

Ai Assissted Coding

Name : B.Sai Varsha

Roll No : 2303A54065

Batch : 48

Lab 9: Documentation Generation – Automatic Documentation and Code Comments

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

Expected Output :

Prompt 1 – Generate Python Function :

The screenshot shows a code editor window titled "AI Assisted Lab". The file "task01.py" contains the following code:

```
1 # Write a clean Python function that takes a list of integers as input and returns:
2 # 1) The sum of even numbers
3 # 2) The sum of odd numbers
4
5 # The function should return both values as a tuple.
6 # Keep the code simple and readable.
7 def sum_even_odd(numbers):
8     sum_even = 0
9     sum_odd = 0
10
11     for num in numbers:
12         if num % 2 == 0:
13             sum_even += num
14         else:
15             sum_odd += num
16
17     return (sum_even, sum_odd)
18
19 # Example usage:
20 numbers = [1, 2, 3, 4, 5, 6]
21 result = sum_even_odd(numbers)
22 print("Sum of even numbers: {result[0]}")
23 print("Sum of odd numbers: {result[1]}")
```

The terminal tab shows the execution of the script:

```
/usr/bin/python3 /Users/anumandararithika/SRU/AI_Assisted_Lab/lab9.3/task01.py
● anumandararithika@Mac AI_Assisted_Lab %
Sum of even numbers: 12
Sum of odd numbers: 9
%
```

Prompt 2 – Generate Google Style Docstring :

The screenshot shows a code editor window titled "AI_Assisted_lab". The left pane displays the Python file "task01.py" with the following content:

```

lab9.3 > ⚡ task01.py > ...
24 # Add a proper Google Style docstring to the following Python function.
25 # Include:
26 # - Short description
27 # - Args section
28 # - Returns section
29 # - Example usage
30 # Here is the function:
31 def sum_even_odd(numbers):
32     """
33         Calculate the sum of even and odd numbers from a list of integers.
34     """
35     Args:
36         numbers (list): A list of integers to be processed.
37     Returns:
38         tuple: A tuple containing the sum of even numbers and the sum of odd numbers.
39     Example:
40     >>> sum_even_odd([1, 2, 3, 4, 5, 6])
41     (12, 9)
42     """
43     sum_even = 0
44     sum_odd = 0
45
46     for num in numbers:
47         if num % 2 == 0:
48             sum_even += num
49         else:
50             sum_odd += num
51
52     return (sum_even, sum_odd)
53 # Example usage:
54 numbers = [1, 2, 3, 4, 5, 6]
55 result = sum_even_odd(numbers)
56 print("Sum of even numbers: (result[0])")
57 print("Sum of odd numbers: (result[1])")

```

The right pane shows the terminal output of running the script:

```

/usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab9.3/task01.py
Sum of even numbers: 12
Sum of odd numbers: 9

```

At the bottom, the status bar indicates "Ln 30, Col 24" and "Python 3.9.6".

Prompt 3 – AI Docstring Generation (for comparison)

The screenshot shows a code editor window titled "AI_Assisted_lab". The left pane displays the Python file "task01.py" with the following content:

```

lab9.3 > ⚡ task01.py > ...
53 # # Example usage:
54 # numbers = [1, 2, 3, 4, 5, 6]
55 # result = sum_even_odd(numbers)
56 # print("Sum of even numbers: (result[0])")
57 # print("Sum of odd numbers: (result[1])")
58
59 # Automatically generate a professional function-level Google Style docstring
60 # for this Python function without modifying the logic.
61
62 # Ensure clarity, completeness, and correctness.
63
64 # [paste function without docstring]
65 def sum_even_odd(numbers):
66     sum_even = 0
67     sum_odd = 0
68
69     for num in numbers:
70         if num % 2 == 0:
71             sum_even += num
72         else:
73             sum_odd += num
74
75     return (sum_even, sum_odd)
76 # # Calling the function
77 numbers = [1, 2, 3, 4, 5, 6]
78 even_sum, odd_sum = sum_even_odd(numbers)
79 print("Sum of even numbers:", even_sum)
80 print("Sum of odd numbers:", odd_sum)
81
82

```

The right pane shows the terminal output of running the script:

```

/usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab9.3/task01.py
Sum of even numbers: 12
Sum of odd numbers: 9

```

At the bottom, the status bar indicates "Ln 78, Col 29" and "Python 3.9.6".

Prompt 4 – Comparison Prompt :

```

lab9.3 > task01.py ...
83 # Compare the following two docstrings:
84 # 1) Manually written docstring:
85 # [paste manual docstring]
86 # 2) AI-generated docstring:
87 # [paste AI docstring]
88 Analyze:
89 # Clarity
90 # Completeness
91 # - Correctness
92 # - Readability
93 # - Any missing information
94 # Provide a structured comparison.
95 # Manually written docstring:
96 """
97 Calculate the sum of even and odd numbers from a list of integers.
98 Args:
99     numbers (list): A list of integers to be processed.
100 Returns:
101     tuple: A tuple containing the sum of even numbers and the sum of odd numbers.
102 Example:
103     >>> sum_even_odd([1, 2, 3, 4, 5, 6])
104     (12, 9)
105 """
106 def sum_even_odd(numbers):
107     sum_even = 0
108     sum_odd = 0
109     for num in numbers:
110         if num % 2 == 0:
111             sum_even += num
112         else:
113             sum_odd += num
114     return (sum_even, sum_odd)
115 """
116 Calculate the sum of even and odd numbers from a list of integers.
117 Args:
118     numbers (list): A list of integers to be processed.
119 Returns:
120     tuple: A tuple containing the sum of even numbers and the sum of odd numbers.
121 Example:
122     >>> sum_even_odd([1, 2, 3, 4, 5, 6])
123     (12, 9)
124 """
# Calling the function

```

Ln 127, Col 42 Spaces: 4 UTF-8 LF () Python 3.9.6

Criteria	Manual Docstring	AI Docstring
Clarity	Clear but short	More detailed
Completeness	Basic explanation	Includes examples
Accuracy	Correct	Correct
Readability	Simple	Professional tone

Explanation :

In this task, I wrote a Python function to calculate the sum of even numbers and odd numbers from a list.

First, I added a **manual Google-style docstring** to explain:

- What the function does
- What input it takes
- What it returns

Then, I used an AI tool to generate a docstring automatically.

After comparing both, I observed that:

- The AI-generated docstring was more detailed.
- My manual docstring was shorter and simple.
- AI documentation looked more professional.
- Both were correct, but AI saved time.

This task helped me understand how AI tools can generate function-level documentation quickly and clearly.

Task 2 – Automatic Inline Comments :

Prompt 1 – Generate Class :

```
task02.py
lab9.3 > task02.py > sru_student > fee_update
1 #Write a Python class named sru_student with:
2 # - name
3 # - roll_no
4 # - hostel_status
5
6 # Methods:
7 # - fee_update(amount)
8 # - display_details()
9 # Keep the structure simple and clean.
10 class sru_student:
11     def __init__(self, name, roll_no, hostel_status):
12         self.name = name
13         self.roll_no = roll_no
14         self.hostel_status = hostel_status
15         self.fee_amount = 0
16     def fee_update(self, amount):
17         """Update the fee amount for the student."""
18         self.fee_amount += amount
19     def display_details():
20         """Prints the details of the student."""
21         print("Name: " + self.name)
22         print("Roll No: " + str(self.roll_no))
23         print("Hostel Status: " + self.hostel_status)
24         print("Fee Amount: " + str(self.fee_amount))
25
26 # Example usage:
27 student1 = sru_student("Alice", "SRU123", "Yes")
28 student1.fee_update(5000)
29 student1.display_details()
30
31 # Add meaningful inline comments to every logical block of the following code.
```

The screenshot shows a code editor window with three tabs: 'task01.py', 'task02.py', and 'task03.py'. The 'task02.py' tab is active, displaying the provided Python code. The code defines a class 'sru_student' with methods for initializing student details, updating fee, and displaying student information. The code is annotated with several inline comments explaining the purpose of each section and method. The terminal below the code editor shows the output of running the script, which creates a student object 'student1' with name 'Alice', roll number 'SRU123', and hostel status 'Yes', and then updates its fee to 5000.

Prompt 2 – Add Manual Inline Comments :

```
task02.py
lab9.3 > task02.py > sru_student > fee_update
30 # Add meaningful inline comments to every logical block of the following code.
31 # Explain what each section does clearly.
32 # do NOT change the logic.
33 # (paste class code)
34 class sru_student:
35     def __init__(self, name, roll_no, hostel_status):
36         """Initialize the student object with name, roll number, and hostel status"""
37         self.name = name
38         self.roll_no = roll_no
39         self.hostel_status = hostel_status
40         self.fee_amount = 0 # Initialize fee amount to zero
41     def fee_update(self, amount):
42         """Update the fee amount for the student."""
43         # Add the given amount to the existing fee amount
44         self.fee_amount += amount
45     def display_details(self):
46         """Display the details of the student."""
47         # Print the student's name, roll number, hostel status, and current fee amount
48         print("Name: " + self.name)
49         print("Roll No: " + self.roll_no)
50         print("Hostel Status: " + self.hostel_status)
51         print("Fee Amount: " + str(self.fee_amount))
52
53 # Create an instance of sru_student with name "Alice", roll number "SRU123", and hostel status "Yes"
54 student1 = sru_student("Alice", "SRU123", "Yes")
55 # # Update the fee amount for the student by adding 5000
56 student1.fee_update(5000)
57 # # Display the details of the student, including the updated fee amount
58 student1.display_details()
```

This screenshot shows the same code editor window as the previous one, but the code has been manually annotated with additional inline comments. These comments provide more detailed explanations for the initialization of the student object, the logic of the 'fee_update' method, and the display of student details. The terminal output remains the same, showing the creation of a student object and its fee update.

Prompt 3 – Evaluate AI Comments :

```

lab9.3 > task01.py > task02.py > ...
60  # Analyze the following Python code with AI-generated inline comments.
61  # Identify:
62  # - Redundant comments
63  # - Missing comments
64  # - Incorrect explanations
65  # - Over-explained parts
66  # Provide a critical evaluation.
67  # [paste AI-commented code]
68
69  class sru_student:
70      def __init__(self, name, roll_no, hostel_status):
71          # Initialize the student object with name, roll number, and hostel status
72          self.name = name
73          self.roll_no = roll_no
74          self.hostel_status = hostel_status
75          self.fee_amount = 0 # Initialize fee amount to zero
76
77      def fee_update(self, amount):
78          """Update the fee amount for the student."""
79          # Add the given amount to the existing fee amount
80          self.fee_amount += amount
81
82      def display_details(self):
83          """Display the details of the student."""
84          # Print the student's name, roll number, hostel status, and current fee amount
85          print("Name: " + self.name)
86          print("Roll No: " + self.roll_no)
87          print("Hostel Status: " + self.hostel_status)
88          print("Fee Amount: " + str(self.fee_amount))
89
90  # Example usage:
91  # Create an instance of sru_student with name "Alice", roll number "SRU123", and hostel status "Yes"

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

anumandlarithika@Mac AI_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab9.3/task02.py
Output:
-----
Name: Alice
Roll No: SRU123
Hostel Status: Yes
Fee Amount: 5000
● anumandlarithika@Mac AI_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab9.3/task02.py
Name: Alice
Roll No: SRU123
Hostel Status: Yes
Fee Amount: 5000
● anumandlarithika@Mac AI_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab9.3/task02.py
Name: Alice
Roll No: SRU123
Hostel Status: Yes
Fee Amount: 5000
% anumandlarithika@Mac AI_Assisted_lab %

```

Ln 93, Col 1 Spaces: 4 UTF-8 LF () Python 3.9.6

Explanation :

In this task, I created a Python class called `sru_student` with attributes like name, roll number, and hostel status.

First, I wrote inline comments manually to explain each part of the code.

Then, I used an AI tool to automatically generate inline comments.

After comparing them, I found:

- AI added comments for almost every line.
- Some AI comments were too detailed.
- Manual comments were shorter and focused.
- AI comments are helpful but need checking.

This task helped me understand how AI can automatically explain code logic.

TASK 3 – Module-Level & NumPy Style Documentation :

Prompt 1 – Generate Calculator Module :

The screenshot shows a terminal window with the following content:

```
lab9.3 > task03.py ...
1 # Write a Python calculator module containing these functions:
2
3 # - add(a, b)
4 # - subtract(a, b)
5 # - multiply(a, b)
6 # - divide(a, b)
7
8 # Keep the code clean and simple.
9 # Define a calculator module with basic arithmetic functions
10 def add(a, b):
11     """Return the sum of a and b."""
12     return a + b
13 def subtract(a, b):
14     """Return the difference of a and b."""
15     return a - b
16 def multiply(a, b):
17     """Return the product of a and b."""
18     return a * b
19 def divide(a, b):
20     """Return the quotient of a and b. Raises ValueError if b is zero."""
21     if b == 0:
22         raise ValueError("Cannot divide by zero.")
23     return a / b
24 # Example usage:
25 if __name__ == "__main__":
26     print("Addition:", add(10, 5))      # Output: 15
27     print("Subtraction:", subtract(10, 5)) # Output: 5
28     print("Multiplication:", multiply(10, 5)) # Output: 50
29     print("Division:", divide(10, 5))    # Output: 2.0
30     # Uncommenting the following line will raise a ValueError
31     # print("Division by zero:", divide(10, 0))
32
33
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+/usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab9.3/task03.py
● anumandlarithika@Mac_AI_Assisted_lab: ~ % /usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab9.3/task03.py
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
% anumandlarithika@Mac_AI_Assisted_lab: ~
```

Prompt 2 – Add NumPy Style Docstrings :

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** Contains tabs for "task01.py", "task02.py", "task03.py", and "task03.py".
- Search Bar:** Displays "AI_Assisted_lab".
- Code Area:** Shows Python code for a function named "add". The code includes docstrings with placeholders for parameters, returns, examples, and code snippets.
- Completion Panel:** A floating panel on the right side displays code completion suggestions for the current line of code.
- Bottom Navigation:** Includes tabs for "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", "TERMINAL", and "PORTS".
- Terminal Output:** Shows command-line interactions with Python, including addition, subtraction, multiplication, division, and assignment operations.

Prompt 3 – Generate Module-Level Docstring :

```
task01.py task02.py task03.py
151 # Generate a professional module-level docstring for this calculator module.
152 # The docstring must include:
153 # - Module overview
154 # - List of functions
155 # - Brief explanation of each function
156 # - Example usage
157
158 # Follow standard Python documentation practices.
159 """
160 Calculator Module
161 This module provides basic arithmetic functions for performing addition, subtraction, multiplication, and division.
162 Functions_____
163 - add(a, b): Returns the sum of a and b.
164 - subtract(a, b): Returns the difference of a and b.
165 - multiply(a, b): Returns the product of a and b.
166 - divide(a, b): Returns the quotient of a and b. Raises ValueError if b is zero.
167 Example Usage:
168
169 >>> from calculator import add, subtract, multiply, divide
170 >>> add(10, 5)
171 15
172 >>> subtract(10, 5)
173 5
174 >>> multiply(10, 5)
175 50
176 >>> divide(10, 5)
177 2.0
178 >>> divide(10, 0)
179 Traceback (most recent call last):
180 ...
181 ValueError: Cannot divide by zero.
182
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
/usr/bin/python3 /Users/anumandararithika/SRU/Al Assisted_lab/lab9.3/task03.py
anumandararithika@Mac Al Assisted_lab % /usr/bin/python3 /Users/anumandararithika/SRU/Al Assisted_lab/lab9.3/task03.py
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
anumandararithika@Mac Al Assisted_lab %
Ln 151, Col 77 Spaces: 4 UTF-8 LF () Python 3.9.6
```

Explanation :

In this task, I created a small calculator module with functions like add, subtract, multiply, and divide.

First, I wrote NumPy-style docstrings manually for each function.

Then, I used AI to generate:

- A module-level docstring
- Function-level docstrings

After comparing them, I observed:

- AI documentation was well structured.
- It followed proper formatting.
- Manual documentation was simple but correct.
- AI helps in maintaining consistency in large projects.

This task helped me understand structured documentation for multiple functions.