

ASSIGNMENT-9.3

Name: V.Pranavith

HT. No: 2303A51488

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

The screenshot shows a Python script named `9.3_ass.py` in a code editor. The code defines a function `sum_even_odd` that takes a list of integers and returns a tuple containing the sum of even numbers and the sum of odd numbers. It includes type checking and handling for non-list inputs. A test block at the bottom demonstrates the function's usage.

```
File Edit Selection View Go Run Terminal Help ← → Q, AIAC
```

```
9.3_ass.py ●
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> [ ]
```

X Open in Browser In 28 Col 16 Spaces: 4 UTF-8 CRLF Python Python 3.11 Go Live

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 code editor with a Python script named `9.3_ass.py`. The code defines a class `sru_student` with methods for initializing student details, updating fee, displaying details, and printing example usage. It also creates two instances of the class, `student1` and `student2`, and prints their details. The code editor interface includes a toolbar with icons for file operations, a search bar, and a terminal tab at the bottom.

```
File Edit Selection View Go Run Terminal Help ← → 🔍 AIACTools
```

```
9.3_ass.py x
9.3_ass.py > ...
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()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
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:\AIACTools>
```

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

The screenshot shows the Visual Studio Code interface. The code editor displays a Python script named `93_ass.py`. The script defines a `Calculator Module` with four functions: `add`, `subtract`, `multiply`, and `divide`. The `add` function adds two numbers. The `subtract` function subtracts one number from another. The `multiply` function multiplies two numbers. The `divide` function divides one number by another, raising a `ValueError` if the denominator is zero. The terminal below the editor shows the output of running the script with inputs 2 and 3, resulting in 5 for addition and 3.0 for multiplication.

```
# Generate a professional module-level docstring for this calculator module. Use NumPy style. Include description and usage example.
"""
Calculator Module
=====
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.

"""

def add(a, b):
    """Add two numbers.
    Args:
        a (float): The first number.
        b (float): The second number.
    Returns:
        float: The sum of a and b.
    Example:
        >>> add(2, 3)
        5
    """
    return a + b

def subtract(a, b):
    """Subtract one number from another.
    Args:
        a (float): The number to be subtracted from.
        b (float): The number to subtract.
    Returns:
        float: The difference of a and b.
    Example:
        >>> subtract(5, 2)
        3
    """
    return a - b

def multiply(a, b):
    """Multiply two numbers.
    Args:
        a (float): The first number.
        b (float): The second number.
    Returns:
        float: The product of a and b.
    Example:
        >>> multiply(2, 3)
        6
    """
    return a * b

def divide(a, b):
    """Divide one number by another.
    Args:
        a (float): The numerator.
        b (float): The denominator.
    Returns:
        float: The quotient of a and b.
    Raises:
        ValueError: If b is zero.
    Example:
        >>> divide(6, 2)
        3.0
    """
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b

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

This screenshot shows the same VS Code session after changes were made to the `divide` function. The code editor now includes a check for zero division, raising a `ValueError` with a descriptive message. The terminal shows the updated behavior: when attempting to divide by zero, it prints the error message "Cannot divide by zero." instead of returning a float value.

```
# Generate a professional module-level docstring for this calculator module. Use NumPy style. Include description and usage example.
"""
Calculator Module
=====
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.

"""

def multiply(a, b):
    """Multiply two numbers.
    Args:
        a (float): The first number.
        b (float): The second number.
    Returns:
        float: The product of a and b.
    Example:
        >>> multiply(2, 3)
        6
    """
    return a * b

def divide(a, b):
    """Divide one number by another.
    Args:
        a (float): The numerator.
        b (float): The denominator.
    Returns:
        float: The quotient of a and b.
    Raises:
        ValueError: If b is zero.
    Example:
        >>> divide(6, 2)
        3.0
    """
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b

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