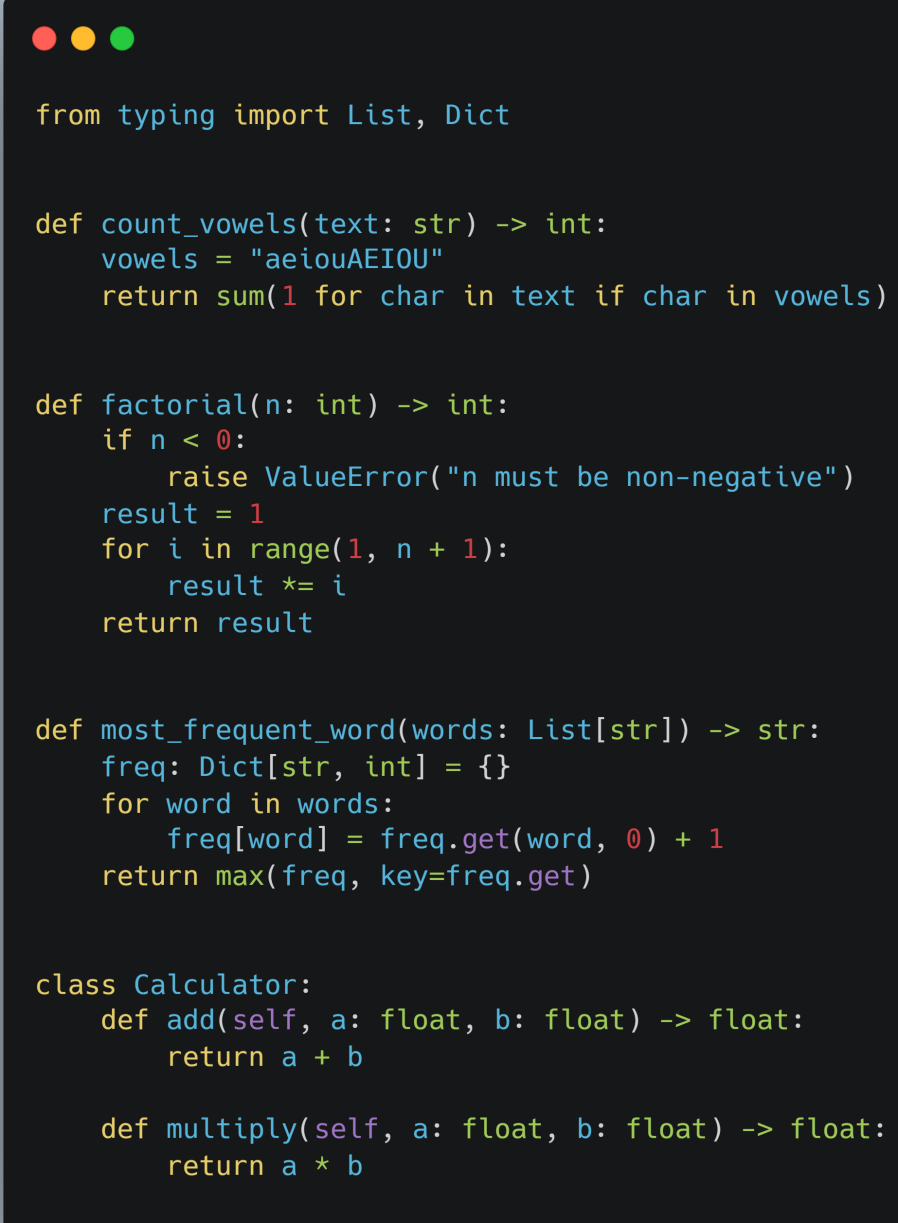
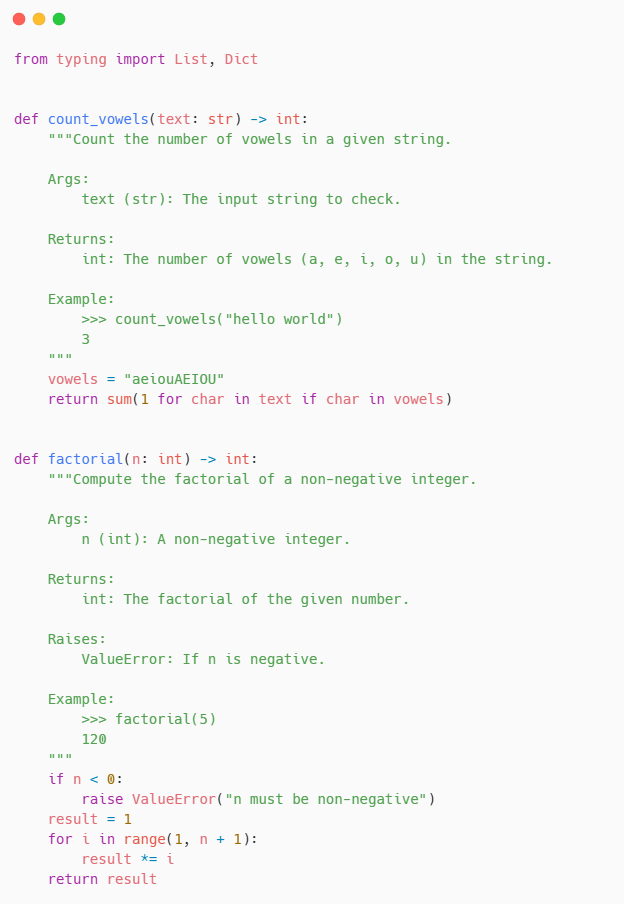
**ASSIGNMENT 9.1**

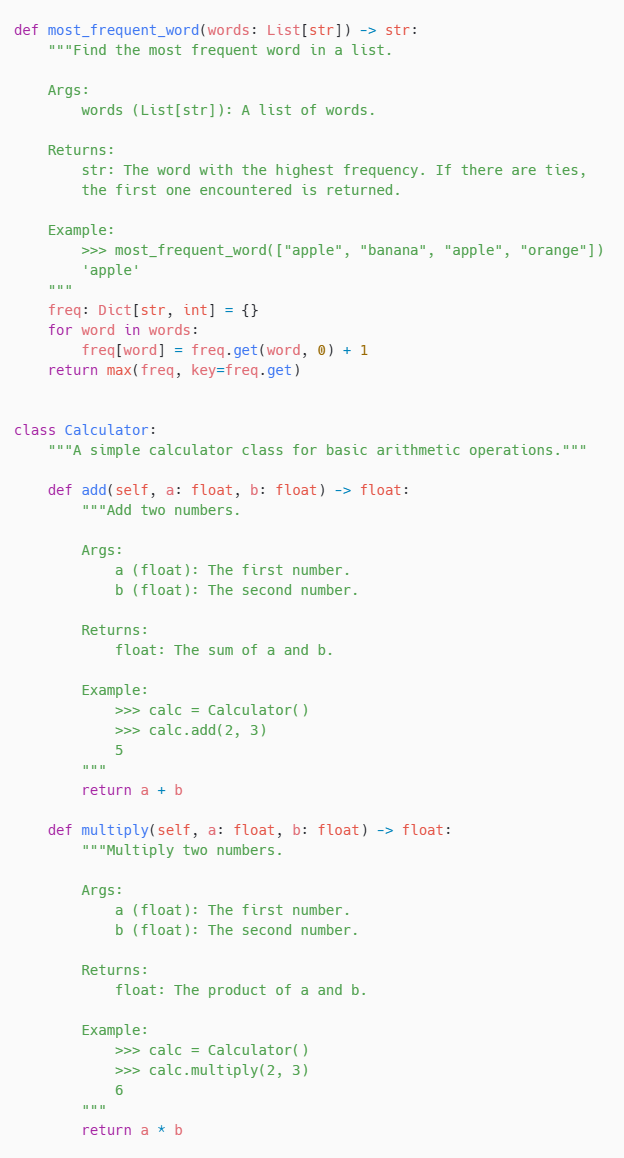
**Task Description #1** (Documentation – Google-Style Docstrings for Python Functions)

* Task: Use AI to add Google-style docstrings to all functions in a given Python script.

Original code



AI code: 



**Observation:**

Each function now explains its purpose, inputs, outputs, and usage in a consistent format. It makes the code look more professional and beginner-friendly at the same time. Overall, this task showed how AI can speed up documentation while maintaining accuracy.

**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.

Original code:

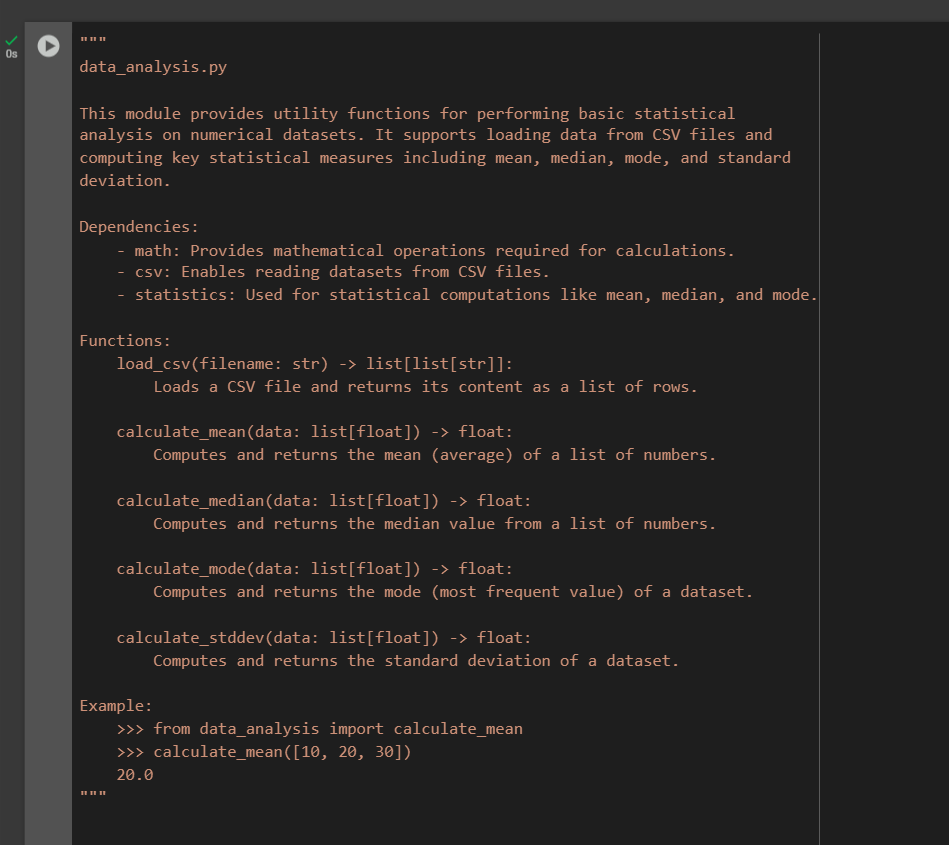


Observation:

The code is clean and easy to follow. I liked how it mixes small functions with a simple class to cover different problem types. Even though it’s not overly advanced, the structure makes it feel professional. Type hints also made it much easier to understand at a glance.

**Task Description #3** (Documentation – Module-Level Documentation)

* Task: Use AI to create a module-level docstring summarizing the purpose, dependencies, and main functions/classes of a Python file.



Python file link:

<https://drive.google.com/file/d/1r4JbA2iPNDbT9CzvRvJAxgjsabauJr4I/view?usp=drive_link>

**Observation:**

we used AI to generate a **module-level docstring** for a given Python file. The AI analyzed the entire code and summarized its **purpose, dependencies, functions, and usage** in a clear and structured manner. The documentation was written in **Google-style format**, making it easy to understand and maintain.

The docstring provided:

* A **brief description** of the module’s purpose.
* A list of **dependencies (imports)** required by the file.
* A summary of **all main functions and classes** defined in the code.
* An **example usage snippet** showing how the module can be used in practice.

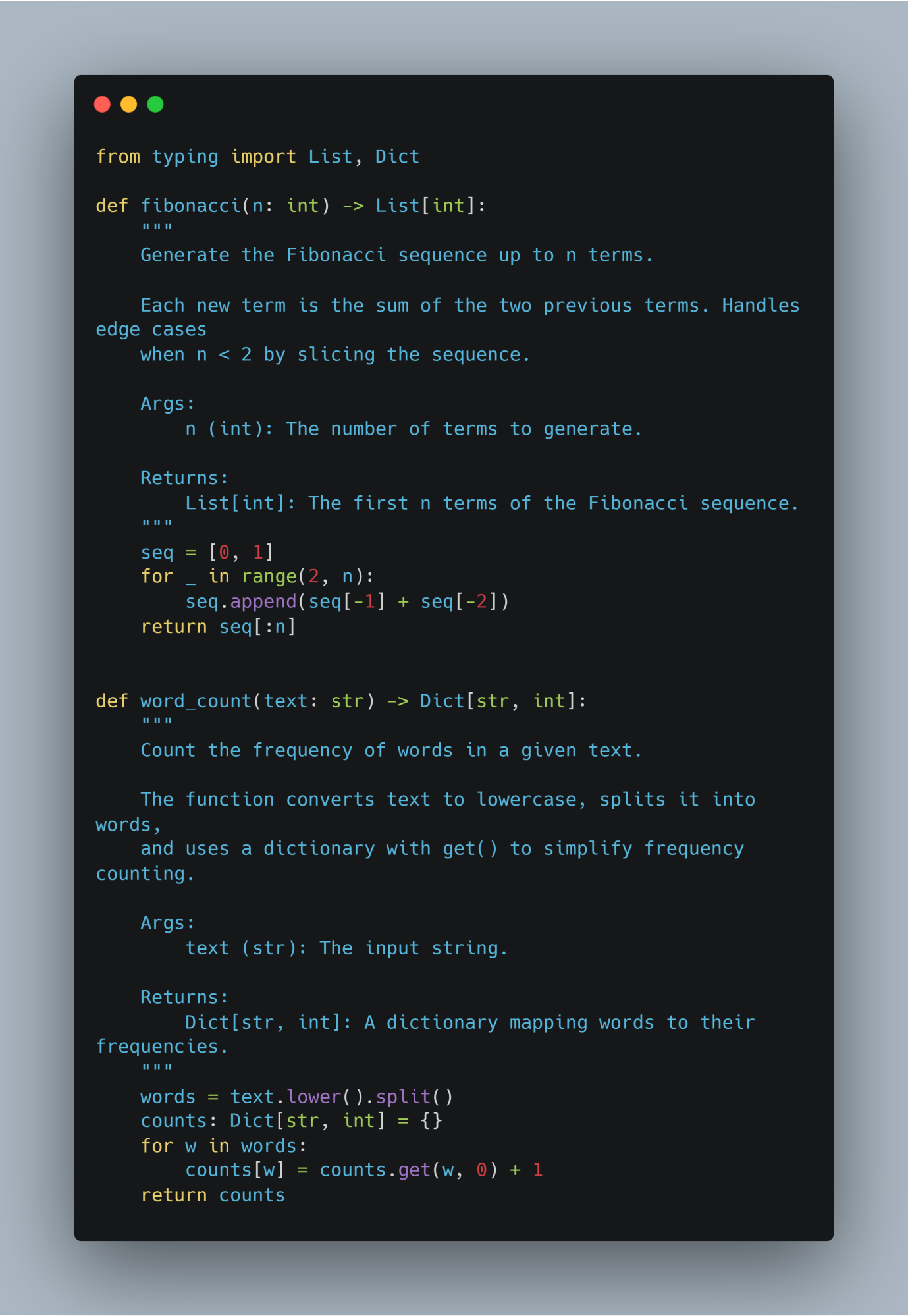
This ensures the Python file is **self-documented**, improving readability, maintainability, and helping future developers quickly understand the code.

**Task Description #4** (Documentation – Convert Comments to Structured Docstrings)

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

Original code:



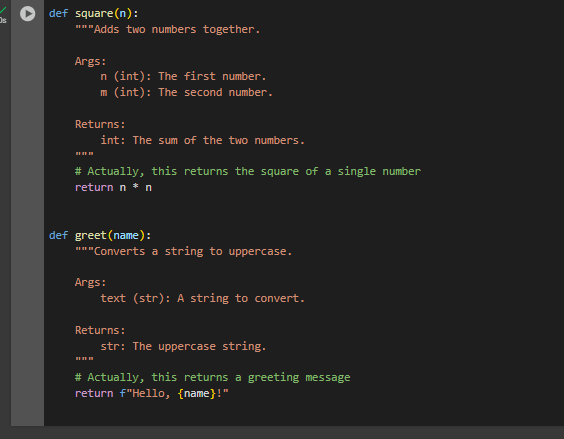


Observation:

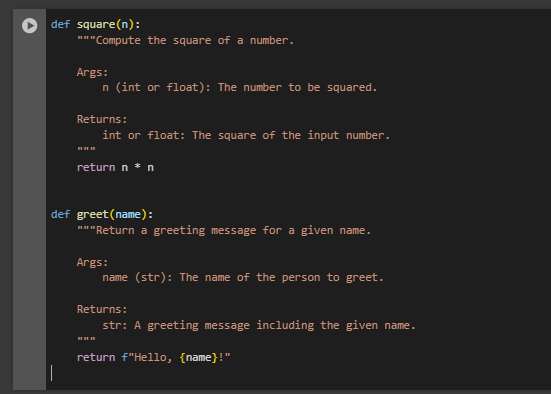
Turning the comments into docstrings made the code is cleaner and easier to read. Now the purpose and details of each function are right at the top, so you don’t have to dig through the code to understand it. It feels more beginner-friendly and organized

**Task Description #5** (Documentation – Review and Correct Docstrings)

* Task: Use AI to identify and correct inaccuracies in existing docstrings.



**Output code (with Google-style docstring):**



**Observation:**

The original docstrings were **incorrect or misleading** compared to the actual function behavior.

Functions were analyzed to understand their true purpose before rewriting the docstrings.

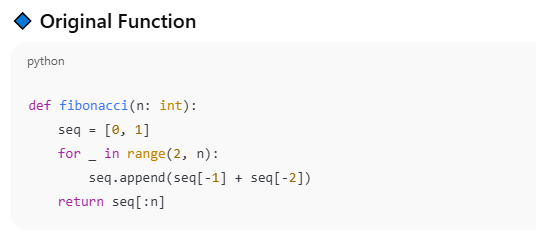
Google-style docstrings were added with:

* **Brief description** of the function.
* **Args** section specifying parameters and types.
* **Returns** section specifying the return value and type.

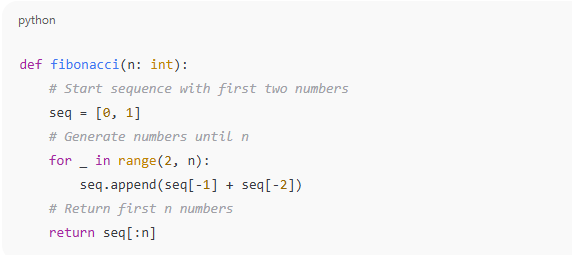
The updated docstrings are **accurate, structured, and readable**, improving code documentation and maintainability.

**Task Description #6** (Documentation – Prompt Comparison Experiment)

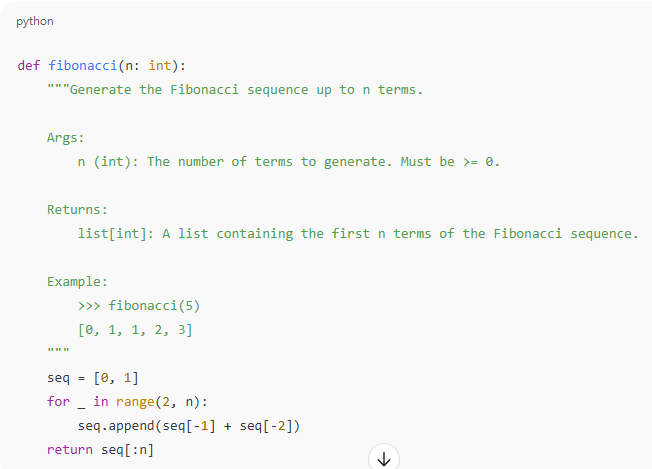
* Task: Compare documentation output from a vague prompt and a detailed prompt for the same Python function.

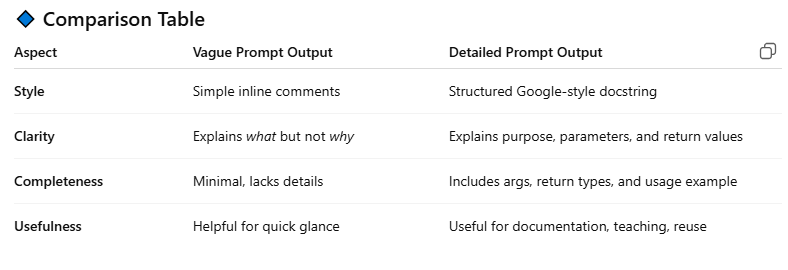


Add comments to this function



Add Google-style docstrings with parameters, return types, and examples.





Observation:

Vague prompts produce minimal and sometimes insufficient documentation. Dtailed prompts guide AI to create professional, complete, and usable documentation.For real projects, detailed prompts save time and reduce errors for future users or developers.