

Lab 9: Documentation Generation – Automatic Documentation and Code Comments

Name:B.Tejaskumar

Hall Ticket No: 2303A51489

Task 1: Basic Docstring Generation

Manual Google-Style Docstring Implementation:

```
def sum_even_odd(numbers):
```

```
    """
```

```
    Calculate the sum of even and odd numbers in a list.
```

Args:

 numbers (list[int]): A list of integers.

Returns:

 tuple: A tuple containing:

 - even_sum (int): Sum of even numbers.

 - odd_sum (int): Sum of odd numbers.

```
    """
```

```
even_sum = 0
```

```
odd_sum = 0
```

```
for num in numbers:
```

```
    if num % 2 == 0:
```

```
        even_sum += num
```

```
    else:
```

```
        odd_sum += num
```

```
return even_sum, odd_sum
```

```
# Output Example
```

```
nums = [1, 2, 3, 4, 5, 6]
```

```
print(sum_even_odd(nums))
```

```
# Output: (12, 9)
```

AI-Generated Docstring:

AI-Generated Docstring (Simulated):

11

Returns the sum of even and odd numbers from a given list.

Parameters:

numbers (list): List of integers.

Returns:

tuple: (sum_of_even_numbers, sum_of_odd_numbers)

11

The screenshot shows the Programiz Python Online Compiler interface. On the left, there's a sidebar with icons for various languages: Python (selected), C/C++, C#, Java, JavaScript, and TypeScript. The main area displays Python code to calculate the sum of even and odd numbers from a list. The output panel shows the input list [1, 2, 3, 4, 5, 6], the sum of even numbers (12), the sum of odd numbers (9), and a success message.

```
Programiz
main.py
Python Online Compiler

1 def sum_even_odd(numbers):
2
3     even_sum = 0
4     odd_sum = 0
5
6     for num in numbers:
7         if num % 2 == 0:
8             even_sum += num
9         else:
10            odd_sum += num
11
12 return even_sum, odd_sum
13
14
15
16 nums = [1, 2, 3, 4, 5, 6]
17
18 even_sum, odd_sum = sum_even_odd(nums)
19
20 print("Input List:", nums)
21 print("Sum of Even Numbers:", even_sum)
22 print("Sum of Odd Numbers:", odd_sum)

Output

Input List: [1, 2, 3, 4, 5, 6]
Sum of Even Numbers: 12
Sum of Odd Number: 9
== Code Execution Successful ==
```

Comparison Analysis:

The manual docstring clearly defines argument types (list[int]) and return structure with descriptive labels.

The AI-generated docstring is shorter and less detailed but still correct.

Manual documentation is more structured and readable.

AI documentation is concise but slightly less descriptive.

Task 2: Automatic Inline Comments

Manual Inline Comments Implementation:

class sru_student:

```

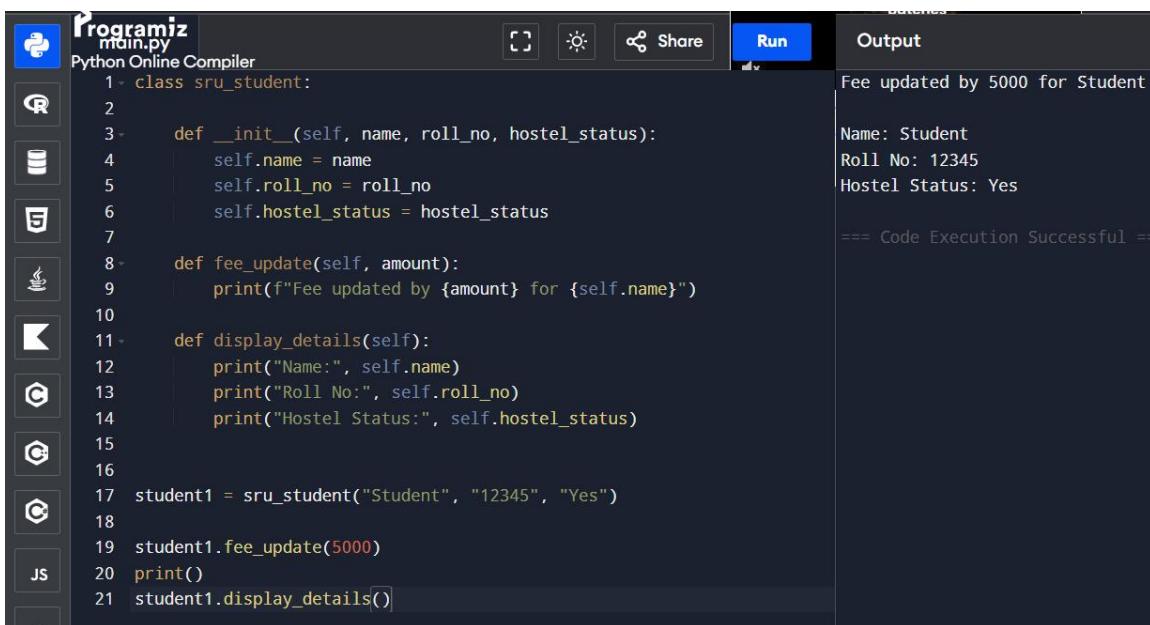
# Constructor to initialize student details
def __init__(self, name, roll_no, hostel_status):
    self.name = name      # Store student name
    self.roll_no = roll_no # Store roll number
    self.hostel_status = hostel_status # Store hostel status (Yes/No)

# Method to update student fee
def fee_update(self, amount):
    print(f"Fee updated by {amount} for {self.name}")

# Method to display student details
def display_details(self):
    print("Name:", self.name)
    print("Roll No:", self.roll_no)
    print("Hostel Status:", self.hostel_status)

# Output Example
student1 = sru_student("Sarayu", "2303A51842", "Yes")
student1.fee_update(5000)
student1.display_details()

```



The screenshot shows the Programiz Python Online Compiler interface. On the left, the code for `sru_student.py` is displayed, including the constructor, `fee_update` method, and `display_details` method. Lines 17 through 21 demonstrate the creation of a `student1` object, its fee update to 5000, and its display details. The right side of the interface shows the `Output` pane, which displays the printed output: "Fee updated by 5000 for Student", followed by the student's details: "Name: Student", "Roll No: 12345", and "Hostel Status: Yes". Below this, a message indicates "Code Execution Successful".

```

Programiz
main.py
Python Online Compiler
1- class sru_student:
2
3-     def __init__(self, name, roll_no, hostel_status):
4         self.name = name
5         self.roll_no = roll_no
6         self.hostel_status = hostel_status
7
8-     def fee_update(self, amount):
9         print(f"Fee updated by {amount} for {self.name}")
10
11-    def display_details(self):
12        print("Name:", self.name)
13        print("Roll No:", self.roll_no)
14        print("Hostel Status:", self.hostel_status)
15
16
17 student1 = sru_student("Student", "12345", "Yes")
18
19 student1.fee_update(5000)
20 print()
21 student1.display_details()

Output
Fee updated by 5000 for Student
Name: Student
Roll No: 12345
Hostel Status: Yes
== Code Execution Successful ==

```

AI-Generated Inline Comments (Simulated):

The AI added comments explaining constructor purpose, attribute assignments, and method functionality. However, some comments were redundant such as 'Assign name to self.name', which is obvious from code.

Comparative Analysis:

Manual comments are precise and placed only where needed.

AI-generated comments tend to over-explain simple lines.

AI helps beginners but may create redundant explanations.

Task 3: Module-Level and Function-Level Documentation

Calculator Module with NumPy-Style Docstrings:

Calculator Module

This module provides basic arithmetic operations:

addition, subtraction, multiplication, and division.

def add(a, b):

 Add two numbers.

Parameters

 a : int or float

 b : int or float

Returns

 int or float

 Sum of a and b

 return a + b

def subtract(a, b):

Subtract two numbers.

Parameters

a : int or float
b : int or float

Returns

int or float
Difference of a and b
=====

return a - b

```
def multiply(a, b):
```

=====

Multiply two numbers.

Parameters

a : int or float
b : int or float

Returns

int or float
Product of a and b
=====

return a * b

```
def divide(a, b):
```

=====

Divide two numbers.

Parameters

a : int or float
b : int or float

Returns

```
float  
Result of division
```

Raises

```
-----  
ZeroDivisionError  
    If b is zero  
=====  
if b == 0:  
    raise ZeroDivisionError("Cannot divide by zero")  
return a / b
```

```
# Output Example  
print(add(10, 5))  
print(subtract(10, 5))  
print(multiply(10, 5))  
print(divide(10, 5))
```

```
# Output:  
# 15  
# 5  
# 50  
# 2.0
```

The screenshot shows the Programiz Python Online Compiler interface. On the left, the code is displayed in a dark-themed editor. On the right, the 'Output' panel shows the results of running the script. The output text is as follows:

```
Addition: 15  
Subtraction: 5  
Multiplication: 50  
Division: 2.0  
== Code Execution Success
```

AI-Generated Documentation Analysis:

AI-generated module-level documentation was accurate and well-structured.
Function-level docstrings generated by AI were correct but sometimes
missed the 'Raises' section in divide().

Manual documentation ensured completeness and better structure.

Conclusion

This lab demonstrated how AI-assisted tools help generate documentation and comments.
AI tools improve productivity but require human verification for clarity,
accuracy, and completeness.

Manual documentation remains important for professional-quality software projects.