SR UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Course Title – AI Assisted Coding

Course Code : 24CS002PC215

Assignment Type : Lab

Academic Year : 2025–2026

Name : Soumith Sripathi

Enroll Number : 2403A51340

Batch Number : 24BTCAICSB14

Week – 9 Date : 25 AUGUST

Task Description #1 (Documentation – Google–Style Docstrings for Python Functions)
Task: Use AI to add Google–style docstrings to all functions in a given Python script.

Instructions:
Prompt AI to generate docstrings without providing any input–output examples.
Ensure each docstring includes:
Function description
Parameters with type hints
Return values with type hints
Example usage
Review the generated docstrings for accuracy and formatting.

CODE&OUTPUT:

```
def multiply_numbers(a: float, b: float) -> float:
  """Multiplies two numbers and returns the product.

  Args:
    a: The first number.
    b: The second number.

  Returns:
    The product of the two numbers.

  Example:
    >>> multiply_numbers(2.0, 3.0)
    6.0
  """
  return a * b

def greet(name: str) -> str:
  """Generates a greeting message.
```

```
    Args:
      name: The name to include in the greeting.

    Returns:
      A greeting string.

    Example:
      >>> greet("Alice")
      'Hello, Alice!'
    """
    return f"Hello, {name}!"

def calculate_average(numbers: list[float]) -> float:
    """Calculates the average of a list of numbers.

    Args:
      numbers: A list of numbers.

    Returns:
      The average of the numbers.
```

```
    Example:
      >>> calculate_average([1.0, 2.0, 3.0, 4.0])
      2.5
    """
    if not numbers:
      return 0.0
    return sum(numbers) / len(numbers)

# Example usage:
product_result = multiply_numbers(5.0, 7.0)
print(f"Product: {product_result}")

greeting_message = greet("Colab User")
print(greeting_message)

average_result = calculate_average([10.0, 20.0, 30.0, 40.0, 50.0])
print(f"Average: {average_result}")
```

```
Product: 35.0
Hello, Colab User!
Average: 30.0
```

EXPLANATION:

Defines a function to multiply two numbers.
Describes the purpose: multiplies two numbers and returns the product.

Starts the arguments section.
Explains the first parameter: the first number.
Explains the second parameter: the second number.
Starts the returns section.
Mentions the function returns the product of the two numbers.
Gives an example usage.
Shows how to call the function with two float values.
Shows the expected result of the multiplication.
Ends the docstring.
Performs the actual multiplication and returns the result.
Defines a function to generate a greeting message.
Describes the purpose: generates a greeting.
Starts the arguments section.
Explains the parameter: name to include in the greeting.
Starts the returns section.
Mentions that it returns a greeting string.
Gives an example usage.
Shows how to call the function with a name.
Shows the expected greeting output.
Ends the docstring.
Returns the greeting message using the name

provided.
Defines a function to calculate the average of a list of numbers.
Describes the purpose: calculates the average.
Starts the arguments section.
Explains the parameter: a list of numbers.
Starts the returns section.
Mentions it returns the average of the numbers.no
Gives an example usage
Shows how to call the function with a list of numbers.
Shows the expected average result.
Ends the docstring.
Checks if the list is empty.
Returns 0.0 if the list is empty.
Returns the average (sum divided by count).

Example Usage Section:
Comments that the following lines demonstrate example usage.
Calls the multiplication function with example values.
Prints the product result.
Calls the greeting function with a name.
Prints the greeting message.

Calls the average function with a list of numbers.
Prints the average result.
Output (from example usage):
Displays the product: 35.0
Displays the greeting: "Hello, Colab User!"
Displays the average: 30.0

Task Description 2 :
(Documentation – Inline Comments for Complex Logic)
• Task: Use AI to add meaningful inline comments to a Python
program explaining only complex logic parts.

• Instructions:
o Provide a Python script without comments to the AI.
Instruct AI to skip obvious syntax explanations and focus only on tricky or non–intuitive code sections.
Verify that comments improve code readability and maintainability.
• **Expected Output :**
o Python code with concise, context–aware inline comments
for complex logic blocks

PROGRAM AND OUTPUT :

```
def process_data(data):
  # This function processes the input data
  processed_data = []
  for item in data:
    if item % 2 == 0:
      processed_data.append(item * 2)
    else:
      processed_data.append(item + 1)
  return processed_data

def analyze_results(results):
  # This function analyzes the processed data
  total = sum(results)
  average = total / len(results) if results else 0
  return total, average

# Sample data
input_data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
# Sample data
input_data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Process and analyze the data
processed_list = process_data(input_data)
total_result, average_result = analyze_results(processed_list)

# Print the results
print("Processed List:", processed_list)
print("Total Result:", total_result)
print("Average Result:", average_result)
```

```
Processed List: [2, 4, 4, 8, 6, 12, 8, 16, 10, 20]
Total Result: 90
Average Result: 9.0
```

EXPLANATION:

Defines a function that will process input data.

Comment: Explains that this function processes the input data.

Creates an empty list to store the processed data.

Starts a loop to go through each item in the input data.
Checks if the current item is even (divisible by 2).
If even, doubles the item and adds it to the processed list.
If not even (i.e., odd), adds 1 to the item and appends it.
Ends the loop. Returns the processed data list.
Defines a function to analyze the processed results.
Comment: Explains this function analyzes the processed data.
Calculates the total sum of the processed results.
Calculates the average of the results (avoids divide–by–zero error).
Returns both the total and the average values.
Comment: Indicates the start of sample input data.
Creates a list of numbers from 1 to 10.
Comment: Indicates the data will now be processed and analyzed.
Calls the data processing function with the input data.
Calls the analysis function on the processed data to get total and average.
Comment: Indicates the results will be printed.
Prints the processed data list.

Prints the total of the processed results.
Prints the average of the processed results.

Task Description –3:

_Documentation – Module–Level Documentation)

• Task: Use AI to create a module–level docstring summarizing thepurpose, dependencies, and main functions/classes of a Python file.

• **Instructions:**

o Supply the entire Python file to AI.

o Instruct AI to write a single multi–line docstring at the topof the file.

o Ensure the docstring clearly describes functionality and
usage without rewriting the entire code.

Expected Output :

o A complete, clear, and concise module–level docstring at the beginning of the file

PROGRAM AND OUTPUT :

📌 Example (before prompt)

python

```python
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

```python
"""This module provides a function to calculate the factorial of a non-negative integer.

It includes a recursive implementation of the factorial function.

Dependencies:
- None (uses built-in Python features)

Functions:
- factorial(n): Calculates the factorial of a non-negative integer n.
"""

def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

EXPLANATION:

1. A function factorial(n) is defined.
2. It takes one input number n.
3. If n is 0 or 1 → factorial is 1.
4. Otherwise, it goes to the else part.
5. There it returns n * factorial(n−1).

6. This means the function calls itself again (recursion).
7. Each call reduces n by 1 until it reaches 1.
8. Then the recursion stops and starts returning results.
9. The results get multiplied step by step back up.
10. Final output is the factorial of n.

Task Description –4 :

(Documentation – Convert Comments to Structured Docstrings)

Task: Use AI to transform existing inline comments into structured function docstrings following Google style.

Instructions:

Provide AI with Python code containing inline comments.

Ask AI to move relevant details from comments into function docstrings.

Verify that the new docstrings keep the meaning intact while improving structure.

PROGRAM AND OUTPUT :

```python
# Code with inline comments converted to structured docstrings (Google style)

def calculate_area(radius):
  """Calculates the area of a circle.

  Args:
    radius: The radius of the circle (float or int).

  Returns:
    The area of the circle (float).
  """
  area = 3.14159 * radius * radius
  return area

def greet_user(name):
  """Greets the user by name.

  Args:
    name: The name of the user (str).
  """
  greeting = f"Hello, {name}!"
  print(greeting)
```

```python
# Example of calling the functions defined above

# Call the calculate_area function and print the result
circle_area = calculate_area(5)
print(f"The area of a circle with radius 5 is: {circle_area}")

# Call the greet_user function
greet_user("Alice")
```

```
The area of a circle with radius 5 is: 78.53975
Hello, Alice!
```

EXPLANATION:

A function calculate_area(radius) is defined.
It uses the formula π × radius² to find the area of a circle.
π is approximated as 3.14159.
The function returns the area as a float value.

Another function greet_user(name) is defined.
It takes a person's name as input.
It creates a message: "Hello, {name}!".
Then it prints the message on the screen.
Example: calculate_area(5) gives 78.53975.
Example: greet_user("Alice") prints Hello, Alice!.

Task Description –5 :
(Documentation – Review and Correct Docstrings)
Task: Use AI to identify and correct inaccuracies in existing docstrings.

Instructions:
Provide Python code with outdated or incorrect docstrings.
Instruct AI to rewrite each docstring to match the current code behavior.

CODE&OUTPUT:

1. add_numbers(a, b) adds two numbers.
2. It takes inputs a and b.
3. Returns the result of a + b.
4. find_maximum(values) finds the largest number in a list.
5. Input is a list of numbers called values.

6.Uses Python’s built–in max() function.
7.Returns the biggest number from the list.
8.format_name(first, last) combines two names.
9.Input is first and last (strings).
10.Returns the full name in the form "First Last".

Task Description 6 :

(Documentation – Prompt Comparison Experiment)
Task: Compare documentation output from a vague prompt and a detailed prompt for the same Python function.

Instructions:

Create two prompts: one simple (“Add comments to this function”) and one detailed (“Add Google–style docstrings with parameters, return types, and examples”).
Use AI to process the same Python function with both prompts.
Analyze and record differences in quality, accuracy, and completeness.

CODE&OUTPUT:

Apply vague prompt

Subtask:

Apply the vague prompt "Add comments to this function" to a sample function and observe the output.

↳ 2 cells hidden

Apply detailed prompt

Subtask:

Apply the detailed prompt "Add Google-style docstrings with parameters, return types, and examples to this function" to the same sample function and observe the output.

Reasoning: Define and print the detailed prompt string as requested by the instructions.

```
[13] detailed_prompt = "Add Google-style docstrings with parameters, return types, and examples to this function"
     print("Detailed Prompt:")
     print(detailed_prompt)
```

```
Detailed Prompt:
Add Google-style docstrings with parameters, return types, and examples to this function
```

Create comparison table

Subtask:

Create a markdown table to compare the outputs based on style, quality, accuracy, and completeness.

Reasoning: Create a markdown string representing the comparison table with placeholder text.

```
[14] markdown_table = """
     | Feature      | Vague Prompt Output                  | Detailed Prompt Output                      |
     |--------------|--------------------------------------|---------------------------------------------|
     | Style        | Analyze language model output for style | Analyze language model output for style     |
     | Quality      | Analyze language model output for quality | Analyze language model output for quality   |
     | Accuracy     | Analyze language model output for accuracy | Analyze language model output for accuracy |
     | Completeness | Analyze language model output for completeness | Analyze language model output for completeness |
     """
     print(markdown_table)
```

Create comparison table

Subtask:

Create a markdown table to compare the outputs based on style, quality, accuracy, and completeness.

↳ 2 cells hidden

Summary:

Data Analysis Key Findings

- A sample Python function `add_numbers` with an existing docstring was defined for testing.
- A "vague prompt" ("Add comments to this function") and a "detailed prompt" ("Add Google-style docstrings with parameters, return types, and examples to this function") were prepared.
- A markdown table was created with columns for "Feature", "Vague Prompt Output", and "Detailed Prompt Output" and rows for "Style", "Quality", "Accuracy", and "Completeness" to facilitate the comparison of the language model outputs.

Insights or Next Steps

- The next step is to actually apply both the vague and detailed prompts to the sample function using a language model and populate the comparison table with the resulting outputs and analysis.
- Analyzing the populated table will provide insights into how prompt specificity affects the style, quality, accuracy, and completeness of generated documentation.

EXPLAINATION:

add_numbers(a, b) is a function.
It takes two inputs, a and b.
It returns their sum.
find_maximum(values) is another function.
It takes a list of numbers as input.
It uses max() to find the largest number.
Then it returns that largest value.
format_name(first, last) is the third function.
It takes first name and last name as input.
It returns them joined as "First Last"