

AI Assisted Coding

Assignment 9.3

Name: Syed Murtaza

Hall ticket no: 2303A51259

Batch no: 19

Task 1: Basic Docstring Generation

Prompt:

Write a Python function that returns the sum of even numbers and the sum of odd numbers in a given list.

Add a Google Style docstring manually and then generate a docstring using AI assistance for the same function.

Code & Output:

```
L9_1.PY x
L9_1.PY > ...
1 #Generate a python program to return the sum of even
2     numbers and sum of odd numbers in a given List of numbers.
3     Function-Level docstring should be included.
4
5 def sum_even_odd(numbers):
6     """
7         This function takes a list of numbers as input and
8             returns the sum of even numbers and the sum of odd
9                 numbers in the list.
10
11    Parameters:
12        numbers (list): A list of integers.
13
14    Returns:
15        tuple: A tuple containing the sum of even numbers and
16            the sum of odd numbers.
17    """
18
19
20
21
22
23
24
25
26

L9_1.PY x
L9_1.PY > ...
1 def sum_even_odd(numbers):
2     sum_even = 0
3     sum_odd = 0
4
5     for num in numbers:
6         if num % 2 == 0:
7             sum_even += num
8         else:
9             sum_odd += num
10
11     return sum_even, sum_odd
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

CHAT

PORTS

TERMINAL

powershell

```
PS C:\Users\known\OneDrive\Desktop\Py\AIAC> python L9_1.py
PS C:\Users\known\OneDrive\Desktop\Py\AIAC> python L9_1.py
Sum of even numbers: 12
Sum of odd numbers: 9
PS C:\Users\known\OneDrive\Desktop\Py\AIAC>
```

CHAT

PORTS

TERMINAL

powershell

```
PS C:\Users\known\OneDrive\Desktop\Py\AIAC> python L9_1.py
PS C:\Users\known\OneDrive\Desktop\Py\AIAC> python L9_1.py
Sum of even numbers: 12
Sum of odd numbers: 9
PS C:\Users\known\OneDrive\Desktop\Py\AIAC>
```

```

❶ L9_1.PY > ...
1 def sum_even_odd(numbers):
2     """
3         Calculates the sum of even and odd numbers in a list.
4
5         This function iterates through a list of integers and
6         computes
7         two separate sums: one for even numbers and one for odd
8         numbers.
9
10    Args:
11        numbers (list[int]): A list of integers to be
12            processed.
13
14    Returns:
15        tuple[int, int]: A tuple containing:
16            - sum_even (int): Sum of all even numbers in
17            the list.
18            - sum_odd (int): Sum of all odd numbers in the
19            list.
20
21    Raises:
22        TypeError: If the input is not a list of integers.
23
24    Example:
25        >>> sum_even_odd([1, 2, 3, 4, 5])
26        (6, 9)
27
28    """
29    if not isinstance(numbers, list):
30        raise TypeError("Input must be a list of integers.")
31
32    sum_even = 0
33
34    for num in numbers:
35        if not isinstance(num, int):
36            raise TypeError("All elements must be integers.
37            ")
38
39        if num % 2 == 0:
40            sum_even += num
41        else:
42            sum_odd += num
43
44    return sum_even, sum_odd

```

> PORTS
▼ TERMINAL
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L9_1.py
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L9_1.py
Sum of even numbers: 12
Sum of odd numbers: 9
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>

> PORTS
▼ TERMINAL
powershell
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L9_1.py
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L9_1.py
Sum of even numbers: 12
Sum of odd numbers: 9
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>

> PORTS
▼ TERMINAL
powershell
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L9_1.py
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L9_1.py
Sum of even numbers: 12
Sum of odd numbers: 9
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>

Explanation:

The manual docstring provides a more descriptive explanation of the function's purpose and return structure. It clearly states the argument type and explains the output in sentence form. The AI-generated docstring is shorter and more concise but still correct. However, it lacks deeper descriptive clarity compared to the manual version. This comparison highlights that AI-generated documentation is accurate but sometimes less detailed, requiring human refinement for completeness.

Task 2: Automatic Inline Comments

Prompt:

Generate a Python class named sru_student with attributes name, roll_no, and hostel_status, and methods fee_update() and display_details(). Add inline comments automatically.

Code & Output:

The screenshot shows a terminal window with two panes. The left pane displays the code for a Python class named SRUStudentClass. The right pane shows the output of running the script in a terminal, displaying the student's details and the updated fee.

```
#Generate a python program for sru_student_class,
#with attributes name,rollno,hostel_status and methods
#like fee_update and display_details.

class SRUStudentClass:
    def __init__(self, name, rollno, hostel_status):
        #Constructor to initialize the attributes
        self.name = name
        self.rollno = rollno
        self.hostel_status = hostel_status
        self.fee = 0

    def fee_update(self, amount):#Method to update
        the fee
        self.fee += amount

class SRUStudentClass:
    print(f"Fee updated. Current fee: {self.fee}")

    def display_details(self):#Method to display the
        details of the student
        print(f"Name: {self.name}")
        print(f"Roll No: {self.rollno}")
        print(f"Hostel Status: {self.hostel_status}")
        print(f"Current Fee: {self.fee}")

# Example usage
student1 = SRUStudentClass("Alice", "12345", "Hostel")
student1.display_details()
student1.fee_update(5000)
student1.display_details()
```

PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L9_1.py
Name: Alice
Roll No: 12345
Hostel Status: Hostel
Current Fee: 0
Fee updated. Current fee: 5000
Name: Alice
Roll No: 12345
Hostel Status: Hostel
Current Fee: 5000
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>

PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L9_1.py
Name: Alice
Roll No: 12345
Hostel Status: Hostel
Current Fee: 0
Fee updated. Current fee: 5000
Name: Alice
Roll No: 12345
Hostel Status: Hostel
Current Fee: 5000
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>

The screenshot shows a code editor with two tabs: L9_1.py and L9_1.py > The code is as follows:

```

1 #Generate a python program for sru_student_class, with
2 #attributes name,rollno,hostel_status and methods like
3 #fee_update and display_details.
4
5 class SRUStudentClass:
6     def __init__(self, name, rollno, hostel_status):
7         #Constructor to initialize the attributes of the
8         #student class
9         self.name = name
10        self.rollno = rollno
11        self.hostel_status = hostel_status
12        self.fee = 0
13
14    def fee_update(self, amount):#Method to update the fee
15        #of the student. It takes an amount as input and adds it
16        #to the current fee.
17        self.fee += amount
18
19 student1 = SRUStudentClass("Alice", "12345", "Hostel")
20 student1.display_details()
21 student1.fee_update(5000)
22 student1.display_details()

```

To the right of the code editor are two terminal panes showing the execution of the script and its output.

Terminal 1 (Left):

```

PS C:\Users\Known\OneDrive\Desktop\Py\AIAC> python L9_1.py
Name: Alice
Roll No: 12345
Hostel Status: Hostel
Current Fee: 0
Fee updated. Current fee: 5000
Name: Alice
Roll No: 12345
Hostel Status: Hostel
Current Fee: 5000
PS C:\Users\Known\OneDrive\Desktop\Py\AIAC>

```

Terminal 2 (Right):

```

PS C:\Users\Known\OneDrive\Desktop\Py\AIAC> python L9_1.py
Name: Alice
Roll No: 12345
Hostel Status: Hostel
Current Fee: 0
Fee updated. Current fee: 5000
Name: Alice
Roll No: 12345
Hostel Status: Hostel
Current Fee: 5000
PS C:\Users\Known\OneDrive\Desktop\Py\AIAC>

```

Explanation:

Manual comments are precise and written with developer intent. AI-generated comments are also correct but slightly generic. AI explains obvious lines but may miss contextual explanations. This shows that AI can accelerate documentation but human review is necessary to ensure meaningful and non-redundant comments.

Task 3: Module-Level and Function-Level Documentation

Prompt:

Generate a Python calculator module with functions add, subtract, multiply, and divide. Add NumPy-style docstrings manually and then generate module-level and function-level documentation using AI assistance.

Code & Output:

The image shows three screenshots of a terminal window from a development environment, likely PyCharm, demonstrating the execution of a Python script named L9_1.py. The script contains definitions for three functions: add, subtract, and multiply, each with detailed docstrings explaining parameters and returns.

L9_1.py Content:

```
1 #generate a python program for calculator containing 3-4 functions
2 #numpy style docstring
3 def add(a, b):
4     """
5         Add two numbers.
6
7         Parameters
8         -----
9         a : float
10            The first number.
11        b : float
12            The second number.
13
14        Returns
15        -----
16        float
17            The sum of a and b.
18    """
19
20 def subtract(a, b):
21     """
22         Subtract two numbers.
23
24         Parameters
25         -----
26         a : float
27            The first number.
28        b : float
29            The second number.
30
31        Returns
32        -----
33        float
34            The difference of a and b.
35
36    return a - b
37
38 def multiply(a, b):
39     """
40         Multiply two numbers.
41
42         Parameters
43         -----
44         a : float
45            The first number.
46        b : float
47            The second number.
48
49        Returns
50        -----
51        float
52            The product of a and b.
53
54    return a * b
```

Terminal Output:

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L9_1.py
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

The terminal shows the results of running the script with the command `python L9_1.py`. The output displays the results of four operations: Addition (15), Subtraction (5), Multiplication (50), and Division (2.0).

```
L9_1.PY
def divide(a, b):
    """
    Divide two numbers.

    Parameters
    -----
    a : float
        The first number.
    b : float
        The second number.

    Returns
    -----
    float
        The quotient of a and b.
    """
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b

L9_2.PY
# Example usage
if __name__ == "__main__":
    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)}")

PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L9_1.py
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

Explanation:

Manual NumPy-style docstrings follow a structured scientific documentation format with sections for parameters and returns. AI-generated documentation is concise and suitable for module overviews but lacks deep parameter-level detailing. AI performs well in summarization, while manual documentation provides stronger technical clarity.

Final Conclusion:

This lab demonstrated the role of AI in generating documentation and comments automatically. AI-assisted tools significantly reduce documentation effort and improve consistency. However, human review remains essential to ensure clarity, correctness, and contextual relevance in software documentation.