AI-Assisted Coding

Name: A.Ashish

Htno:2403A51249

TASK __1

Task Description#1

Basic Docstring Generation

- Write python function to return sum of even and odd numbers in the given list.
- Incorporate manual docstring in code with Google Style • Use an AI-assisted tool (e.g., Copilot, Cursor AI) to generate a docstring describing the function.
- Compare the AI-generated docstring with your manually written one.

Expected Outcome#1: Students understand how AI can produce function-level documentation.

PROMPT:

Write a Python function that takes a list of integers and returns the sum of even and odd numbers separately. Add a docstring using Google-style format.

Manually written code:

```python
def sum_even_odd(numbers):
    """
    Returns the sum of even and odd numbers from a list of integers.

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

    Returns:
        tuple[int, int]: A tuple containing two elements:
            - The sum of even numbers.
            - 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


# Take user input
user_input = input("Enter a list of integers separated by spaces: ")

# Convert the input string into a list of integers
numbers = list(map(int, user_input.strip().split()))

# Call the function and print the result
even_sum, odd_sum = sum_even_odd(numbers)
print(f"Sum of even numbers: {even_sum}")
print(f"Sum of odd numbers: {odd_sum}")
```

```
Enter a list of integers separated by spaces: 4 5  9 12 6 3 8
Sum of even numbers: 30
Sum of odd numbers: 17
```

AI-generated Code:

```python
def sum_even_odd(numbers):
    """
    Calculates the sum of even and odd numbers in a list.

    Parameters:
        numbers (list): 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
```

Comparison:

Manual vs. AI Docstring

Aspect	Manual (Google Style)	AI-Generated (Typical Style)
Style Format	Follows Google Style Guide exactly	Follows a more generic format, similar to NumPy or Sphinx style
Argument Type Info	Specifies list[int] for type clarity	Uses list, without specifying it's a list of integers
Return Type Detail	Specifies tuple[int, int] and explains both elements	Just says tuple, and gives a general description

OBSERVATION:

The manual docstring is more detailed and follows the Google style guide, clearly specifying argument and return types. The AI-generated docstring is concise and understandable but lacks detailed type info and precise return value description. AI docstrings are useful for quick drafts but often need manual refinement for clarity and completeness. Overall, combining both can speed up documentation while maintaining quality.

TASK__2

Task Description#2

Automatic Inline Comments

- Write python program for sru_student class with attributes like name, roll no., hostel_status and fee_update method and display_details method.
- Write comments manually for each line/code block

- Ask an AI tool to add inline comments explaining each line/step. • Compare the AI-generated comments with your manually written one.

Expected Output#2:

Students critically analyze AI-generated code comments.

PROMPT:

Write a Python class sru_student with attributes name, roll_no, and hostel_status, and methods fee_update() and display_details().Add manual inline comments explaining each line of your code.Use an AI tool to generate automatic inline comments for the same code.Compare the two sets of comments and write a brief reflection discussing

MANUALLY WRITTEN CODE:

```python
# Define the sru_student class
class sru_student:
    # Initialize the student with name, roll number, and hostel status
    def __init__(self, name, roll_no, hostel_status):
        self.name = name                # Store the student's name
        self.roll_no = roll_no          # Store the student's roll number
        self.hostel_status = hostel_status  # Store the student's hostel status (Yes/No)

    # Update the fee payment status
    def fee_update(self, status):
        self.fee_paid = status          # Set the fee payment status (True or False)

    # Display all student details
    def display_details(self):
        print(f"Name: {self.name}")                  # Print student's name
        print(f"Roll Number: {self.roll_no}")        # Print student's roll number
        print(f"Hostel Status: {self.hostel_status}") # Print if student stays in hostel
        # Print fee status or 'Not updated' if fee_paid attribute does not exist
        print(f"Fee Paid: {getattr(self, 'fee_paid', 'Not updated')}")

# Take user input for student details
name = input("Enter student name: ")                    # Input student name as string
roll_no = int(input("Enter roll number: "))             # Input roll number as integer
hostel_status = input("Enter hostel status (Yes/No): ") # Input hostel status as string

# Create an instance of sru_student with the provided details
student = sru_student(name, roll_no, hostel_status)

# Take user input for fee payment status
fee_input = input("Has the student paid the fee? (yes/no): ").strip().lower()
# Convert user input to boolean value
fee_status = True if fee_input == "yes" else False
```

```python
# Update the fee payment status of the student
student.fee_update(fee_status)

# Display all student details including fee payment status
student.display_details()
```

```
Enter student name: vamshi
Enter roll number: 2016
Enter hostel status (Yes/No): yes
Has the student paid the fee? (yes/no): yes
Name: vamshi
Roll Number: 2016
Hostel Status: yes
Fee Paid: True
```

AI-GENERATED CODE:

```python
class sru_student:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name  # Assign student's name
        self.roll_no = roll_no  # Assign student's roll number
        self.hostel_status = hostel_status  # Assign hostel status

    def fee_update(self, status):
        self.fee_paid = status  # Update fee payment status

    def display_details(self):
        print(f"Name: {self.name}")  # Print student's name
        print(f"Roll Number: {self.roll_no}")  # Print student's roll number
        print(f"Hostel Status: {self.hostel_status}")  # Print hostel status
        print(f"Fee Paid: {getattr(self, 'fee_paid', 'Not updated')}")  # Print fee status or default

# Get user input
name = input("Enter student name: ")  # Input name
roll_no = int(input("Enter roll number: "))  # Input roll number
hostel_status = input("Enter hostel status (Yes/No): ")  # Input hostel status

student = sru_student(name, roll_no, hostel_status)  # Create student object

fee_input = input("Has the student paid the fee? (yes/no): ").strip().lower()  # Input fee status
fee_status = True if fee_input == "yes" else False  # Convert input to boolean

student.fee_update(fee_status)  # Update fee status
student.display_details()  # Display student info
```

```
Enter student name: vamshi
Enter roll number: 2016
Enter hostel status (Yes/No): yes
Has the student paid the fee? (yes/no): yes
Name: vamshi
Roll Number: 2016
Hostel Status: yes
Fee Paid: True
```

COMPARISON:

Manual comments are more detailed and explain the purpose behind each line, making them easier to understand for beginners. AI-generated comments are shorter and focus on describing what the code does without much detail. While AI comments are quick and consistent, they may miss deeper explanations. Combining both approaches can help create clear and e icient documentation.

OBSERVATION:

The sru_student class e ectively models a student with attributes for name, roll number, and hostel status. It includes methods to update fee payment status and display all student details clearly. The use of getattr in display_details handles cases where the fee status might not be updated yet. The code also incorporates user inputs for dynamic data entry, making it interactive and practical

TASK__3

Task Description#3

- Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).
- Incorporate manual docstring in code with NumPy Style • Use AI assistance to generate a module-level docstring + individual function docstrings.
- Compare the AI-generated docstring with your manually written one.

Expected Output#3:

Students learn structured documentation for multi-function scripts

PROMPT:

Write a Python script with four calculator functions: add, subtract, multiply, and divide.
Add manual docstrings to each function using the NumPy style. Then generate a module-level docstring and function docstrings using AI. Finally, compare the manual and AI-generated docstrings.

MANUALLY WRITTEN CODE:

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

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

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

def divide(a, b):
    """
    Divide first number by second number.

    Parameters
    ----------
    a : float
        Numerator.
    b : float
        Denominator.

    Returns
    -------
    float
        Quotient of a and b.

    Raises
    ------
    ValueError
        If b is zero.
    """
```

```
"""
Calculator Module
================

This module provides basic calculator functions including addition, subtraction,
multiplication, and division.

"""

def add(a, b):
    """
    Add two numbers.

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

    Returns
    -------
    float
        Sum of a and b.
    """
    return a + b

def subtract(a, b):
    """
    Subtract second number from first number.

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

    Returns
    -------
```

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


    # Example usage (optional)
    if __name__ == "__main__":
        x = float(input("Enter first number: "))
        y = float(input("Enter second number: "))

        print(f"{x} + {y} = {add(x, y)}")
        print(f"{x} - {y} = {subtract(x, y)}")
        print(f"{x} * {y} = {multiply(x, y)}")
        try:
            print(f"{x} / {y} = {divide(x, y)}")
        except ValueError as e:
            print(e)
```

```
Enter first number: 3
Enter second number: 5
3.0 + 5.0 = 8.0
3.0 - 5.0 = -2.0
3.0 * 5.0 = 15.0
3.0 / 5.0 = 0.6
```

AI-GENERATED CODE:

```python
def main():
    """
    Main function to interact with the user, get inputs,
    and perform calculator operations.
    """
    try:
        a = float(input("Enter the first number: "))
        b = float(input("Enter the second number: "))
    except ValueError:
        print("Invalid input. Please enter numeric values.")
        return

    print(f"Sum: {add(a, b)}")
    print(f"Difference: {subtract(a, b)}")
    print(f"Product: {multiply(a, b)}")

    try:
        print(f"Quotient: {divide(a, b)}")
    except ValueError as err:
        print(err)

if __name__ == "__main__":
    main()
```

```
Enter the first number: 4
Enter the second number: 5
Sum: 9.0
Difference: -1.0
Product: 20.0
Quotient: 0.8
```

COMPARISON:

- The manual docstrings use NumPy style with detailed sections like Parameters and Returns.
- The AI-generated docstrings follow a simpler, more compact style closer to the Google or Sphinx style, with Args and Returns.
- Both clearly explain function purpose, inputs, outputs, and errors.
- Manual docstrings are more structured; AI docstrings are more concise.

OBSERVATION:

The manual input code is simple and easy to understand but lacks error handling for invalid inputs. The AI-generated code adds input validation and organizes the logic inside a main() function, making it more robust and maintainable. Overall, the AI version improves user experience and code structure without adding much complexity.