

AI ASSISTED CODING

ENROLLMENT NUMBER : 2403A52019

Name of the Student : Peravena Navadeep

BATCH : B 02

ASSIGNMENT : 9.3

TASK DESCRIPTION:

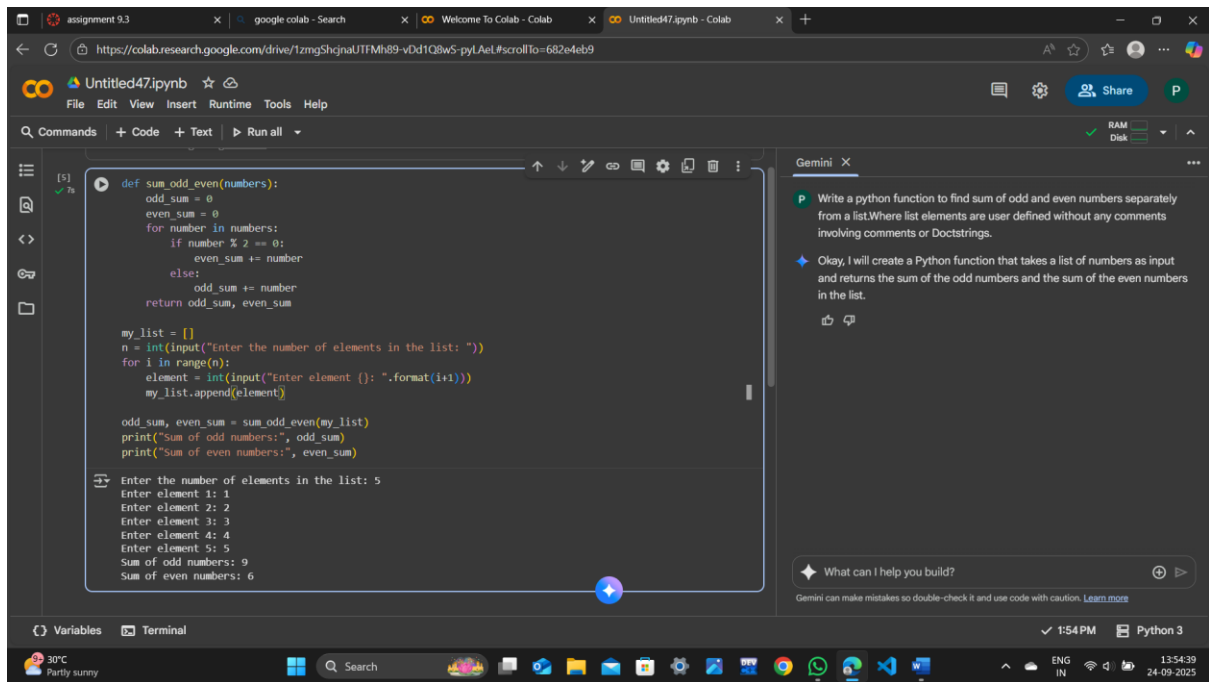
Write python function to return sum of even and odd numbers in the given list.

- 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.

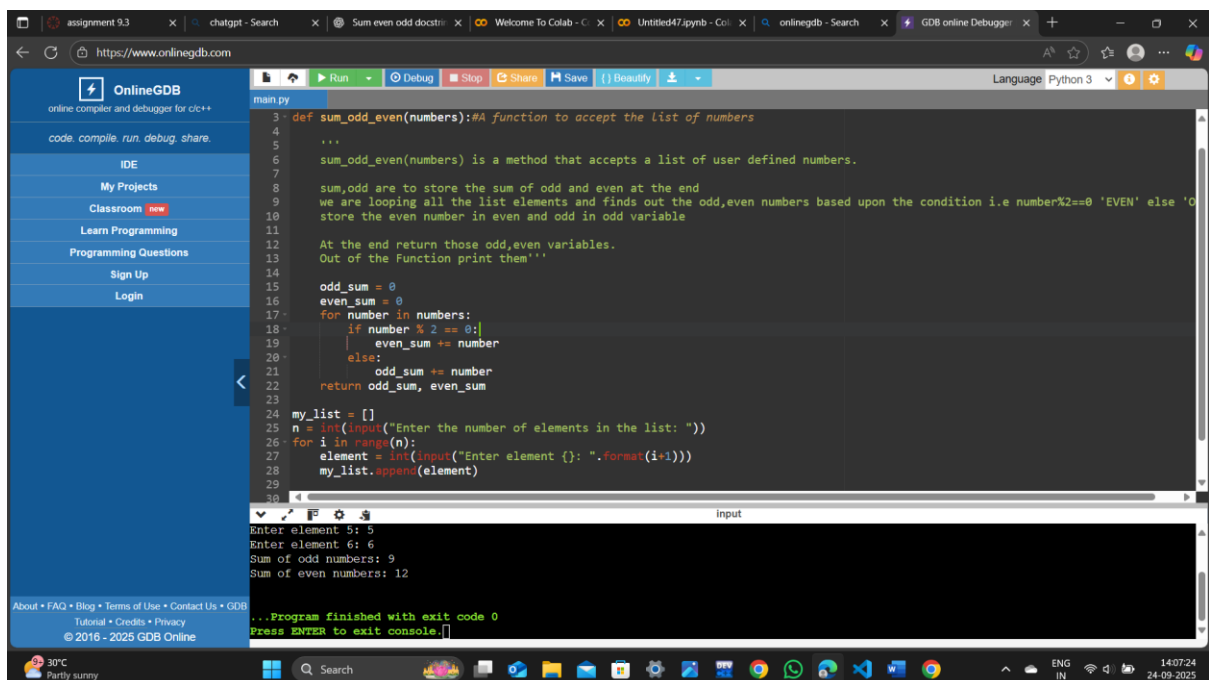
Prompt:

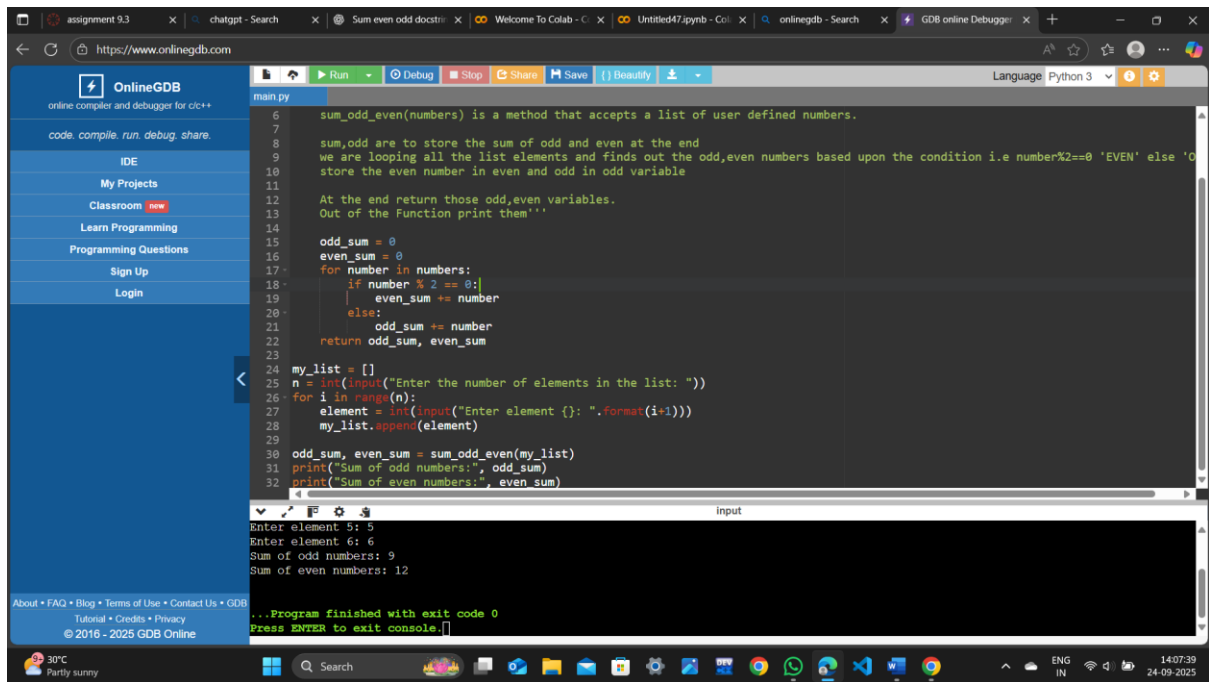
Write a python function to find sum of odd and even numbers separately from a list. Where list elements are user defined without any comments involving comments or Doctstrings.

CODE:



Incorporate manual docstring in code with Google Style





The screenshot shows the OnlineGDB web interface. The browser tabs include 'assignment 9.3', 'chatgpt - Search', 'Sum even odd docstr...', 'Welcome To Colab - C...', 'Untitled47.py:nb - Col...', 'onlinegdb - Search', and 'GDB online Debugger'. The address bar shows 'https://www.onlinegdb.com'. The left sidebar contains navigation links: 'OnlineGDB', 'code, compile, run, debug, share.', 'IDE', 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Sign Up', and 'Login'. The main editor area shows a Python file named 'main.py' with the following code:

```
6  sum_odd_even(numbers) is a method that accepts a list of user defined numbers.
7
8  sum,odd are to store the sum of odd and even at the end
9  we are looping all the list elements and finds out the odd,even numbers based upon the condition i.e number%2==0 'EVEN' else 'O
10 store the even number in even and odd in odd variable
11
12 At the end return those odd,even variables.
13 Out of the Function print them'''
14
15 odd_sum = 0
16 even_sum = 0
17 for number in numbers:
18     if number % 2 == 0:
19         even_sum += number
20     else:
21         odd_sum += number
22 return odd_sum, even_sum
23
24 my_list = []
25 n = int(input("Enter the number of elements in the list: "))
26 for i in range(n):
27     element = int(input("Enter element {}: ".format(i+1)))
28     my_list.append(element)
29
30 odd_sum, even_sum = sum_odd_even(my_list)
31 print("Sum of odd numbers:", odd_sum)
32 print("Sum of even numbers:", even_sum)
```

The output window shows the following input and output:

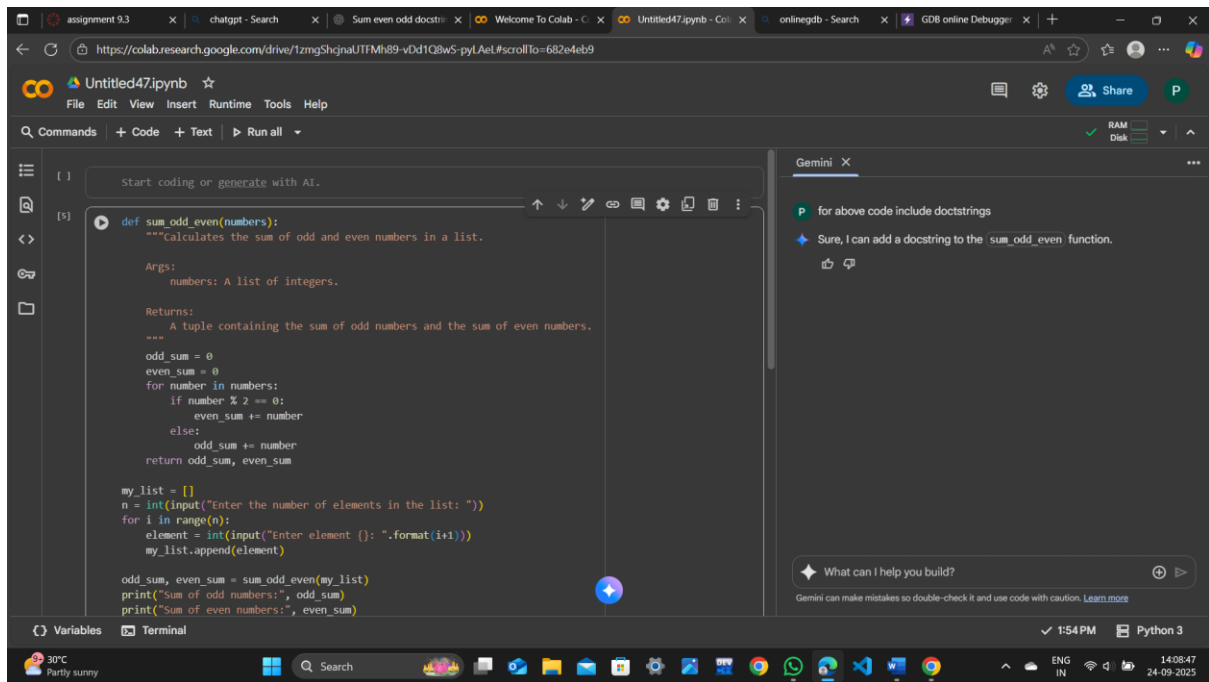
```
input
Enter element 5: 5
Enter element 6: 6
Sum of odd numbers: 9
Sum of even numbers: 12
...Program finished with exit code 0
Press ENTER to exit console.
```

The bottom status bar shows '30°C Partly sunny', a search bar, and system icons for language (ENG IN), network, and time (14:07:39 24-09-2025).

Use an AI-assisted tool (e.g., Copilot, Cursor AI) to generate a docstring describing the function.

PROMPT:

For the code that is been generated by You, Include docstrings in it



OBSERVATION:

Compare the AI-generated docstring with your manually written one.

On Comparison of the Docstrings of mine and AI tool i.e Gemini. Mine is better and understandable by beginners, where as gemini it is a little bit terminology is involved.

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.

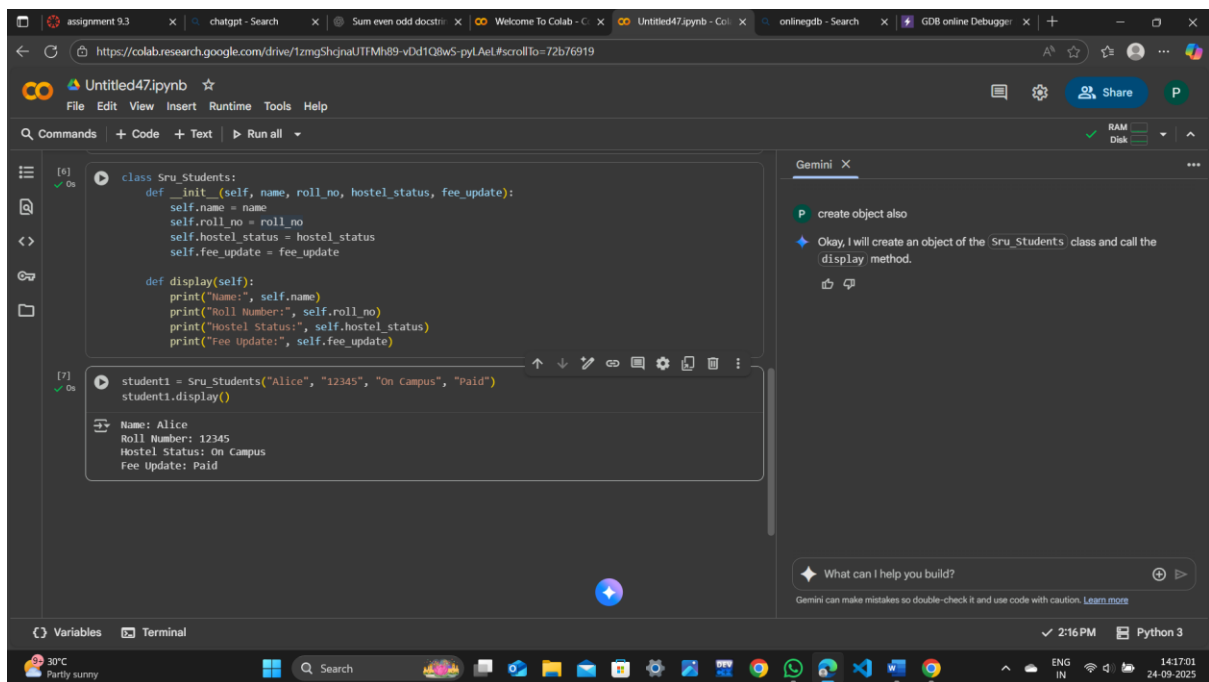
Expected Output#2: Students critically analyze AI-generated code comments.

Prompt:

Write a python class Sru_Students with attributes like name,roll_no,hostel_status,fee_update

and a method 'display' that displays information without any docstrings.

CODE:



The screenshot shows a Google Colab notebook interface. The main code cell contains the following Python code:

```
[6] class Sru_Students:
    def __init__(self, name, roll_no, hostel_status, fee_update):
        self.name = name
        self.roll_no = roll_no
        self.hostel_status = hostel_status
        self.fee_update = fee_update

    def display(self):
        print("Name:", self.name)
        print("Roll Number:", self.roll_no)
        print("Hostel Status:", self.hostel_status)
        print("Fee Update:", self.fee_update)

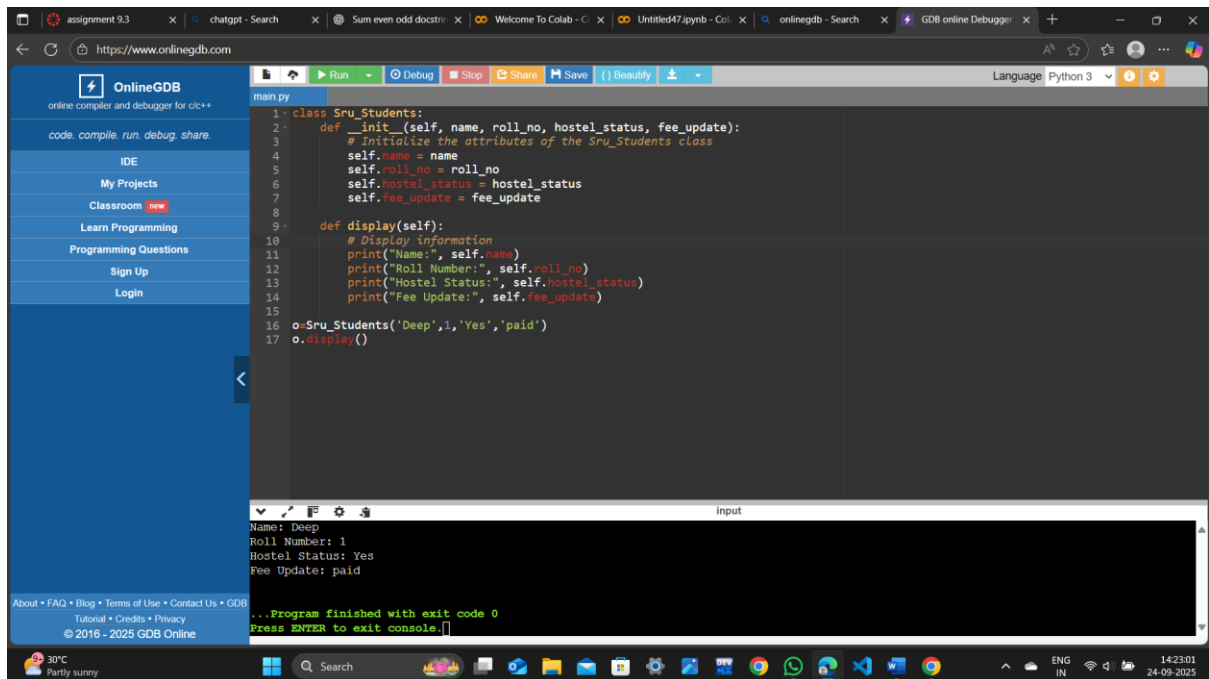
[7] student1 = Sru_Students("Alice", "12345", "On Campus", "Paid")
student1.display()
```

The output of the code execution is displayed below the code cell:

```
Name: Alice
Roll Number: 12345
Hostel Status: On Campus
Fee Update: Paid
```

On the right side of the notebook, there is a Gemini chat interface. The chat history shows a prompt "create object also" and a response from Gemini: "Okay, I will create an object of the 'Sru_Students' class and call the display method." The chat input field at the bottom contains the text "What can I help you build?".

Write comments manually for each line/code block



The screenshot shows the OnlineGDB website interface. The left sidebar contains navigation links: OnlineGDB, code, compile, run, debug, share, IDE, My Projects, Classroom (highlighted), Learn Programming, Programming Questions, Sign Up, and Login. The main editor area displays a Python program with the following code:

```
1: class Sru_Students:
2:     def __init__(self, name, roll_no, hostel_status, fee_update):
3:         # Initialize the attributes of the Sru_Students class
4:         self.name = name
5:         self.roll_no = roll_no
6:         self.hostel_status = hostel_status
7:         self.fee_update = fee_update
8:
9:     def display(self):
10:        # Display information
11:        print("Name:", self.name)
12:        print("Roll Number:", self.roll_no)
13:        print("Hostel Status:", self.hostel_status)
14:        print("Fee Update:", self.fee_update)
15:
16: o=Sru_Students('Deep',1,'Yes','paid')
17: o.display()
```

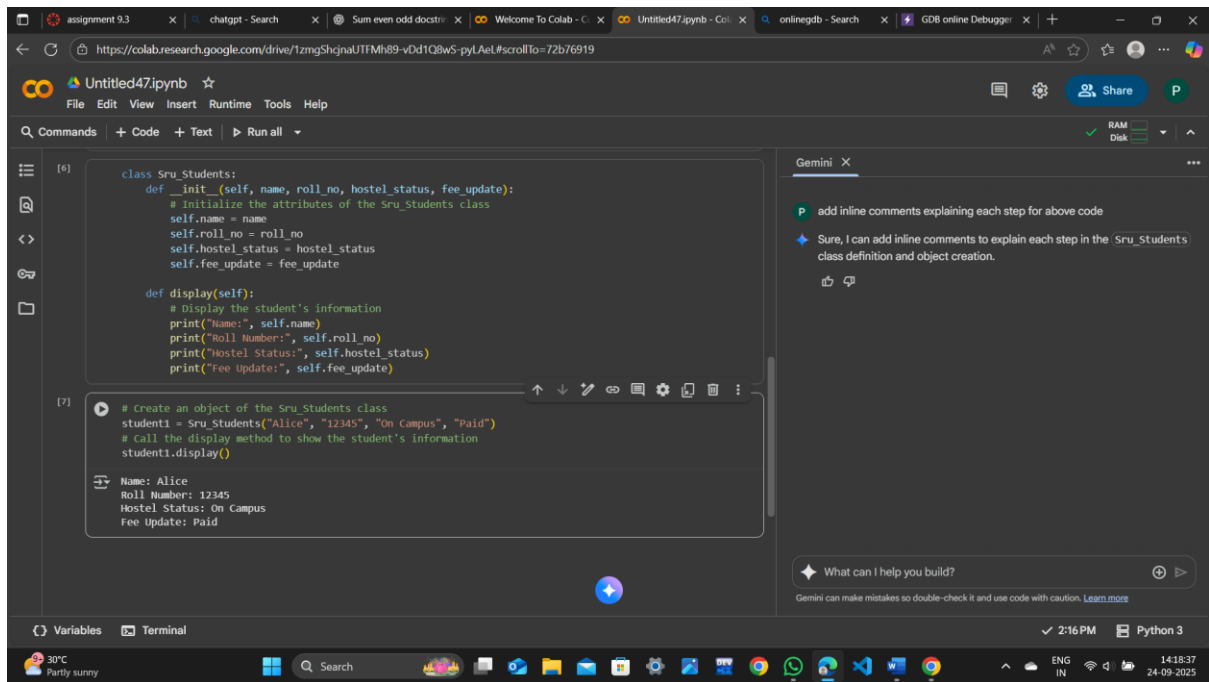
The output console at the bottom shows the following text:

```
Name: Deep
Roll Number: 1
Hostel Status: Yes
Fee Update: paid
...Program finished with exit code 0
Press ENTER to exit console.[]
```

Ask an AI tool to add inline comments explaining each line/step

PROMPT:

Add inline comments explaining each step for above code



OBSERVATION:

Compare the AI-generated comments with your manually written one

I have Analysed the inline comments that are given by me and Gemini. What I observed is Gemini inline comments are far better than my comments. Since it has involved class, Attributes in the code. So, it has better approach than me

TASK DESCRIPTION -3:

Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).

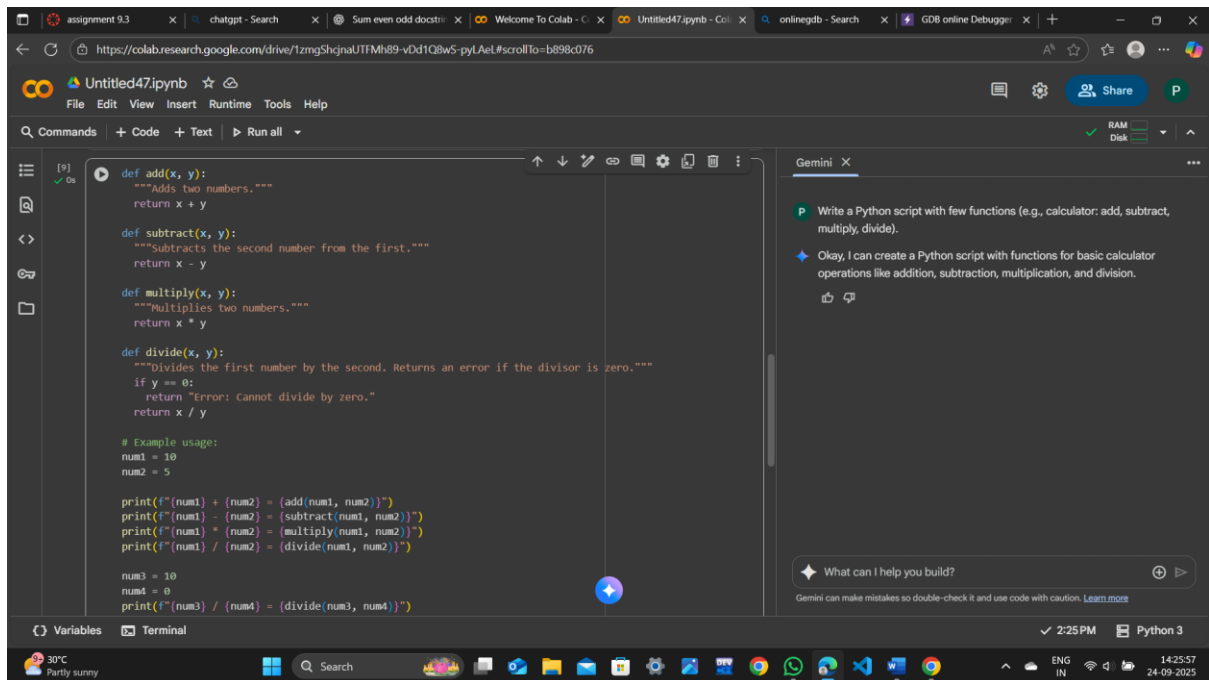
- Incorporate manual docstring in code with NumPy Style
- Use AI assistance to generate a module-level docstring + individual function docstrings.
- Compare the AI-generated docstring with your manually written one.

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

PROMPT:

Write a Python script with few functions (e.g., calculator: add, subtract, multiply,

divide).



```
[9] ✓ Os
def add(x, y):
    """Adds two numbers."""
    return x + y

def subtract(x, y):
    """Subtracts the second number from the first."""
    return x - y

def multiply(x, y):
    """Multiplies two numbers."""
    return x * y

def divide(x, y):
    """Divides the first number by the second. Returns an error if the divisor is zero."""
    if y == 0:
        return "Error: cannot divide by zero."
    return x / y

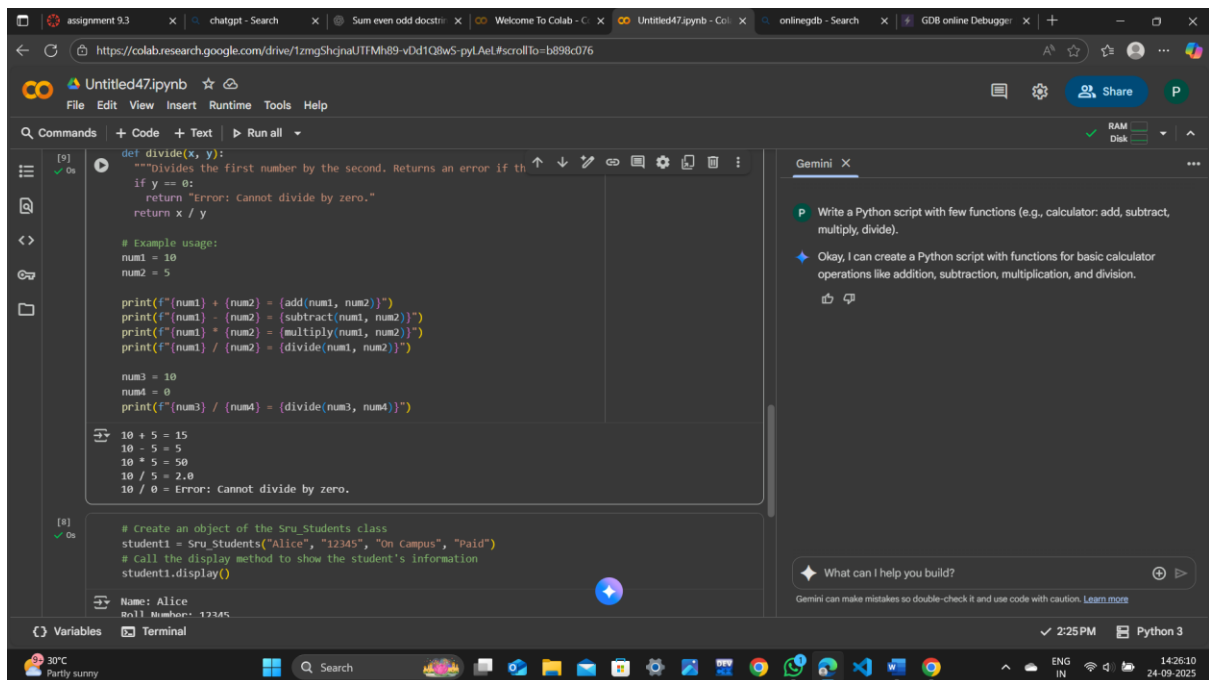
# Example usage:
num1 = 10
num2 = 5

print(f"{num1} + {num2} = {add(num1, num2)}")
print(f"{num1} - {num2} = {subtract(num1, num2)}")
print(f"{num1} * {num2} = {multiply(num1, num2)}")
print(f"{num1} / {num2} = {divide(num1, num2)}")

num3 = 10
num4 = 0
print(f"{num3} / {num4} = {divide(num3, num4)}")
```

Variables Terminal

2:25 PM Python 3



```
[9] ✓ Os
def divide(x, y):
    """Divides the first number by the second. Returns an error if the divisor is zero."""
    if y == 0:
        return "Error: cannot divide by zero."
    return x / y

# Example usage:
num1 = 10
num2 = 5

print(f"{num1} + {num2} = {add(num1, num2)}")
print(f"{num1} - {num2} = {subtract(num1, num2)}")
print(f"{num1} * {num2} = {multiply(num1, num2)}")
print(f"{num1} / {num2} = {divide(num1, num2)}")

num3 = 10
num4 = 0
print(f"{num3} / {num4} = {divide(num3, num4)}")
```

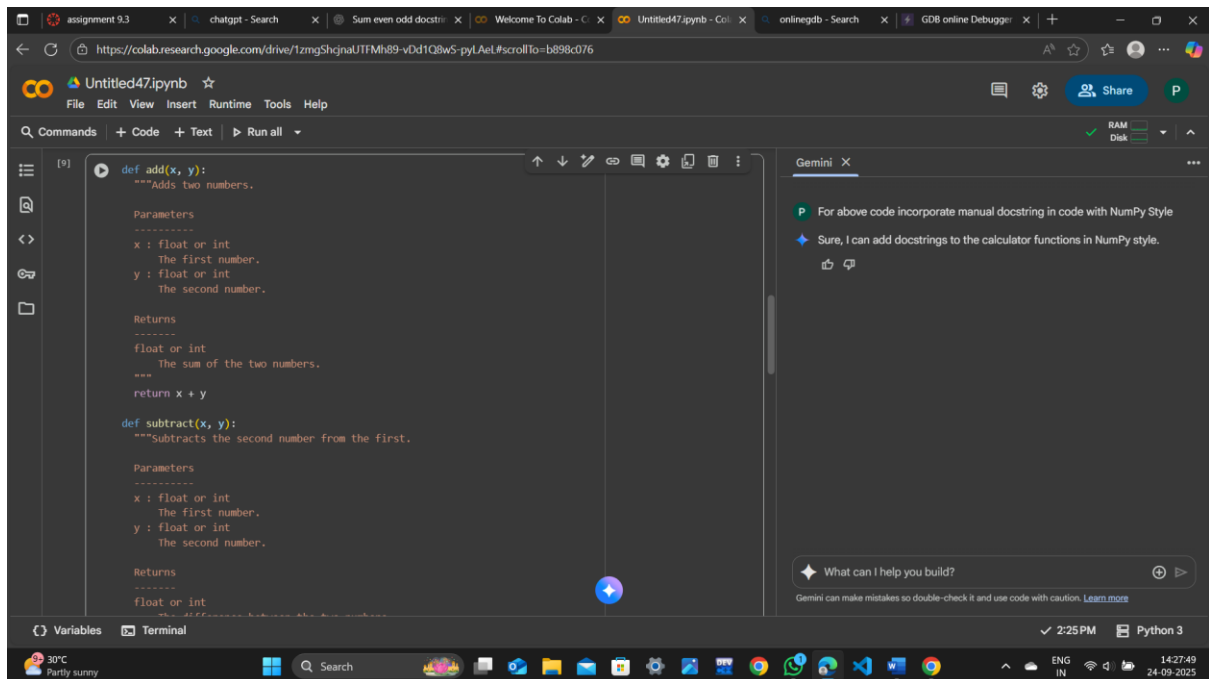
Variables Terminal

2:25 PM Python 3

Incorporate manual docstring in code with NumPy Style

PROMPT:

For Above code incorporate manual docstring in code with NumPy Style



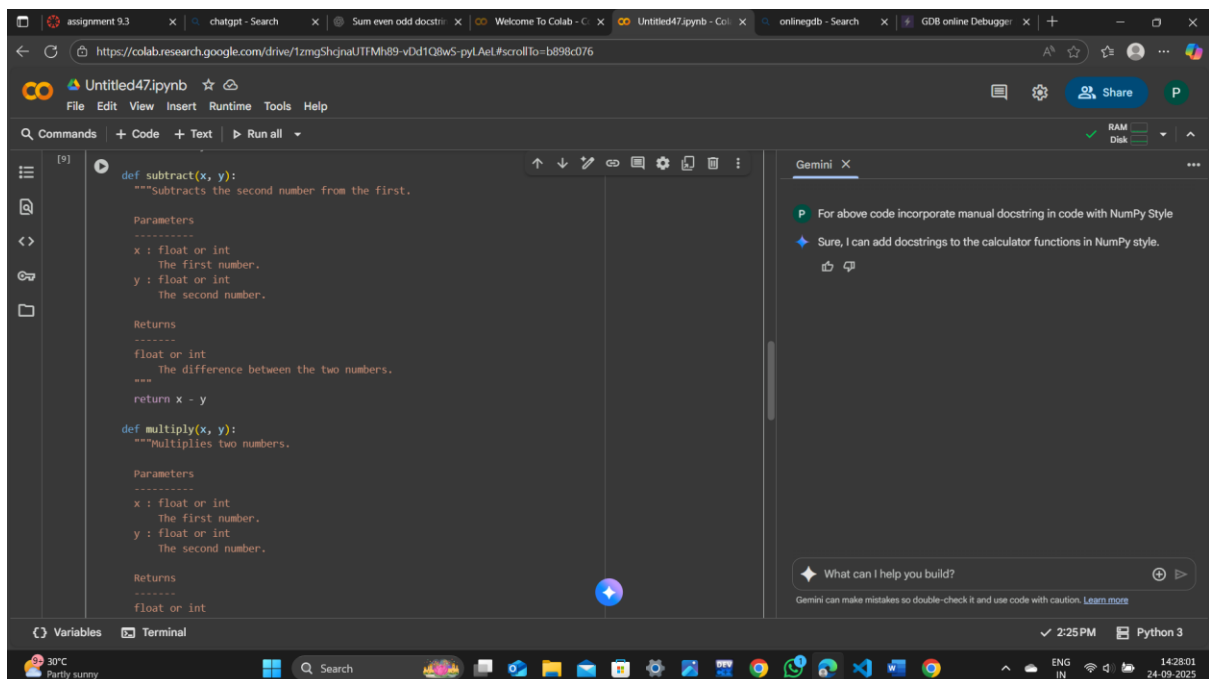
The screenshot shows a Jupyter Notebook interface with a code cell containing a Python function `add(x, y)`. The function has a manual docstring in NumPy style, including 'Parameters' and 'Returns' sections. A Gemini chat window on the right shows a prompt and a response.

```
def add(x, y):  
    """Adds two numbers.  
  
    Parameters  
    -----  
    x : float or int  
        The first number.  
    y : float or int  
        The second number.  
  
    Returns  
    -----  
    float or int  
        The sum of the two numbers.  
    """  
    return x + y  
  
def subtract(x, y):  
    """Subtracts the second number from the first.  
  
    Parameters  
    -----  
    x : float or int  
        The first number.  
    y : float or int  
        The second number.  
  
    Returns  
    -----  
    float or int  
        The difference between the two numbers.  
    """  
    return x - y  
  
def multiply(x, y):  
    """Multiplies two numbers.  
  
    Parameters  
    -----  
    x : float or int  
        The first number.  
    y : float or int  
        The second number.  
  
    Returns  
    -----  
    float or int  
        The product of the two numbers.  
    """  
    return x * y
```

Gemini chat window:

P For above code incorporate manual docstring in code with NumPy Style

Sure, I can add docstrings to the calculator functions in NumPy style.



The screenshot shows a Jupyter Notebook interface with a code cell containing a Python function `subtract(x, y)`. The function has a manual docstring in NumPy style, including 'Parameters', 'Returns', and a description. A Gemini chat window on the right shows a prompt and a response.

```
def subtract(x, y):  
    """Subtracts the second number from the first.  
  
    Parameters  
    -----  
    x : float or int  
        The first number.  
    y : float or int  
        The second number.  
  
    Returns  
    -----  
    float or int  
        The difference between the two numbers.  
    """  
    return x - y  
  
def multiply(x, y):  
    """Multiplies two numbers.  
  
    Parameters  
    -----  
    x : float or int  
        The first number.  
    y : float or int  
        The second number.  
  
    Returns  
    -----  
    float or int  
        The product of the two numbers.  
    """  
    return x * y
```

Gemini chat window:

P For above code incorporate manual docstring in code with NumPy Style

Sure, I can add docstrings to the calculator functions in NumPy style.

```
def divide(x, y):  
    """Divides the first number by the second.  
  
    Parameters  
    -----  
    x : float or int  
        The numerator.  
    y : float or int  
        The denominator.  
  
    Returns  
    -----  
    float or int or str  
        The quotient of the two numbers, or an error message if the denominator is zero.  
    """  
    if y == 0:  
        return "Error: Cannot divide by zero."  
    return x / y  
  
# Example usage:  
num1 = 10  
num2 = 5  
  
print(f"{num1} + {num2} = {add(num1, num2)}")  
print(f"{num1} - {num2} = {subtract(num1, num2)}")  
print(f"{num1} * {num2} = {multiply(num1, num2)}")  
print(f"{num1} / {num2} = {divide(num1, num2)}")  
  
num3 = 10  
num4 = 0  
print(f"{num3} / {num4} = {divide(num3, num4)}")
```

Gemini X

P For above code incorporate manual docstring in code with NumPy Style

Sure, I can add docstrings to the calculator functions in NumPy style.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

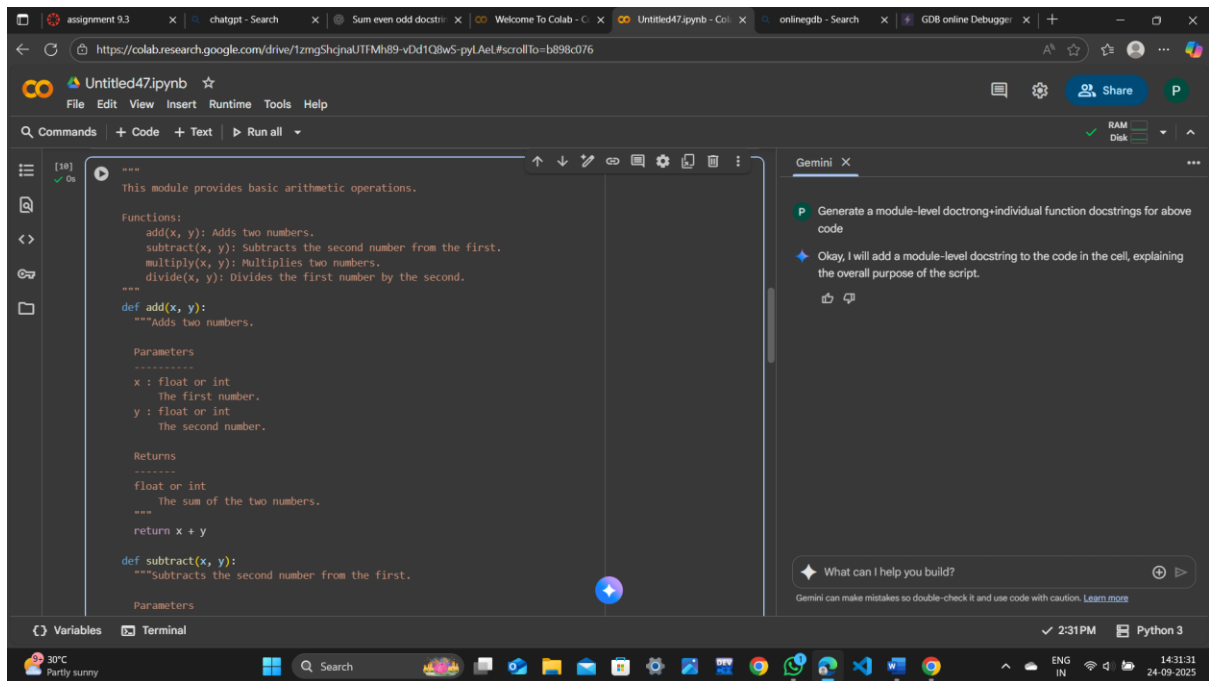
```
10 + 5 = 15  
10 - 5 = 5  
10 * 5 = 50  
10 / 5 = 2.0  
10 / 0 = Error: Cannot divide by zero.
```

```
# Create an object of the Sru_Students class  
student1 = Sru_Students("Alice", "12345", "On Campus", "Paid")  
# Call the display method to show the student's information  
student1.display()  
  
Name: Alice
```

Use AI assistance to generate a module-level docstring + individual function docstrings.

PROMPT:

Generate a module-level doctring+individual function docstrings for above code



The screenshot shows a Google Colab notebook titled "Untitled47.ipynb". The code cell contains a Python module with the following structure:

```
"""
This module provides basic arithmetic operations.

Functions:
    add(x, y): Adds two numbers.
    subtract(x, y): Subtracts the second number from the first.
    multiply(x, y): Multiplies two numbers.
    divide(x, y): Divides the first number by the second.
"""

def add(x, y):
    """Adds two numbers.

    Parameters
    -----
    x : float or int
        The first number.
    y : float or int
        The second number.

    Returns
    -----
    float or int
        The sum of the two numbers.
    """
    return x + y

def subtract(x, y):
    """Subtracts the second number from the first.

    Parameters
    -----
    x : float or int
        The first number.
    y : float or int
        The second number.

    Returns
    -----
    float or int
        The difference between the two numbers.
    """
    return x - y

def multiply(x, y):
    """Multiplies two numbers.

    Parameters
    -----
    x : float or int
        The first number.
    y : float or int
        The second number.

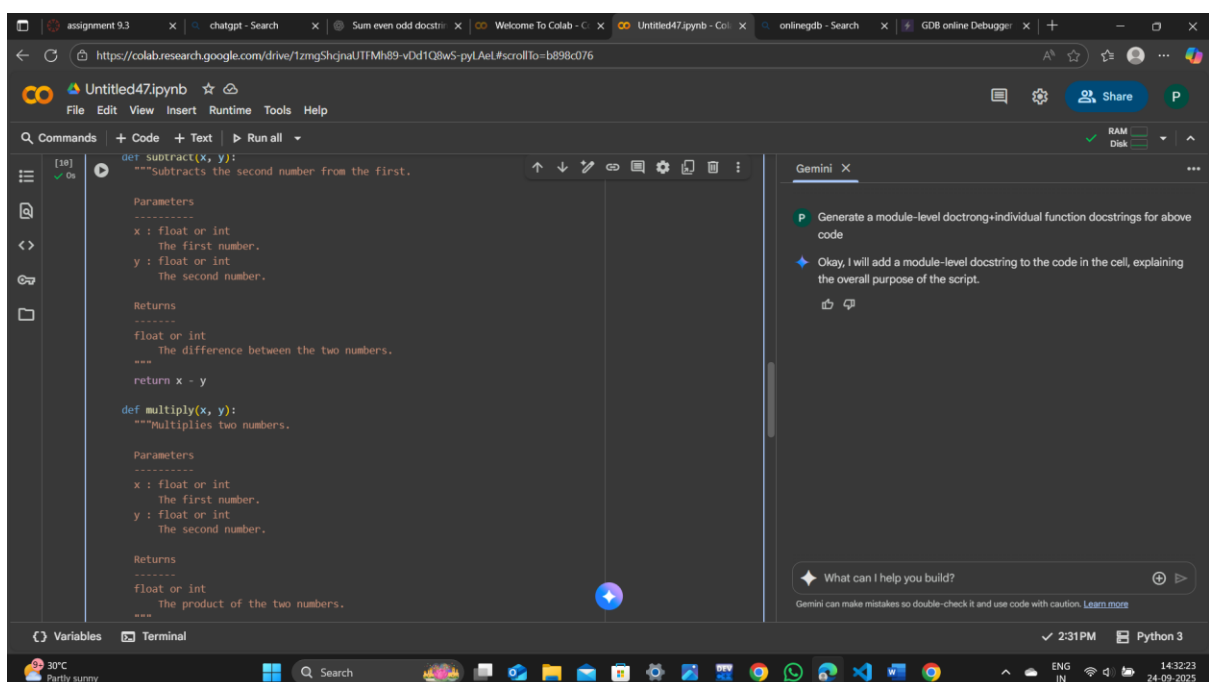
    Returns
    -----
    float or int
        The product of the two numbers.
    """
    return x * y

def divide(x, y):
    """Divides the first number by the second.

    Parameters
    -----
    x : float or int
        The first number.
    y : float or int
        The second number.

    Returns
    -----
    float or int
        The quotient of the two numbers.
    """
    return x / y
```

The Gemini chat interface on the right shows the prompt: "Generate a module-level doctring+individual function docstrings for above code". The response is: "Okay, I will add a module-level doctring to the code in the cell, explaining the overall purpose of the script."



The screenshot shows a Google Colab notebook titled "Untitled47.ipynb". The code cell contains a Python module with the following structure:

```
"""
This module provides basic arithmetic operations.

Functions:
    add(x, y): Adds two numbers.
    subtract(x, y): Subtracts the second number from the first.
    multiply(x, y): Multiplies two numbers.
    divide(x, y): Divides the first number by the second.
"""

def add(x, y):
    """Adds two numbers.

    Parameters
    -----
    x : float or int
        The first number.
    y : float or int
        The second number.

    Returns
    -----
    float or int
        The sum of the two numbers.
    """
    return x + y

def subtract(x, y):
    """Subtracts the second number from the first.

    Parameters
    -----
    x : float or int
        The first number.
    y : float or int
        The second number.

    Returns
    -----
    float or int
        The difference between the two numbers.
    """
    return x - y

def multiply(x, y):
    """Multiplies two numbers.

    Parameters
    -----
    x : float or int
        The first number.
    y : float or int
        The second number.

    Returns
    -----
    float or int
        The product of the two numbers.
    """
    return x * y

def divide(x, y):
    """Divides the first number by the second.

    Parameters
    -----
    x : float or int
        The first number.
    y : float or int
        The second number.

    Returns
    -----
    float or int
        The quotient of the two numbers.
    """
    return x / y
```

The Gemini chat interface on the right shows the prompt: "Generate a module-level doctring+individual function docstrings for above code". The response is: "Okay, I will add a module-level doctring to the code in the cell, explaining the overall purpose of the script."

```
def divide(x, y):  
    """Divides the first number by the second.  
  
    Parameters  
    -----  
    x : float or int  
        The numerator.  
    y : float or int  
        The denominator.  
  
    Returns  
    -----  
    float or int or str  
        The quotient of the two numbers, or an error message if the denominator is zero.  
    """  
    if y == 0:  
        return "Error: Cannot divide by zero."  
    return x / y  
  
# Example usage:  
num1 = 10  
num2 = 5  
  
print(f"{num1} + {num2} = {add(num1, num2)}")  
print(f"{num1} - {num2} = {subtract(num1, num2)}")  
print(f"{num1} * {num2} = {multiply(num1, num2)}")  
print(f"{num1} / {num2} = {divide(num1, num2)}")  
  
num3 = 10  
num4 = 0  
print(f"{num3} / {num4} = {divide(num3, num4)}")
```

```
# Example usage:  
num1 = 10  
num2 = 5  
  
print(f"{num1} + {num2} = {add(num1, num2)}")  
print(f"{num1} - {num2} = {subtract(num1, num2)}")  
print(f"{num1} * {num2} = {multiply(num1, num2)}")  
print(f"{num1} / {num2} = {divide(num1, num2)}")  
  
num3 = 10  
num4 = 0  
print(f"{num3} / {num4} = {divide(num3, num4)}")  
  
10 + 5 = 15  
10 - 5 = 5  
10 * 5 = 50  
10 / 5 = 2.0  
10 / 0 = Error: Cannot divide by zero.
```

Compare the AI-generated docstring with your manually written one

OBSERVATION:

As per my Observation.Both docstrings looking good and has meaning as per the code that is been given,In my docstring it has simple English letters and words where as in Gemini code ,it has used some Terminology.

Apart from that Everything is good when we compare the docstrings .Externally We took help to get module level docstring from GEMINI.