

**Program :**B.tech(CSE)

**Specialization :**AIML

**Course Title :**AI Assisted Coding

**Course Code :**24CS002PC215

**Semester :**3rd semester

**Academic Session :**2025-2026

**Name of Student :**Kurapati Ashritha

**Enrollment No. :**2403A52023

**Batch No. :**02

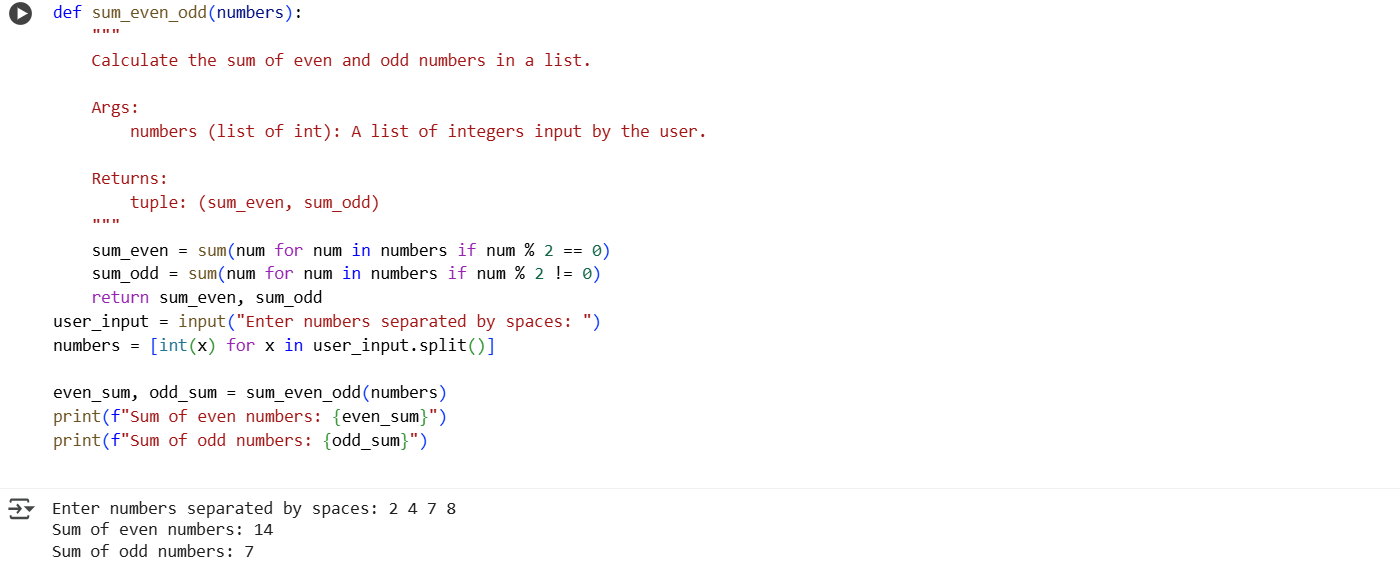
**Date :**24/09/2025

#LAB ASSIGNMENT

#TASK DESCRIPTION-1:

• 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.

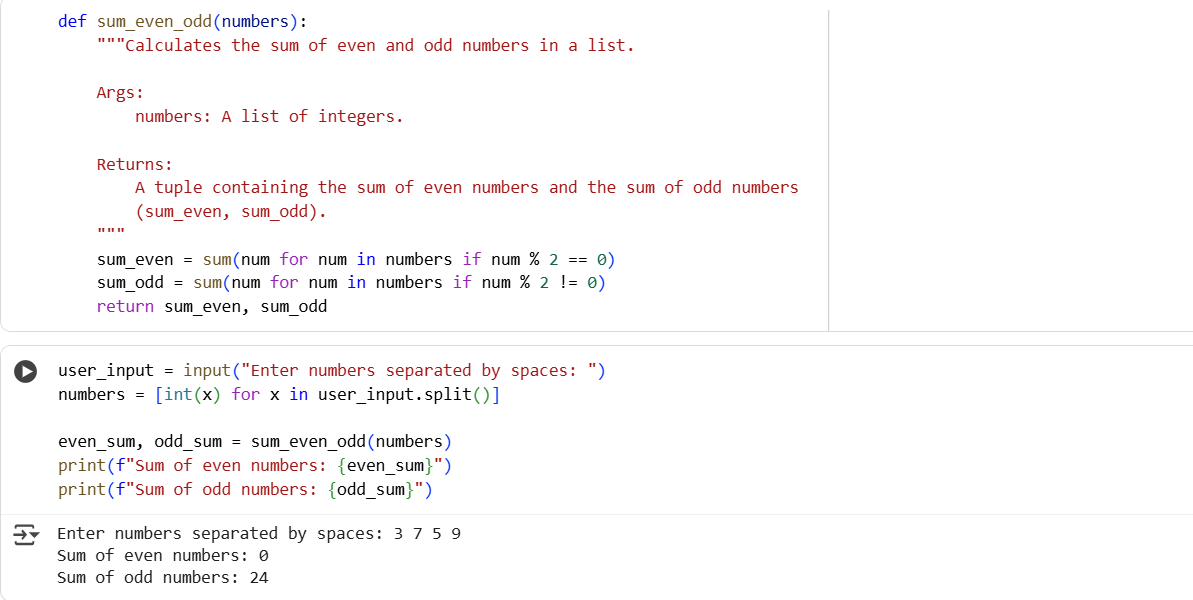
#MANUAL CODE:

#CODE GENERATED BY AI:

Prompt:

 Generate a Google-style docstring for a Python function `sum\_even\_odd` that takes a list of integers and returns a tuple with the sum of even and odd numbers with dynamic input.

CODE:



# OUTPUT:

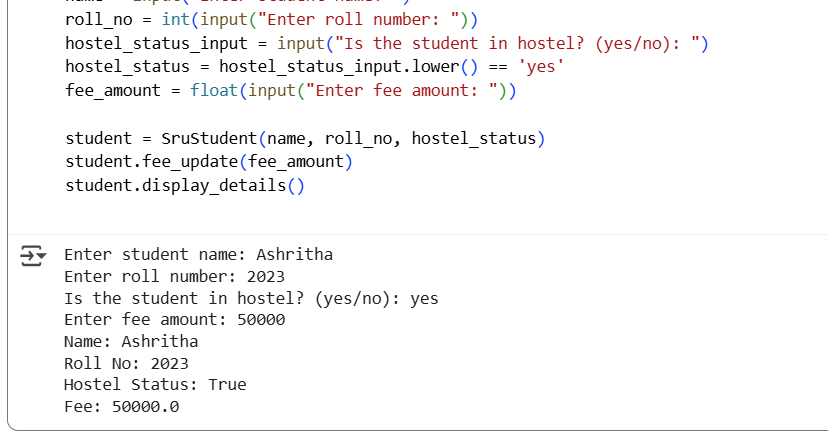
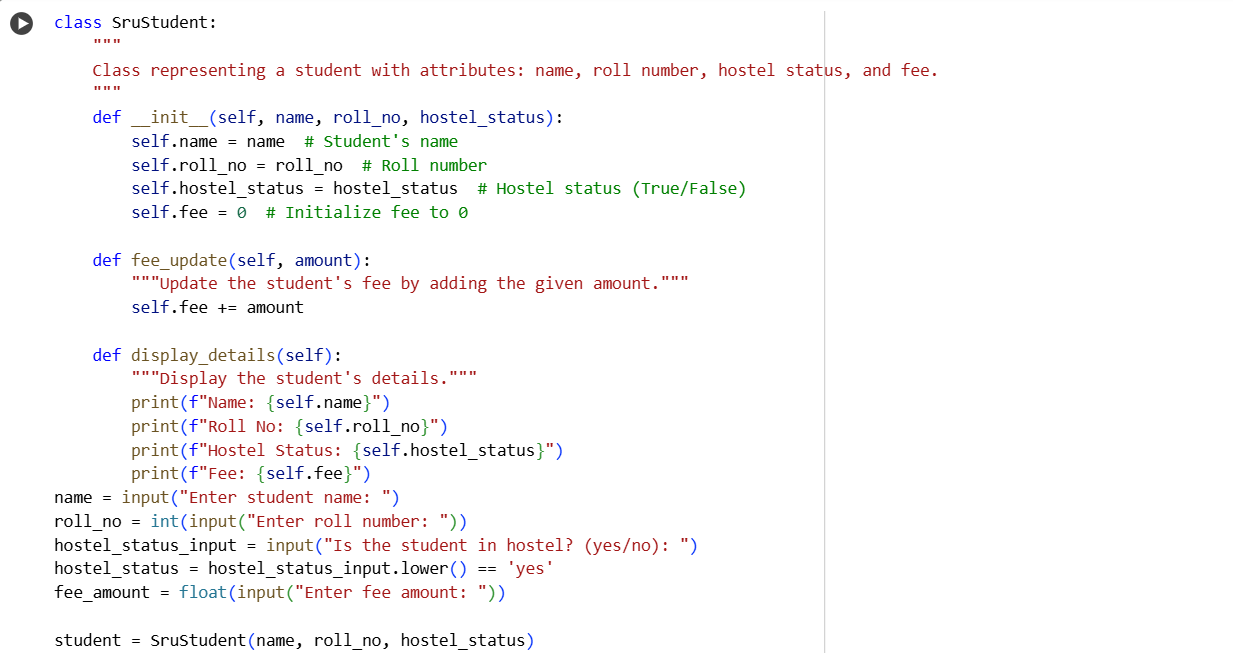
* Cell c1d0ece5 (the function definition with the docstring): This cell is better for understanding what the sum\_even\_odd function does at its core. The Google-style docstring clearly explains its purpose, arguments, and return value, allowing you to grasp the function's functionality without delving into the implementation details. This is crucial for code reusability and maintainability.
* Cell e5020bed (the input and function call): This cell is better for understanding how to use the sum\_even\_odd function with dynamic user input and how to display the results. It demonstrates a practical application of the function.

Therefore, Cell c1d0ece5 is better for understanding the function's logic and purpose, while Cell e5020bed is better for understanding how to interact with the user and utilize the function.

#TASK DESCRIPTION-2:

• 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.

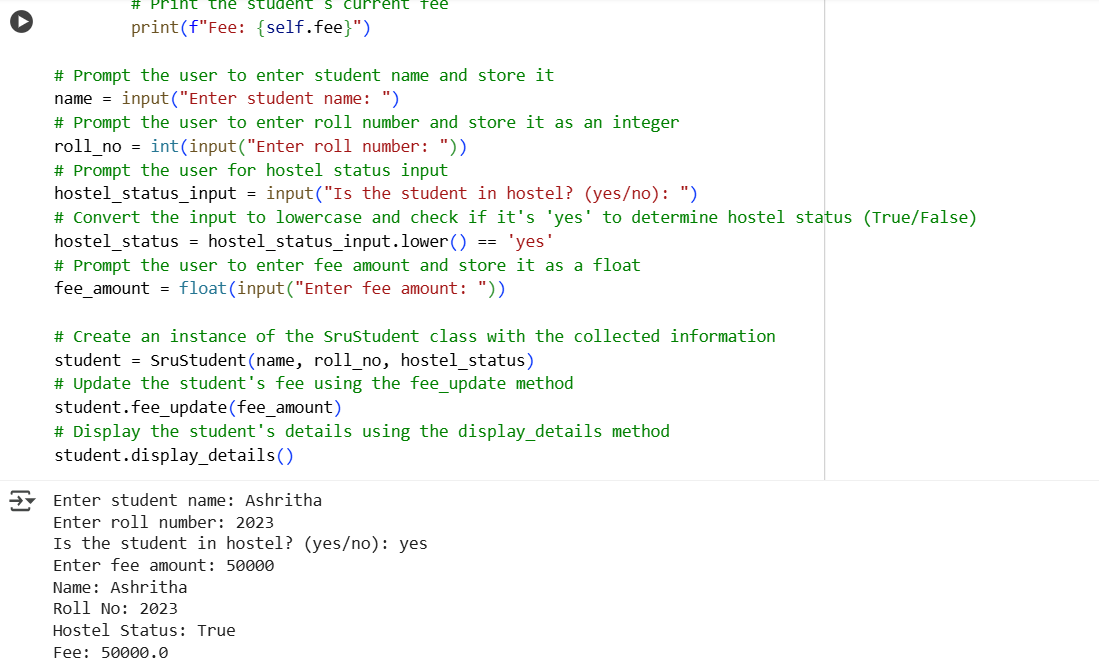
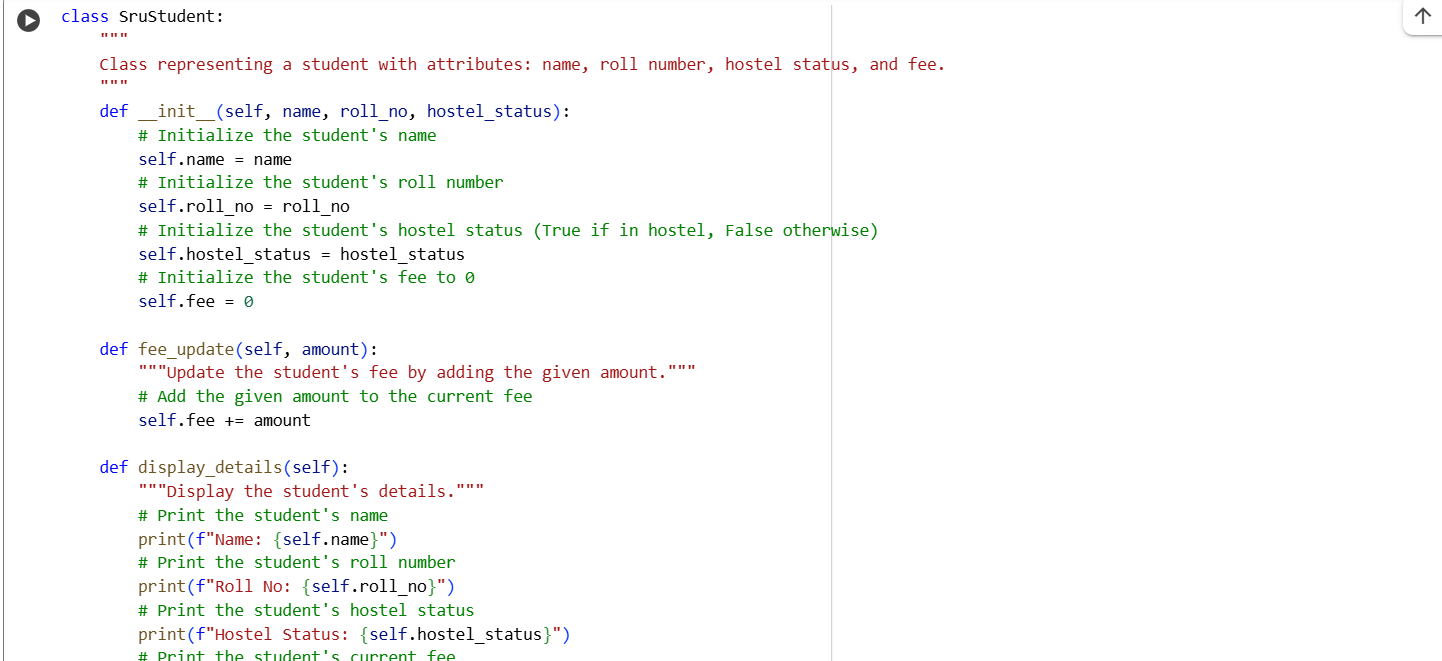
#MANUAL CODE:



#GENERATED BY AI:

Prompt:

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



#OUTPUT:

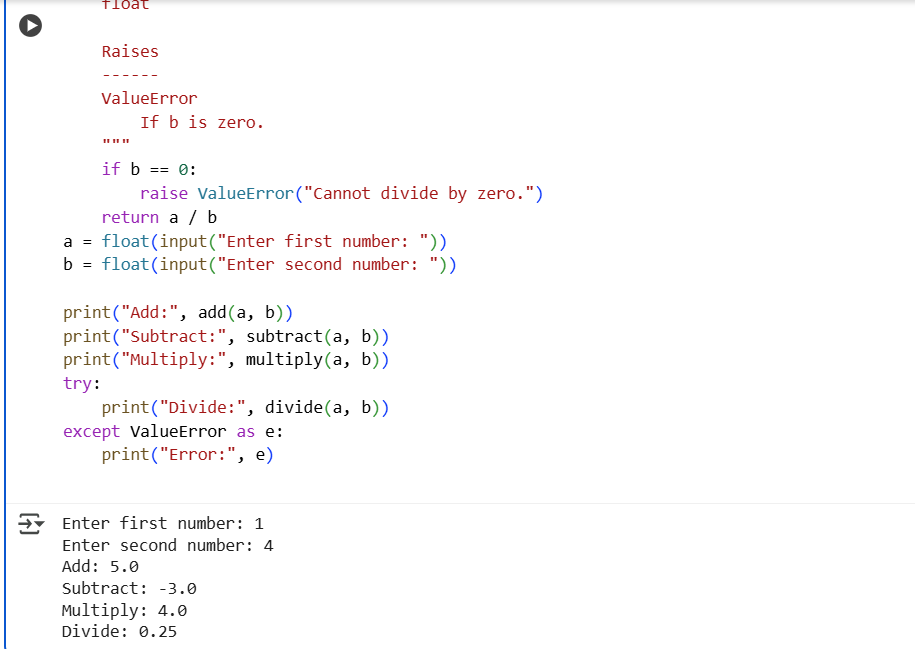
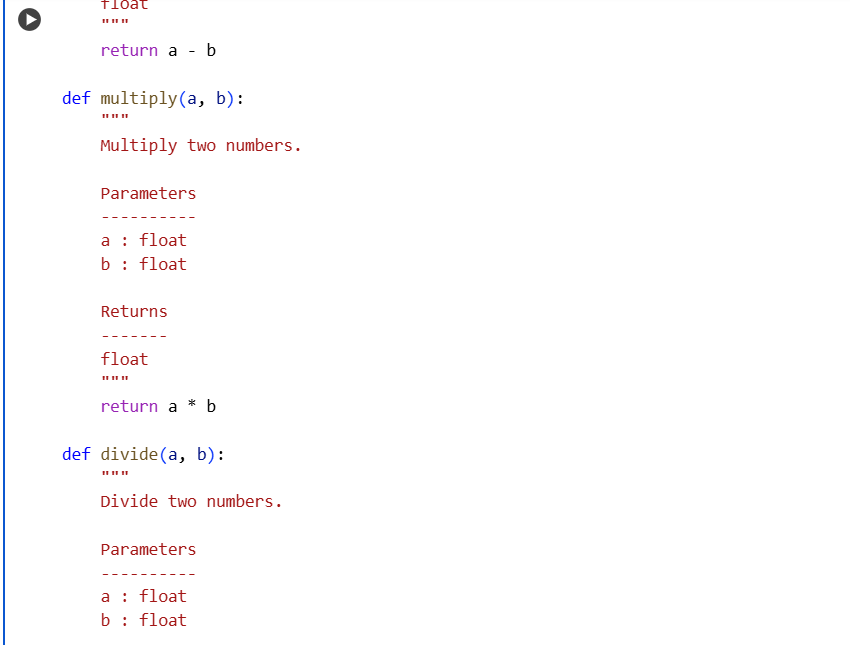
Analyze AI-generated code comments:

* **Placement:** The comments are placed directly after the line of code they describe. This makes it easy to see which comment belongs to which line.
* **Detail Level:** The comments are quite detailed, explaining the purpose of initializing each attribute (self.name, self.roll\_no, self.hostel\_status, self.fee) and the action performed by each line within the fee\_update and display\_details methods.
* **Clarity:** The language used is straightforward and easy to understand, even for someone relatively new to Python.
* **Effectiveness:** These inline comments are effective in explaining the immediate action of each line of code. They are particularly helpful for understanding:
  + What each attribute is being set to during initialization.
  + How the fee\_update method modifies the fee attribute.
  + What information is being printed by each line in the display\_details method.

#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 CODE:

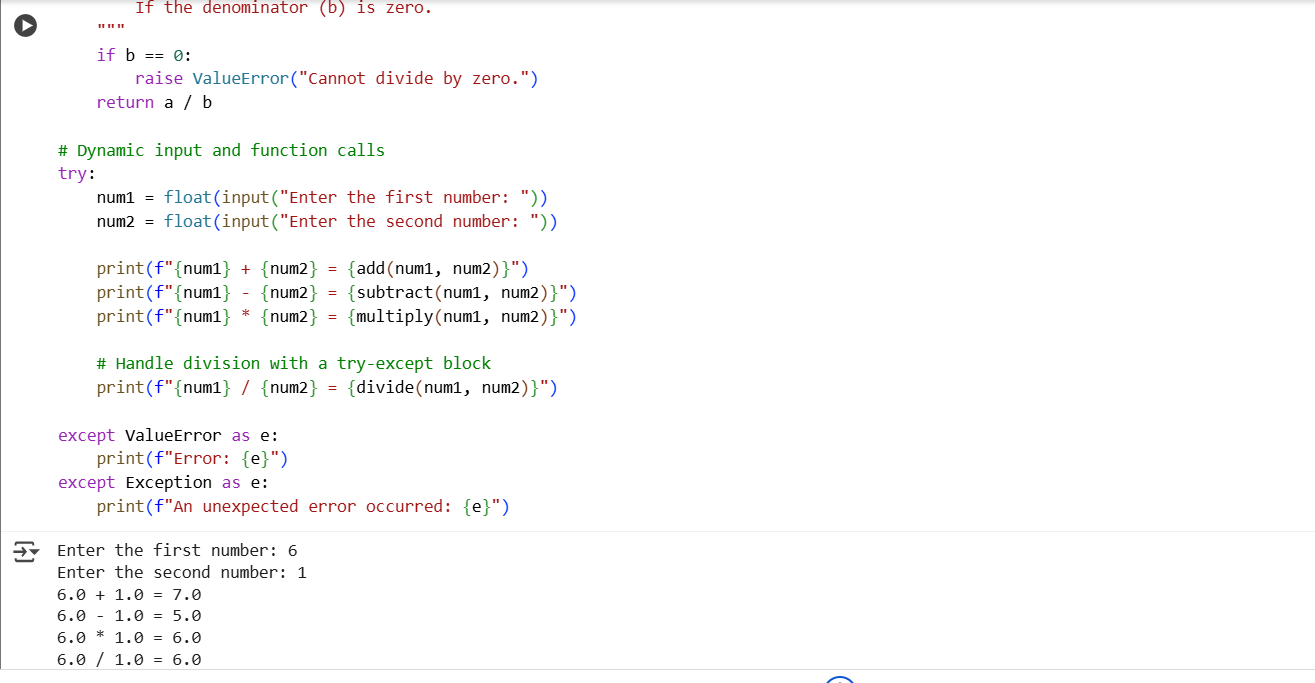
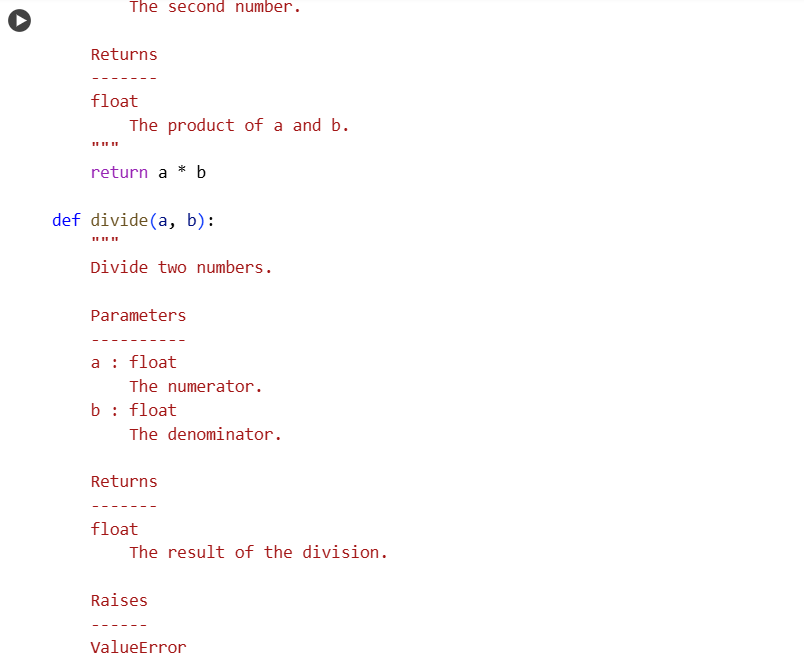
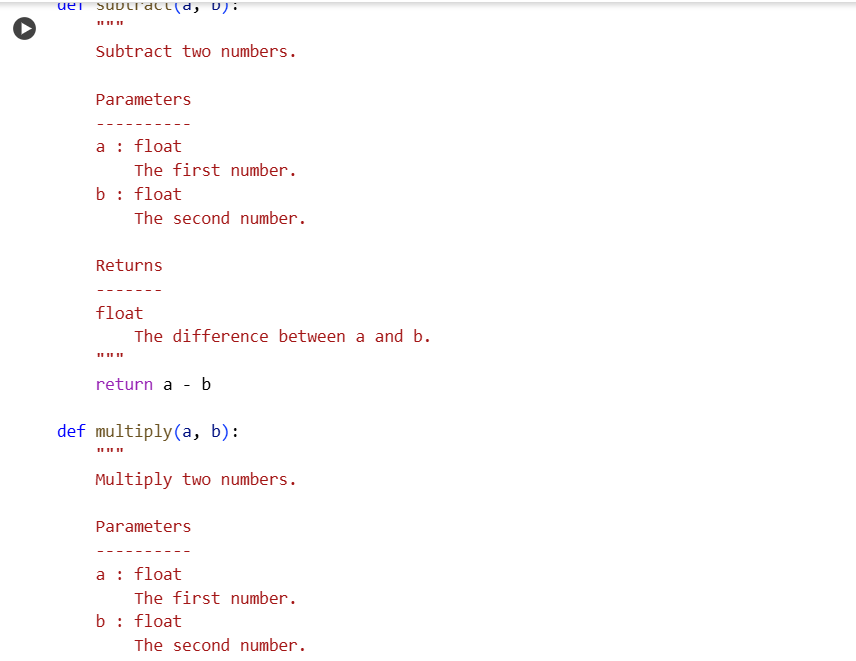
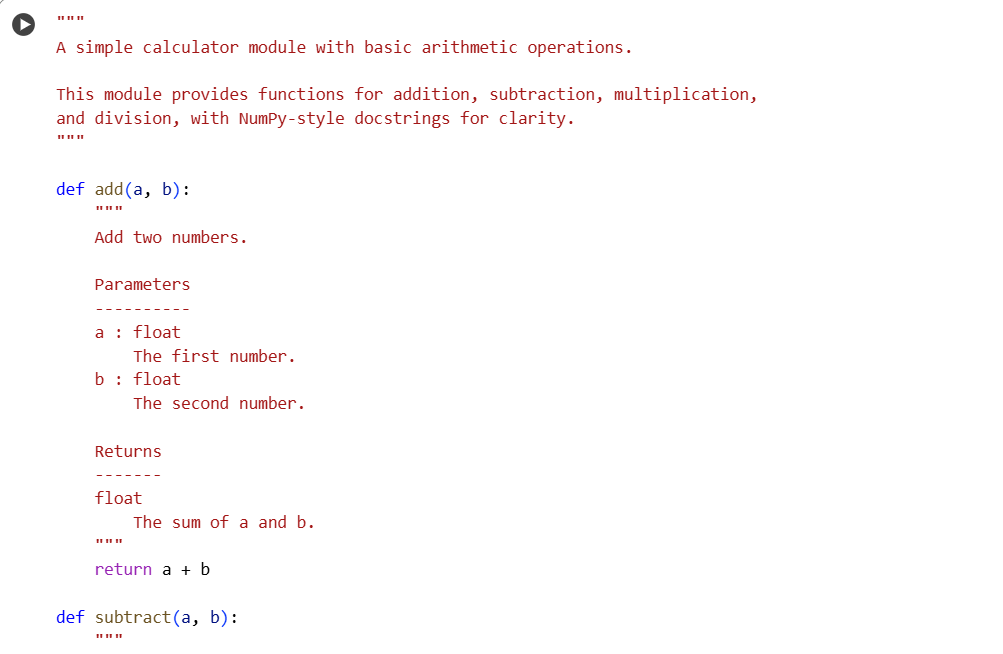


#GENERATED BY AI:

Prompt:

 Generate a module-level docstring and individual NumPy-style docstrings for a Python calculator module with functions: add, subtract, multiply, divide. Include parameter descriptions, return types, and exceptions with dynamic input.

CODE:



#OUTPUT:

* Cell RrMuWq0T9kP2: This was the original calculator code you had. It included basic functions for addition, subtraction, multiplication, and division, along with docstrings that provided a general description of each function, their parameters, and return types. It also included the input and function calls directly after the function definitions.
* Cell b48bf52e: This is the new code I just generated based on your request. It also includes the same four basic functions. However, it has a few key differences:
  + Module-Level Docstring: It includes a docstring at the very beginning of the cell, explaining the overall purpose of the "module" (in this case, the code in the cell acting as a module).
  + NumPy-style Docstrings: The docstrings for each function follow the NumPy style, which is a more structured and detailed format for documenting scientific and numerical code. This style includes specific sections like Parameters, Returns, and Raises, making it very clear what the function expects, what it gives back, and what potential errors it might raise.
  + Error Handling: Both cells include error handling for division by zero using a try-except block, but the new cell also includes a more general except Exception block to catch any other unexpected errors.
  + Dynamic Input: Both cells include dynamic input to get numbers from the user and then call the functions.