**Program :**

B.tech(CSE)



**Specialization :**AIML

**Course Title :**AI Assisted Coding

**Course Code :**24CS002PC215

**Semester :**3rd semester

**Academic Session :**2025-2026

**Name of Student :**Kaveti Manohar

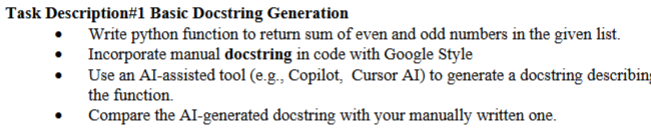
**Enrollment No. :**2403A52079

**Batch No. :**02

**Date :**24/09/2025

LAB ASSIGNMENT

Task Description -1:



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:

A screenshot of a computer

AI-generated content may be incorrect.

# AI -- CODE:

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.

A screenshot of a computer

AI-generated content may be incorrect.

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

#EXPLANANTION:

This code block prints a comparison between the manually written docstring and the simulated AI-generated docstring for the sum\_even\_odd function. It highlights the key differences, such as the inclusion of an 'Example' section in the manual docstring and the use of type hints in the AI-generated one, while noting that both accurately describe the function's purpose, arguments, and return values.

#MANUAL CODE:

A screenshot of a computer

AI-generated content may be incorrect.

#MANUAL OUTPUT:

A screenshot of a computer

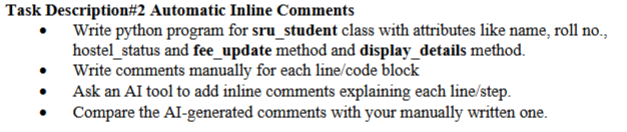
AI-generated content may be incorrect.

#COMPARISION OF BOTH AI AND MANUAL CODES:

"Manual docstring uses Google style (Args, Returns), AI one is shorter and in plain language."

This comparison highlights that the manual docstring follows the Google style guide, which includes explicit sections for arguments and return values. The AI-generated docstring, on the other hand, is shorter and uses a more plain language description.

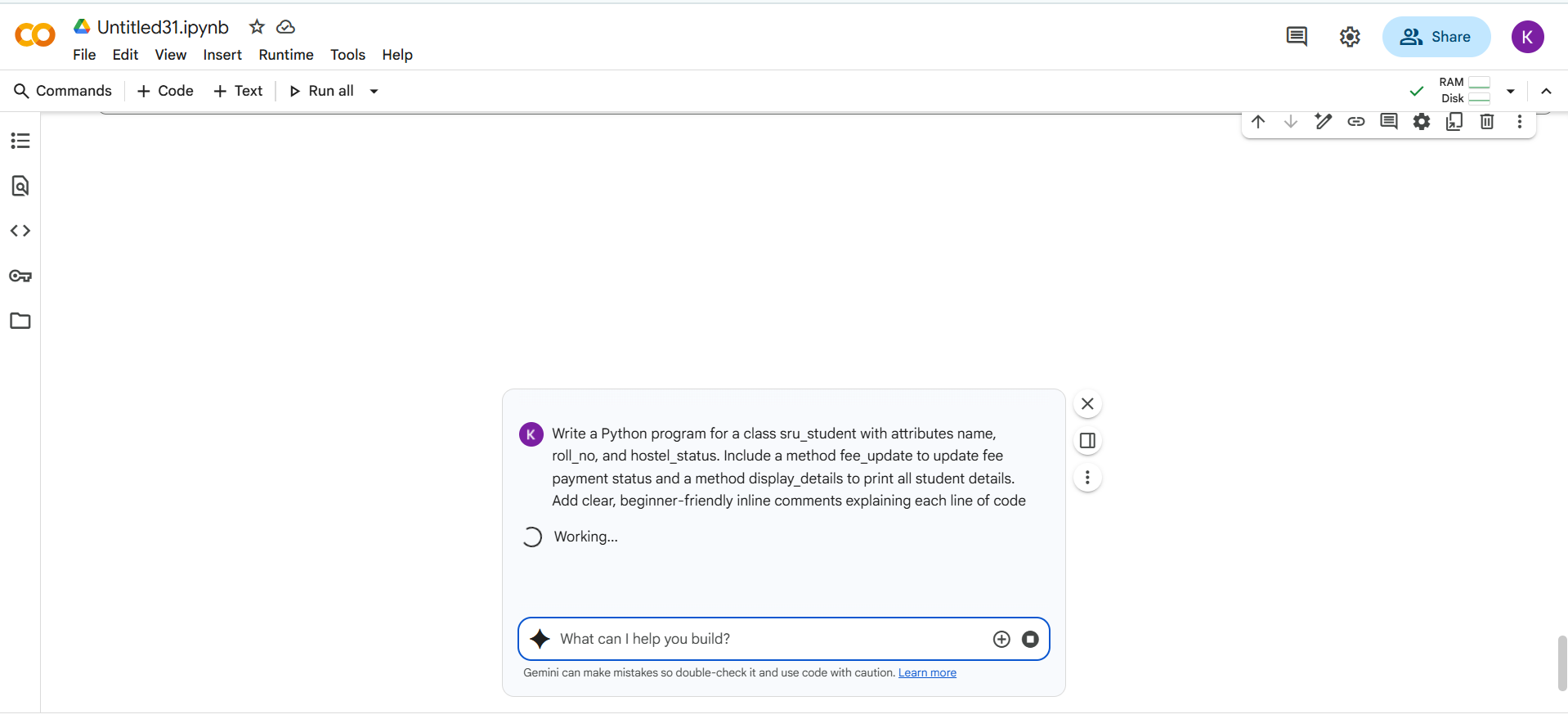
Task Description -2:



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:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

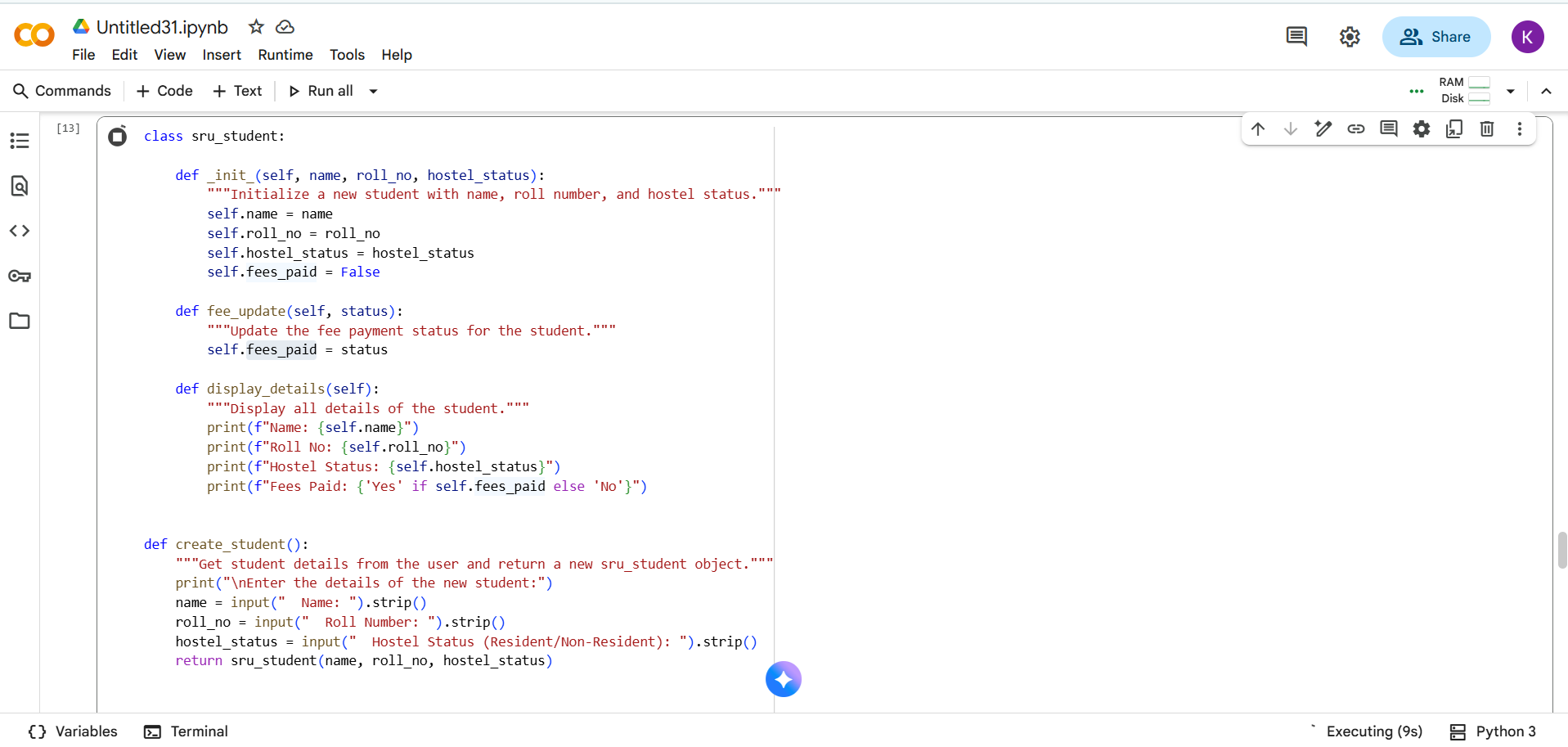
AI-generated content may be incorrect.

EXPLANATION:

This code defines a Python class named sru\_student to represent student information.

* The \_\_init\_\_ method is the constructor. It's called when you create a new sru\_student object. It takes the student's name, roll\_no, and hostel\_status as input and stores them as attributes of the object. It also initializes fee\_paid to False.
* The fee\_update method takes a paid\_status argument (which should be True or False) and updates the fee\_paid attribute of the student object accordingly.
* The display\_details method prints the name, roll\_no, hostel\_status, and fee\_paid status of the student object in a formatted way.

MANUAL CODE:



A screenshot of a computer

AI-generated content may be incorrect.

#OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

#COMPARISION OF AI CODE AND MANUAL CODE:

As I explained before, the main differences between the two code blocks are:

* **Errors:** The code in cell h-uxwiT0BPUV has errors in the \_\_init\_\_ method name and the fee\_update method syntax that prevent it from running. The code in cell b39d7fa2 is correct.
* **User Interface:** Cell h-uxwiT0BPUV includes a menu-driven user interface for managing multiple students, whereas cell b39d7fa2 just shows a basic example with one student.
* **Documentation Style:** Cell h-uxwiT0BPUV uses docstrings, while cell b39d7fa2 uses inline comments.

Task Description -3:

A white paper with black text

AI-generated content may be incorrect.

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:

A screenshot of a computer

AI-generated content may be incorrect.

#CODE WITH OUTPUT:

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

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:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

#Comparing Manual code and AI code:

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