PROGRAM : B.TECH/CSE

SPECIALIZATION : AIML

COURSE TITLE : AI ASSISTED CODING

COURSE CODE : 24CS101PC214

SEMESTER : 3RD

NAME OF THE STUDENT : SAINI.DEEPTHI

ENROLLMENT NO : 2403A52014

BATCH NO : 02

Task Description#1 Basic Docstring Generation

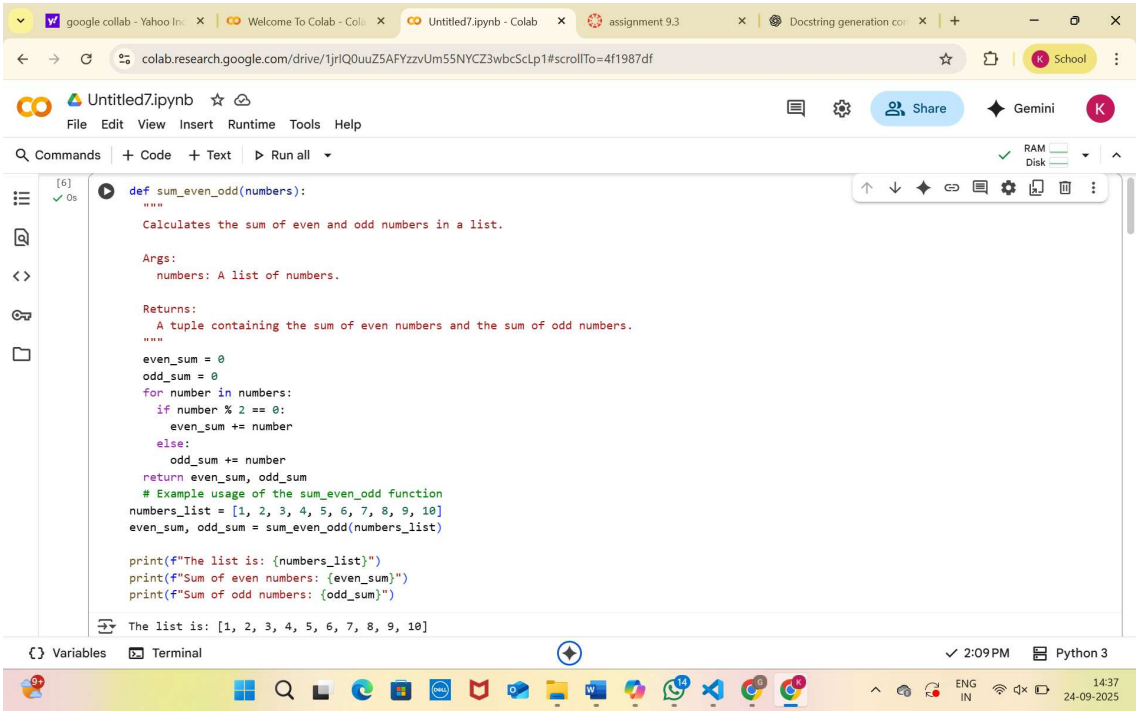
• Write python function to return sum of even and odd numbers in the given list.
• Incorporate manual docstring in code with Google Style
• Use an AI-assisted tool (e.g., Copilot, Cursor AI) to generate a docstring describing the function.
• Compare the AI-generated docstring with your manually written one.

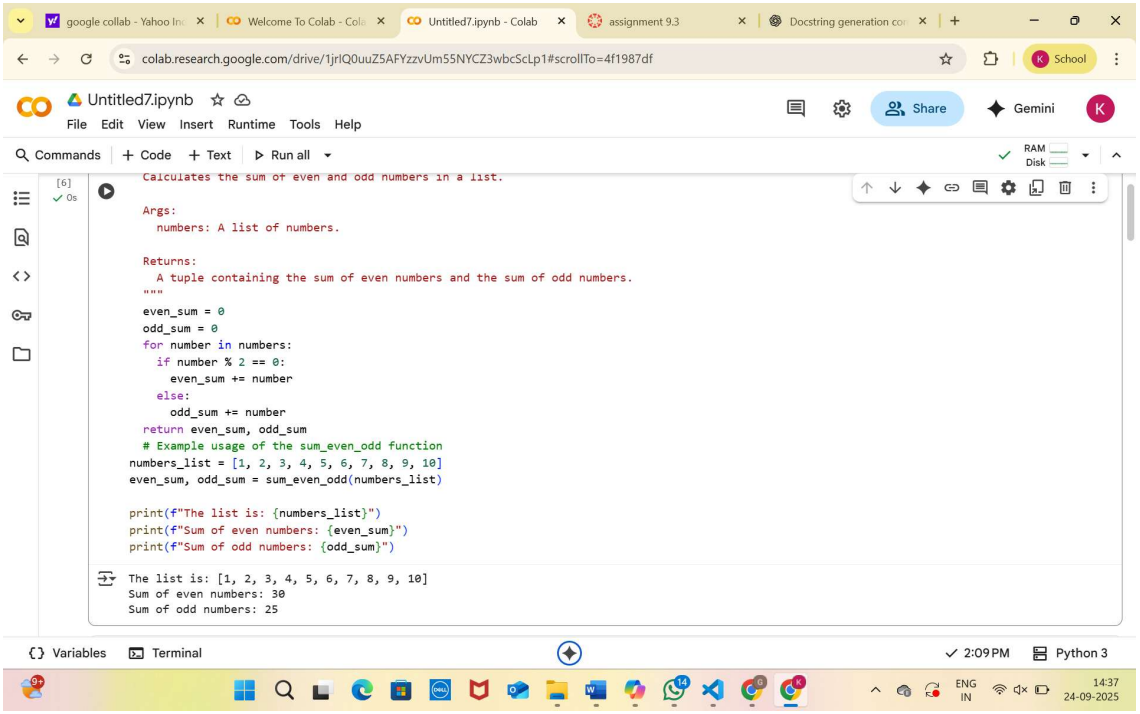
Expected Outcome#1: Students understand how AI can produce function-level documentation

Prompt:

Write a Python function that returns the sum of even and odd numbers in a given list. generate the docstring function automatically.

Code&Output:





Observations:

- This function efficiently separates and sums even and odd numbers.

- It iterates through the list once, making it performant.

- The use of a docstring clearly explains the function's purpose and parameters.

Task Description#2 Automatic Inline Comments

• 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.

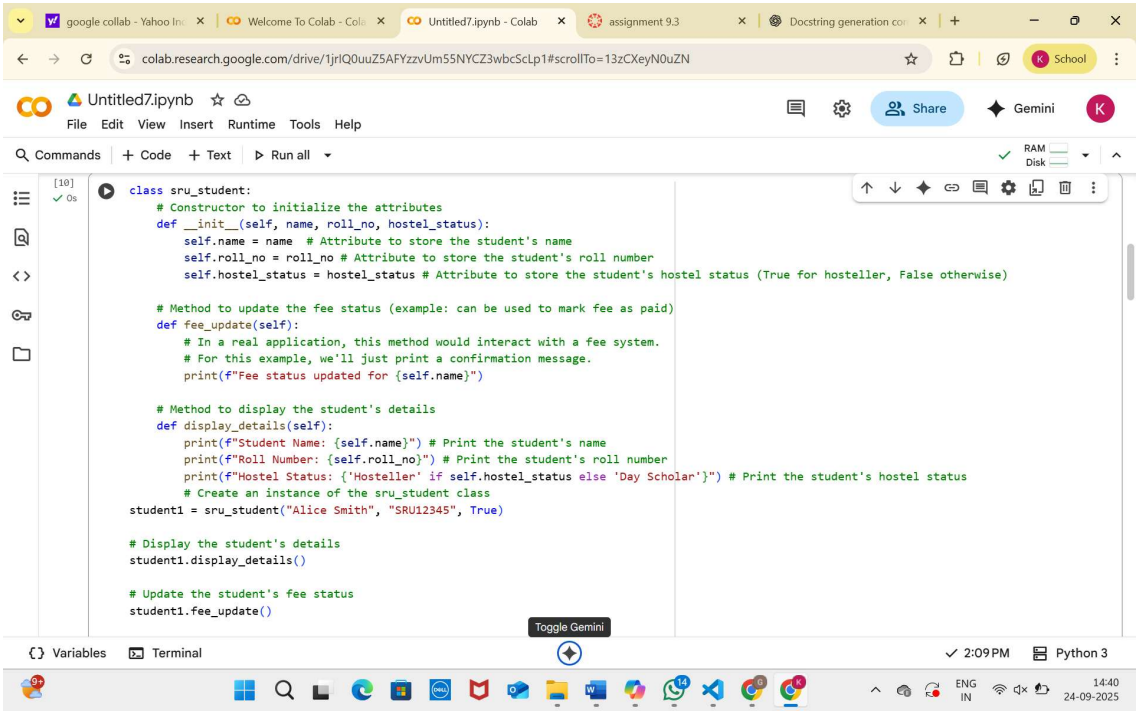
• Compare the AI-generated comments with your manually written one.

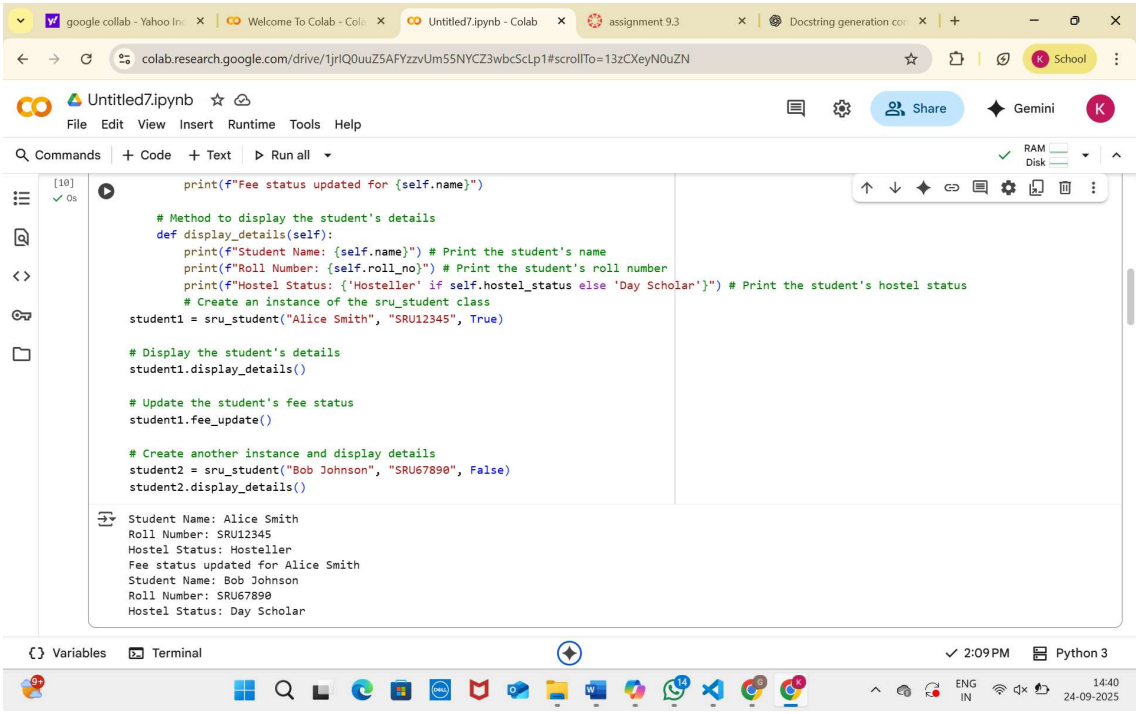
Expected Output#2: Students critically analyze AI-generated code comments

Prompt:

Write a Python class named sru_student with attributes (name, roll_no, hostel_status) and methods fee_update and display_details. automatically add inline comments explaining each line of code.

Code&Output:





Observations:

- Classes are useful for grouping data (attributes) and functionality (methods).

- The constructor (`__init__`) is used to set up the initial state of an object.

- Methods allow objects to perform actions or display information.

Task Description#3

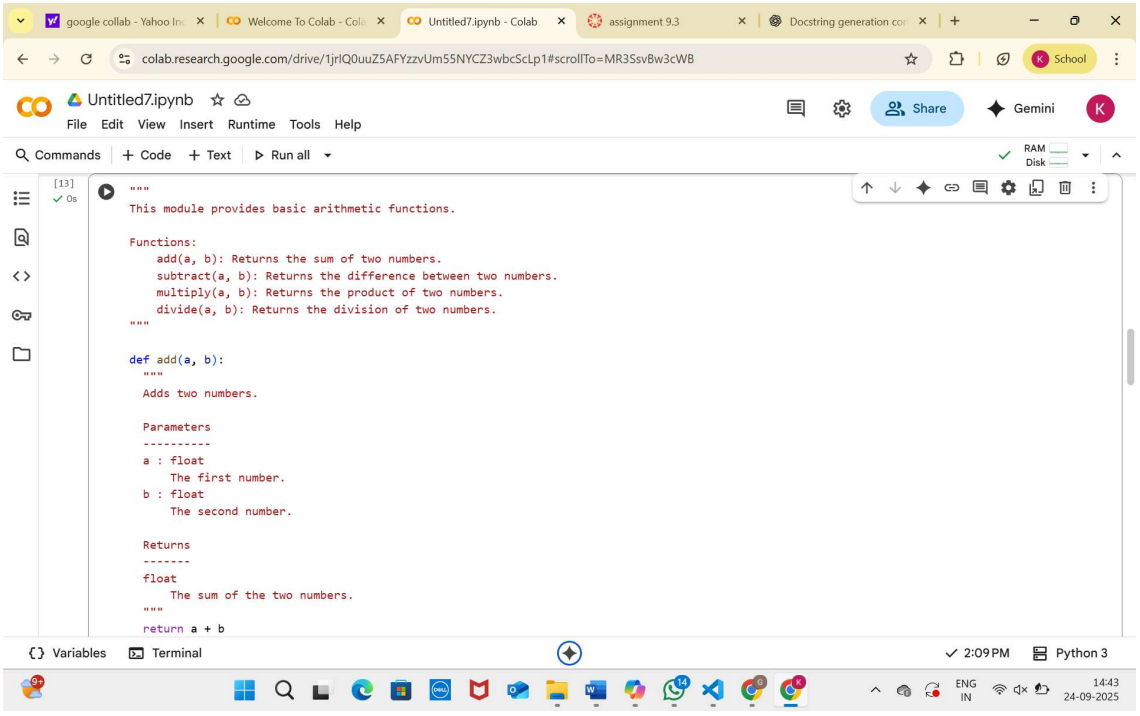
• Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).
• Incorporate manual docstring in code with NumPy Style
• Use AI assistance to generate a module-level docstring + individual function docstrings.
• Compare the AI-generated docstring with your manually written one.

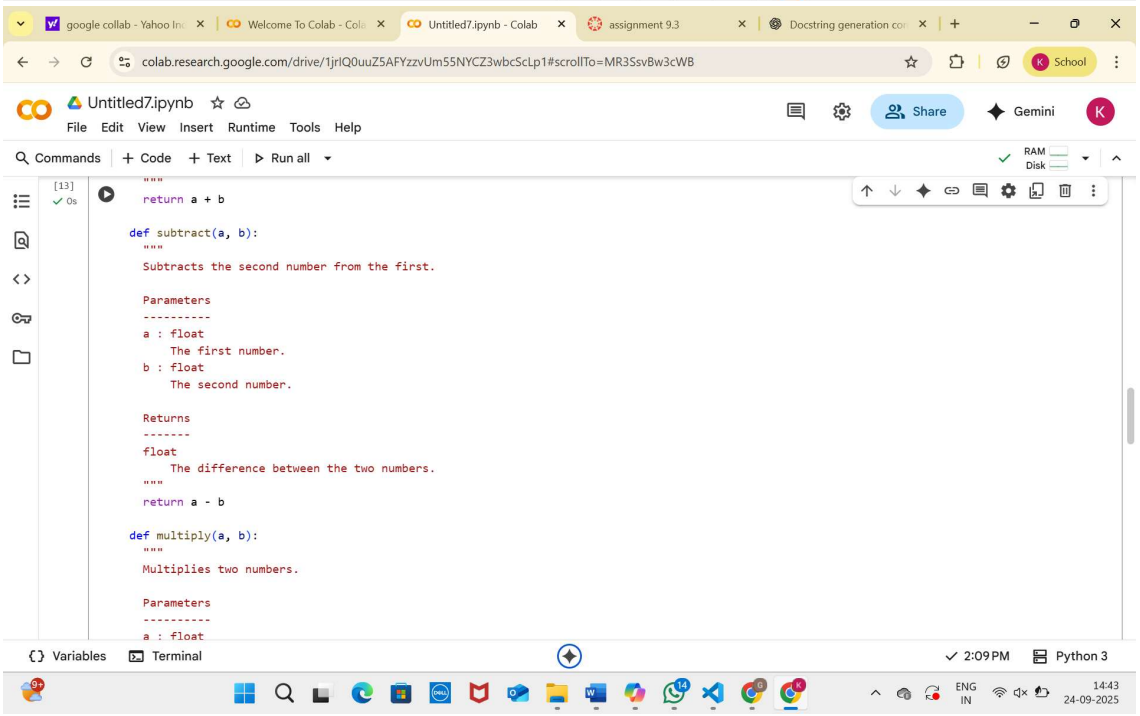
Expected Output#3: Students learn structured documentation for multi-function scripts

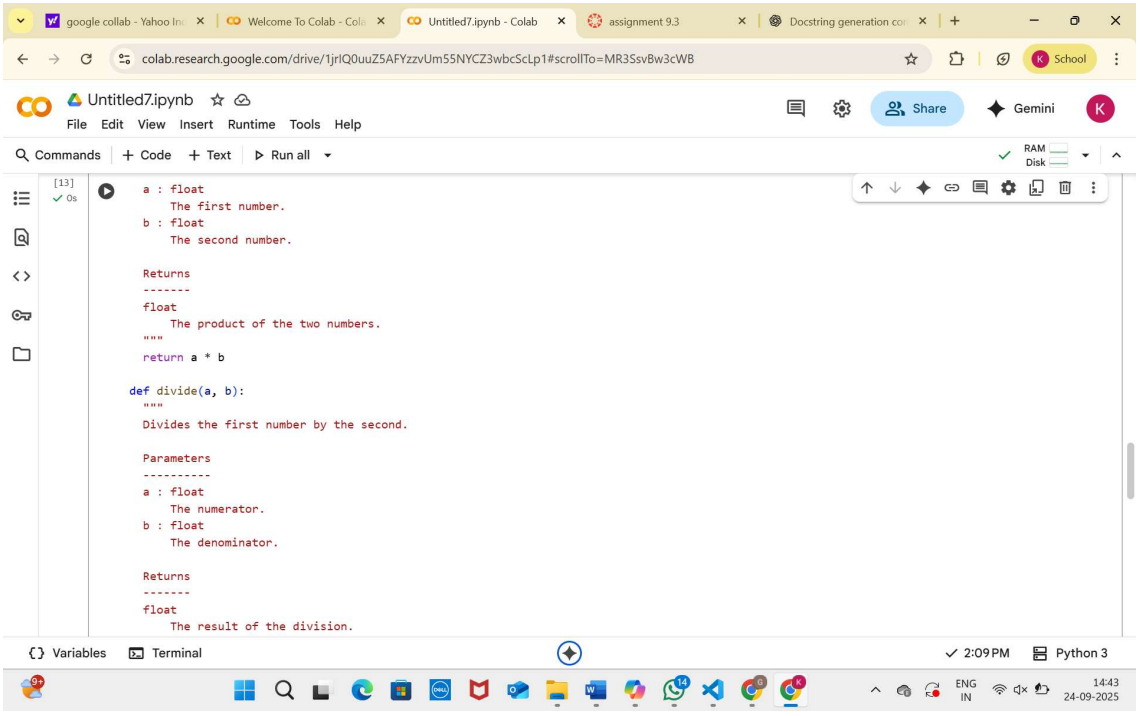
Prompt:

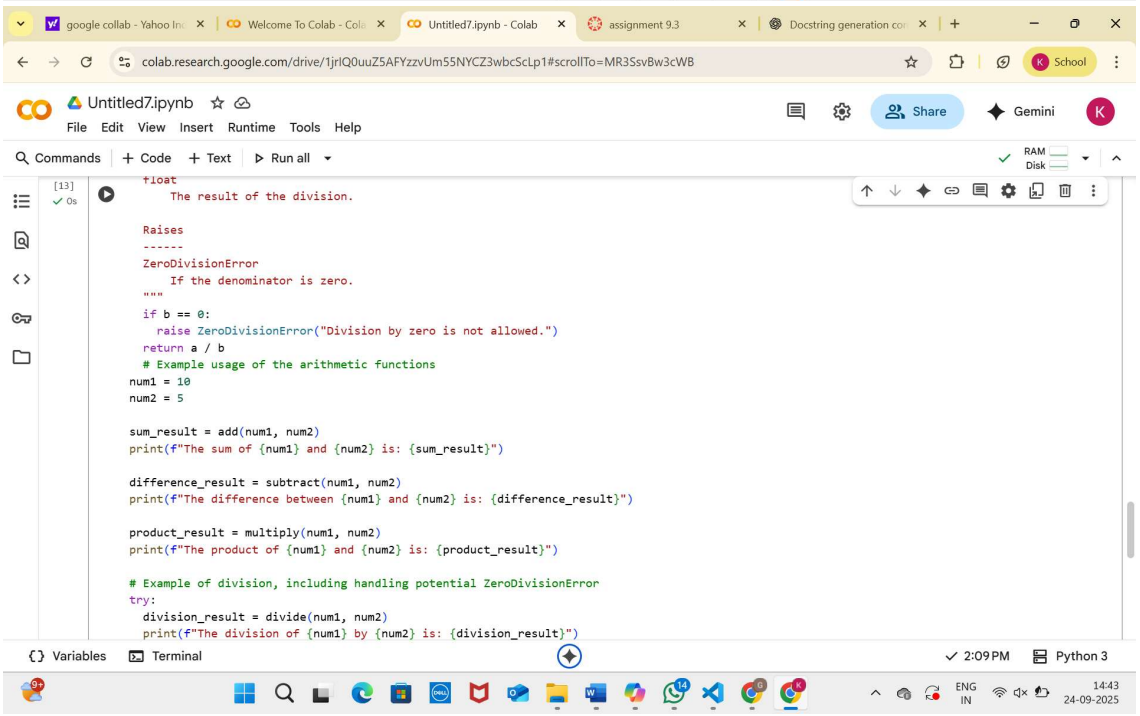
Write a Python script with 3–4 functions (e.g., add, subtract, multiply, divide) generate: A module-level docstring. Function-level docstrings. Expected Outcome: Students observe how AI creates structured documentation (NumPy style or similar) for a full script.

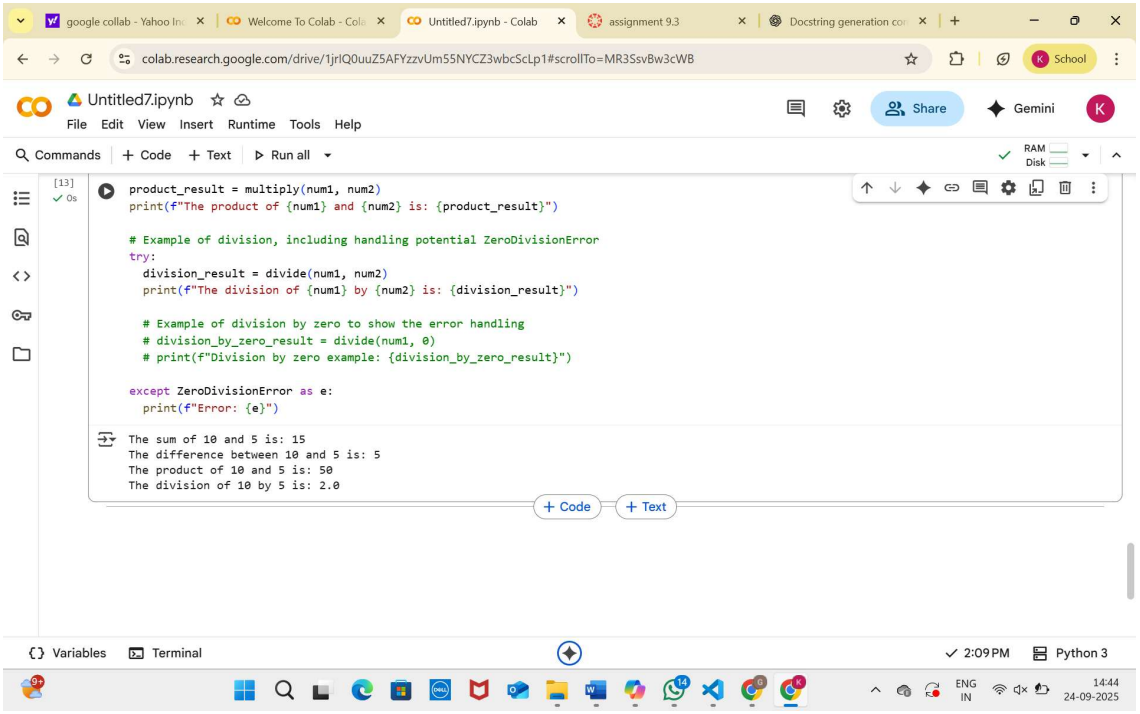
Code&Output:


google collab - Yahoo Inc
Welcome To Colab - Colab
Untitled7.ipynb - Colab
assignment 9.3
Docstring generation cor
colab.research.google.com/drive/1jrlQ0uuZ5AFYzzvUm55NYCZ3wbcScLp1#scrollTo=MR3SsvBw3cWB
School
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM
Disk
[13]
0s
"""
This module provides basic arithmetic functions.
Functions:
add(a, b): Returns the sum of two numbers.
subtract(a, b): Returns the difference between two numbers.
multiply(a, b): Returns the product of two numbers.
divide(a, b): Returns the division of two numbers.
"""
def add(a, b):
"""
Adds two numbers.
Parameters
----
a : float
The first number.
b : float
The second number.
Returns
----
float
The sum of the two numbers.
"""
return a + b
Variables
Terminal
2:09 PM
Python 3
ENG
IN
14:43
24-09-2025


google collab - Yahoo Inc
Welcome To Colab - Colab
Untitled7.ipynb - Colab
assignment 9.3
Docstring generation cor
colab.research.google.com/drive/1jrlQ0uuZ5AFYzzvUm55NYCZ3wbcScLp1#scrollTo=MR3SsvBw3cWB
School
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM
Disk
[13]
0s
"""
return a + b
def subtract(a, b):
"""
Subtracts the second number from the first.
Parameters
----
a : float
The first number.
b : float
The second number.
Returns
----
float
The difference between the two numbers.
"""
return a - b
def multiply(a, b):
"""
Multiplies two numbers.
Parameters
----
a : float
Variables
Terminal
2:09 PM
Python 3
ENG
IN
14:43
24-09-2025


google collab - Yahoo Inc
Welcome To Colab - Colab
Untitled7.ipynb - Colab
assignment 9.3
Docstring generation cor
colab.research.google.com/drive/1jrlQ0uuZ5AFYzzvUm55NYCZ3wbcScLp1#scrollTo=MR3SsvBw3cWB
School
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM
Disk
[13]
0s
a : float
The first number.
b : float
The second number.
Returns
float
The product of the two numbers.
return a * b
def divide(a, b):
Divides the first number by the second.
Parameters
a : float
The numerator.
b : float
The denominator.
Returns
float
The result of the division.
Variables
Terminal
2:09 PM
Python 3
ENG
IN
14:43
24-09-2025


google collab - Yahoo Inc
Welcome To Colab - Colab
Untitled7.ipynb - Colab
assignment 9.3
Docstring generation cor
colab.research.google.com/drive/1jrlQ0uuZ5AFYzzvUm55NYCZ3wbcScLp1#scrollTo=MR3SsvBw3cWB
School
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM
Disk
[13]
0s
float
The result of the division.
Raises
ZeroDivisionError
If the denominator is zero.
if b == 0:
raise ZeroDivisionError("Division by zero is not allowed.")
return a / b
# Example usage of the arithmetic functions
num1 = 10
num2 = 5
sum_result = add(num1, num2)
print(f"The sum of {num1} and {num2} is: {sum_result}")
difference_result = subtract(num1, num2)
print(f"The difference between {num1} and {num2} is: {difference_result}")
product_result = multiply(num1, num2)
print(f"The product of {num1} and {num2} is: {product_result}")
# Example of division, including handling potential ZeroDivisionError
try:
division_result = divide(num1, num2)
print(f"The division of {num1} by {num2} is: {division_result}")
Variables
Terminal
2:09 PM
Python 3
ENG
IN
14:43
24-09-2025



Observations:

- AI tools can effectively generate structured docstrings for Python code.

- Module-level docstrings provide a high-level overview of the script's purpose.

- Function-level docstrings detail the parameters, return values, and potential errors of each function.