

Assignment 9.3(Ai Assisted Coding)

Htno:2303a51305

Btno:05

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

- Python function with manual Google-style docstring
- AI-generated docstring for the same function
- Comparison explaining differences between manual and AI-generated documentation
- Improved understanding of AI-generated function-level documentation.

Code:

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files: fibonacci.py, Untitled-1, docstring.py (selected), .cursor, .vscode, docstring.py, fibonacci.py, file_operations.py, sample.txt.
- Editor:** Displays the content of docstring.py:

```
1 def sum_even_odd(numbers):
2     """
3         Return the sum of even numbers and sum of odd
4
5     Args:
6         numbers: A list of integers
7
8     Returns:
9         A tuple of (sum_of_evens, sum_of_odds)
10    """
11    sum_even = sum(n for n in numbers if n % 2 == 0)
12    sum_odd = sum(n for n in numbers if n % 2 != 0)
13    return sum_even, sum_odd
14
15
16 # Example usage
17 if __name__ == "__main__":
18     nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```
- Terminal:** Shows the command PS C:\Users\sneha\OneDrive\Desktop\ai coding>.
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, TERMINAL, and other icons like powershell and Python.
- Right Panel:** Features a "Build with Agent" section with a message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." It also shows a "docstring.py" card with options to "Describe what to build next" and "Agent" settings.

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files: fibonacci.py, Untitled-1, docstring.py (selected), .cursor, .vscode, docstring.py, fibonacci.py, file_operations.py, sample.txt.
- Editor:** Displays the content of docstring.py:

```
1 def sum_even_odd(numbers):
2     return sum_even, sum_odd
3
4
5 # Example usage
6 if __name__ == "__main__":
7     nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
8     even_sum, odd_sum = sum_even_odd(nums)
9     print(f"Sum of even numbers: {even_sum}")
10    print(f"Sum of odd numbers: {odd_sum}")
11    print(sum_even_odd.__doc__)
```
- Terminal:** Shows the command PS C:\Users\sneha\OneDrive\Desktop\ai coding>.
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, TERMINAL, and other icons like powershell and Python.
- Right Panel:** Features a "Build with Agent" section with a message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." It shows a "docstring.py" card with options to "Describe what to build next" and "Agent" settings.

Output:

The screenshot shows two instances of the VS Code interface, each displaying a Python file named `docstring.py`. The code defines a function `sum_even_odd` that takes a list of integers and returns the sum of even and odd numbers.

Top Instance:

```

def sum_even_odd(numbers):
    return sum_even, sum_odd

# Example usage
if __name__ == "__main__":
    nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    even_sum, odd_sum = sum_even_odd(nums)
    print(f"Sum of even numbers: {even_sum}")
    print(f"Sum of odd numbers: {odd_sum}")

```

Bottom Instance:

```

def sum_even_odd(numbers):
    return sum_even, sum_odd

# Example usage
if __name__ == "__main__":
    nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    even_sum, odd_sum = sum_even_odd(nums)
    print(f"Sum of even numbers: {even_sum}")

```

In both instances, the AI has generated the following documentation:

Args:

- numbers: A list of integers
- numbers: A list of integers

Returns:

- A tuple of (sum_of_evens, sum_of_odds)

The bottom instance also shows the command line prompt `PS C:\Users\sneha\OneDrive\Desktop\ai coding>`.

Explanation:

-->The manual docstring is more detailed and follows proper Google style. It clearly explains the input type, return values, and includes an example, making it easier to understand.

-->The AI-generated docstring is correct but shorter. It provides basic information but lacks detailed explanation and an example.

Conclusion: AI saves time, but manual improvement makes documentation clearer and more complete.

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

- Python class with manually written inline comments
- AI-generated inline comments added to the same code
- Comparative analysis of manual vs AI comments
- Critical discussion on strengths and limitations of AI-generated comments

Code:

The screenshot shows a Google Colab notebook titled "Untitled32.ipynb". The code defines a class `srw_student` with methods for initializing student details, updating fees based on hostel status, and displaying student information. The code is as follows:

```
[10]  ✓ 0s
class srw_student:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name          # Store student's name
        self.roll_no = roll_no     # Store student's roll number
        self.hostel_status = hostel_status # Store hostel status (True/False)
        self.fee = 50000           # Base academic fee

    def fee_update(self):
        if self.hostel_status:      # If student stays in hostel
            self.fee += 20000         # Add hostel fee
        else:
            self.fee += 0             # No extra hostel fee
        return self.fee

    def display_details(self):
        print("Name:", self.name)    # Print student name
        print("Roll No:", self.roll_no) # Print roll number
        print("Hostel Status:", self.hostel_status) # Print hostel info
        print("Total Fee:", self.fee)   # Print final fee
```

The screenshot shows the same Google Colab notebook with AI-generated inline comments. The code remains identical to the first screenshot, but includes detailed comments explaining each line. For example, the `__init__` method now includes comments for each variable assignment.

```
[12]  ✓ 0s
#Ai Generated inline comments code
class srw_student:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name # Student name
        self.roll_no = roll_no # Roll number
        self.hostel_status = hostel_status # Hostel status
        self.fee = 50000 # Initial fee

    def fee_update(self):
        if self.hostel_status: # Check hostel
            self.fee += 20000 # Increase fee
        else: # Otherwise
            self.fee += 0 # No change
        return self.fee # Return fee

    def display_details(self):
        print("Name:", self.name) # Print name
        print("Roll No:", self.roll_no) # Print roll
        print("Hostel Status:", self.hostel_status) # Print hostel
```

Output:

```

Name: Sneha
Roll No: 23SR001
Hostel Status: True
Total Fee: 70000

```

Explanation:

Comparison: Manual vs AI Comments

Feature	Manual Comments	AI Comments	Analysis
Detail Level	More descriptive	Short/basic	AI lacks detailed explanation
Clarity	Explains purpose clearly	Basic meaning only	Manual helps beginners more
Redundancy	Minimal	Some redundant (e.g., "Print name")	AI often states obvious
Logic Explanation	Explains fee logic clearly	Just "increase fee"	AI misses context
Completeness	Covers all important logic	Covers lines but shallow	AI comments are surface-level

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

- Python script with manual NumPy-style docstrings
- AI-generated module-level and function-level documentation
- Comparison between AI-generated and manual documentation
- Clear understanding of structured documentation for multi-function scripts.

Code:

The image shows two screenshots of a Google Colab notebook titled "Untitled32.ipynb".

Screenshot 1:

```
"""  
Calculator module providing basic arithmetic operations.  
  
This module includes functions for addition, subtraction,  
multiplication, and division. It can be reused in different  
projects requiring simple mathematical operations.  
"""  
  
def add(a, b):  
    """  
    Add two numbers.  
  
    Parameters  
    -----  
    a : int or float  
        First number.  
    b : int or float  
        Second number.  
  
    Returns  
    -----  
    """  
    return a + b
```

Screenshot 2:

```
"""  
Sum of a and b.  
"""  
return a + b  
  
def subtract(a, b):  
    """  
    Subtract second number from first number.  
  
    Parameters  
    -----  
    a : int or float  
        First number.  
    b : int or float  
        Second number.  
  
    Returns  
    -----  
    int or float  
        Difference of a and b.  
    """  
return a - b
```

A Snipping Tool window is visible in the bottom right corner, indicating a screenshot has been taken.

The screenshot shows a Google Colab notebook titled "Untitled32.ipynb". The code cell at index [16] contains the following Python code:

```
def multiply(a, b):
    """
    Multiply two numbers.

    Parameters
    -----
    a : int or float
        First number.
    b : int or float
        Second number.

    Returns
    -----
    int or float
        Product of a and b.
    """
    return a * b

def divide(a, b):
    """
    Divide a by b.
    """
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
```

The notebook interface includes a sidebar with various icons for file operations, a toolbar with file navigation and search, and a status bar at the bottom.

word - Search | 2303a51305-Assignment 9.3(a) | google collab - Search | Untitled32.ipynb - Colab | +

Untitled32.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text | Run all

RAM Disk

[16] 0s Divide first number by second number.

Parameters

a : int or float
 Numerator.

b : int or float
 Denominator.

Returns

float
 Result of division.

Raises

ZeroDivisionError
 If b is zero.

if b == 0:
 raise ZeroDivisionError("Cannot divide by zero")

Variables Terminal

10:40 AM Python 3

The screenshot shows a Google Colab interface with the following details:

- Top Bar:** word - Search, 2303a51305-Assignment 9.3(a), google colab - Search, Untitled32.ipynb - Colab.
- Header:** Untitled32.ipynb, File, Edit, View, Insert, Runtime, Tools, Help, Share, RAM (green checkmark), Disk.
- Toolbar:** Commands, + Code, + Text, Run all.
- Code Cell:** [16] 0s

```
If b is zero.  
    """  
    if b == 0:  
        raise ZeroDivisionError("Cannot divide by zero")  
    return a / b  
  
# 📄 Printing Docstrings  
print("MODULE DOCSTRING:\n", __doc__)  
print("\nADD FUNCTION DOCSTRING:\n", add.__doc__)  
print("\nSUBTRACT FUNCTION DOCSTRING:\n", subtract.__doc__)  
print("\nMULTIPLY FUNCTION DOCSTRING:\n", multiply.__doc__)  
print("\nDIVIDE FUNCTION DOCSTRING:\n", divide.__doc__)  
  
# 📄 Testing functions  
print("\nOUTPUTS:")  
print("Add:", add(10, 5))  
print("Subtract:", subtract(10, 5))  
print("Multiply:", multiply(10, 5))  
print("Divide:", divide(10, 5))
```

- Bottom Bar:** Variables, Terminal.
- Snipping Tool Overlay:** A window titled "Snipping Tool" is overlaid on the Colab interface. It shows a screenshot of the desktop with the Snipping Tool window itself. The message in the Snipping Tool window says: "Screenshot copied to clipboard" and "Automatically saved to screenshots folder".

Output:

The screenshot shows a Google Colab notebook titled "Untitled32.ipynb". The code cell at the top contains the following Python code:

```
print("\nOUTPUTS:")
print("Add:", add(10, 5))
print("Subtract:", subtract(10, 5))
print("Multiply:", multiply(10, 5))
print("Divide:", divide(10, 5))
```

Below the code cell, there is a section titled "MODULE DOCSTRING:" containing the following text:

```
Calculator module providing basic arithmetic operations.

This module includes functions for addition, subtraction,
multiplication, and division. It can be reused in different
projects requiring simple mathematical operations.
```

Further down, there is a section titled "ADD FUNCTION DOCSTRING:" containing the following text:

```
Add two numbers.

Parameters
-----
a : int or float
    First number.
```

The screenshot shows a Google Colab notebook titled "Untitled32.ipynb". The left sidebar contains icons for file operations like New, Open, Save, and Delete. The main workspace displays two code cells. The first cell contains a function definition for adding two numbers:

```
Add two numbers.  
Parameters  
-----  
a : int or float  
    First number.  
b : int or float  
    Second number.  
  
Returns  
-----  
int or float  
    Sum of a and b.
```

The second cell contains a function definition for subtracting one number from another:

```
SUBTRACT FUNCTION DOCSTRING:  
  
Subtract second number from first number.  
  
Parameters  
-----  
a : int or float  
    First number.  
b : int or float
```

At the bottom, there are tabs for "Variables" and "Terminal", along with a blue circular "Run" button. The status bar at the bottom right shows the time as 10:40 AM and the language as Python 3.

```

a : int or float
    First number.
b : int or float
    Second number.

Returns
-----
int or float
    Difference of a and b.

MULPY FUNCTION DOCSTRING:

    Multiply two numbers.

Parameters
-----
a : int or float
    First number.
b : int or float
    Second number.

Returns
-----
int or float
    Product of a and b.

DIVIDE FUNCTION DOCSTRING:

```

Explanation:

Aspect	Manual NumPy Docstrings	AI-Generated Docstrings	Analysis
Format	Proper NumPy style	Generic style	Manual follows structured standard
Detail	Clear sections	Short descriptions	AI is brief
Error	Mentions	Often missing	Manual more complete
Handling	ZeroDivisionError		

Readability Highly structured

Simple and readable

AI easier but less formal

Conclusion:

AI is helpful for generating initial documentation, but manual refinement is required for professional-quality module and function documentation.

