**HTNO:2403A51344**

**Name: Bollam Sathvika**

**Batch no:24BTCAICSB14**

**Lab 9 – Documentation Generation: Automatic Documentation and  
Code Comments**

**Task 1:**

**Prompt:** Add Google-style docstrings to all functions in this Python script. Each docstring should include a clear function description, parameter names with type hints, return values with type hints, and an example usage. Do not include input-output examples. Ensure the formatting follows Google’s Python style guide.

**CODE:**

****

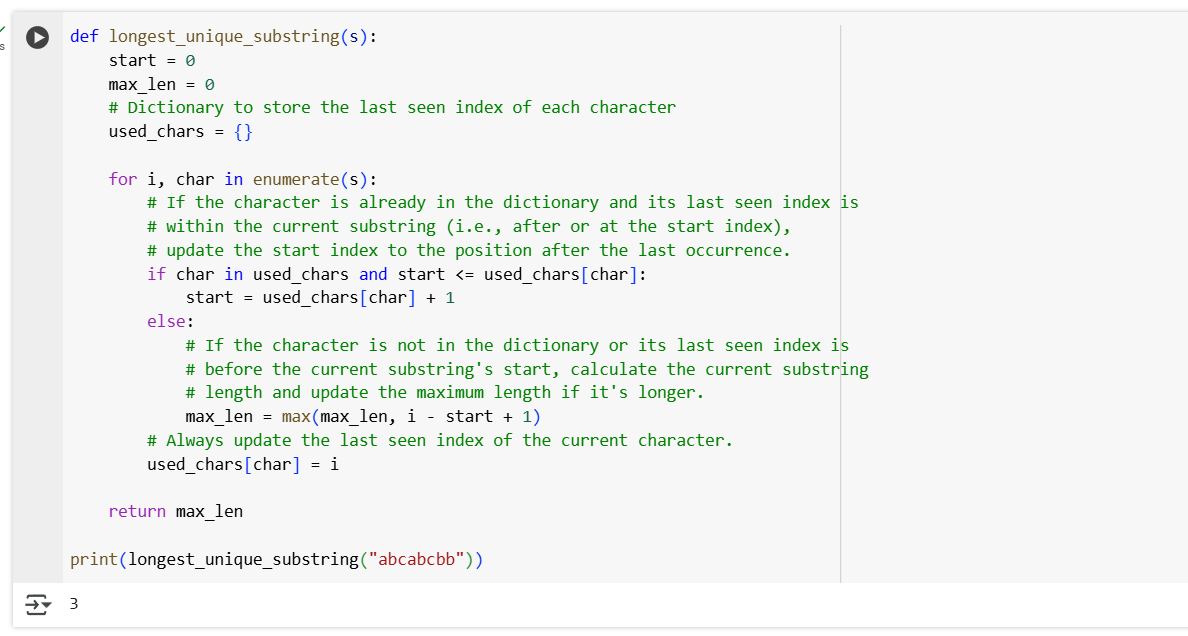
**Observation:**

* **add\_numbers(a: int, b: int) -> int**: This function takes two integer arguments, a and b, and returns their sum as an integer. The docstring explains its purpose and the arguments and return value with their types.
* **greet(name: str) -> str**: This function takes a string argument name and returns a greeting message including the provided name. The docstring describes its function and the types of its argument and return value.
* **calculate\_area(length: float, width: float) -> float**: This function calculates the area of a rectangle. It takes two float arguments, length and width, and returns their product as a float. The docstring clarifies its purpose and the types of its arguments and return value.

Each function has a Google-style docstring that explains what the function does, the expected types of its arguments (Args:), and the type of the value it returns (Returns:).

**Task2:**

**Prompt:** I will provide you with a Python script that has no comments. Your task is to add meaningful inline comments ONLY for complex or non-intuitive parts of the code. Guidelines: - Do NOT explain obvious syntax like loops, variable assignments, or function definitions. - Focus only on tricky logic, edge-case handling, or algorithmic reasoning. - Keep comments concise, context-aware, and helpful for someone maintaining the code. - Do NOT rewrite the code, just add inline comments.

**Code:  
**

**Observation:**

The code in cell  implements an efficient algorithm to find the length of the longest substring without repeating characters. It uses a sliding window approach with a dictionary to keep track of the last seen index of each character, allowing it to quickly adjust the window's start when a repeating character is encountered. This avoids the need to check every possible substring, making it quite performant.

**Task 3:**

**Prompt:** I will provide you with a Python file.

Your task is to generate a \*\*module-level docstring\*\* at the very top of the file.

Requirements:

- Summarize the purpose of the module.

- Mention any key dependencies or libraries used.

- List and briefly describe the main functions and/or classes.

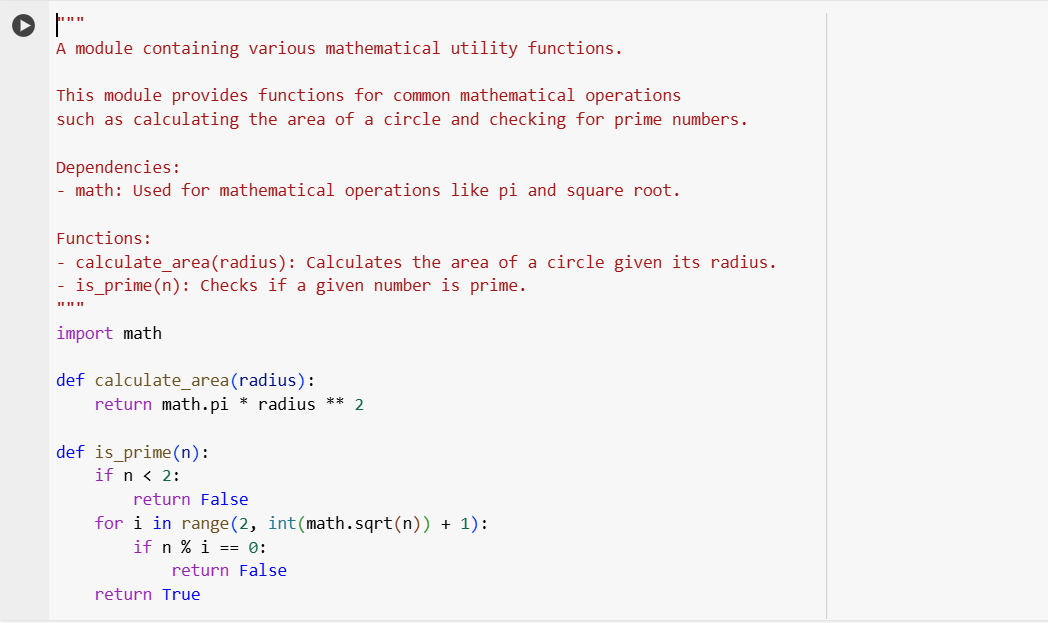
- Keep it concise and professional (3–8 lines).

- Do NOT rewrite the code or add inline comments, just produce the docstring.

Here is the Python file:

https://colab.research.google.com/drive/12fSYTxRN41QHCiEx2BmnShwirpwbQ4o2?usp=drive\_link

**Code:**

****

**Observation:**

The visible cells contain a Python module named math\_utils with two functions:

* calculate\_area(radius): Calculates the area of a circle given its radius.
* is\_prime(n): Checks if a given number is a prime number.

The module includes a docstring explaining its purpose, dependencies, and the functions it provides.

**Task 4:**

**Prompt:** I will provide you with Python code that contains inline comments.

Your task is to convert those inline comments into structured \*\*Google-style function docstrings\*\*.

Requirements:

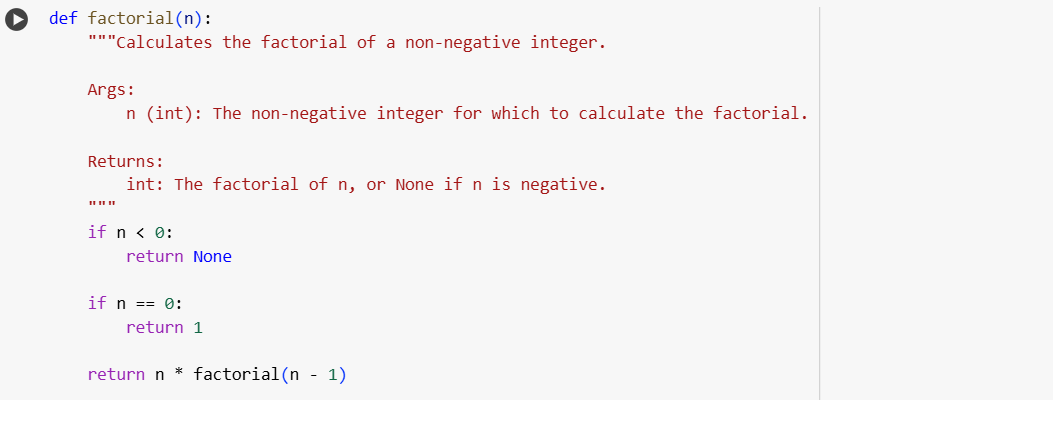
- Use Google-style docstrings (with Args, Returns, Raises if applicable).

- Move relevant details from inline comments into the function docstring.

- Remove redundant inline comments after creating docstrings.

- Do NOT alter the actual code logic.

**CODE:**

****

**Observation:**

The visible cells contain two Python code snippets.

* The first snippet defines a module with functions for calculating the area of a circle (calculate\_area) and checking for prime numbers (is\_prime).
* The second snippet defines a recursive function to calculate the factorial of a non-negative integer (factorial).

Both functions now have detailed Google-style docstrings explaining their purpose, arguments, and return values.

**TASK 5:**

**Prompt:**

I will provide you with Python code that contains outdated or incorrect docstrings.

Your task is to review and rewrite these docstrings so they accurately reflect the current behavior of the code.

Requirements:

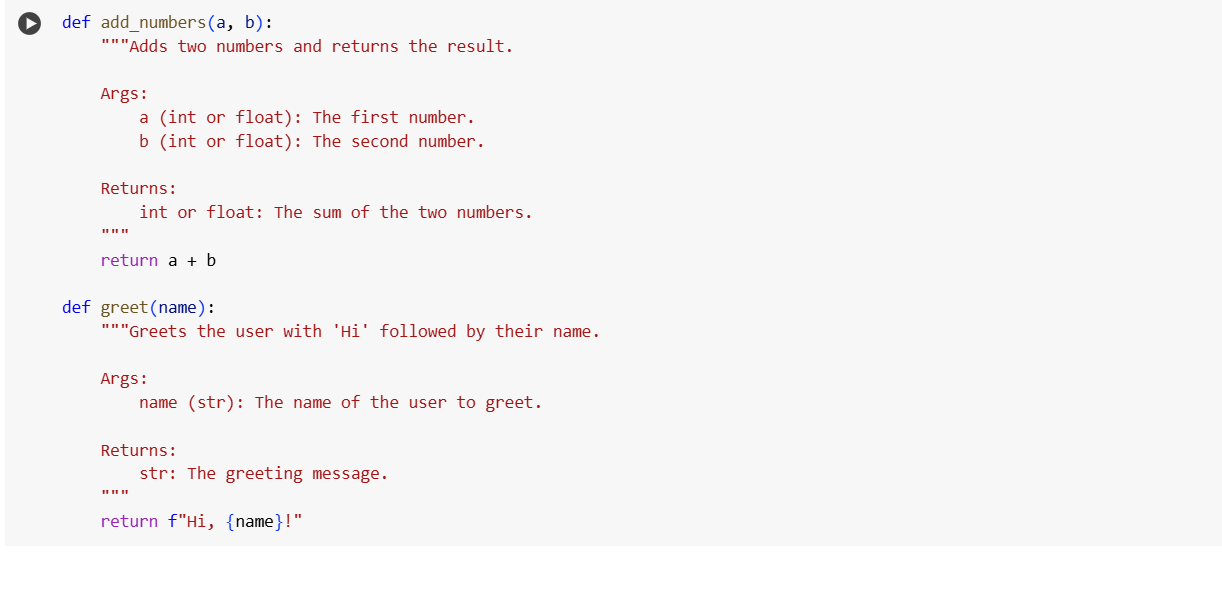
- Correct any inaccuracies in the docstrings.

- Use Google-style formatting (Args, Returns, Raises).

- Ensure the updated docstrings are concise, clear, and match what the function actually does.

- Do not modify the code logic, only fix the docstrings.

**Code:**

****

**Observation:**

The Python code contain with functions that now have corrected and well-formatted Google-style docstrings.

* One cell contains the factorial function, which calculates the factorial of a non-negative integer.
* Another cell contains the add\_numbers function, which adds two numbers, and the greet function, which returns a greeting message.

All the docstrings accurately describe the function's purpose, arguments, and return values.

**Task 6:**

**Prompt1:** Add comments to this function.

**Prompt2:** Add a Google-style docstring to this function.

Include:

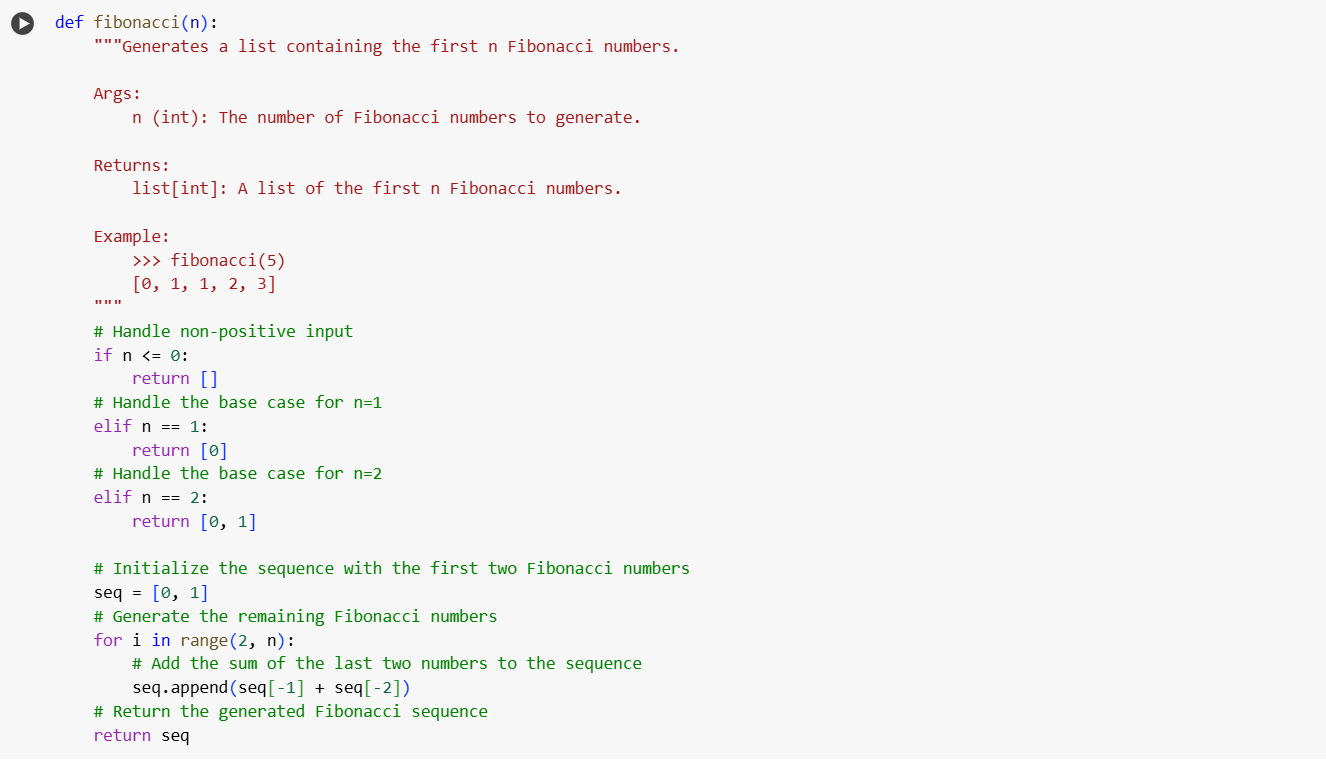
- A short description of what the function does.

- Parameter descriptions with types.

- Return value description with type.

- An example usage.

**Code:**

****

**Observation:**

The visible cell contains the fibonacci function. This function now has a complete Google-style docstring that explains what it does, its parameters, what it returns, and even includes an example of how to use it.