

ASSIGNMENT-9.3

Name: J.Vyshnavi

HT. No: 2303A51895

Batch: 08

Lab 9: Documentation Generation – Automatic Documentation and Code Comments

Task 1: Basic Docstring Generation

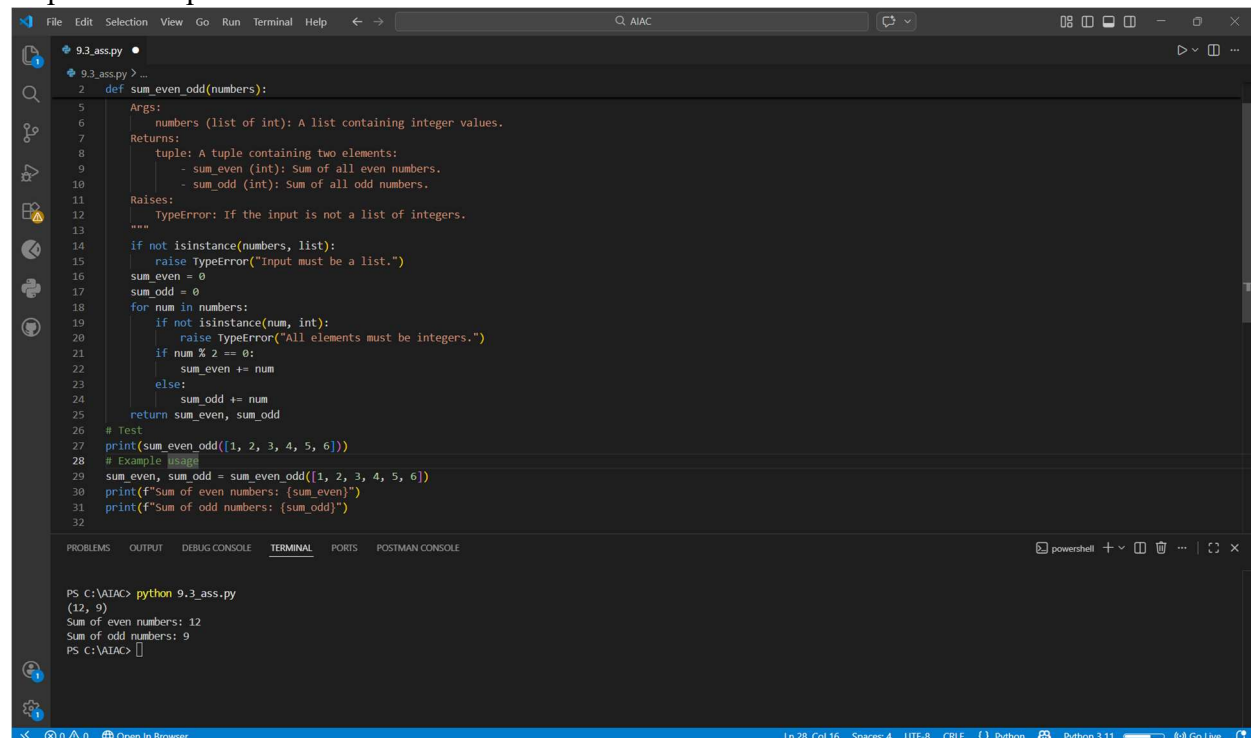
Scenario

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



```
File Edit Selection View Go Run Terminal Help
9.3_ass.py
2 def sum_even_odd(numbers):
5     Args:
6         numbers (list of int): A list containing integer values.
7     Returns:
8         tuple: A tuple containing two elements:
9             - sum_even (int): Sum of all even numbers.
10            - sum_odd (int): Sum of all odd numbers.
11     Raises:
12         TypeError: If the input is not a list of integers.
13     """
14     if not isinstance(numbers, list):
15         raise TypeError("Input must be a list.")
16     sum_even = 0
17     sum_odd = 0
18     for num in numbers:
19         if not isinstance(num, int):
20             raise TypeError("All elements must be integers.")
21         if num % 2 == 0:
22             sum_even += num
23         else:
24             sum_odd += num
25     return sum_even, sum_odd
26 # Test
27 print(sum_even_odd([1, 2, 3, 4, 5, 6]))
28 # Example usage
29 sum_even, sum_odd = sum_even_odd([1, 2, 3, 4, 5, 6])
30 print(f"Sum of even numbers: {sum_even}")
31 print(f"Sum of odd numbers: {sum_odd}")
32

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\AIAC> python 9.3_ass.py
(12, 9)
Sum of even numbers: 12
Sum of odd numbers: 9
PS C:\AIAC>
```

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

The screenshot shows a VS Code editor window with a Python file named `9.3_ass.py`. The code defines a `sru_student` class with methods `__init__`, `fee_update`, and `display_details`. It also includes example usage code that creates two student objects and prints their details. The terminal at the bottom shows the output of the script, which matches the expected output.

```

33 # Add meaningful inline comments for each logical block in this Python class. Avoid redundant comments and explain purpose clearly.
34 class sru_student:
35     def __init__(self, name, roll_no, hostel_status):
36         # Initialize student attributes
37         self.name = name
38         self.roll_no = roll_no
39         self.hostel_status = hostel_status
40
41     def fee_update(self, amount):
42         # Update the student's fee
43         self.fee = amount
44         print("Fee updated successfully")
45
46     def display_details(self):
47         # Display all student details
48         print("Name:", self.name)
49         print("Roll No:", self.roll_no)
50         print("Hostel Status:", self.hostel_status)
51         print("Fee:", getattr(self, 'fee', "Not Updated"))
52
53 # Example usage
54 student1 = sru_student("Alice", "12345", "Yes")
55 student2 = sru_student("Bob", "67890", "No")
56 print("Student 1 Details:")
57 student1.display_details()
58 print("\nStudent 2 Details:")
59 student2.display_details()

```

Terminal Output:

```

Name: Alice
Roll No: 12345
Hostel Status: Yes
Fee: Not Updated

Student 2 Details:
Name: Bob
Roll No: 67890
Hostel Status: No
Fee: Not Updated
PS C:\AIAC>

```

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

This screenshot shows the first part of a Python calculator module in a VS Code editor. The code includes a docstring, a module-level docstring, and three functions: `add`, `subtract`, and `multiply`. Each function has a docstring describing its purpose, arguments, and return value, along with an example. The `add` function returns the sum of two numbers, `subtract` returns the difference, and `multiply` returns the product. The terminal at the bottom shows the command `python 9_3_ass.py` being executed, resulting in the output: `5`, `3`, `6`, and `3.0`.

```
60 #Generate a professional module-level docstring for this calculator module. Use NumPy style. Include description and usage example.
61 ---
62 Calculator Module
63 -----
64 This module provides basic arithmetic operations such as addition, subtraction, multiplication, and division. It is designed to be simple and easy to use for basic calculations.
65 ---
66 ---
67 def add(a, b):
68     """Add two numbers.
69     Args:
70         a (float): The first number.
71         b (float): The second number.
72     Returns:
73         float: The sum of a and b.
74     Example:
75         >>> add(2, 3)
76         5
77     """
78     return a + b
79 def subtract(a, b):
80     """Subtract one number from another.
81     Args:
82         a (float): The number to be subtracted from.
83         b (float): The number to subtract.
84     Returns:
85         float: The difference of a and b.
86     Example:
87         >>> subtract(5, 2)
88         3
89     """
90     return a - b
91 def multiply(a, b):
92     """Multiply two numbers.
93     Args:
94         a (float): The first number.
95         b (float): The second number.
96     Returns:
97         float: The product of a and b.
98     Example:
99         >>> multiply(2, 3)
100         6
101     """
```

This screenshot shows the second part of the Python calculator module. It includes the `divide` function, which takes a numerator and a denominator and returns their quotient. It also includes a `print` statement that calls all four functions. The terminal at the bottom shows the command `python 9_3_ass.py` being executed, resulting in the output: `5`, `3`, `6`, and `3.0`.

```
91 def multiply(a, b):
92     """Multiply two numbers.
93     Args:
94         a (float): The first number.
95         b (float): The second number.
96     Returns:
97         float: The product of a and b.
98     Example:
99         >>> multiply(2, 3)
100         6
101     """
102     return a * b
103 def divide(a, b):
104     """Divide one number by another.
105     Args:
106         a (float): The numerator.
107         b (float): The denominator.
108     Returns:
109         float: The quotient of a and b.
110     Raises:
111         ValueError: If b is zero.
112     Example:
113         >>> divide(6, 2)
114         3.0
115     """
116     if b == 0:
117         raise ValueError("Cannot divide by zero.")
118     return a / b
119 print(add(2, 3))
120 print(subtract(5, 2))
121 print(multiply(2, 3))
122 print(divide(6, 2))
```