**HALL TICKET NUMBER: 2403A51365**

**NAME: RAYAPUDI HARITHA**

**BATCH: 24BTCAICSB14**

**AssignmentNumber:10.2**

Lab 10: Code Review and Quality: Using AI to improve code quality and readability  
Lab Objectives:  
• To understand the importance of code readability, maintainability, and quality.  
• To explore how AI-assisted coding tools can review code and suggest improvements.  
• To practice identifying code smells, redundant code, and poor naming conventions.  
• To apply AI tools for refactoring and improving readability.  
• To critically evaluate AI feedback and integrate it into real projects  
Week5

Lab Outcomes (LOs):  
After completing this lab, students will be able to:  
• Use AI-assisted tools (e.g., GitHub Copilot, Cursor AI) to review Python code.  
• Identify and correct syntax issues, code smells, and inefficient logic.  
• Improve readability by applying consistent formatting, naming, and comments.  
• Refactor code with AI suggestions while ensuring functionality is preserved.  
• Apply best practices for writing clean, maintainable, and professional code.

Task Description#1 AI-Assisted Code Review (Basic Errors)

**Prompt:-**

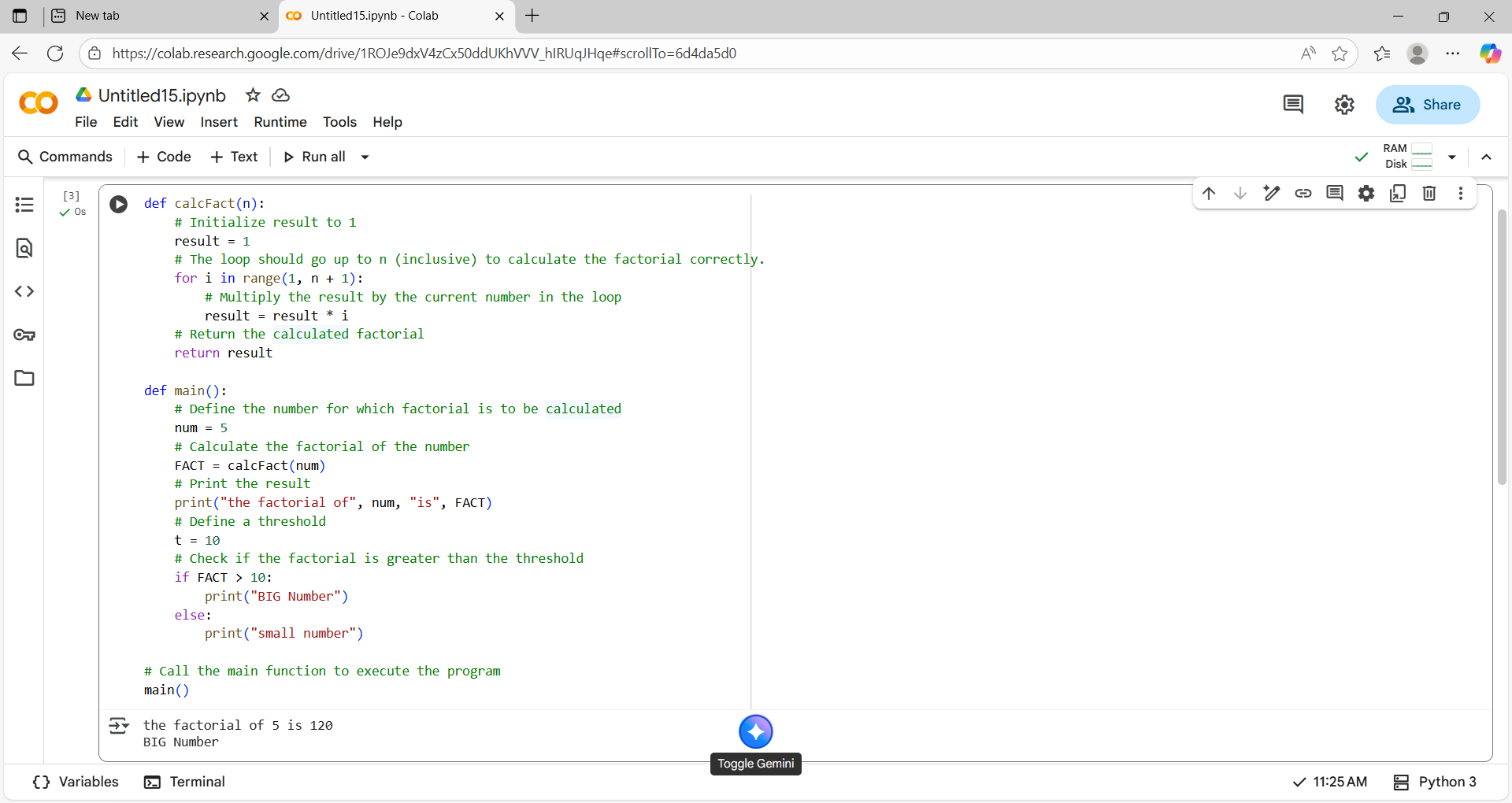
• Write python program as shown below.

• Use an AI assistant to review and suggest corrections

A screenshot of a computer program

AI-generated content may be incorrect.

Code And Output:-

**

**Code Explanation:-**

* *The code defines two functions: calcFact and main.*
* *The calcFact function calculates the factorial of a given number.*
* *It initializes a result variable to 1.*
* *It uses a for loop to iterate from 1 up to the input number (inclusive).*
* *Inside the loop, it multiplies the result by the current number.*
* *The main function demonstrates the usage of calcFact.*
* *It calculates the factorial of the number 5.*
* *It prints the calculated factorial.*
* *It checks if the factorial is greater than a threshold (10).*
* *Based on the comparison, it prints either "BIG Number" or "small number".*

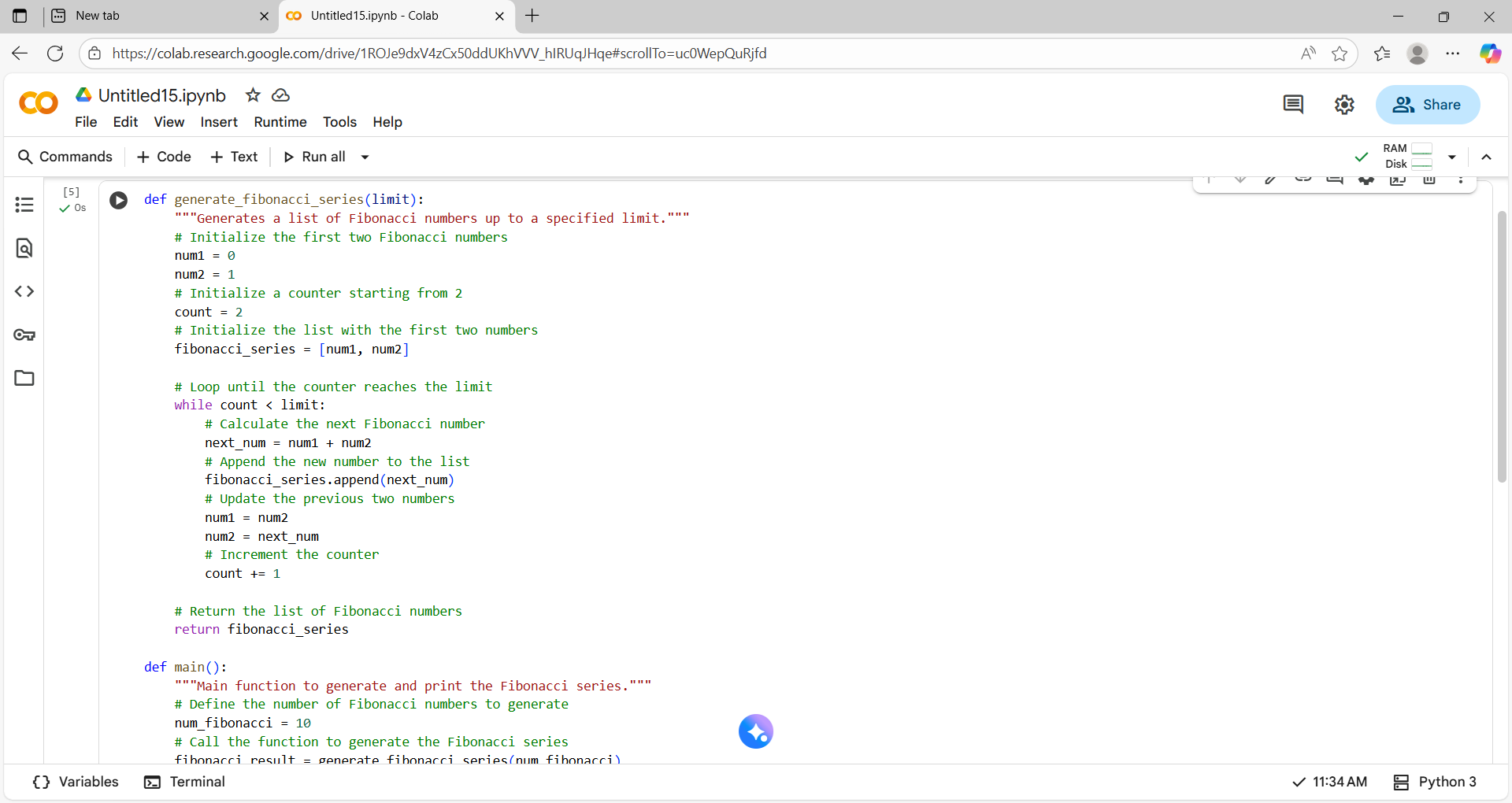
*Task Description#2 Automatic Inline Comments*

**Prompt:-**

*• Write the Python code for Fibonacci as shown below and execute.  
• Ask AI to improve variable names, add comments, and apply PEP8 formatting  
(cleaned up).  
• Students evaluate which suggestions improve readability most. one.*A screenshot of a computer

AI-generated content may be incorrect.

Code And Output:-

**

*A screenshot of a computer

AI-generated content may be incorrect.*

**Code Explanation:-**

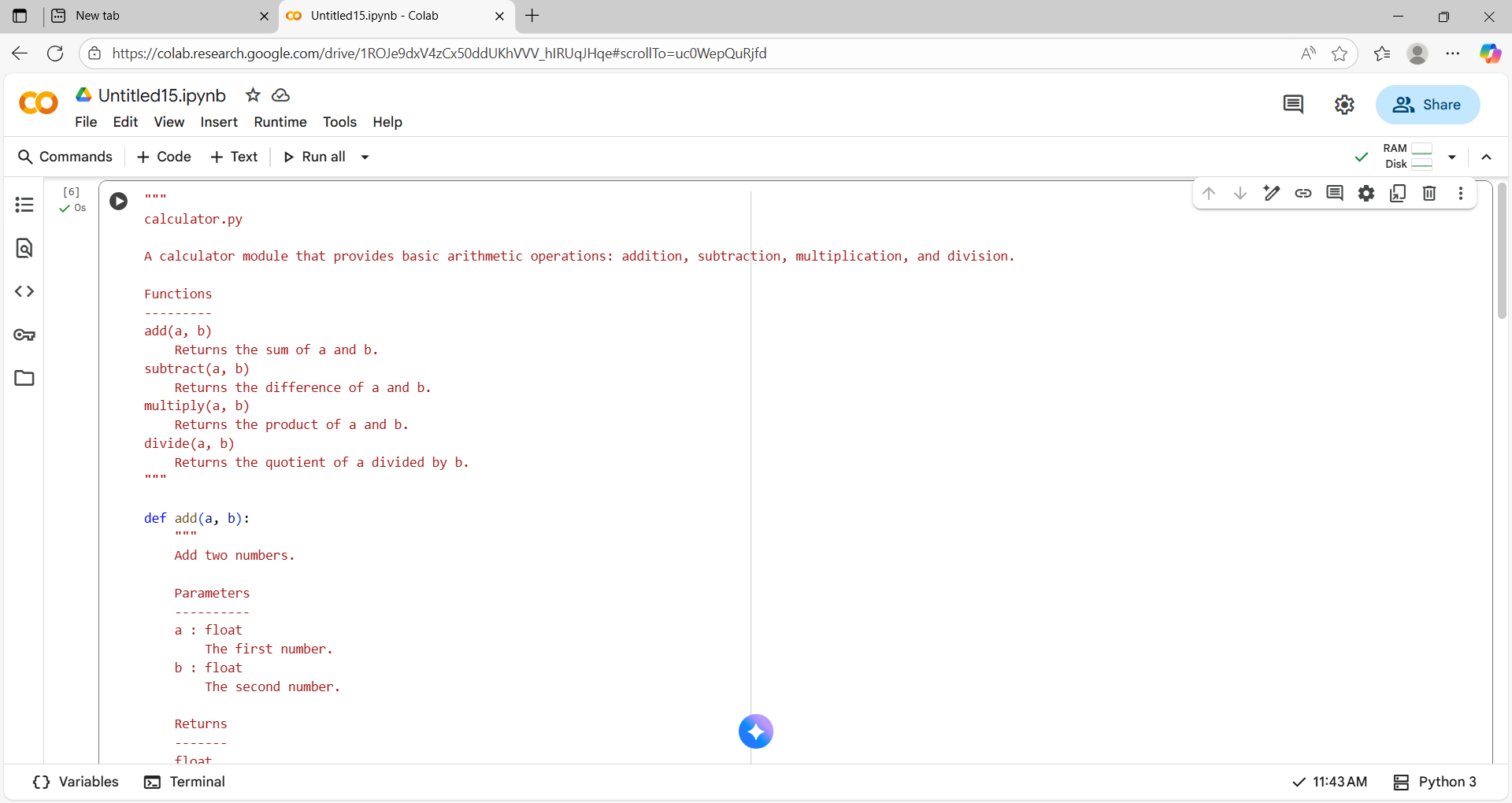
* *The code defines two functions: generate\_fibonacci\_series and main.*
* *The generate\_fibonacci\_series function generates a list of Fibonacci numbers up to a specified limit.*
* *It initializes the first two Fibonacci numbers, num1 and num2, to 0 and 1.*
* *It uses a while loop to generate subsequent Fibonacci numbers.*
* *Inside the loop, it calculates the next\_num by summing the previous two.*
* *The next\_num is appended to the fibonacci\_series list.*
* *The values of num1 and num2 are updated in each iteration.*
* *The main function calls generate\_fibonacci\_series with a limit of 10.*
* *It stores the result in fibonacci\_result.*
* *Finally, it prints the generated Fibonacci series.*

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.  
Common Examples of Code Smells  
• Long Function – A single function tries to do too many things.  
• Duplicate Code – Copy-pasted logic in multiple places.  
• Poor Naming – Variables or functions with confusing names (x1, foo, data123).  
• Unused Variables – Declaring variables but never using them.  
• Magic Numbers – Using unexplained constants (3.14159 instead of PI).  
• Deep Nesting – Too many if/else levels, making code hard to read.  
• Large Class – A single class handling too many responsibilities.  
Why Detecting Code Smells is Important  
• Makes code easier to read and maintain.  
• Reduces chance of bugs in future updates.  
• Helps in refactoring (improving structure without changing behavior).  
• Encourages clean coding practices  
Dead Code – Code that is never executed.

**Prompt:-**

*• 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.*

Code And Output:-

**

*A screenshot of a computer

AI-generated content may be incorrect.*

*A screenshot of a computer

AI-generated content may be incorrect.*

*A screenshot of a computer

AI-generated content may be incorrect.*

*Comparesion table:-*

| ***Aspect*** | ***AI-Generated Docstring*** | ***Manually Written Docstring*** |
| --- | --- | --- |
| ***Clarity*** | *Often clear but may include redundant or generic phrases* | *Tailored for the specific function, concise and focused* |
| ***Structure*** | *Follows standard format (e.g., summary, parameters, returns) but may lack nuance* | *Structured with attention to readability and developer expectations* |
| ***Detail Level*** | *May include excessive or insufficient detail depending on prompt quality* | *Balanced detail—enough to understand usage without overwhelming* |
| ***Context Awareness*** | *Limited understanding of broader project or domain context* | *Written with full awareness of project goals and integration points* |
| ***Examples*** | *May include generic examples or none at all* | *Includes relevant, tested examples that reflect real use cases* |
| ***Tone*** | *Neutral and formal, sometimes robotic* | *Can be adapted to team style—friendly, formal, or technical* |
| ***Error Handling*** | *May omit edge cases or exceptions* | *Explicitly documents expected errors and how they’re handled* |
| ***Consistency*** | *Depends on prompt and model behavior* | *Maintains consistent style across codebase* |
| ***Update Responsiveness*** | *Needs re-generation if code changes* | *Can be manually updated alongside code changes* |
| ***Best Use Case*** | *Quick documentation for simple functions or prototyping* | *Production-level code, complex logic, or public APIs* |

**Code Explanation:-**

* *The code defines a Python module named calculator.py.*
* *It provides basic arithmetic operations: addition, subtraction, multiplication, and division.*
* *Each operation is implemented as a separate function.*
* *Each function includes a docstring explaining its purpose, parameters, and return value.*
* *The divide function includes error handling for division by zero.*
* *The module includes an if \_\_name\_\_ == "\_\_main\_\_": block.*
* *This block demonstrates how to use the defined functions.*
* *It calculates and prints the results of sample arithmetic operations.*
* *The output shows the results of adding, subtracting, multiplying, and dividing 5 and 3.*
* *This structure allows the module to be run directly or imported into other scripts.*