

AI Assisted Coding

Assignment – 9.3

M.Sathwik || 2303A51483 || Batch:- 08

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.py > sum_even_odd
1  #Create a Python function that returns the sum of even and odd numbers from a list.
2  # Add a manual Google-style docstring, then generate an AI docstring version.
3  # Compare both and analyze clarity, correctness, and completeness.
4  def sum_even_odd(numbers):
5      """
6          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.
7          Parameters:
8              numbers (list): A list of integers or floats.
9          Returns:
10             tuple: A tuple where the first element is the sum of even numbers and the second element is the sum of odd numbers.
11             Example:
12             sum_even_odd([1, 2, 3, 4, 5]) → (6, 9)
13             sum_even_odd([]) → (0, 0)
14             sum_even_odd([-1, -2, -3]) → (-2, -4)
15             """
16             even_sum = 0
17             odd_sum = 0
18             for num in numbers:
19                 if isinstance(num, (int, float)): # Check if the element is a number
20                     if num % 2 == 0:
21                         even_sum += num
22                     else:
23                         odd_sum += num
24             return even_sum, odd_sum
25             # Test cases
26             print(sum_even_odd([1, 2, 3, 4, 5])) # Expected output: (6, 9)
27             print(sum_even_odd([])) # Expected output: (0, 0)
28             print(sum_even_odd([-1, -2, -3])) # Expected output: (-2, -4)
29             # AI-generated docstring version
30             def sum_even_odd(numbers):
31                 """
32                     Calculate the sum of even and odd numbers in a given list.
33                     Args:
34                         numbers (list): A list of integers or floats to be processed.
```

```

52     Calculate the sum of even and odd numbers's in a given list.
53
54     Args:
55         numbers (list): A list of integers or floats to be processed.
56
57     Returns:
58         tuple: A tuple containing two elements: the first is the sum of even numbers, and the second is the sum of odd numbers.
59
60     Examples:
61         >>> sum_even_odd([1, 2, 3, 4, 5])
62         (6, 9)
63
64         >>> sum_even_odd([])
65         (0, 0)
66
67         >>> sum_even_odd([-1, -2, -3])
68         (-2, -4)
69
70     """
71
72     even_sum = 0
73     odd_sum = 0
74     for num in numbers:
75         if isinstance(num, (int, float)): # Check if the element is a number
76             if num % 2 == 0:
77                 even_sum += num
78             else:
79                 odd_sum += num
80     return even_sum, odd_sum
81
82     # Test cases
83     print(sum_even_odd([1, 2, 3, 4, 5])) # Expected output: (6, 9)
84     print(sum_even_odd([])) # Expected output: (0, 0)
85     print(sum_even_odd([-1, -2, -3])) # Expected output: (-2, -4)
86
87     # Analysis:
88     # Both docstrings provide a clear and concise explanation of the function's purpose, parameters, return value, and examples.
89     # The manual docstring is straightforward and easy to understand, while the AI-generated docstring follows
90     # a more formal structure with sections for arguments, returns, and examples.
91     # Both docstrings are correct and complete, covering all necessary information for users to understand how to use the function effectively.
92

```

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

```

64  #Write a Python class sru_student with attributes name, roll_no, hostel_status and methods fee_update() and display_details().
65  # First add manual inline comments for each line/block.
66  # Then generate an AI-commented version of the same code.
67  # Compare both and identify missing, redundant, and incorrect AI comments.
68  class sru_student:
69      def __init__(self, name, roll_no, hostel_status):
70          # Initialize the student's name, roll number, and hostel status
71          self.name = name
72          self.roll_no = roll_no
73          self.hostel_status = hostel_status
74          self.fee = 0 # Initialize fee to 0
75
76      def fee_update(self, amount):
77          # Update the student's fee by adding the specified amount
78          self.fee += amount
79
80      def display_details(self):
81          # Display the student's details including name, roll number, hostel status, and fee
82          print(f"Name: {self.name}")
83          print(f"Roll No: {self.roll_no}")
84          print(f"Hostel Status: {self.hostel_status}")
85          print(f"Fee: {self.fee}")
86
87  # AI-commented version
88  class sru_student:
89      def __init__(self, name, roll_no, hostel_status):
90          # Constructor to initialize the student's attributes
91          self.name = name # Student's name
92          self.roll_no = roll_no # Student's roll number
93          self.hostel_status = hostel_status # Student's hostel status
94          self.fee = 0 # Initialize fee to 0
95
96      def fee_update(self, amount):
97          # Method to update the student's fee by adding the specified amount
98          self.fee += amount # Add the amount to the existing fee

```

```

88      def __init__(self, name, roll_no, hostel_status):
89          # Constructor to initialize the student's attributes
90          self.name = name # Student's name
91          self.roll_no = roll_no # Student's roll number
92          self.hostel_status = hostel_status # Student's hostel status
93          self.fee = 0 # Initialize fee to 0
94
95      def fee_update(self, amount):
96          # Method to update the student's fee by adding the specified amount
97          self.fee += amount # Add the amount to the existing fee
98
99      def display_details(self):
100         # Method to display the student's details
101         print(f"Name: {self.name}") # Print the student's name
102         print(f"Roll No: {self.roll_no}") # Print the student's roll number
103         print(f"Hostel Status: {self.hostel_status}") # Print the student's hostel status
104         print(f"Fee: {self.fee}") # Print the current fee of the student
105
106  # Analysis:
107  # The AI comments are generally accurate and provide a clear explanation of each line of code.
108  # There are some redundant comments, such as "Constructor to initialize the student's attributes" which is already implied by the method name `__init__`.
109  # Additionally, the comment "Add the amount to the existing fee" is somewhat redundant.
110  # The method name `fee_update` already suggests that it updates the fee. Overall, while

```

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

```
13 # Create a Python calculator module with 3-4 functions (add, subtract, multiply, divide).
14 # First write manual NumPy-style docstrings for each function.
15 # Then generate an AI-written module-level docstring and function-level docstrings.
16 # Compare manual vs AI documentation and evaluate structure, accuracy, and readability.
17 def add(a, b):
18     """
19     Add two numbers.
20
21     Parameters
22     -----
23     a : int or float
24         The first number to be added.
25     b : int or float
26         The second number to be added.
27
28     Returns
29     -----
30     int or float
31         The sum of a and b.
32
33     Examples
34     -----
35     >>> add(2, 3)
36     5
37     >>> add(1.5, 2.5)
38     4.0
39     """
40     return a + b
41 def subtract(a, b):
42     """
43     Subtract one number from another.
44     Parameters
45     -----
46     a : int or float
47         The number from which to subtract.
48     b : int or float
49         The number to be subtracted.
50
51     Returns
52     -----
53     int or float
54         The difference of a and b.
55     Examples
56     -----
57     >>> subtract(5, 3)
58     2
59     """
60     return a - b
61 def multiply(a, b):
62     """
63     Compute the product of a and b.
64     Examples
65     -----
66     >>> multiply(2, 3)
67     6
68     >>> multiply(1.5, 2.5)
69     3.75
70     """
71     return a * b
72 def divide(a, b):
73     """
74     Divide one number by another.
75     Parameters
76     -----
77     a : int or float
78         The number to be divided.
79     b : int or float
80         The number by which to divide.
81
82     Returns
83     -----
84     int or float
85         The quotient of a and b.
86     Raises
87     -----
88     ZeroDivisionError
89         If b is zero.
```

```

199 |     -----
200 |     >>> divide(6, 3)
201 |     2.0
202 |     >>> divide(7, 2)
203 |     3.5
204 |     >>> divide(5, 0)
205 |     Traceback (most recent call last):
206 |     ...
207 |     ZeroDivisionError: division by zero
208 |     """
209 |
210 |     if b == 0:
211 |         raise ZeroDivisionError("division by zero")
212 |     return a / b
213 | # AI-generated module-level docstring
214 | """
215 |
216 | This module provides basic arithmetic operations including addition, subtraction, multiplication, and division.
217 | Functions
218 | -----
219 |
220 | - add(a, b): Returns the sum of a and b.
221 | - subtract(a, b): Returns the difference of a and b.
222 | - multiply(a, b): Returns the product of a and b.
223 | - divide(a, b): Returns the quotient of a and b, raises ZeroDivisionError if b is zero.
224 | """
225 | # AI-generated function-level docstrings
226 | def add(a, b):
227 |     """
228 |     Add two numbers and return the result.
229 |     Parameters:
230 |     a (int or float): The first number to be added.
231 |     b (int or float): The second number to be added.
232 |     Returns:
233 |     int or float: The sum of a and b.
234 |     """
235 |     return a + b
236 | def subtract(a, b):
237 |     """
238 |     Subtract one number from another and return the result.
239 |     Parameters:
240 |     a (int or float): The number from which to subtract.
241 |     b (int or float): The number to be subtracted.
242 |     Returns:
243 |     int or float: The difference of a and b.
244 |     """
245 |     return a - b
246 | def multiply(a, b):
247 |     """
248 |     Multiply two numbers and return the result.
249 |     Parameters:
250 |     a (int or float): The first number to be multiplied.
251 |     b (int or float): The second number to be multiplied.
252 |     Returns:
253 |     int or float: The product of a and b.
254 |     """
255 |     return a * b
256 | def divide(a, b):
257 |     """Divide one number by another and return the result. Raises ZeroDivisionError if b is zero.
258 |     Parameters:
259 |     a (int or float): The number to be divided.
260 |     b (int or float): The number by which to divide.
261 |     Returns:
262 |     int or float: The quotient of a and b.
263 |     Raises:
264 |     ZeroDivisionError: If b is zero.
265 |     """
266 |     if b == 0:
267 |         raise ZeroDivisionError("division by zero")
268 |     return a / b
269 | # Analysis:
270 | # The manual NumPy-style docstrings are well-structured and provide detailed information about the parameters, return values,
271 | # and examples for each function.
272 | # The AI-generated module-level docstring is concise and effectively summarizes the purpose of the module and its functions.
273 | # The AI-generated function-level docstrings are accurate and provide clear explanations of the parameters, return values,
274 | # and exceptions for each function. However, they are less detailed than the manual docstrings,
275 | # lacking examples and a more formal structure. Overall, both the manual and AI-generated docstrings are accurate and readable,
276 | # but the manual docstrings offer more comprehensive information for users.

```