
Program	:B.tech(CSE)
Specialization	:AIML
Course Title	:AI Assisted Coding
Course Code	:24CS002PC215
Semester	:3 rd semester
Academic Session	:2025-2026
Name of Student	:Kurapati Akshitha
Enrollment No.	:2403A52021
Batch No.	:02
Date	:24/09/2025

LAB ASSIGNMENT-9.2

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
-

Manual:-

```
[7]
✓ 0s
def sum_even_odd(numbers):
    """
    Manual Docstring (Google Style)
    Calculate sum of even and odd numbers in a list.

    Args:
        numbers (list): List of integers

    Returns:
        tuple: Sum of even numbers, Sum of odd numbers
    """
    even_sum = sum(num for num in numbers if num % 2 == 0)
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    return even_sum, odd_sum
```

Ai Generated:-

```
[6]
✓ 0s
def sum_even_odd(numbers):
    """Calculates the sum of even and odd numbers in a list.

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

    Returns:
        tuple: A tuple containing the sum of even numbers and the sum of odd numbers.
    """
    even_sum = sum(num for num in numbers if num % 2 == 0)
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    return even_sum, odd_sum
```

OUTPUT: compare both the codes in the page manual and another clearly

The two code cells containing the sum_even_odd function more clearly.

Cell KzYffotQzlsE:

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

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

```
    Args:
```

```
        numbers (list): A list of numbers.
```

```
    Returns:
```

```
        tuple: A tuple containing the sum of even numbers and the sum of odd numbers.
```

```
    """
```

```
    even_sum = sum(num for num in numbers if num % 2 == 0)
```

```
odd_sum = sum(num for num in numbers if num % 2 != 0)
return even_sum, odd_sum
```

Cell D3vP0m2y0H7A:

```
def sum_even_odd(numbers):
    """
    Manual Docstring (Google Style)
    Calculate sum of even and odd numbers in a list.

    Args:
        numbers (list): List of integers

    Returns:
        tuple: Sum of even numbers, Sum of odd numbers
    """
    even_sum = sum(num for num in numbers if num % 2 == 0)
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    return even_sum, odd_sum
```

Similarities:

- Both cells define a function named `sum_even_odd` that takes one argument, `numbers`.
- Both functions have the exact same logic to calculate the sum of even and odd numbers using list comprehensions and the `sum()` function.
- Both functions return a tuple containing two values: the sum of even numbers and the sum of odd numbers.

Differences:

- **Docstring Content and Formatting:** This is the primary difference.
 - Cell KzYffotQzlsE has a more detailed and standard Google-style docstring with a concise summary line and clear descriptions for Args and Returns.
 - Cell D3vP0m2y0H7A also attempts a Google-style docstring but includes a line "Manual Docstring (Google Style)" and has

slightly different spacing and less descriptive text in the summary line and Returns section.

In essence, the core functionality of the functions in both cells is identical. The difference lies solely in the style and content of the docstrings. Cell KzYffotQzlsE has a more conventional and complete Google-style docstring.

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

Manual:-

```
# Ai generated
class sru_student:
    # Constructor method to initialize student attributes
    def __init__(self, name, roll_no, hostel_status):
        self.name = name # Assign the provided name to the student object
        self.roll_no = roll_no # Assign the provided roll number to the student object
        self.hostel_status = hostel_status # Assign the provided hostel status to the student object (e.g., 'Yes' or 'No')
        self.fee_status = 0 # Initialize the fee status to 0 for a new student

    # Method to update the student's fee status
    def fee_update(self, amount):
        self.fee_status = amount # Update the fee_status attribute with the provided amount

    # Method to display the details of the student
    def display_details(self):
        # Print a formatted string containing the student's name, roll number, hostel status, and fee status
        print(f"Name: {self.name}, Roll No: {self.roll_no}, Hostel: {self.hostel_status}, Fee: {self.fee_status}")
```

Ai Generated:- Add detailed inline comments to this Python class. Explain the purpose of each attribute and method line by line.

```

# Manual inline comments
class sru_student:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name # Name of the student
        self.roll_no = roll_no # Roll number of the student
        self.hostel_status = hostel_status # Hostel status (Yes/No)
        self.fee_status = 0 # Initialize fee status to 0

    def fee_update(self, amount):
        self.fee_status = amount # Update the student's fee

    def display_details(self):
        # Print all student details
        print(f"Name: {self.name}, Roll No: {self.roll_no}, Hostel: {self.hostel_status}, Fee: {self.fee_status}")

```

Output:-

Compare these manual inline comments with the AI-generated inline comments below. Highlight: - Missing information - Differences in clarity or detail - Suggestions to improve the manual comments

I can compare the manual inline comments in cell OMT2VIXj5o8v with the AI-generated comments in cell VUWPTxHA5HCM.

Here's a comparison based on your points:

Manual Inline Comments (Cell OMT2VIXj5o8v):

Manual inline comments

class sru_student:

```

    def __init__(self, name, roll_no, hostel_status):
        self.name = name # Name of the student
        self.roll_no = roll_no # Roll number of the student
        self.hostel_status = hostel_status # Hostel status (Yes/No)
        self.fee_status = 0 # Initialize fee status to 0

```

```

    def fee_update(self, amount):
        self.fee_status = amount # Update the student's fee

```

```
def display_details(self):  
    # Print all student details  
    print(f"Name: {self.name}, Roll No: {self.roll_no}, Hostel: {self  
.hostel_status}, Fee: {self.fee_status}")
```

AI-Generated Inline Comments (Cell VUWPTxHA5HCM):

Ai generated

```
class sru_student:
```

```
    # Constructor method to initialize student attributes
```

```
    def __init__(self, name, roll_no, hostel_status):
```

```
        self.name = name # Assign the provided name to the studen  
t object
```

```
        self.roll_no = roll_no # Assign the provided roll number to t  
he student object
```

```
        self.hostel_status = hostel_status # Assign the provided hos  
tel status to the student object (e.g., 'Yes' or 'No')
```

```
        self.fee_status = 0 # Initialize the fee status to 0 for a new st  
udent
```

```
    # Method to update the student's fee status
```

```
    def fee_update(self, amount):
```

```
        self.fee_status = amount # Update the fee_status attribute w  
ith the provided amount
```

```
    # Method to display the details of the student
```

```
    def display_details(self):
```

```
        # Print a formatted string containing the student's name, roll  
number, hostel status, and fee status
```

```
print(f"Name: {self.name}, Roll No: {self.roll_no}, Hostel: {self
.hostel_status}, Fee: {self.fee_status}")
```

Comparison:

- **Missing Information:**

- Both sets of comments could benefit from explicitly mentioning the expected data types for the attributes (name, roll_no, hostel_status, fee_status).
- Neither set explains *why* fee_status is initialized to 0. Is it a default state, or does 0 represent something specific (like no fees paid)?
- The manual comments don't have a comment explaining the overall purpose of the __init__ method, like the AI-generated comments do ("Constructor method to initialize student attributes").

- **Differences in Clarity or Detail:**

- The AI-generated comments are generally more detailed and descriptive. For example, in __init__, they explain that the values are being "assigned to the student object." They also explicitly state "(e.g., 'Yes' or 'No')" for hostel_status, which is helpful.
- The comment for display_details in the AI-generated version is more specific, mentioning it prints a "formatted string containing the student's name, roll number, hostel status, and fee status," whereas the manual version simply says "Print all student details."

- **Suggestions to Improve the Manual Comments:**

- Add a comment before the `__init__` method explaining its role as the constructor.
- Include expected data types for attributes in the inline comments (e.g., `# Name of the student (string)`).
- Clarify the initial value of `fee_status` (e.g., `# Initialize fee status to 0 (representing no fees paid)`).
- Make the comment for `display_details` more specific about what information is being printed.
- Consider adding a class-level docstring to explain the overall purpose of the `sru_student` class.

In summary, the AI-generated comments are more detailed and provide slightly better clarity on the purpose of different parts of the code. The manual comments are concise but could be improved by adding more context and detail, particularly regarding data types and the initial state of attributes.

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

Manual:-

Basic Calculator code:-

[12]
✓ 0s

```
# Calculator functions
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

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

Manual Numpy-style Function Docstrings:-

[16]
✓ 0s

```
"""
Calculator Module
This module provides basic arithmetic operations: add, subtract, multiply, and divide.
"""
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
```

```
[16] ✓ 0s ▶ 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
        Division of a by b

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

Ai Generated code:-

[17]
✓ 0s

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

This module provides basic arithmetic operations.

Functions
-----
add(a, b)
    Add two numbers.
subtract(a, b)
    Subtract two numbers.
multiply(a, b)
    Multiply two numbers.
divide(a, b)
    Divide two numbers.
"""
def add(a, b):
    """Add two numbers.

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

    Returns
    -----
    int or float
    """
```

[17]
✓ 0s

```
    b : int or float
        The second number.

    Returns
    -----
    int or float
        The sum of `a` and `b`.
    """
    return a + b

def subtract(a, b):
    """Subtract two numbers.

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

    Returns
    -----
    int or float
        The difference of `a` and `b`.
    """
    return a - b

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

    Returns
    -----
    int or float
        The product of `a` and `b`.
    """
    return a * b
```

```
[17]
✓ 0s def multiply(a, b):
    """Multiply two numbers.

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

    Returns
    -----
    int or float
        The product of `a` and `b`.
    """
    return a * b

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

    Parameters
    -----
    a : int or float
        The numerator.
    b : int or float
        The denominator.

    Returns
    -----
    float
        The result of dividing `a` by `b`.
```

```

    Returns
    -----
    float
        The result of dividing `a` by `b`.

    Raises
    -----
    ValueError
        If the denominator `b` is zero.
    """
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
```

Output:- Compare the manual NumPy-style docstrings with the AI-generated docstrings for this calculator module.

Highlight:

- Missing details
- Differences in clarity
- Suggestions to improve the manual docstrings

I can now compare it to the AI-generated NumPy-style docstrings in cell qgdQ0k1a8oDZ.

Here's a comparison highlighting the differences, missing details, and suggestions for improvement:

Manual Code (Provided in prompt):

```
# Calculator functions
```

```
def add(a, b):  
    return a + b
```

```
def subtract(a, b):  
    return a - b
```

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

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

Calculator Module

This module provides basic arithmetic operations: add, subtract, multiply, and divide.

```
"""
```

```
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

Division of a by b

Raises

ValueError

If b is zero

"""

if b == 0:

raise ValueError("Cannot divide by zero")

return a / b

AI-Generated NumPy-style Docstrings (Cell qgdQ0k1a8oDZ):

```
"""
```

Calculator Module

This module provides basic arithmetic operations: add, subtract, multiply, and divide.

```
"""
```

```
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
    Division of a by b
```

Raises

ValueError

 If b is zero

"""

if b == 0:

 raise ValueError("Cannot divide by zero")

return a / b

Comparison:

- **Missing Details:**

- The provided manual code has the function definitions repeated, first without docstrings and then with docstrings. This is redundant and would cause errors if run in a single block.
- While the docstrings in the second set of functions in the manual code are close to NumPy style, they are missing the structured format for the module-level docstring (e.g., the ===== under the module name and the Functions section with brief descriptions).
- Neither the manual nor the AI-generated docstrings explicitly mention the data types of the parameters within the brief summary line of the function docstring, although they are listed under the Parameters section.

- **Differences in Clarity:**

- The AI-generated docstrings in cell qgdQ0k1a8oDZ follow the standard NumPy-style format more closely, especially the module-level docstring structure. This consistency contributes to better readability and clarity for users familiar with this style.

- The function-level docstrings in both appear quite similar in clarity regarding parameters and returns.
- **Suggestions to Improve the Manual Docstrings:**
 - Remove the initial set of function definitions without docstrings to avoid redundancy and potential errors.
 - Adopt the standard NumPy-style format for the module-level docstring, including the underlines and the Functions section.
 - Ensure consistent spacing and indentation throughout the docstrings to adhere to the style guide.
 - (Optional but recommended) Include a brief mention of the parameter types in the first line of the function docstring for quick understanding, e.g., Add two numbers (int or float).

In summary, the AI-generated docstrings in cell qgdQ0k1a8oDZ adhere more closely to the standard NumPy style guide, particularly for the module-level docstring structure. The manual code provided in the prompt has redundancy and slightly deviates from the standard format. Adopting the full NumPy-style structure and removing the duplicate function definitions would significantly improve the manual code's clarity and correctness.

THANK YOU

