

AI Assisted Coding

Lab 9: Documentation Generation – Automatic Documentation and Code Comments

Name: Mujtaba Ahmed

Hall ticket no: 2303A510C2

Batch no: 19

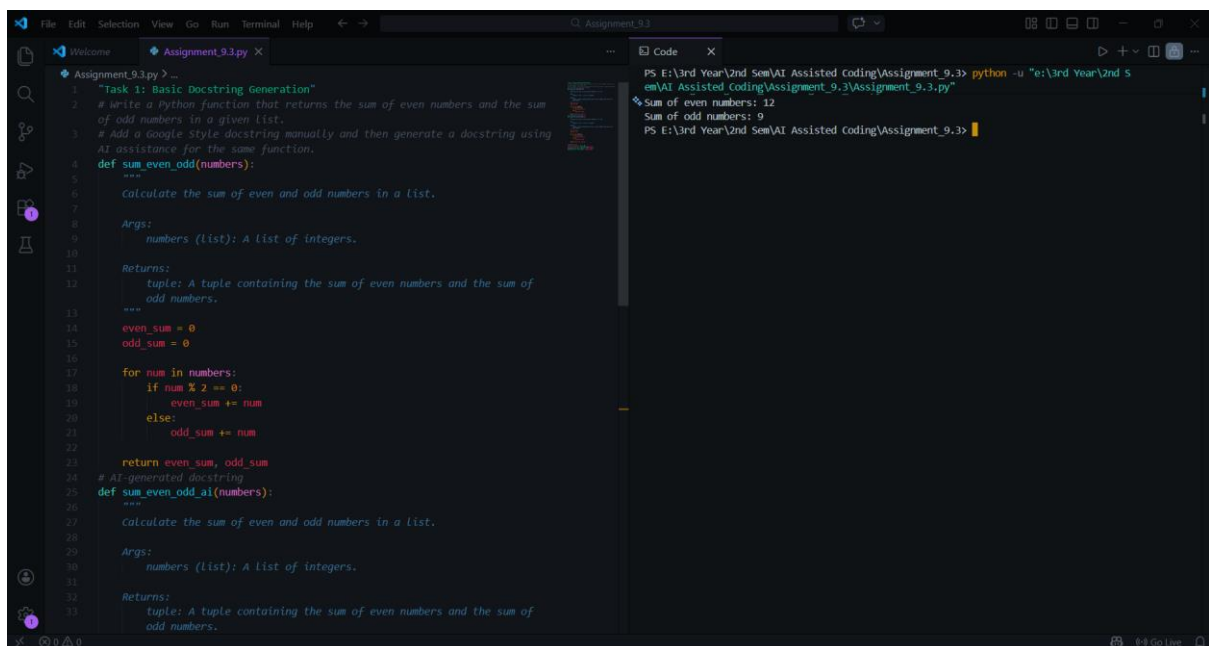
Task 1: Basic Docstring Generation

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:



```
File Edit Selection View Go Run Terminal Help
Assignment_9.3.py X
Welcome
Task 1: Basic Docstring Generation
# 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.
def sum_even_odd(numbers):
    """
    Calculate the sum of even and odd numbers in a list.

    Args:
        numbers (list): A list of integers.

    Returns:
        tuple: A tuple containing the sum of even numbers and the 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
# AI-generated docstring
def sum_even_odd_ai(numbers):
    """
    Calculate the sum of even and odd numbers in a list.

    Args:
        numbers (list): A list of integers.

    Returns:
        tuple: A tuple containing the sum of even numbers and the 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

PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_9.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_9.3\Assignment_9.3.py"
Sum of even numbers: 12
Sum of odd numbers: 9
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_9.3>
```

The screenshot shows a code editor with a Python file named 'Assignment_9.3.py'. The code defines a function `sum_even_odd(numbers)` that calculates the sum of even and odd numbers in a list. It includes two docstrings: a manually written one and an AI-generated one. The manually written docstring is more detailed, specifying the parameter type and the return value. The AI-generated docstring is brief. Below the docstrings, the function logic is shown, including initialization of `even_sum` and `odd_sum`, a loop to iterate over the numbers, and a return statement. An example usage is provided at the bottom. To the right, a terminal window shows the command `python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_9.3.py"` and its output: `Sum of even numbers: 12` and `Sum of odd numbers: 9`.

```
def sum_even_odd(numbers):  
    return even_sum, odd_sum  
    # AI-generated docstring  
    def sum_even_odd_ai(numbers):  
        """  
        Calculate the sum of even and odd numbers in a list.  
        Args:  
            numbers (list): A list of integers.  
        Returns:  
            tuple: A tuple containing the sum of even numbers and the 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  
    # Example usage  
    numbers = [1, 2, 3, 4, 5, 6]  
    even_sum, odd_sum = sum_even_odd(numbers)  
    print(f'Sum of even numbers: {even_sum}')  
    print(f'Sum of odd numbers: {odd_sum}')
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_9.3> python -u "e:\3rd Year\2nd 5  
em\AI Assisted Coding\Assignment_9.3\Assignment_9.3.py"  
Sum of even numbers: 12  
Sum of odd numbers: 9  
PS E:\3rd Year\2nd Sem\AI Assisted coding\Assignment_9.3>
```

Explanation:

The manually written docstring offers a clearer and more detailed description of the function's objective and the structure of its return value. It specifies the parameter type and explains the output in well-formed sentences, making it easier to understand. In contrast, the AI-generated docstring is brief and to the point, and while it is technically accurate, it does not provide the same level of depth or explanation. This comparison shows that AI-produced documentation can be correct and efficient, but it may still need human editing to improve clarity and 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:

```
51
52 "Task 2: Automatic Inline Comments"
53 # 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.
54 class sru_student:
55     def __init__(self, name, roll_no, hostel_status):
56         # Initialize the student's name, roll number, and
           hostel_status
57         self.name = name
58         self.roll_no = roll_no
59         self.hostel_status = hostel_status
60
61     def fee_update(self, new_fee):
62         # Update the fee for the student
63         self.fee = new_fee
64
65     def display_details(self):
66         # Display the student's details
67         print(f"Name: {self.name}")
68         print(f"Roll No: {self.roll_no}")
69         print(f"Hostel Status: {self.hostel_status}")
70 # Example usage
71 student1 = sru_student("Alice", "SRU123", "Active")
72 student1.fee_update(5000)
73 student1.display_details()
74
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_9.3> python -u "e:\3rd Year\2nd Sem\AI Assist
ed Coding\Assignment_9.3\Assignment_9.3.py"
Name: Alice
Roll No: SRU123
Hostel Status: Active
PS E:\3rd Year\2nd Sem\AI Assisted coding\Assignment_9.3>
```

Explanation:

Manually written comments are usually focused and reflect the developer's specific intent. AI-generated comments are accurate as well, but they can sound more general and sometimes explain code that is already obvious. This indicates that AI can speed up the documentation process, but human review is still important to ensure the comments are relevant, clear, and not repetitive.

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 screenshot shows the Visual Studio Code editor with a file named `Assignment_9.3.py`. The code defines a function `add(a, b)` with a docstring that includes a task description, a module-level comment, and NumPy-style docstrings for parameters and returns. The function body simply returns `a + b`. A terminal window on the right shows the command `python -u "E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_9.3.py"` being executed, with the output `5` and `2.0` displayed.

```
74
75 "Task 3: Module-Level and Function-Level Documentation"
76 # Generate a Python calculator module with functions add,
77 subtract, multiply, and divide. Add NumPy-style docstrings
78 manually and then generate module-level and function-level
79 documentation using AI assistance.
80
81 def add(a, b):
82     """
83     Add two numbers.
84
85     Parameters
86     -----
87     a : int or float
88         The first number.
89     b : int or float
90         The second number.
91
92     Returns
93     -----
94     int or float
95         The sum of a and b.
96     """
97     return a + b
98
99 def subtract(a, b):
100     """
101     Subtract two numbers.
102     Parameters
103     -----
104     a : int or float
105         The first number.
106     b : int or float
107         The second number.
108
109     Returns
110     -----
111     int or float
112         The difference of a and b.
113     """
114     return a - b
115
116 def multiply(a, b):
117     """
118     Multiply two numbers.
119     Parameters
120     -----
121     a : int or float
122         The first number.
123     b : int or float
124         The second number.
125
126     Returns
127     -----
128     int or float
129         The product of a and b.
130     """
131     return a * b
132
133 def divide(a, b):
134     """
135     Divide two numbers.
136     Parameters
137     -----
138     a : int or float
139         The first number.
140     b : int or float
141         The second number.
142
143     Returns
144     -----
145     float
146         The quotient of a and b.
147     """
148     return a / b
149
150 if __name__ == "__main__":
151     # Test the functions
152     print(add(2, 3))
153     print(subtract(5, 3))
154     print(multiply(2, 3))
155     print(divide(6, 3))
```

This screenshot shows the same Visual Studio Code editor, but the code has been scrolled down to the `multiply(a, b)` function. The docstring for `multiply` follows the same format as the `add` function, with a task description, a module-level comment, and NumPy-style docstrings. The function body returns `a * b`. The terminal window on the right shows the same command as the first screenshot, with the output `5` and `2.0` displayed.

```
94
95 def subtract(a, b):
96     """
97     Subtract two numbers.
98     Parameters
99     -----
100     a : int or float
101         The first number.
102     b : int or float
103         The second number.
104
105     Returns
106     -----
107     int or float
108         The difference of a and b.
109     """
110     return a - b
111
112 def multiply(a, b):
113     """
114     Multiply two numbers.
115     Parameters
116     -----
117     a : int or float
118         The first number.
119     b : int or float
120         The second number.
121
122     Returns
123     -----
124     int or float
125         The product of a and b.
126     """
127     return a * b
128
129 def divide(a, b):
130     """
131     Divide two numbers.
132     Parameters
133     -----
134     a : int or float
135         The first number.
136     b : int or float
137         The second number.
138
139     Returns
140     -----
141     float
142         The quotient of a and b.
143     """
144     return a / b
145
146 if __name__ == "__main__":
147     # Test the functions
148     print(add(2, 3))
149     print(subtract(5, 3))
150     print(multiply(2, 3))
151     print(divide(6, 3))
```

```
def divide(a, b):  
    """  
    Divide two numbers.  
    Parameters  
    -----  
    a : int or float  
        The first number.  
    b : int or float  
        The second number.  
    Returns  
    -----  
    int or float  
        The quotient of a and b.  
    Raises  
    -----  
    ValueError  
        If b is zero.  
    """  
    if b == 0:  
        raise ValueError("Cannot divide by zero.")  
    return a / b  
  
# Example usage  
print(add(10, 5))      # Output: 15  
print(subtract(10, 5)) # Output: 5  
print(multiply(10, 5)) # Output: 50  
print(divide(10, 5))   # Output: 2.0
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_9.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_9.3\Assignment_9.3.py"  
15  
5  
50  
2.0  
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_9.3>
```

Explanation:

Manual NumPy-style docstrings use a well-defined structure with clear sections for parameters and return values, which makes them more detailed and technically precise. AI-generated documentation is usually shorter and good for general summaries, but it may not provide in-depth parameter explanations. AI is efficient for summarizing, while manual documentation offers 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.