## AI ASSISTED CODING

## ASSIGNMENT 9.2

2403A51260

SWAPNA MADISHETTI

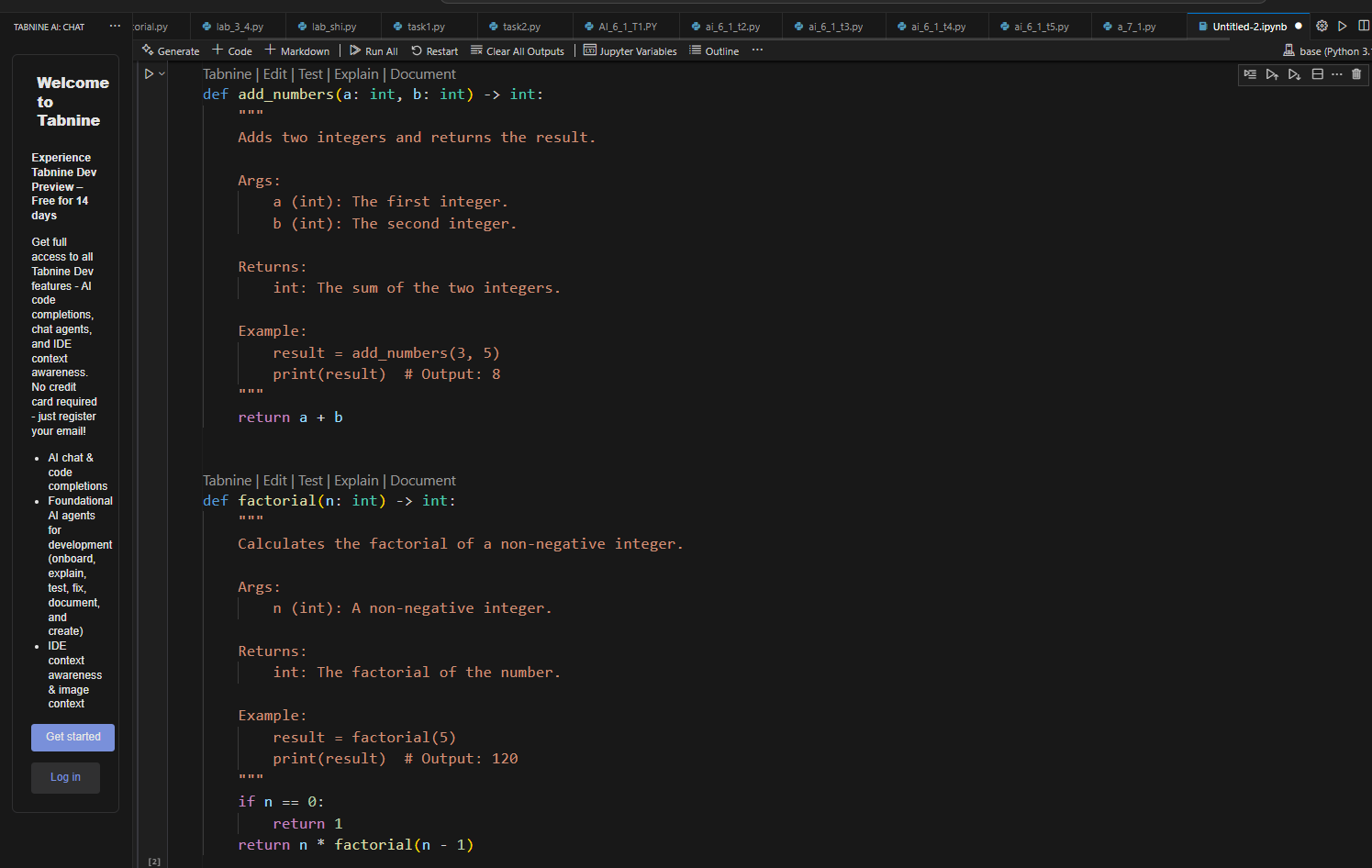
BATCH-11

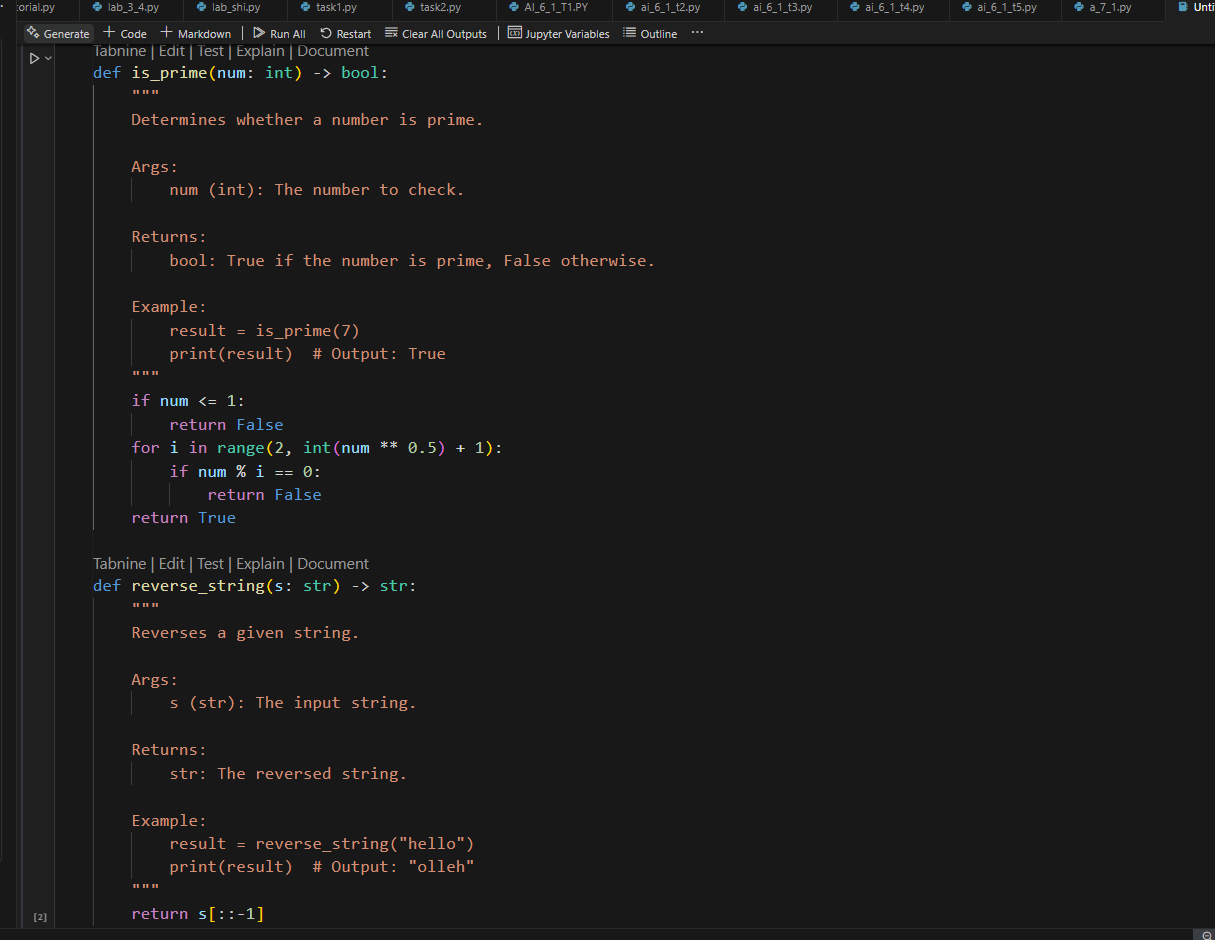
**TASK-1**:(Documentation – Google-Style Docstrings for  
Python Functions)

Use AI to add Google-style docstrings to all functions in a  
given Python script.  
• Instructions:  
o Prompt AI to generate docstrings without providing any  
input-output examples.  
o Ensure each docstring includes:  
▪ Function description  
▪ Parameters with type hints  
▪ Return values with type hints  
▪ Example usage  
o Review the generated docstrings for accuracy and  
formatting.

**Prompt**: Give me a python script with all functions documented using  
correctly formatted Google-style docstrings

**Response:**





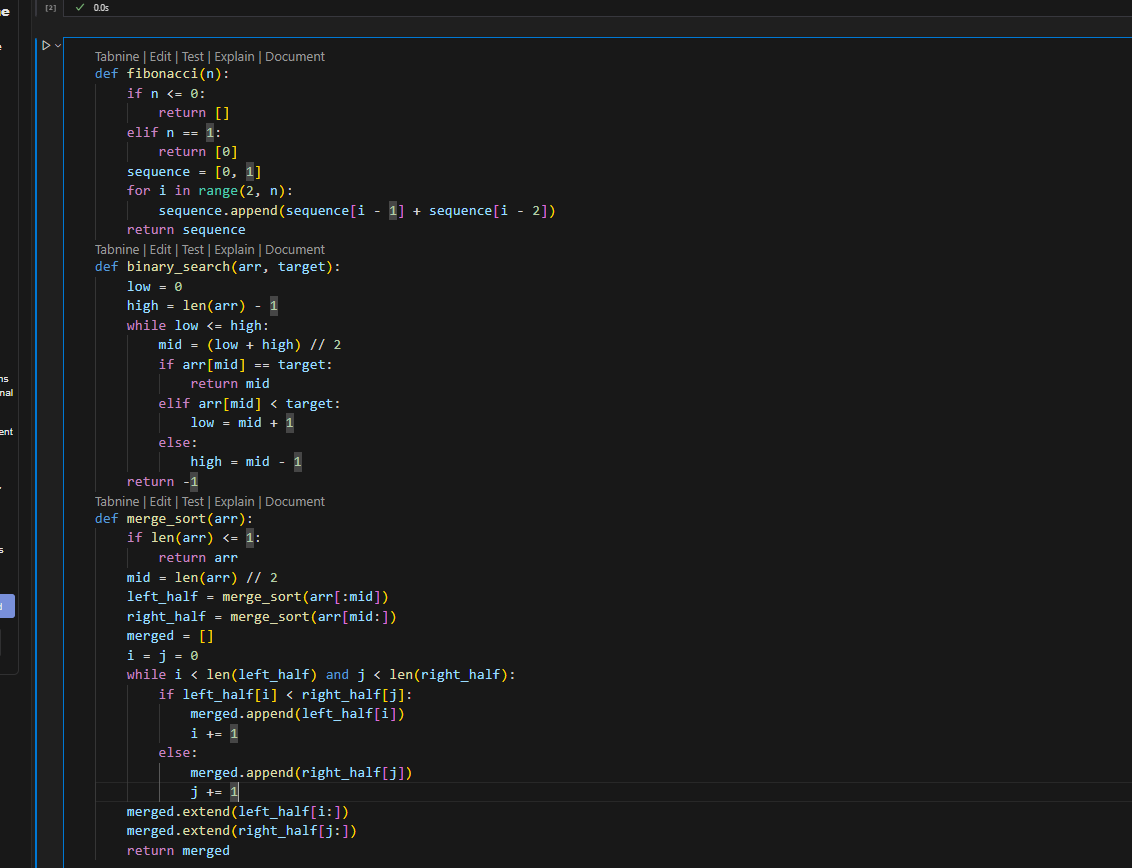
This script includes:

* A function to add two numbers.
* A function to compute factorial.
* A function to check if a number is prime.
* A function to reverse a string.

All functions have:

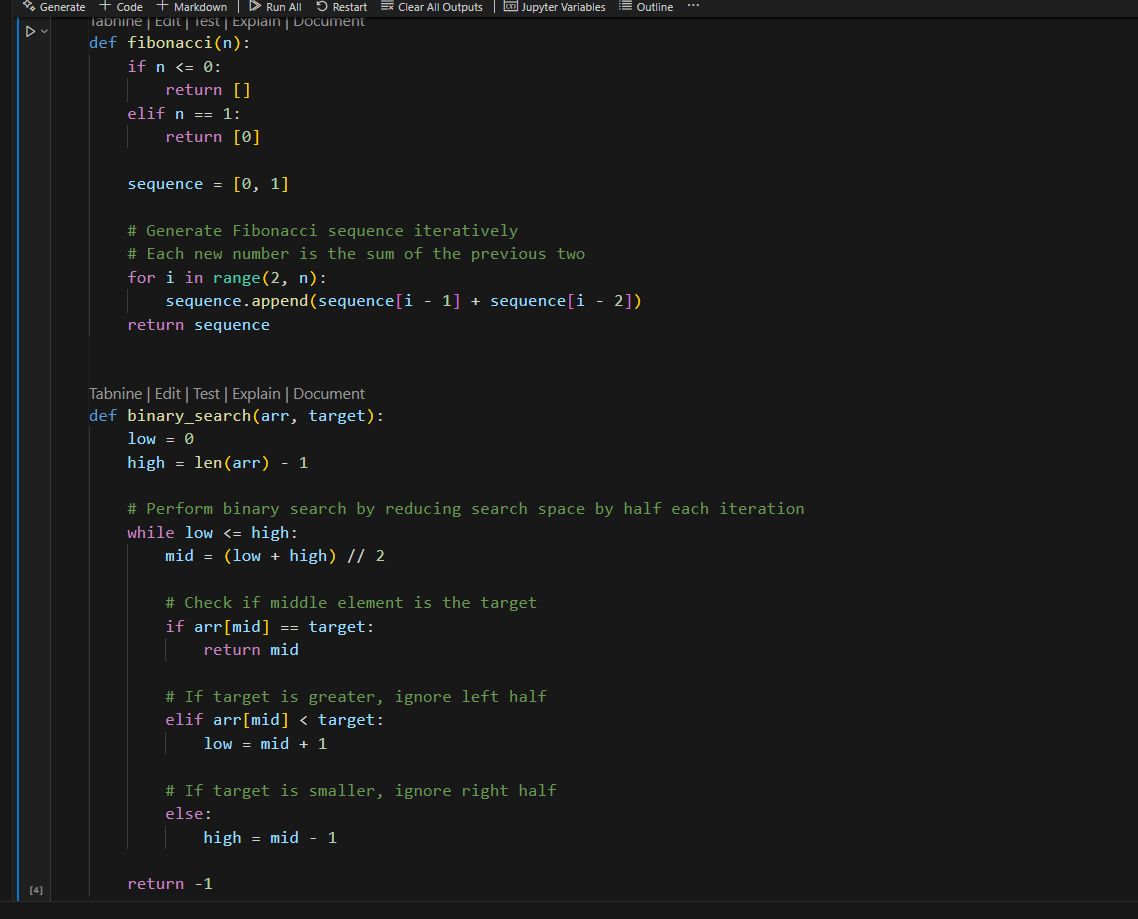
* Clear description
* Type hints in Args and Returns
* Example usage in docstrings

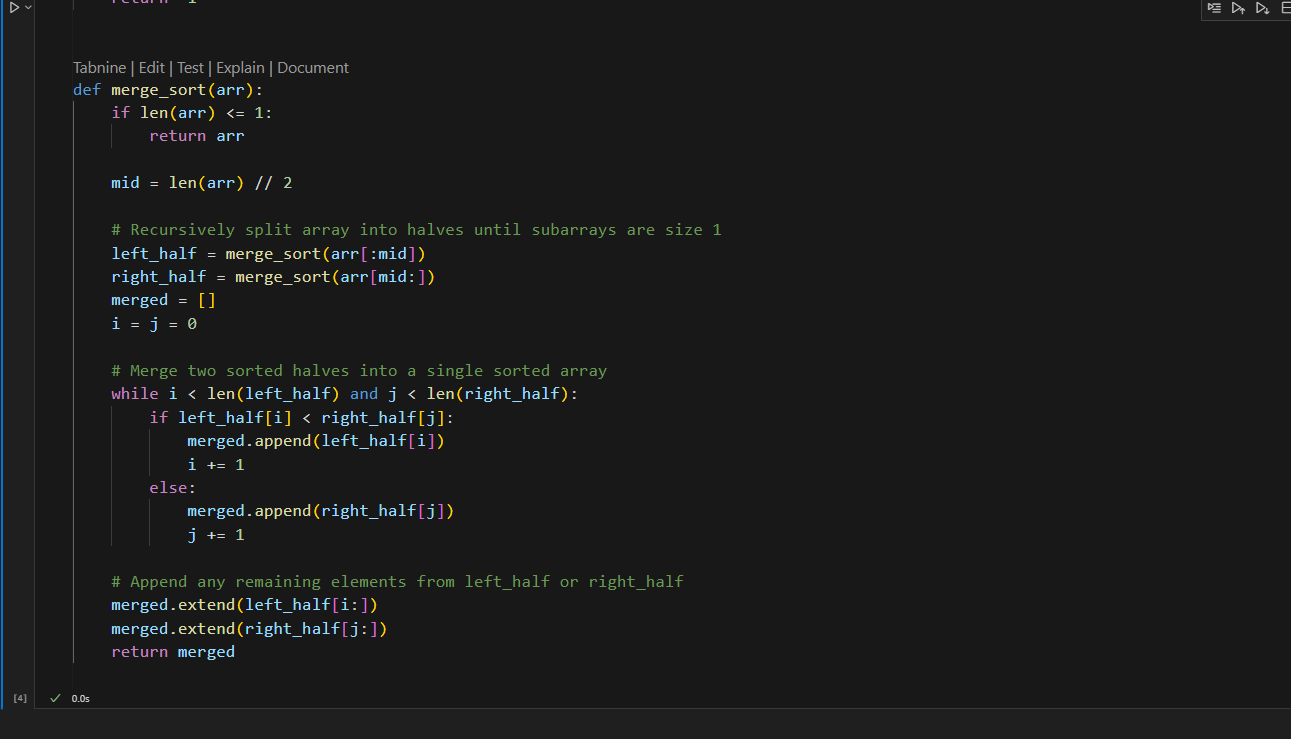
**TASK-2:** (Documentation – Inline Comments for Complex  
Logic)  
Use AI to add meaningful inline comments to a Python  
program explaining only complex logic parts.  
• Instructions:  
o Provide a Python script without comments to the AI.  
o Instruct AI to skip obvious syntax explanations and focus  
only on tricky or non-intuitive code sections.  
o Verify that comments improve code readability and  
maintainability.  
**Original Python Script (Without Comments):**

****

**Prompt:** Modify the code with concise, context-aware inline comments  
for complex logic blocks

**Documented Python Script with Inline Comments on Complex Logic:**

****

****

**Why These Comments Are Helpful:**

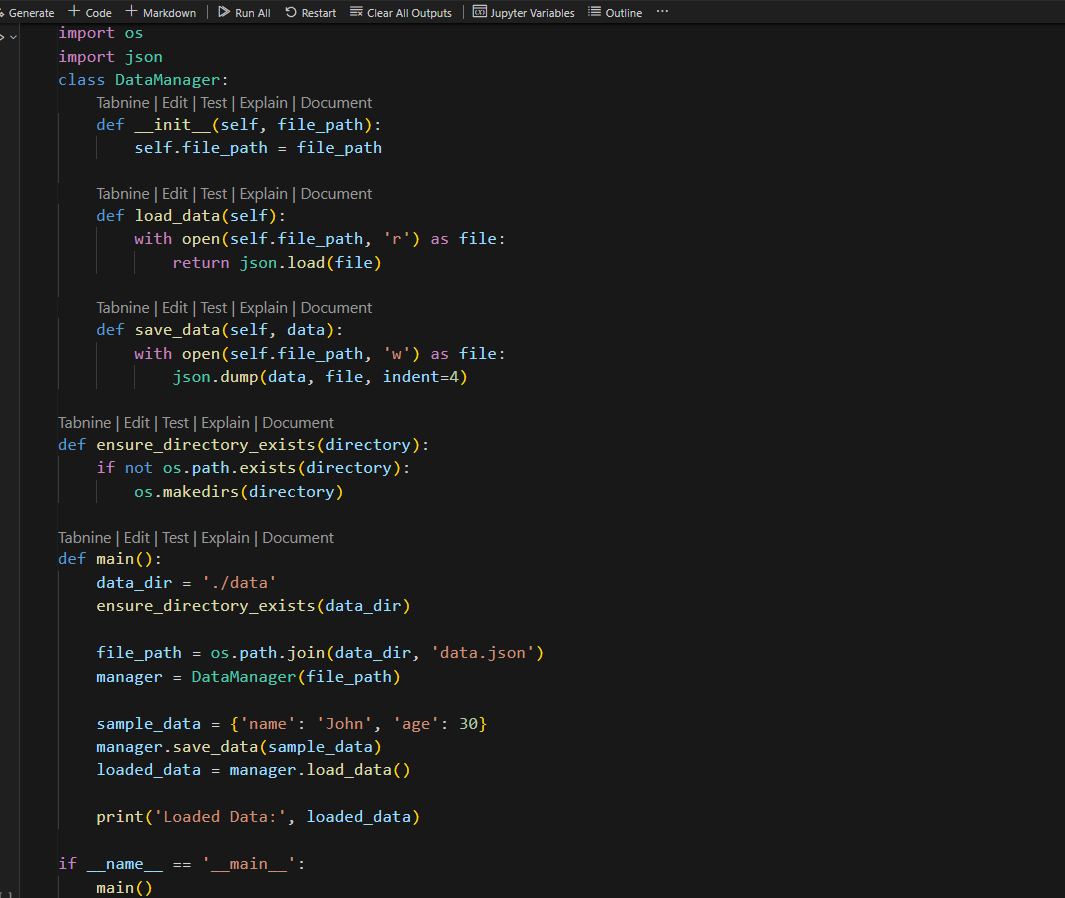
* They explain whycertain decisions are made, like the iterative Fibonacci generation, binary search logic, and recursive splitting + merging in merge sort.
* Obvious syntax (e.g., simple assignments or return statements) is skipped.
* Only non-trivial logic is commented for clarity and maintainability.

**TASK-3:**(Documentation – Module-Level Documentation)  
Use AI to create a module-level docstring summarizing the  
purpose, dependencies, and main functions/classes of a Python  
file.

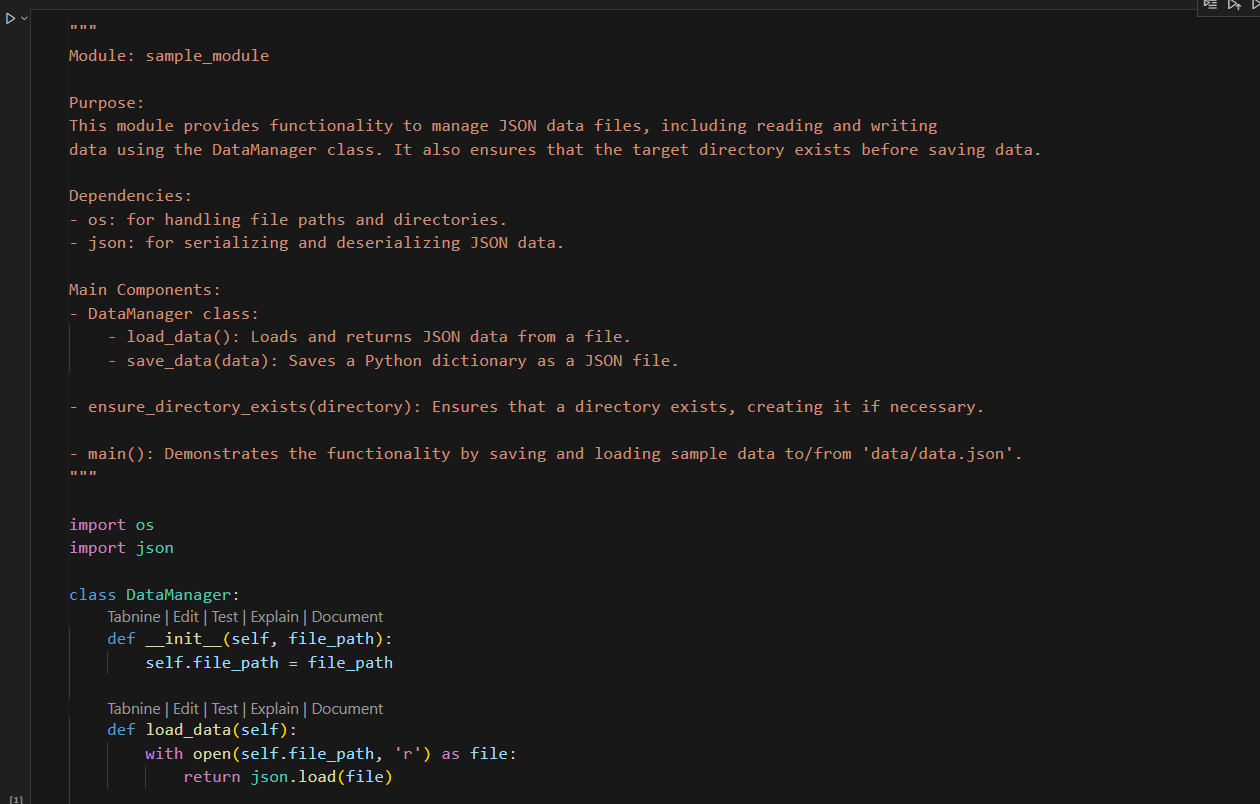
• Instructions:  
o Supply the entire Python file to AI.  
o Instruct AI to write a single multi-line docstring at the top  
of the file.  
o Ensure the docstring clearly describes functionality and  
usage without rewriting the entire code.

**Prompt:** write a single multi-line docstring that describes functionality and  
usage without rewriting the entire code at the top  
of the file.

Example python file:



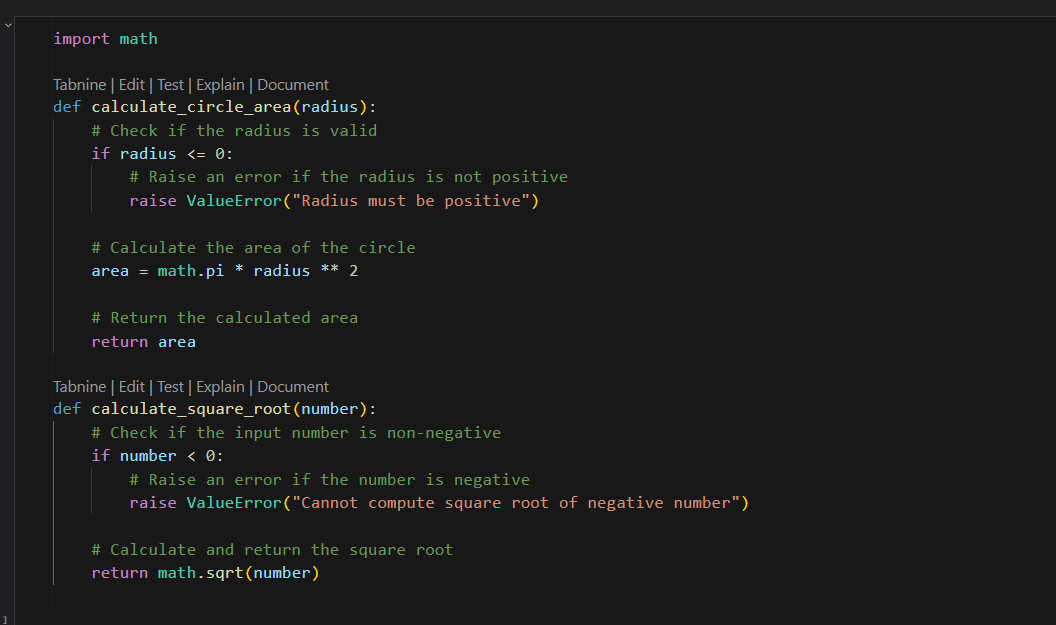
**Python File with Docstring:**





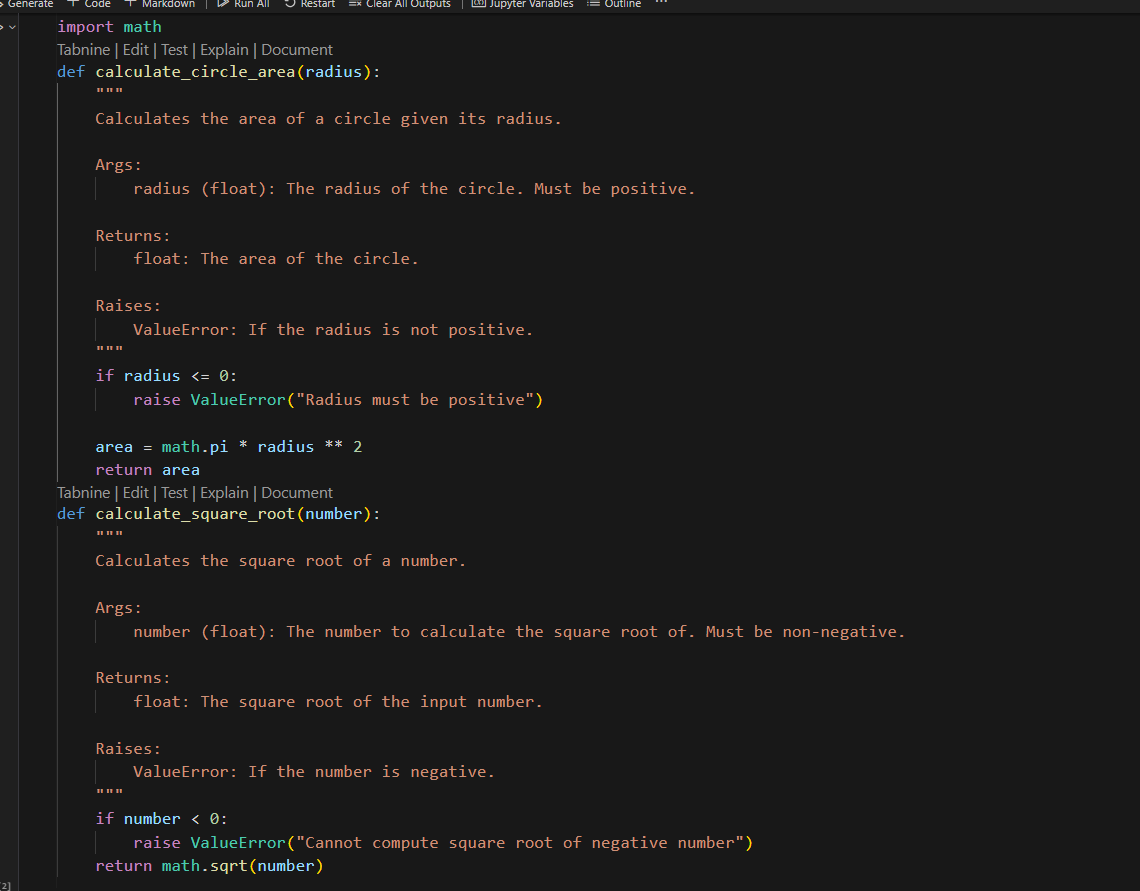
**TASK-4:**Documentation – Convert Comments to  
Structured Docstrings)  
Use AI to transform existing inline comments into  
structured function docstrings following Google style.  
• Instructions:  
o Provide AI with Python code containing inline comments.  
o Ask AI to move relevant details from comments into  
function docstrings.  
o Verify that the new docstrings keep the meaning intact  
while improving structure.  
• Expected Output #4:  
o Python code with comments replaced by clear,  
standardized docstrings

**Example Python Code with Inline Comments (Before):**

****

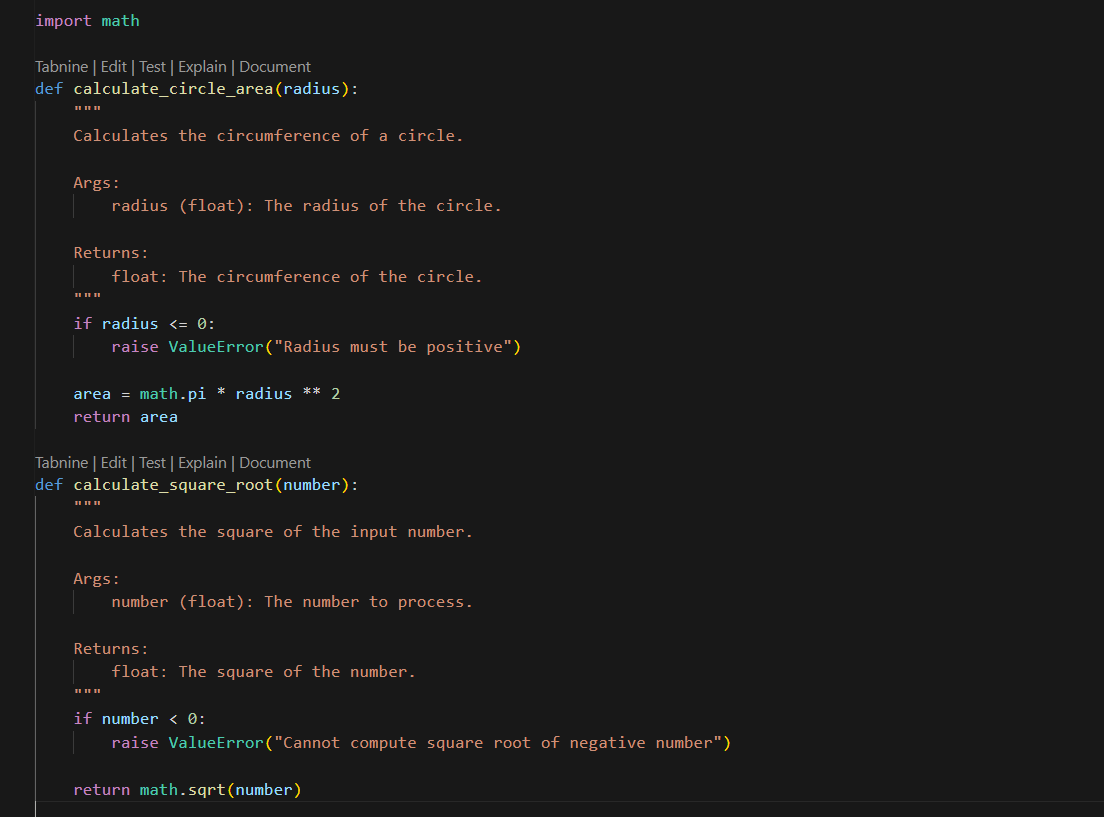
**Prompt:** Modify Python code with comments replaced by clear,  
standardized docstrings.

**Final Code with Structured Docstrings:**



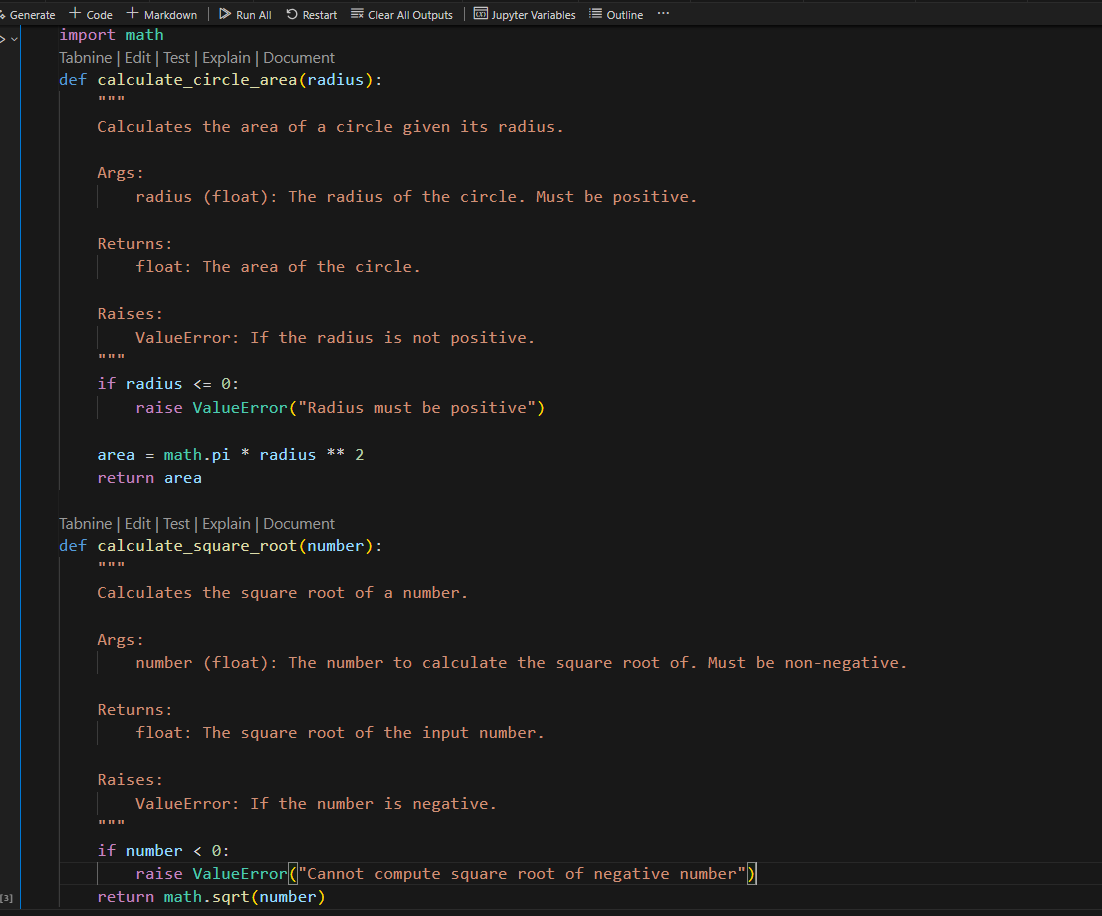
**Task-5:**(Documentation – Review and Correct  
Docstrings)  
Use AI to identify and correct inaccuracies in existing  
docstrings.  
• Instructions:  
o Provide Python code with outdated or incorrect  
docstrings.  
o Instruct AI to rewrite each docstring to match the current  
code behavior.  
o Ensure corrections follow Google-style formatting

**Example Python Code with Incorrect/Outdated Docstrings:**

****

**Prompt:** Modify Python file with updated, accurate, and standardized  
docstrings

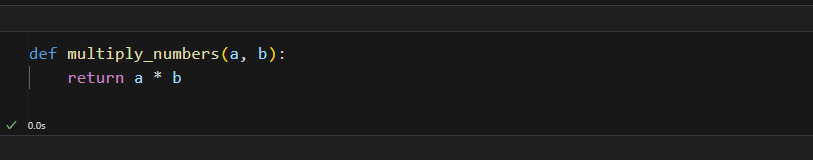
**Corrected and Updated Docstrings :**

****

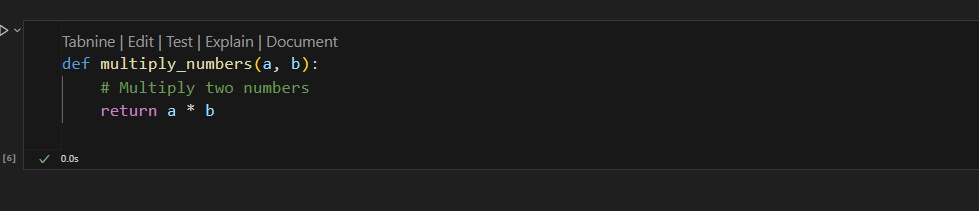
**TASK-6:** (Documentation – Prompt Comparison  
Experiment)

Compare documentation output from a vague prompt and a  
detailed prompt for the same Python function.  
• Instructions:  
o Create two prompts: one simple (“Add comments to this  
function”) and one detailed (“Add Google-style docstrings  
with parameters, return types, and examples”).  
o Use AI to process the same Python function with both  
prompts.  
o Analyze and record differences in quality, accuracy, and  
completeness.  
• Expected Output #6:  
o A comparison table showing the results from both  
prompts with observations

**Sample Python Function to Document:**

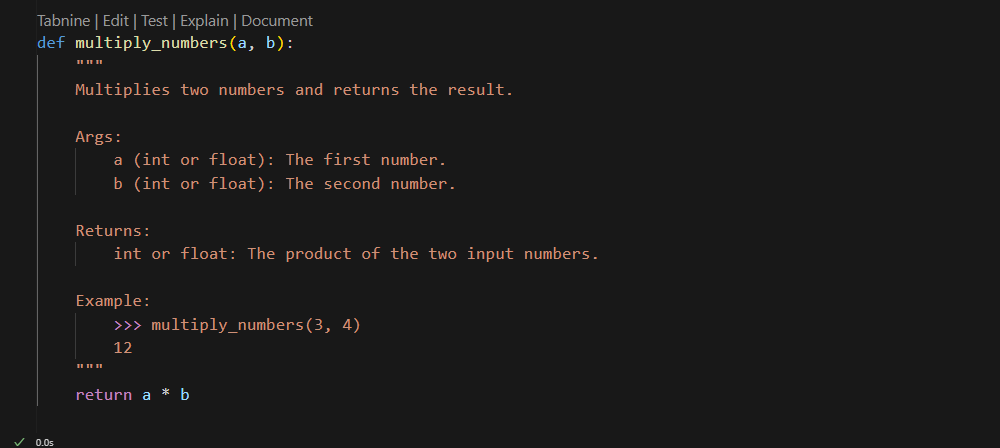
****

**Prompt-1:**Add comments to this function**.**



**Prompt-2:**Add Google-style docstrings with parameters, return types, and examples.

**AI Output:**

****

**Comparision table:**

| **Criteria** | **Vague Prompt Output** | **Detailed Prompt Output** |
| --- | --- | --- |
| **Comments vs. Docstring** | Simple inline comment: # Multiply two numbers | Structured, Google-style docstring with sections: Args, Returns, Example |
| **Parameter Types** | Not specified | Explicitly documents that a and b can be int or float |
| **Return Type** | Not specified | Clearly specified (int or float) |
| **Example Usage** | None | Includes an example of usage: >>> multiply\_numbers(3, 4) → 12 |
| **Completeness** | Minimal and vague | Complete, well-structured, and precise |
| **Accuracy** | Correct but lacks detail | Correct, complete, and informative |
| **Professional Quality** | Low for production or documentation standards | High, suitable for production-ready code documentation |