the sum of even and odd numbers

from a list.

def sum_even_odd_ai(numbers):

if not isinstance(numbers, list):

raise ValueError("Input must be a list")

even_sum = 0

odd_sum = 0

for num in numbers:

if not isinstance(num, (int, float)):

raise ValueError("List must contain only numbers")

if num % 2 == 0:

even_sum += num

else:

odd_sum += num

return even_sum, odd_sum

Compare manual docstring and AI-generated docstring for

clarity, correctness, and completeness. Explain differences.

def docstring_comparison():

print("\n--- Docstring Comparison ---")

print("\nClarity:")

```
print("Manual docstring is simple and direct.")
print("AI-generated docstring provides more detailed explanation.")

print("\nCorrectness:")
print("Both docstrings correctly describe parameters, return values, and exceptions.")

print("\nCompleteness:")
print("AI docstring includes additional explanation about function working.")
print("Manual docstring focuses mainly on usage.")

print("\nConclusion:")
print("AI-generated documentation is slightly more descriptive and detailed.")

if __name__ == "__main__":
    sample_list = [1, 2, 3, 4, 5, 6]

    even, odd = sum_even_odd(sample_list)

    print("Sum of Even Numbers:", even)
    print("Sum of Odd Numbers:", odd)

    docstring_comparison()
```

OUTPUT:

```
Sum of Even Numbers: 12
Sum of Odd Numbers: 9

--- Docstring Comparison ---

Clarity:
Manual docstring is simple and direct.
AI-generated docstring provides more detailed explanation.

Correctness:
Both docstrings correctly describe parameters, return values, and exceptions.

Completeness:
AI docstring includes additional explanation about function working.
Manual docstring focuses mainly on usage.

Conclusion:
AI-generated documentation is slightly more descriptive and detailed.
PS C:\Users\cheti>
```

ANALYSIS:

- Function works correctly.
- Validation is decent but could be stricter.
- Docstring comparison is theoretical, not practical.
- Minor improvement needed for float handling and redundancy removal.

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

Expected Output

- Python class with manually written inline comments
- AI-generated inline comments added to the same code
- Comparative analysis of manual vs AI comments
- Critical discussion on strengths and limitations of AI-generated comments

PROMPT:

#Write a Python program to create student management module for sru_student class .

#IT should have the following attributes name, roll_no, hostel_status

Methods: fee_update() and display_details()

write inline comments for each line or logical block

CODE:

#Write a Python program to create student management module for sru_student class .

#IT should have the following attributes name, roll_no, hostel_status

Methods: fee_update() and display_details()

write inline comments for each line or logical block

class SruStudent:

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

 # Initialize the attributes of the SruStudent class

 self.name = name # Store the name of the student

 self.roll_no = roll_no # Store the roll number of the student

 self.hostel_status = hostel_status # Store the hostel status of the student

def fee_update(self, amount):

 # This method is a placeholder for updating the fee details

 print(f"Updating fee for {self.name} by amount: {amount}")

def display_details(self):

 # This method displays the details of the student

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

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

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

Example usage

student1 = SruStudent("SUCHIRA", "SRU1146", "Hostel A")

```
student1.display_details() # Display the details of student1
student1.fee_update(5000) # Update the fee for student1
student2 = SruStudent("VYSHU", "SRU1254", "Hostel B")
student2.display_details() # Display the details of student2
student2.fee_update(4500) # Update the fee for student2
```

OUTPUT:

```
Updating fee for Thrishla by amount: 4500
PS C:\Users\91701\OneDrive\Desktop\AI Assistance(1146)> & C:\Users\91701\OneDrive\Desktop\AI Assistance(1146)/9.3.py"
Name: Suchira
Roll No: SRU146
Hostel Status: Hostel A
Updating fee for Suchira by amount: 5000
Name: Thrishla
Roll No: SRU123
Hostel Status: Hostel B
Updating fee for Thrishla by amount: 4500
PS C:\Users\91701\OneDrive\Desktop\AI Assistance(1146)>
```

ANALYSIS:

- Class structure is correct and requirements are satisfied.
- Code is clean, readable, and properly commented.
- OOP concepts are correctly demonstrated.
- fee_update() only prints message; no real fee tracking implemented.
- No input validation for student details.
- Implementation is basic and can be enhanced for full functionality.

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

Expected Output

- Python script with manual NumPy-style docstrings
- AI-generated module-level and function-level documentation
- Comparison between AI-generated and manual documentation
- Clear understanding of structured documentation for multi-function scripts

PROMPT:

#write a python program to build a small calculator module that will be shared across multiple projects.

#it should have the following functions: add(), subtract(), multiply(), divide()

#Manually write NumPy Style docstrings for each function.

Compare manual docstrings and AI-generated docstrings for clarity, correctness, and completeness. Explain differences.

Run the docstring comparison analysis

CODE:

#write a python program to build a small calculator module that will be shared across multiple projects.

#it should have the following functions: add(), subtract(), multiply(), divide()

#Manually write NumPy Style docstrings for each function.

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

```
    """
```

```
    Compute the sum of two numbers.
```

```
    Parameters
```

```
    -----
```

```
    a : float
```

```
        The first number.
```

```
    b : float
```

```
        The second number.
```

```
    Returns
```

```
    -----
```

```
    float
```


The sum of a and b.

"""

 return a + b

def subtract(a, b):

 """

 Compute the difference of two numbers.

 Parameters

 a : float

 The first number.

 b : float

 The second number.

 Returns

 float

 The difference of a and b.

 """

 return a - b

def multiply(a, b):

 """

 Compute the product of two numbers.

 Parameters

 a : float

 The first number.

 b : float

 The second number.

Returns

float

The product of a and b.

"""

return a * b

def divide(a, b):

"""

Compute the quotient of two numbers.

Parameters

a : float

The dividend.

b : float

The divisor.

Returns

float or str

The quotient of a and b, or an error message if division by zero occurs.

"""

if b == 0:

return "Error: Division by zero is not allowed."

return a / b

Example usage

num1 = 10

num2 = 5

print(f"Addition: {add(num1, num2)}")

print(f"Subtraction: {subtract(num1, num2)}")

```

print(f"Multiplication: {multiply(num1, num2)}")
print(f"Division: {divide(num1, num2)}")

# Compare manual docstrings and AI-generated docstrings for clarity, correctness, and completeness. Explain differences.

def docstring_comparison():
    print("\n--- Docstring Comparison ---")

    print("\nClarity:")
    print("Manual docstrings are clear and concise.")
    print("AI-generated docstrings provide more detailed explanations.")

    print("\nCorrectness:")
    print("Both sets of docstrings correctly describe the parameters, return values, and exceptions.")

    print("\nCompleteness:")
    print("AI-generated docstrings include additional information about the function's behavior and edge cases.")
    print("Manual docstrings focus on the basic functionality without extra details.")

    print("\nConclusion:")
    print("AI-generated documentation is more comprehensive, while manual documentation is straightforward and to the point.")

if __name__ == "__main__":
    docstring_comparison() # Run the docstring comparison analysis

```

OUTPUT:

```

Addition: 15
Subtraction: 5
Multiplication: 50
Division: None

--- Docstring Comparison ---

Clarity:
Manual docstrings are clear and concise.
AI-generated docstrings provide more detailed explanations.

```

Correctness:

Both sets of docstrings correctly describe the parameters, return values, and exceptions.

Completeness:

AI-generated docstrings include additional information about the function's behavior and edge cases.

Manual docstrings focus on the basic functionality without extra details.

Conclusion:

AI-generated documentation is more comprehensive, while manual documentation is straightforward and concise.

PS C:\Users\cheti>

ANALYSIS:

- All calculator functions (add, subtract, multiply, divide) are logically correct for basic arithmetic.
- NumPy-style docstrings are properly structured with Parameters and Returns sections.
- Code is modular and reusable across projects.
- `divide()` contains a logical error: `return a / b` is unreachable due to incorrect indentation.
- Returning a string for division by zero creates inconsistent return types (float or str).
- No input validation for non-numeric values.
- Docstring comparison section is theoretical, not an actual automated comparison.
- Overall implementation is good but needs minor fixes for correctness and robustness.