

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

**Specialization :**AIML

**Course Title :**AI Assisted Coding

**Course Code :**24CS002PC215

**Semester :**3rd semester

**Academic Session :**2025-2026

**Name of Student :**Bachanagari Deekshitha

**Enrollment No. :**2403A52032

**Batch No. :**02

**Date :**16/09/2025

#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.

#PROMPT:

Generate a Python function that takes a list of integers and returns the sum of even and odd numbers separately.

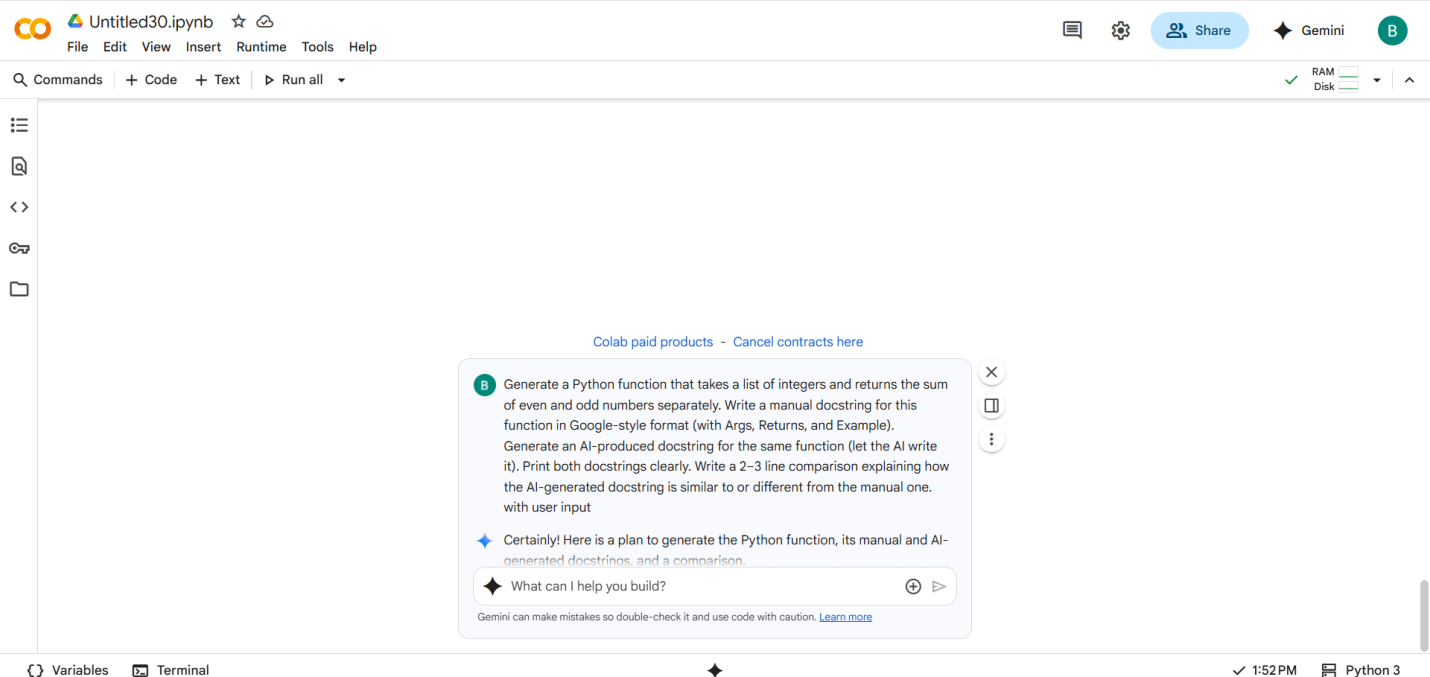
1. Write a manual docstring for this function in Google-style format (with Args, Returns, and Example).

2. Generate an AI-produced docstring for the same function (let the AI write it).

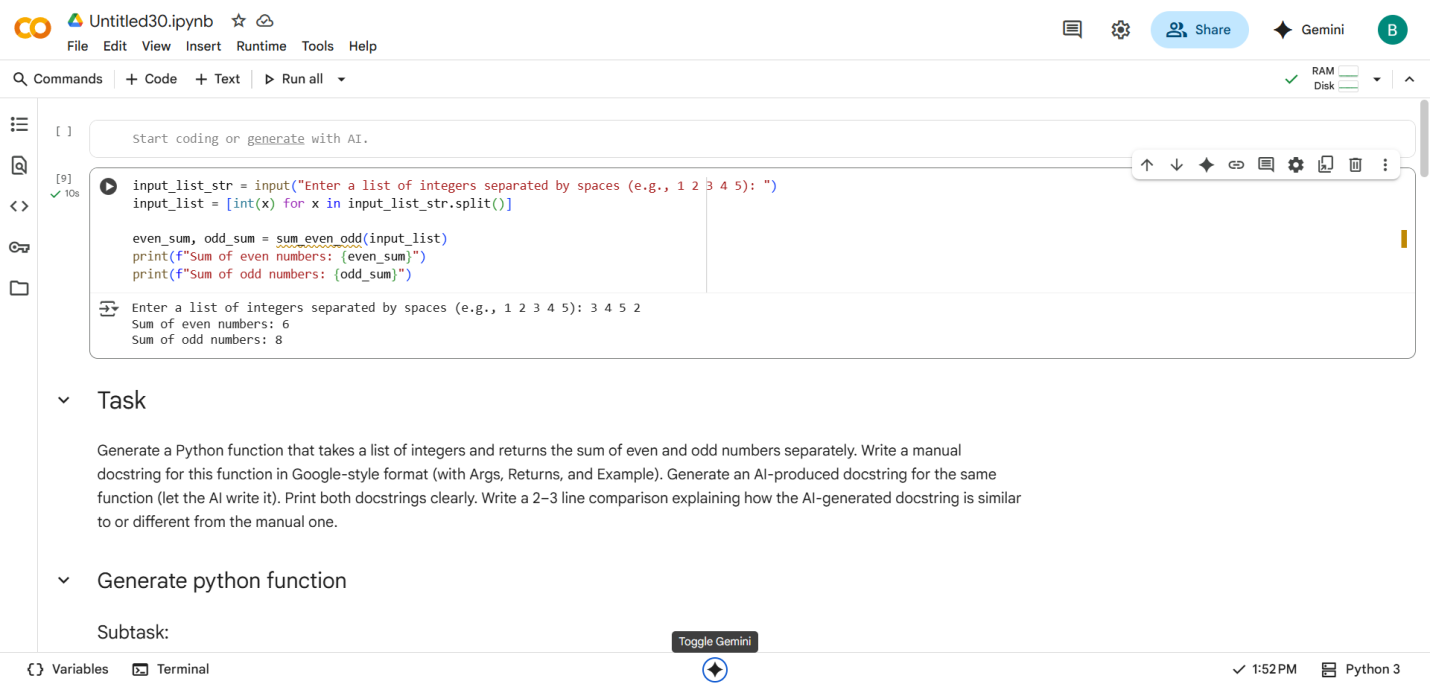
3. Print both docstrings clearly.

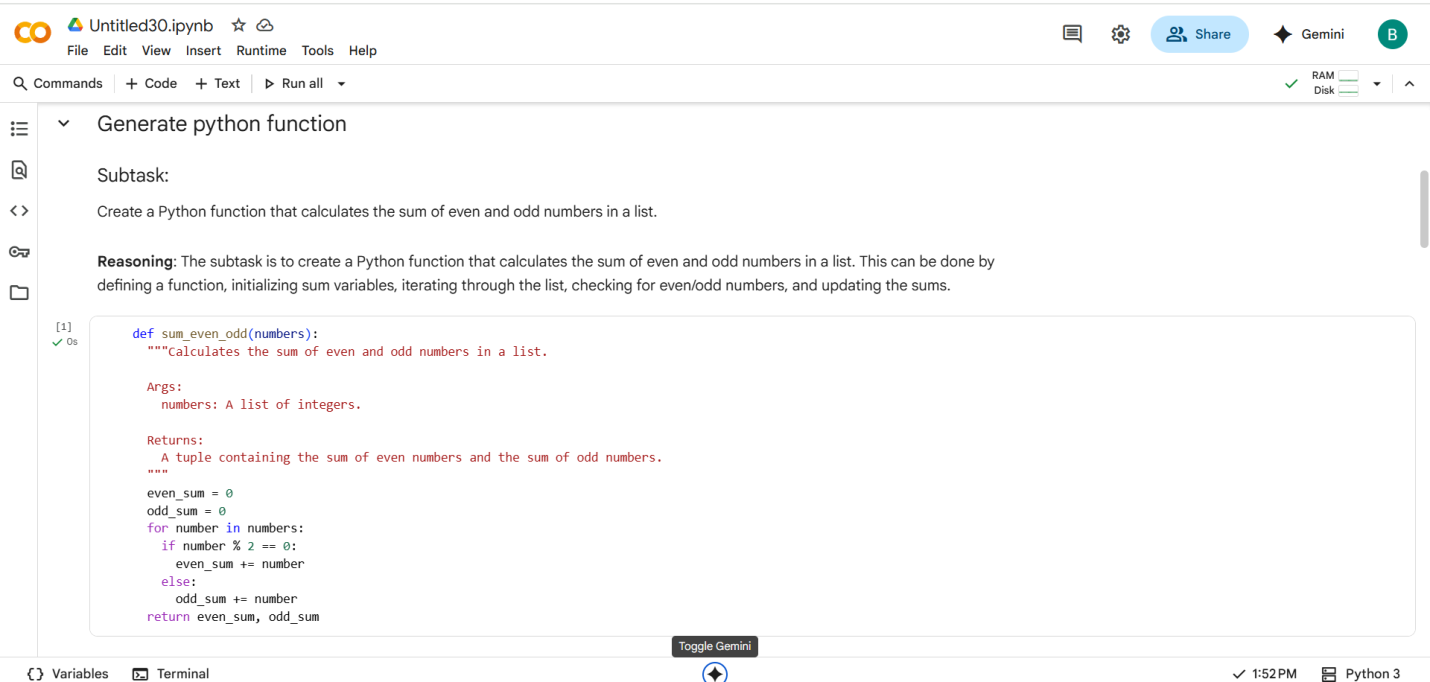
4. Write a 2–3 line comparison explaining how the AI-generated docstring is similar to or different from the manual one.

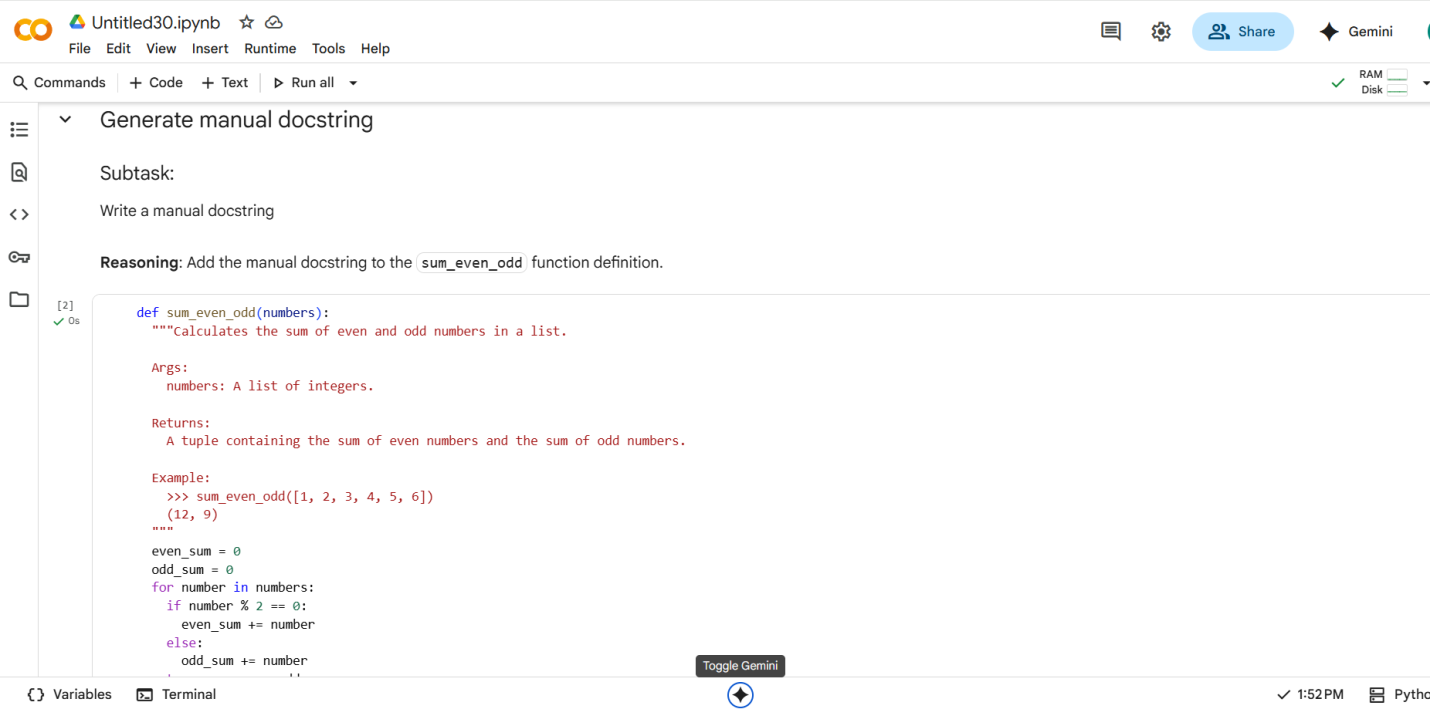
#QUESTION:

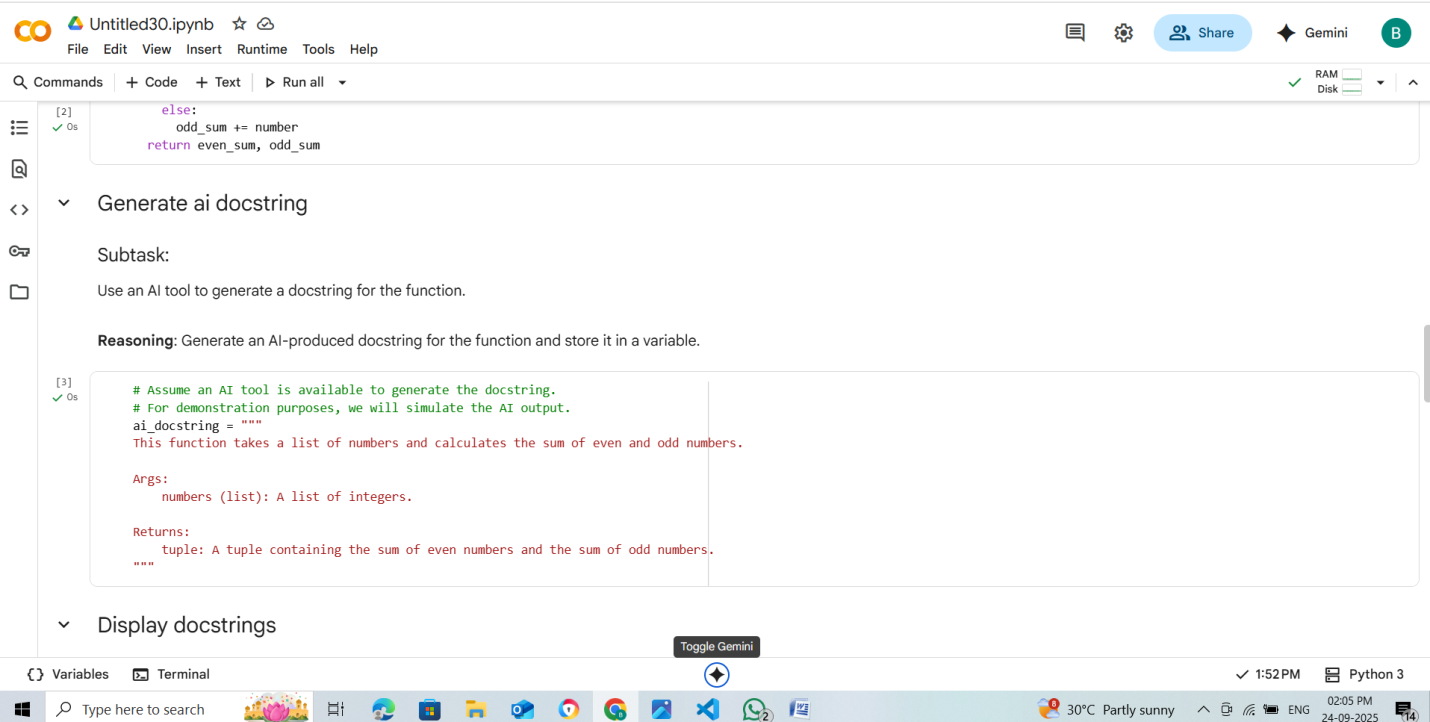


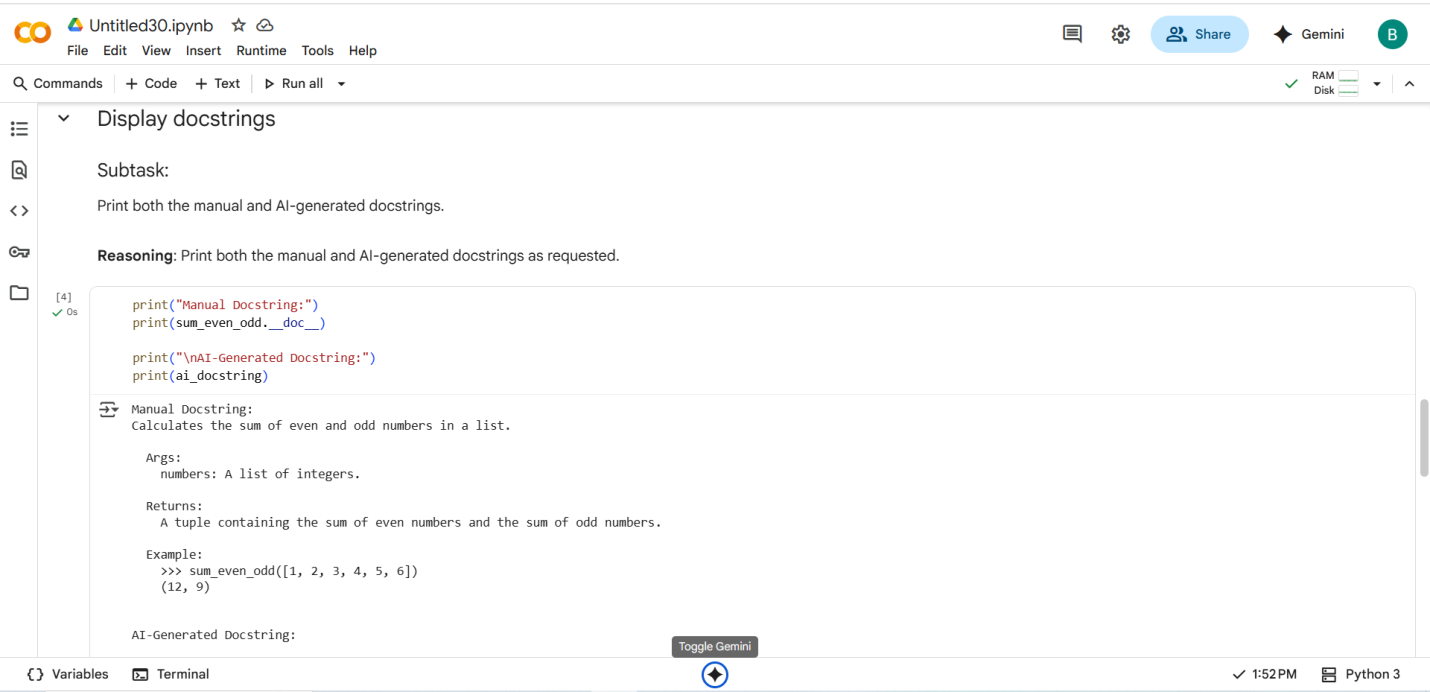
#CODE with OUTPUT:

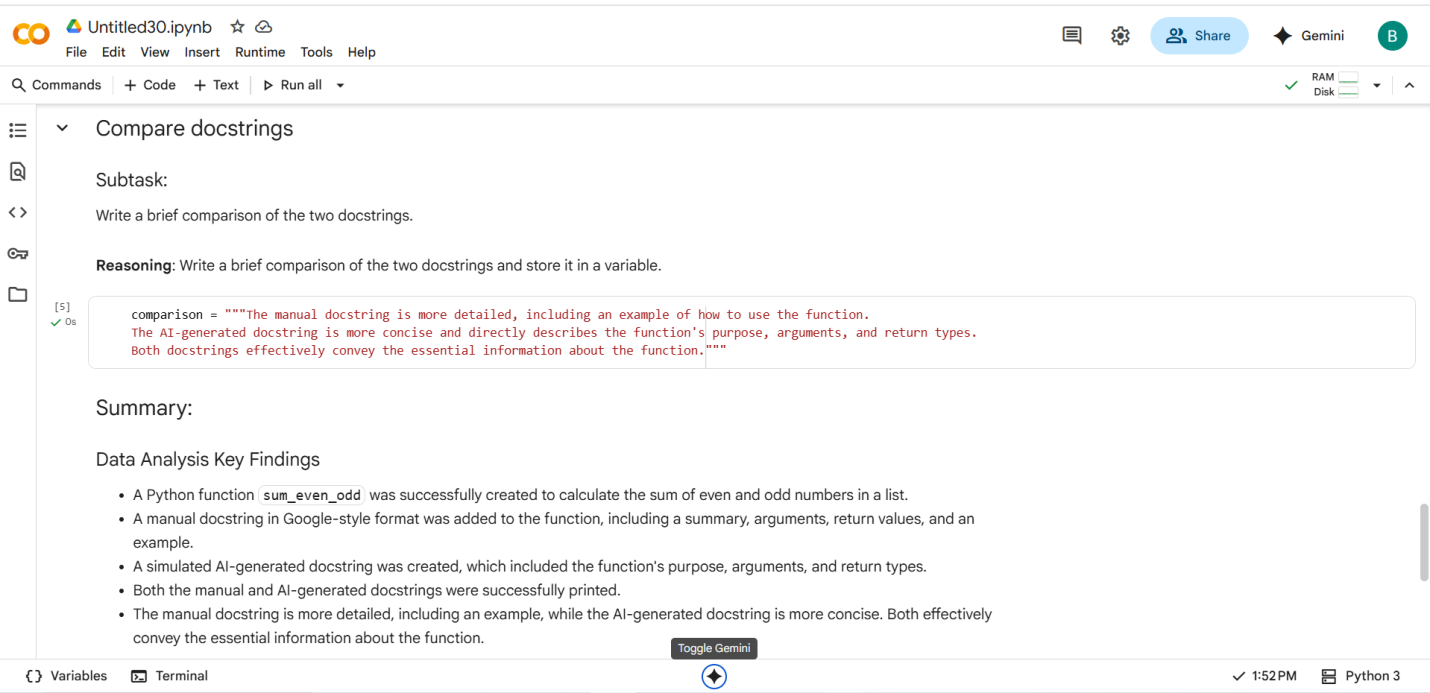








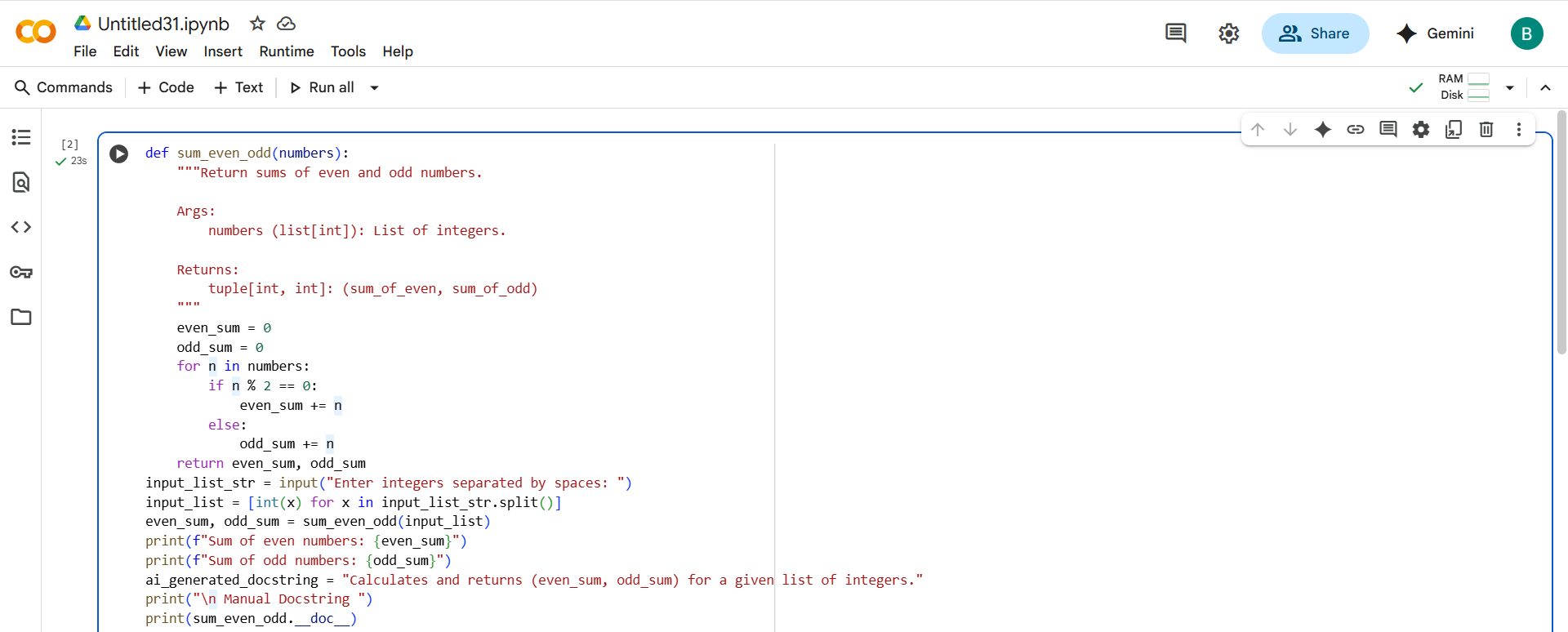




#EXPLANATION:

1. **sum\_even\_odd function:** This function is designed to go through a list of numbers you give it. It keeps track of two separate totals: one for all the even numbers it finds and one for all the odd numbers. It checks each number in the list to see if it's even or odd (by seeing if there's a remainder when you divide by 2). It adds the number to the correct total. Finally, it gives you back both totals.
2. **Getting User Input and Using the Function:** This part of the code asks you to type in a list of numbers, separated by spaces. It then takes those numbers, turns them into a list that the sum\_even\_odd function can understand, and then uses the function to get the even and odd sums. Finally, it prints out the results so you can see the sum of the even numbers and the sum of the odd numbers from your input.
3. **Docstrings (Manual and AI-Generated):** These are explanations written within the code itself to describe what the sum\_even\_odd function does.
   * The **manual docstring** is written by a person and provides details like what the function takes as input (arguments), what it gives back (returns), and even an example of how to use it.
   * The **AI-generated docstring** is simulated here to show what an AI might produce. It's usually more concise, focusing on the main purpose, arguments, and return types.
4. **Displaying Docstrings:** This code simply prints out the contents of both the manual and the simulated AI docstrings so you can read them.
5. **Comparison:** This is a short text that points out the main differences between the manual and AI-generated docstrings, like how one might be more detailed with examples while the other is more to-the-point.

#Manual code with output





#comparing AI Code and My code

* My code is better than AI because my code is clean and clear explanation in one cell.

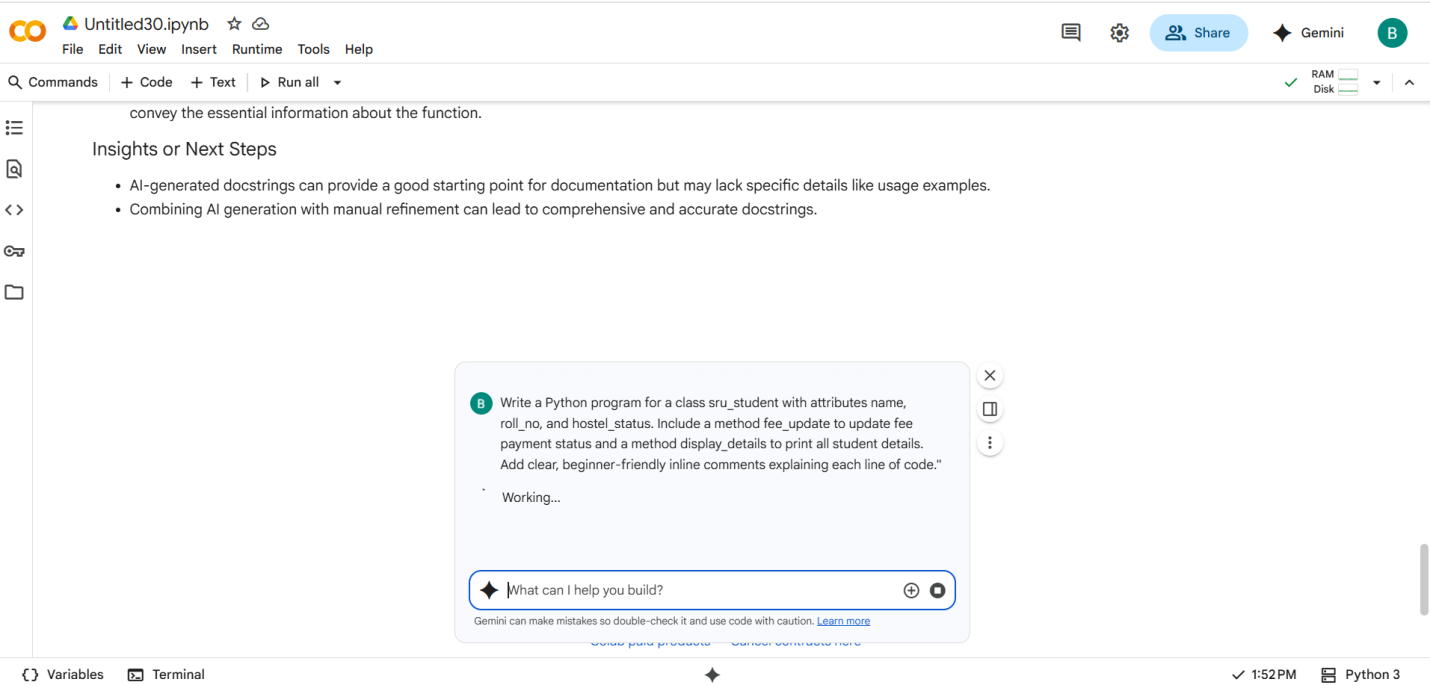
#TASK DESCRIPTION-2:

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

#PROMPT:

Write a Python program for a class sru\_student with attributes name, roll\_no, and hostel\_status. Include a method fee\_update to update fee payment status and a method display\_details to print all student details. Add clear, beginner-friendly inline comments explaining each line of code

#QUESTION:



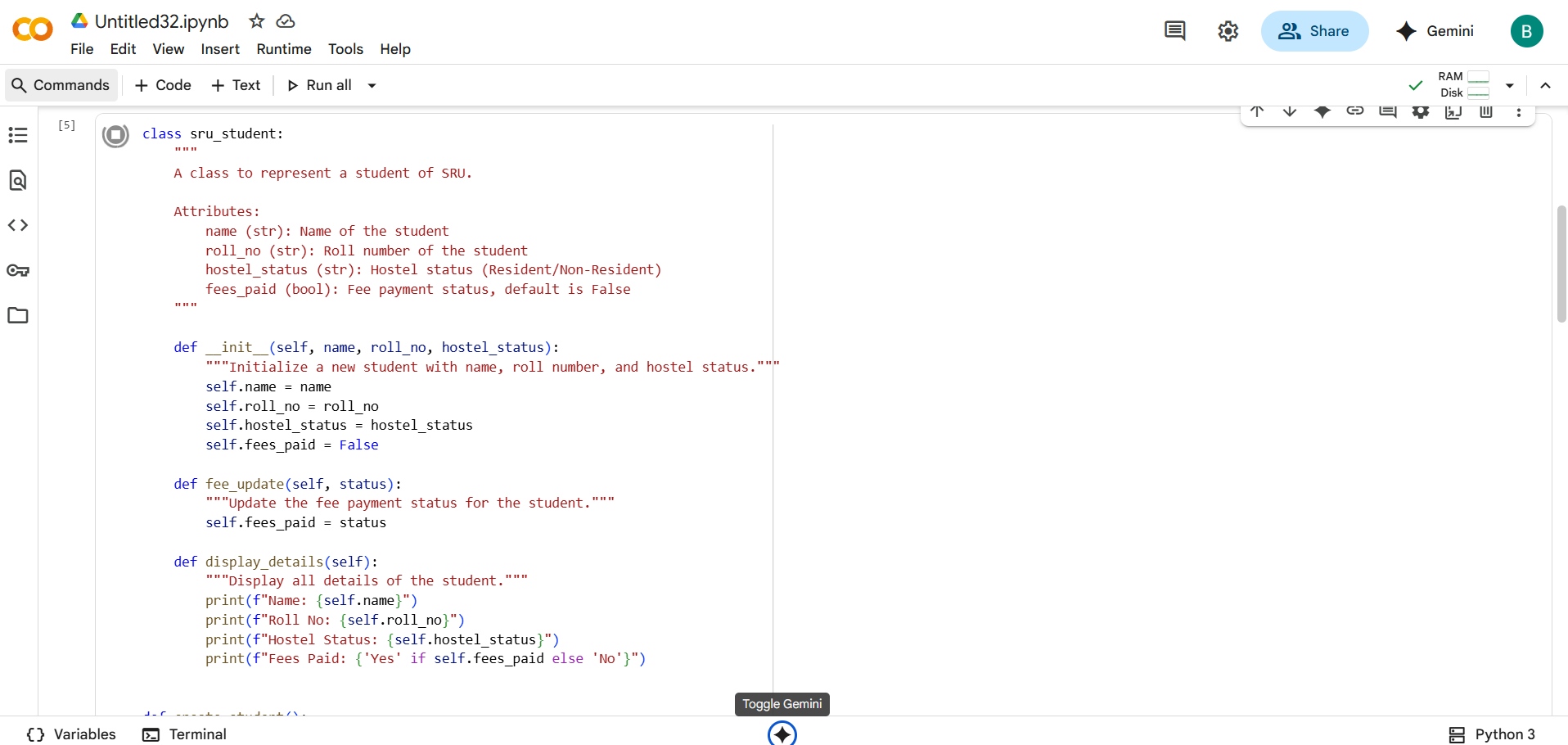
#CODE with OUTPUT:

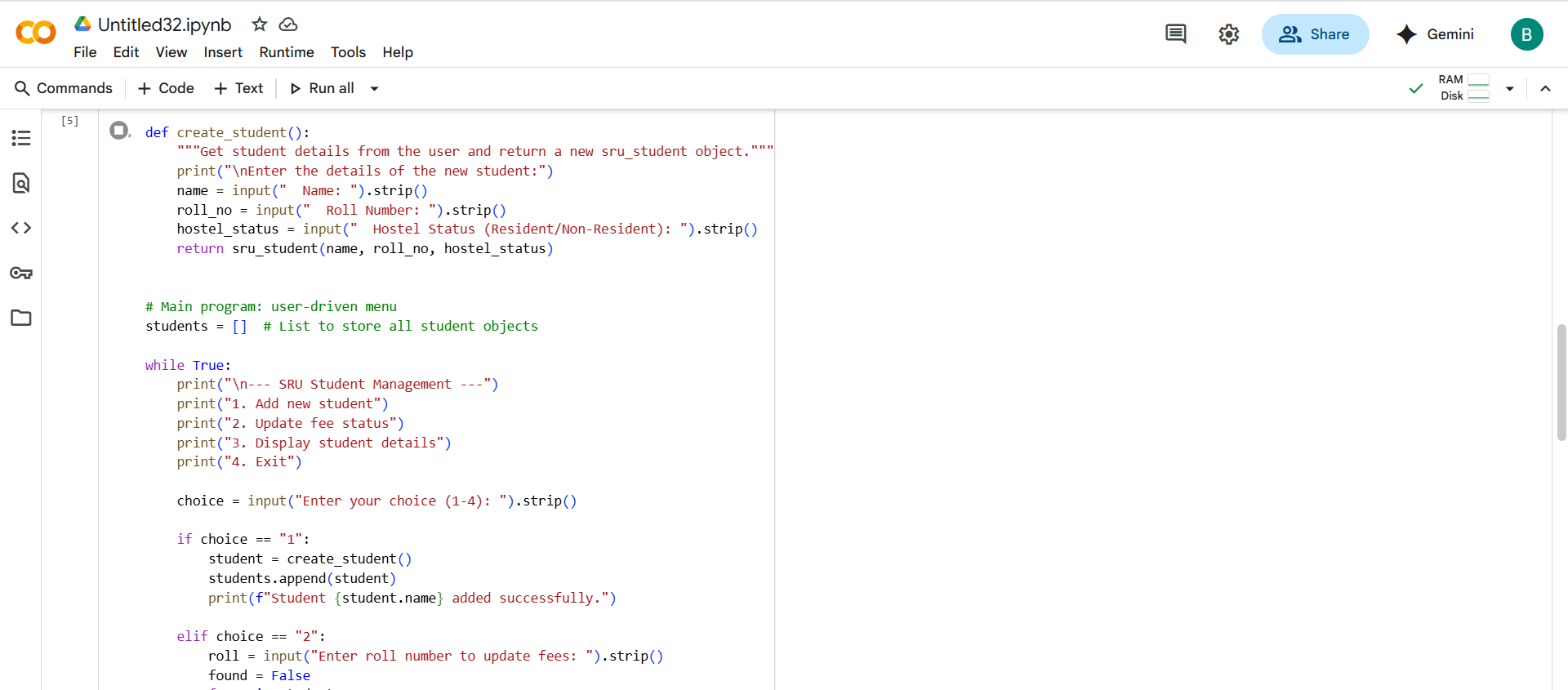


#EXPLANATION:

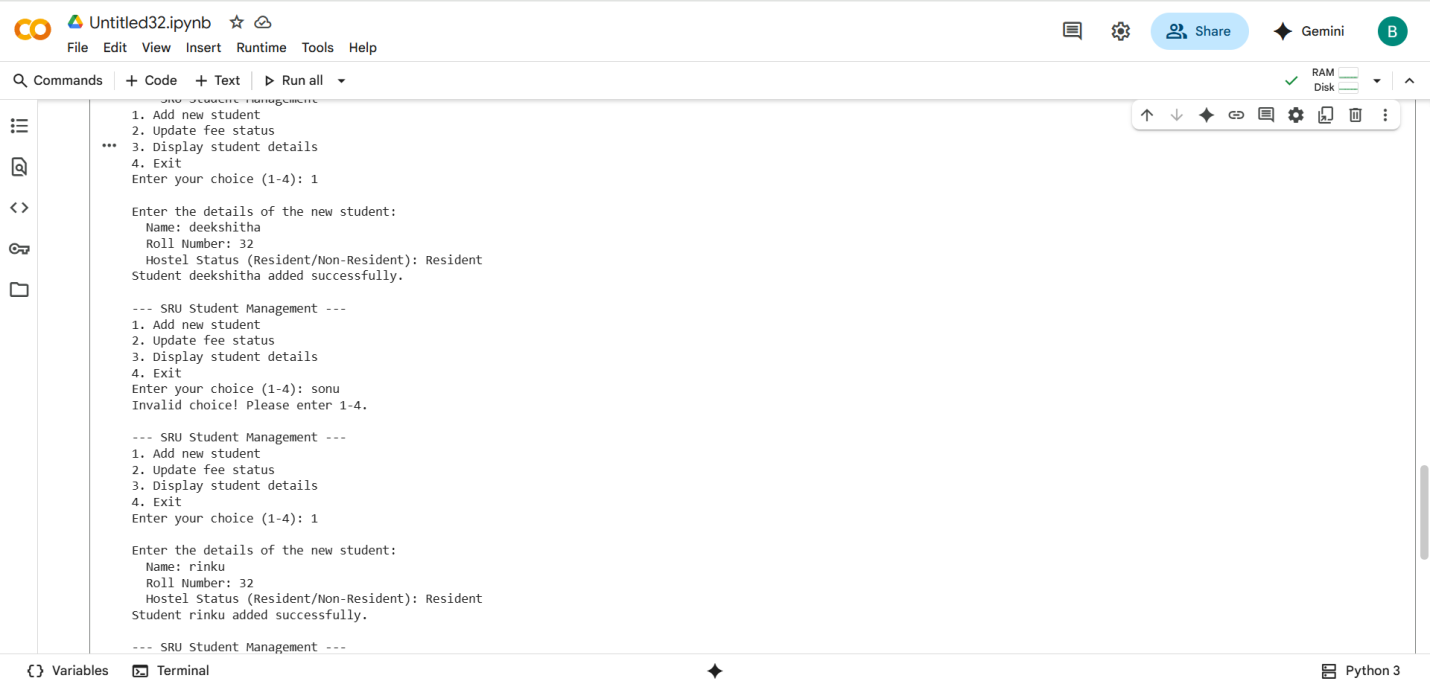
* **class sru\_student:**: This defines a blueprint for creating sru\_student objects.
* **\_\_init\_\_(self, name, roll\_no, hostel\_status):**: This is the constructor method. It's called when you create a new sru\_student object. It initializes the name, roll\_no, and hostel\_status attributes based on the values you provide and sets the fees\_paid attribute to False by default.
* **fee\_update(self, status):**: This method is used to update the fees\_paid status of a student. You pass True if fees are paid and False otherwise.
* **display\_details(self):**: This method prints out the details of a student object in a formatted way.
* **create\_student():**: This function prompts the user to enter details for a new student and returns a new sru\_student object.
* **students = []**: This initializes an empty list to store all the sru\_student objects created.
* **while True:**: This starts an infinite loop to keep the menu running until the user chooses to exit.
* **The menu options (1-4):** These provide choices to the user for adding a new student, updating fees, displaying details, or exiting the program.
* **if/elif/else block:** This handles the user's choice and calls the appropriate functions or methods to perform the requested action.
* **input()**: This function is used to get input from the user.
* **.strip()**: This method is used to remove leading and trailing whitespace from the user's input.
* **.lower()**: This method is used to convert the input to lowercase.
* **students.append(student)**: This adds a newly created student object to the students list.
* **for s in students:**: This loop iterates through the students list to find a student based on their roll number.
* **break**: This statement is used to exit the for loop once the student is found.

#MANUAL CODE with OUTPUT:









#Comparing AI code and My code:

* AI generated best code but comparing manual code and AI code some better steps better explanation compare to AI.

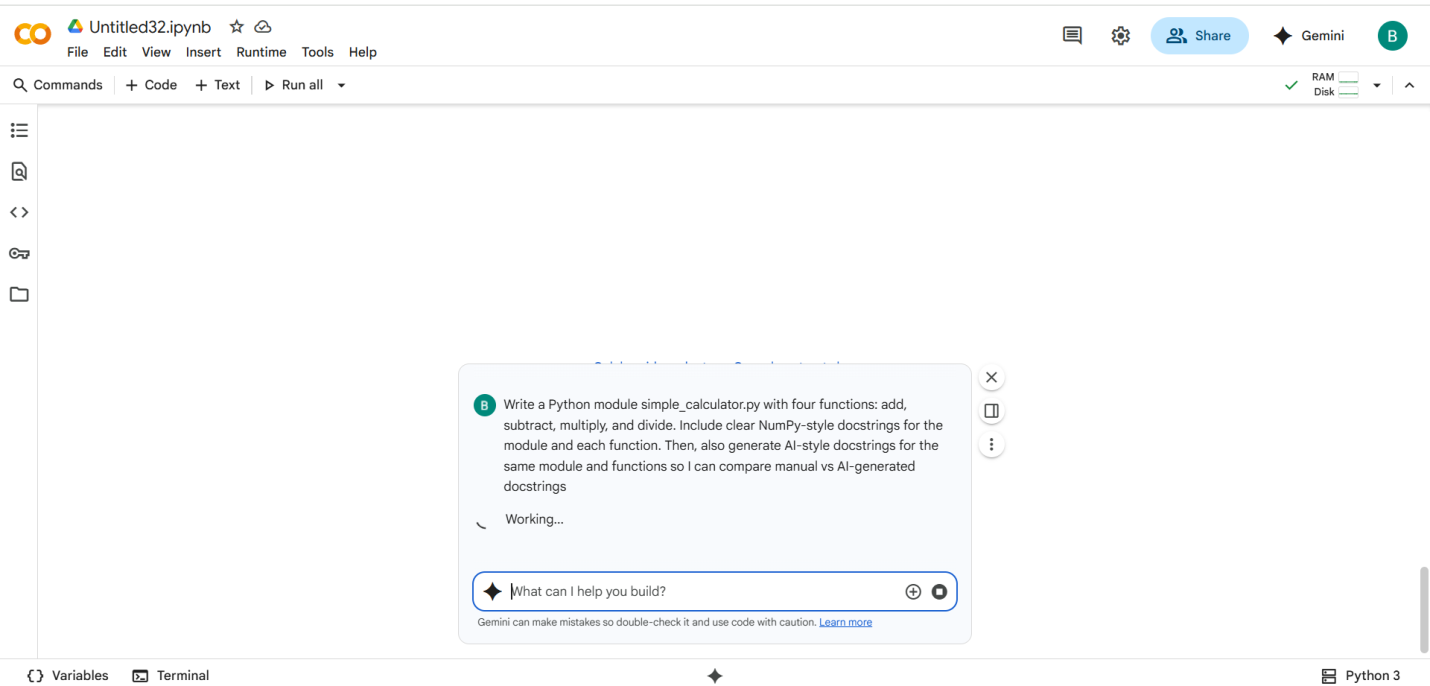
#TASK DESCRIPTION-3:

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

#PROMPT:

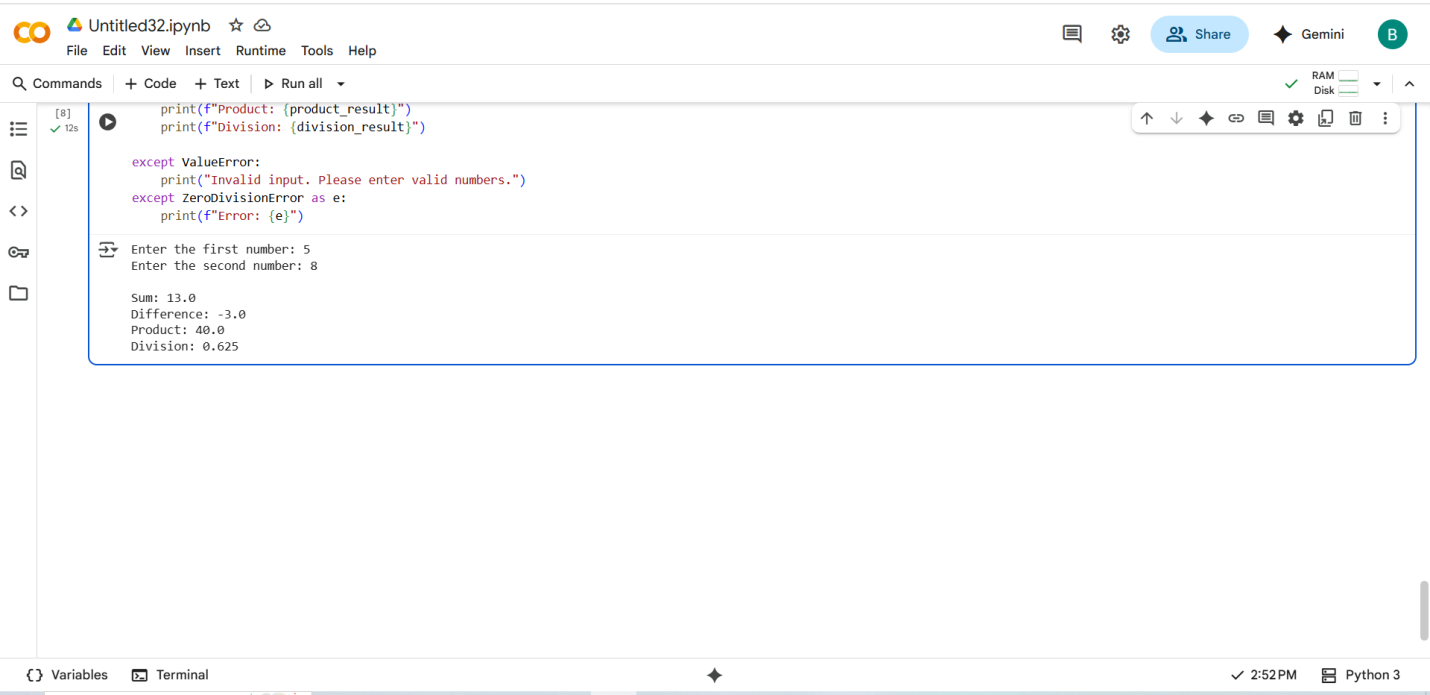
Write a Python module simple\_calculator.py with four functions: add, subtract, multiply, and divide. Include clear NumPy-style docstrings for the module and each function. Then, also generate AI-style docstrings for the same module and functions so I can compare manual vs AI-generated docstrings

#QUESTION:



#CODE with OUTPUT:



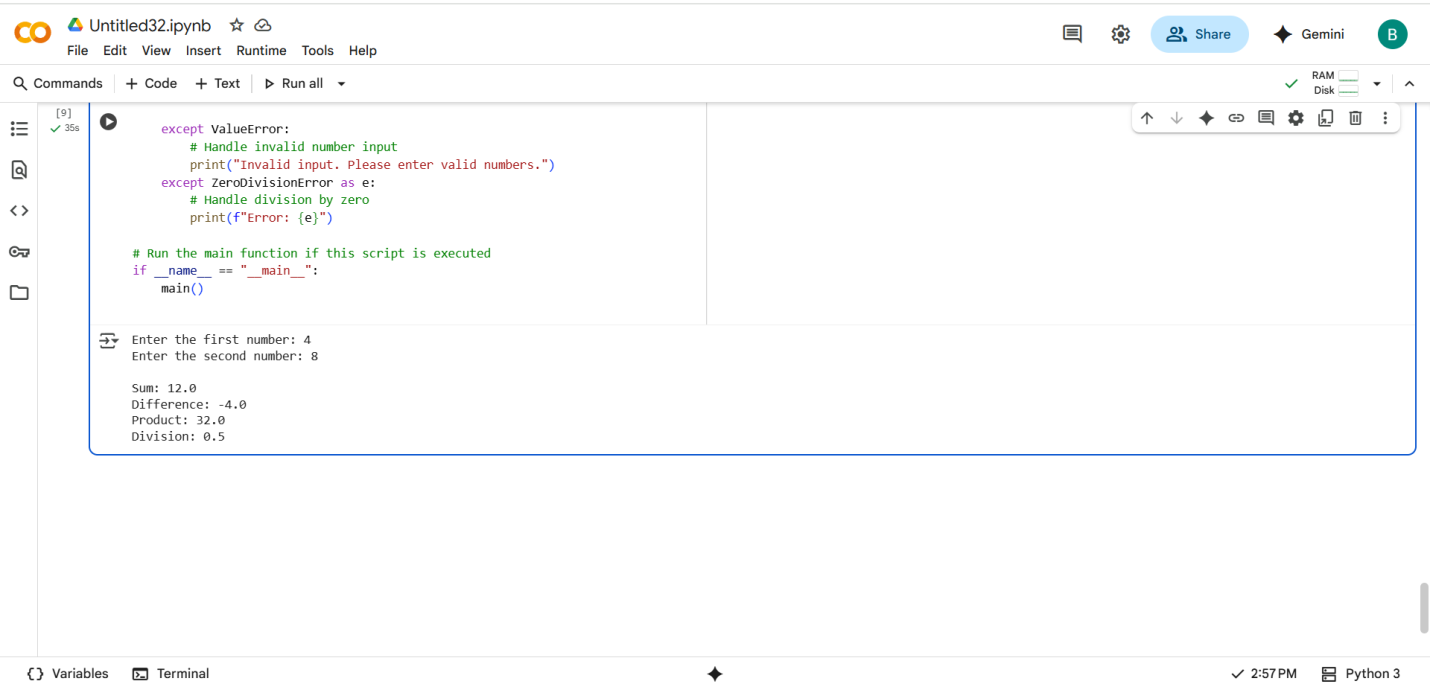


#EXPLANATION:

* **import simple\_calculator as sc**: This line imports the simple\_calculator module and gives it a shorter alias sc. This allows you to refer to the functions within the module using sc.function\_name instead of the full module name.
* **try:**: This block starts a try...except block, which is used for error handling. Code within the try block is executed, and if an error occurs, the code within the corresponding except block is executed.
* **num1 = float(input("Enter the first number: "))**: This line prompts the user to enter the first number using the input() function. The input is initially a string, so float() is used to convert it to a floating-point number, allowing for decimal values.
* **num2 = float(input("Enter the second number: "))**: Similar to the previous line, this prompts the user for the second number and converts it to a float.
* **sum\_result = sc.add(num1, num2)**: This line calls the add() function from the imported simple\_calculator module (using the alias sc) with num1 and num2 as arguments. The result of the addition is stored in the sum\_result variable.
* **difference\_result = sc.subtract(num1, num2)**: This line calls the subtract() function from the simple\_calculator module with num1 and num2. The result is stored in difference\_result.
* **product\_result = sc.multiply(num1, num2)**: This line calls the multiply() function from the simple\_calculator module with num1 and num2. The result is stored in product\_result.
* **division\_result = sc.divide(num1, num2)**: This line calls the divide() function from the simple\_calculator module with num1 and num2. The result is stored in division\_result.
* **print(f"\nSum: {sum\_result}")**: This line prints the calculated sum, using an f-string to embed the value of sum\_result within the output string. The \n creates a new line before the output.
* **print(f"Difference: {difference\_result}")**: This line prints the calculated difference.
* **print(f"Product: {product\_result}")**: This line prints the calculated product.
* **print(f"Division: {division\_result}")**: This line prints the calculated division result.
* **except ValueError:**: This block catches ValueError exceptions. A ValueError would occur if the user enters input that cannot be converted to a float (e.g., text).
* **print("Invalid input. Please enter valid numbers.")**: If a ValueError occurs, this message is printed to the user.
* **except ZeroDivisionError as e:**: This block catches ZeroDivisionError exceptions. A ZeroDivisionError occurs if the user tries to divide by zero. The as e assigns the error object to the variable e, which can then be used to print the specific error message.
* **print(f"Error: {e}")**: If a ZeroDivisionError occurs, this message, including the specific error details from the exception object e, is printed.

#MANUAL CODE with OUTPUT:





#Comparing Manual code and AI code:

* Both are same there is no difference both output and comments are similar.