]SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName:M. Tech		**Assignment Type: Lab**	**AcademicYear:**2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
rCourseCode	24CS002PC215	**CourseTitle**	AI Assisted Problem Solving Using Python
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	10.11.2025	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber:9.3(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	Lab 9: Documentation Generation: Automatic documentation and code comments **Lab Objectives:** • To understand the importance of documentation and code comments in software development. • To explore how AI-assisted coding tools can generate meaningful documentation and inline comments. • To practice generating function-level and module-level docstrings automatically. • To evaluate the quality, accuracy, and limitations of AI-generated documentation. • To develop a small automated tool for documentation generation in Python.. **Lab Outcomes (LOs):** After completing this lab, students will be able to: • Apply AI-assisted coding tools to generate docstrings and inline comments for Python code. • Critically analyze AI-generated documentation for correctness,	Week4 - Wednesday

completeness, and readability.

- Create structured documentation (function-level, module-level) following standard formats.

- Design and implement a mini documentation generator tool to automate code commenting and docstring creation.

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.

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.

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

Push documentation whole workspace as .md file in GitHub

	Repository **Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots**	

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.

PROMPT: Write a Python function to return the sum of even and odd numbers in a list. Add a **manual docstring** in Google Style, then use AI to generate its own docstring and compare both.

Manual docstring in Google Style:

```python
def sum_even_odd_manual(numbers):
    """
    Calculates the sum of even and odd numbers in a given list.

    Args:
        numbers (list): A list of integers entered by the user.

    Returns:
        tuple: A tuple containing two integers:
            - The first value is the sum of even numbers.
            - The second value is the sum of odd numbers.

    Example:
        >>> sum_even_odd_manual([1, 2, 3, 4, 5])
        (6, 9)
    """
    even_sum = 0
    odd_sum = 0

    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num

    return even_sum, odd_sum


# --- User Input Section ---
numbers = list(map(int, input("Enter numbers separated by spaces: ").split()))
```

```
                odd_sum += num

    return even_sum, odd_sum


# --- User Input Section ---
numbers = list(map(int, input("Enter numbers separated by spaces: ").split()))
even_sum, odd_sum = sum_even_odd_manual(numbers)
print("Manual Docstring Function Result:")
print("Sum of Even Numbers:", even_sum)
print("Sum of Odd Numbers:", odd_sum)
```

AI-Generated Docstring Version:

SK 1.py | TASK 1.1.py | def sum_even_odd_ai(numbers): Untitled-1 | TASK 3.py

```
def sum_even_odd_ai(numbers):
    """
    Returns the sum of even and odd numbers from the given list.

    Parameters:
        numbers (list): List of integers.

    Returns:
        tuple: (sum_of_even, sum_of_odd)
    """
    even_sum = sum(num for num in numbers if num % 2 == 0)
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    return even_sum, odd_sum


# --- User Input Section ---
numbers = list(map(int, input("Enter numbers separated by spaces: ").split()))
even_sum, odd_sum = sum_even_odd_ai(numbers)
print("AI-Generated Docstring Function Result:")
print("Sum of Even Numbers:", even_sum)
print("Sum of Odd Numbers:", odd_sum)
```

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

Practical output:
Manual docstring in Google Style:

```
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rimsha\OneDrive\Deskto
tempCodeRunnerFile.python"
Enter numbers separated by spaces: 1 2 3 4 5 6 7 8
Manual Docstring Function Result:
Sum of Even Numbers: 20
Sum of Odd Numbers: 16
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment>
```

AI-Generated Docstring Version:

```
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rims
tempCodeRunnerFile.python"
Enter numbers separated by spaces: 1 2 3 4 5 6 7 8 9 2
AI-Generated Docstring Function Result:
Sum of Even Numbers: 22
Sum of Odd Numbers: 25
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment>
```

Comparison Table

Aspect	Manual Docstring (Google Style)	AI-Generated Docstring
Structure	Follows Google Style (Args, Returns, Example)	Generic short form (Parameters, Returns)
Detail Level	More descriptive and clear for beginners	Concise and minimal
Clarity	Explains every parameter and output explicitly	Summarizes in short
Professional Use	Best for documentation and learning	Quick and efficient for developers
Example Included	Yes	No

Task Description#2 Automatic Inline Comments

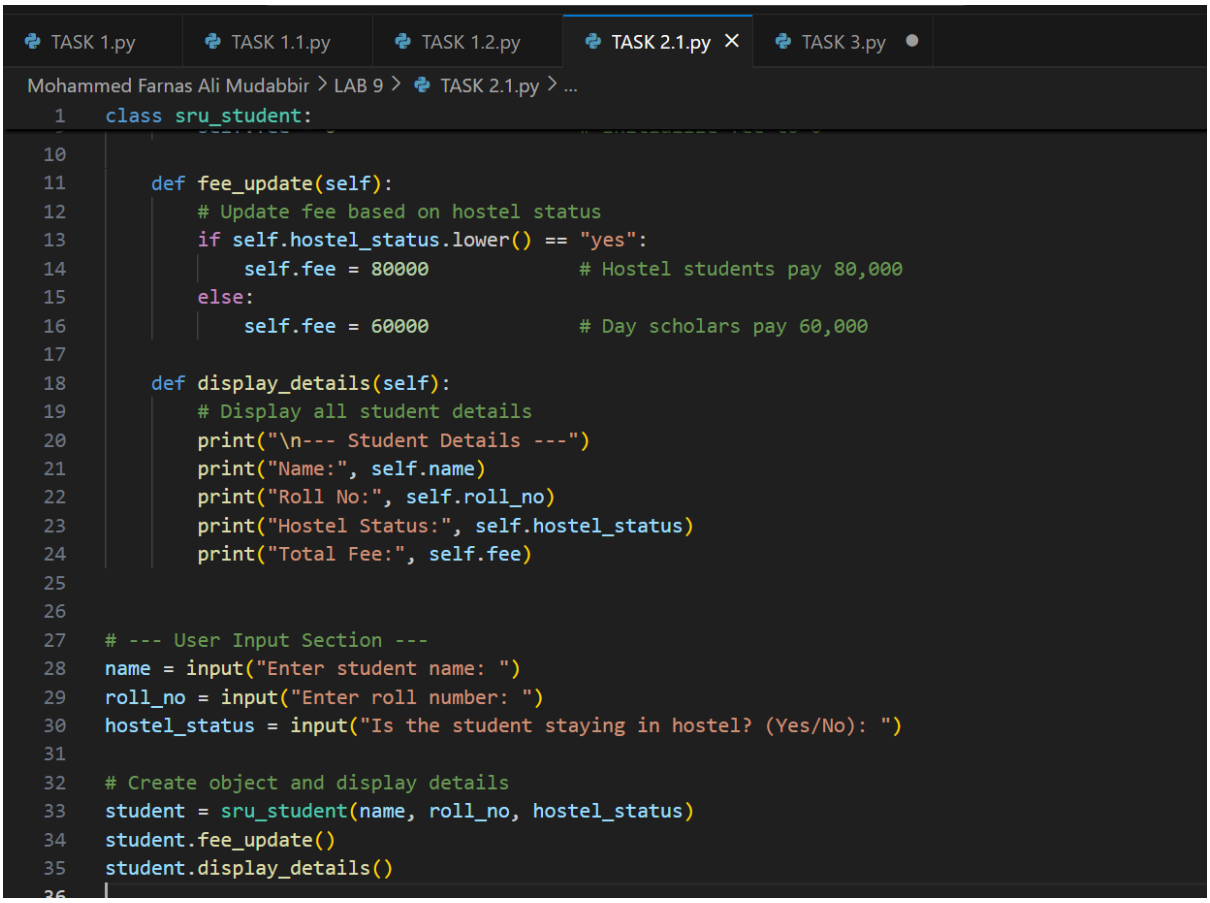
- Write python program for **sru_student** class with attributes like name, roll no., hostel_status and **fee_update** method and **display_details** method.

- 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.

PROMPT: Write a Python program for an sru_student class with attributes name, roll_no, and hostel_status, including methods fee_update() and display_details(). Add manual comments for each line, then let AI generate inline comments and compare both.

Manual docstring in Google Style:



AI-Generated Docstring Version:

```python
class sru_student:
    def fee_update(self):
        if self.hostel_status.lower() == "yes":
            self.fee = 80000  # Set fee to 80000 for hostel students
        else:
            self.fee = 60000  # Set fee to 60000 for non-hostel students

    def display_details(self):
        # Print student details and total fee
        print("\n--- Student Details ---")
        print("Name:", self.name)
        print("Roll No:", self.roll_no)
        print("Hostel Status:", self.hostel_status)
        print("Total Fee:", self.fee)


# Take input from user for student details
name = input("Enter student name: ")
roll_no = input("Enter roll number: ")
hostel_status = input("Is the student staying in hostel? (Yes/No): ")

# Create a student object and display fee details
student = sru_student(name, roll_no, hostel_status)
student.fee_update()
student.display_details()
```

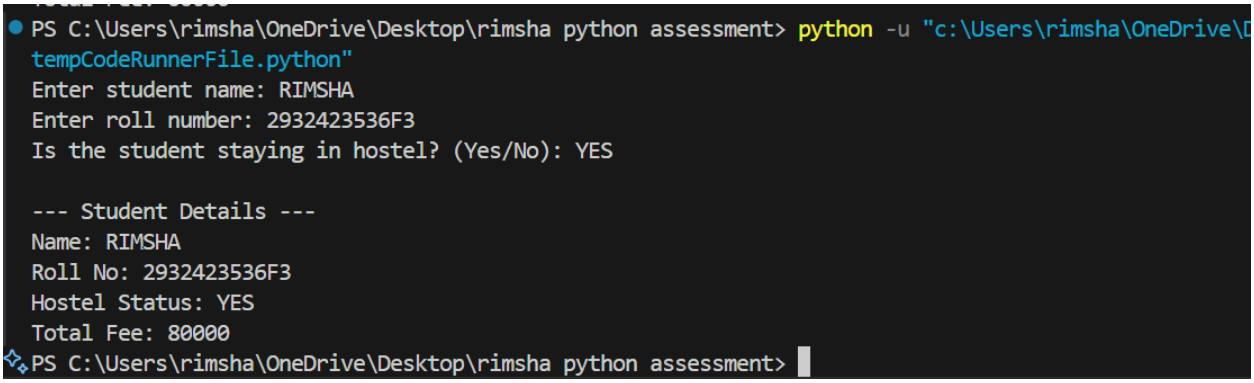
Expected Output#2: Students critically analyze AI-generated code comments.

Practical output:
Manual docstring in Google Style:

```
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rimsha\One
9 assessment\TASK 2.1 MANUAL.py"
Enter student name: RAHIMATHUNNISA RIMSHA
Enter roll number: 2205A41L24
Is the student staying in hostel? (Yes/No): NO

--- Student Details ---
Name: RAHIMATHUNNISA RIMSHA
Roll No: 2205A41L24
Hostel Status: NO
Total Fee: 60000
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment>
```

AI-Generated Docstring Version:



Manual vs AI-Generated Comment Comparison

Aspect	Manual Comments (Written by Student)	AI-Generated Comments (From Copilot/Cursor AI)
Style	Simple, clear, and descriptive	Concise, professional, and formal
Purpose	Explains **why** each line is written and what it means	Explains **what** each line of code does
Tone	Friendly and educational (beginner-friendly)	Technical and developer-focused
Detail Level	More detailed — focuses on logic and reasoning behind code	Brief — focuses only on the functionality of code
Example	"# Update fee based on hostel	"# Check if student is a hostel

Example	"# Update fee based on hostel status" → tells the purpose of that step	"# Check if student is a hostel resident and update fee accordingly" → states the direct action
Readability	Easier for beginners to follow and understand logic	More suitable for experienced programmers
Depth of Explanation	Explains initialization, logic, and output purpose	Describes code behavior without deeper reasoning
Usefulness in Learning	✅ Excellent for understanding flow and concept	⚙️ Good for documentation and quick code review
Accuracy	Human-level clarity and contextual understanding	AI may sometimes give generic or repeated phrasing

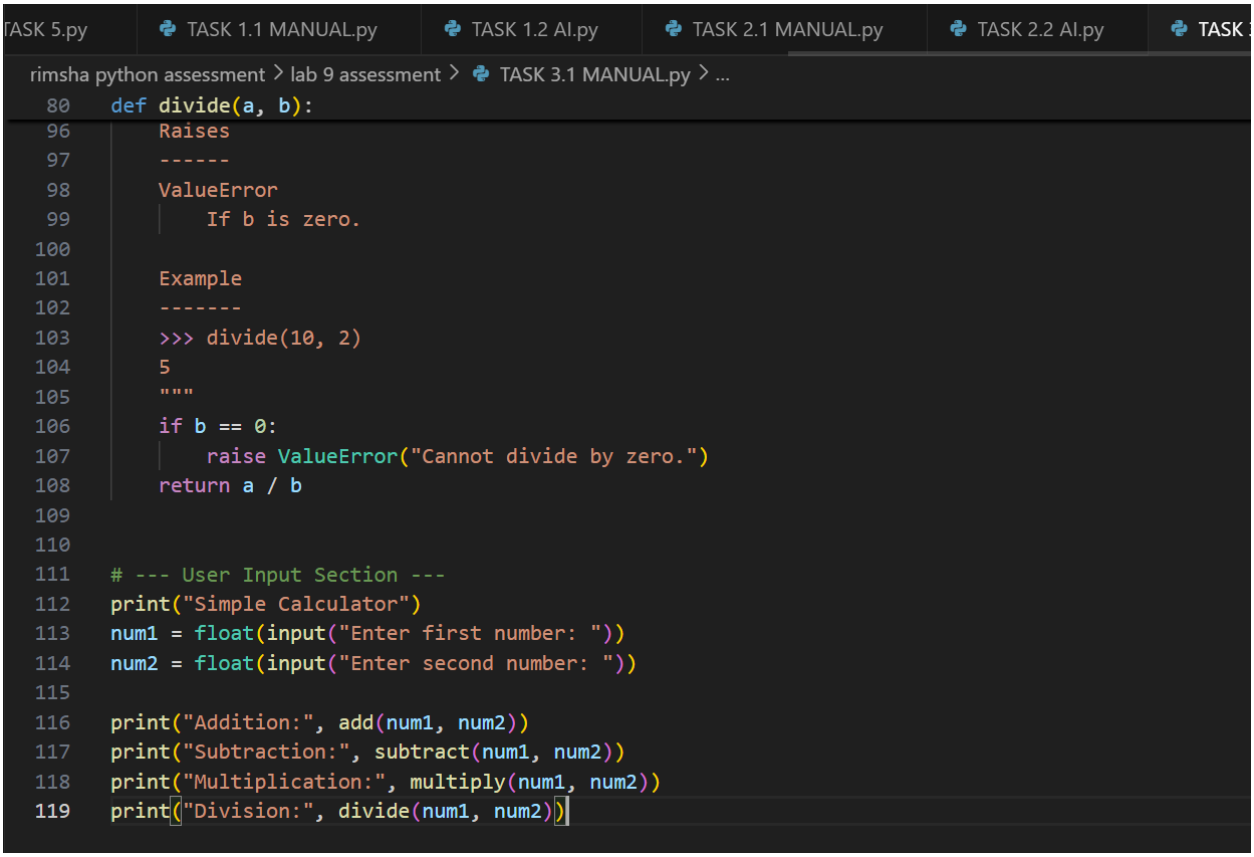
Manual Comment	AI-Generated Comment
`# Initialize student attributes`	`# Initialize the student object with name, roll number, and hostel status`
`# Update fee based on hostel status`	`# Check if student is a hostel resident and update fee accordingly`
`# Display all student details`	`# Print student details and total fee`

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 script with 4 functions — add, subtract, multiply, and divide. Add manual NumPy-style docstrings for each function. Then use AI to generate module-level and function docstrings, and compare both.

Manual docstring in Google Style:



TASK 1.py | TASK 1.1.py | TASK 1.2.py | TASK 2.1.py | TASK 2.2.py | Untitled-1 | TASK 3.py

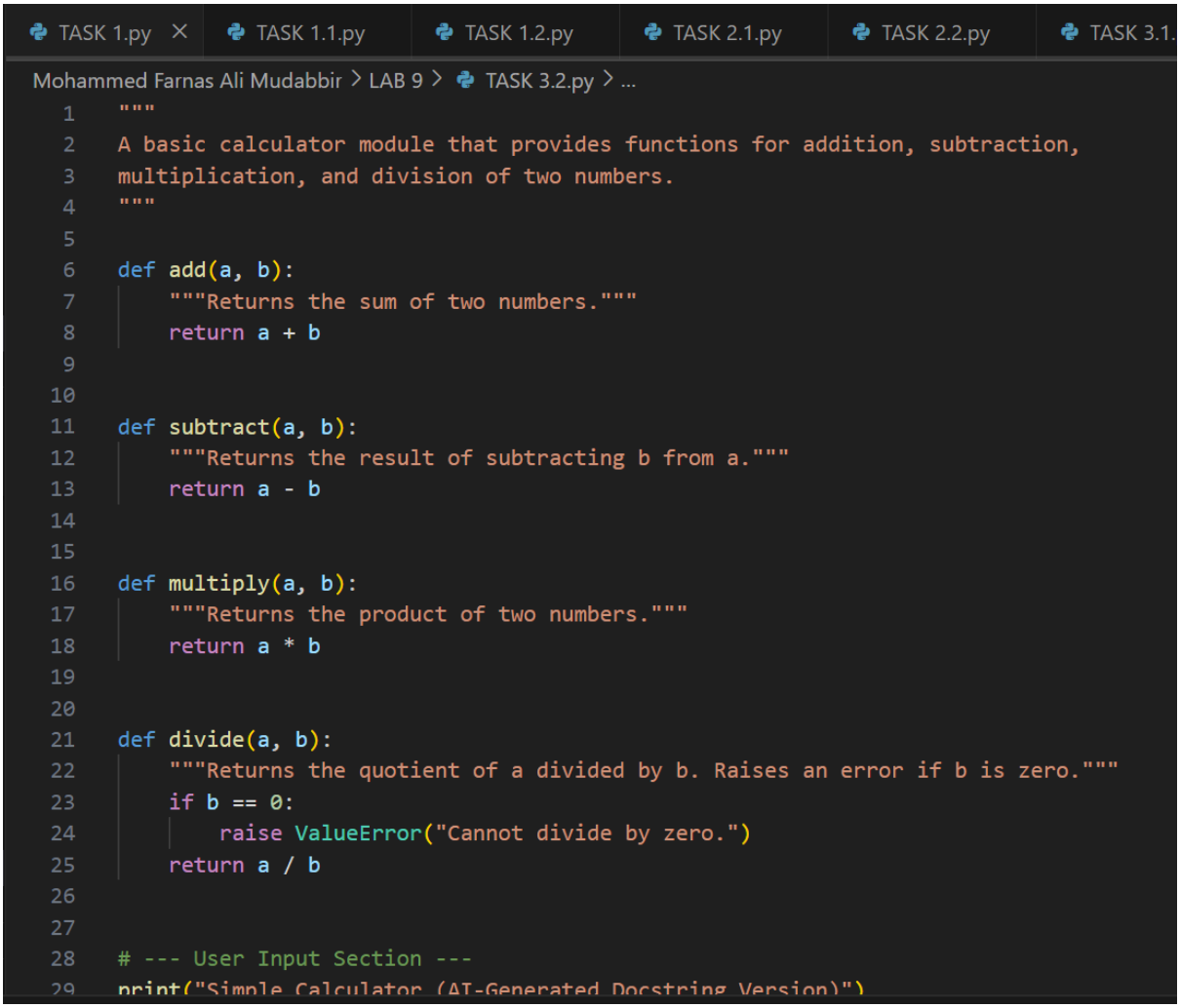
```
31
32  def subtract(a, b):
33      """
34      Subtract two numbers.
35
36      Parameters
37      ----------
38      a : float
39          First number.
40      b : float
41          Second number to subtract from the first.
42
43      Returns
44      -------
45      float
46          Result of a - b.
47
48      Example
49      -------
50      >>> subtract(10, 4)
51      6
52      """
53      return a - b
54
55
56  def multiply(a, b):
57      """
58      Multiply two numbers.
59
60      Parameters
```

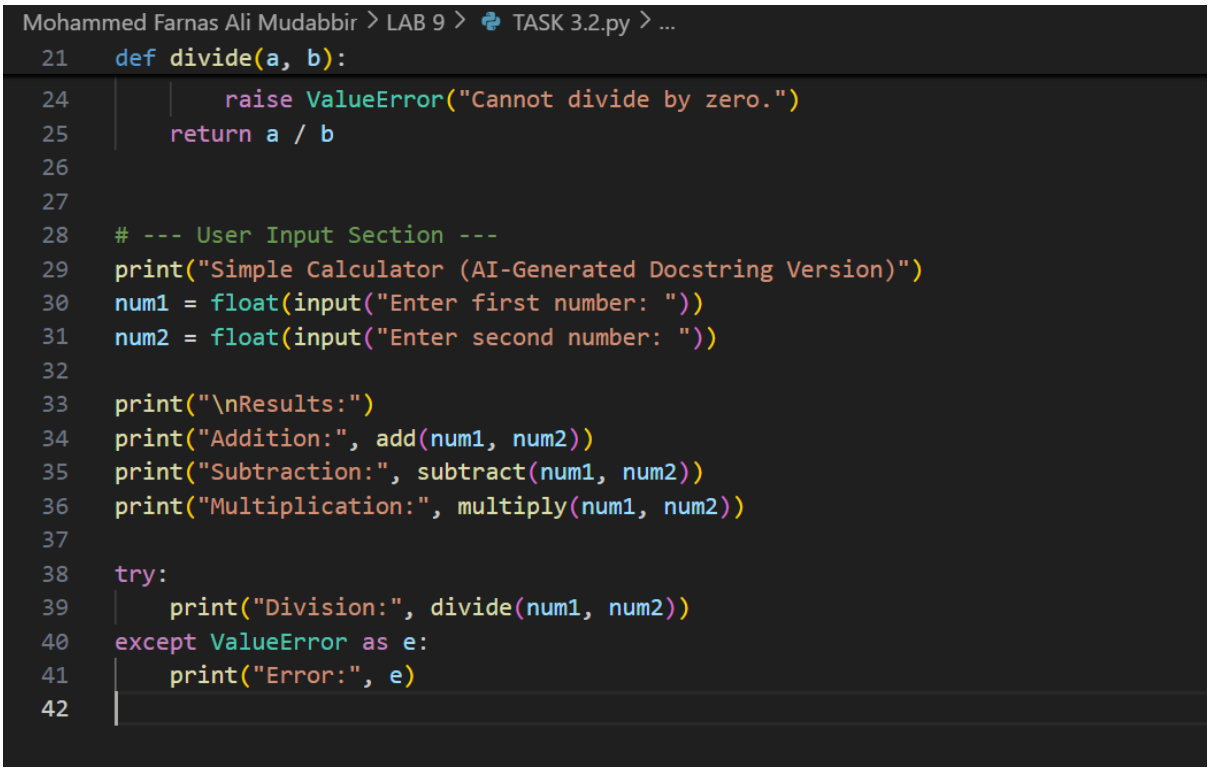
TASK 1.py | TASK 1.1.py | TASK 1.2.py | TASK 2.1.py | TASK 2.2.py | Untitled-1 | TASK 3.py

```
56  def multiply(a, b):
60      Parameters
61      ----------
62      a : float
63          First number.
64      b : float
65          Second number.
66
67      Returns
68      -------
69      float
70          Product of a and b.
71
72      Example
73      -------
74      >>> multiply(2, 5)
75      10
76      """
77      return a * b
78
79
80  def divide(a, b):
81      """
82      Divide two numbers.
83
84      Parameters
85      ----------
86      a : float
87          Numerator.
88      b : float
```

```
80  def divide(a, b):
88      b : float
89          Denominator.
90
91      Returns
92      -------
93      float
94          Result of division (a / b).
95
96      Raises
97      ------
98      ValueError
99          If b is zero.
100
101     Example
102     -------
103     >>> divide(10, 2)
104     5
105     """
106     if b == 0:
107         raise ValueError("Cannot divide by zero.")
108     return a / b
109
110
111 # --- User Input Section ---
112 print("Simple Calculator")
113 num1 = float(input("Enter first number: "))
114 num2 = float(input("Enter second number: "))
115
109
110
111 # --- User Input Section ---
112 print("Simple Calculator")
113 num1 = float(input("Enter first number: "))
114 num2 = float(input("Enter second number: "))
115
116 print("Addition:", add(num1, num2))
117 print("Subtraction:", subtract(num1, num2))
118 print("Multiplication:", multiply(num1, num2))
119 print("Division:", divide(num1, num2))
120
```

AI-Generated Docstring Version:

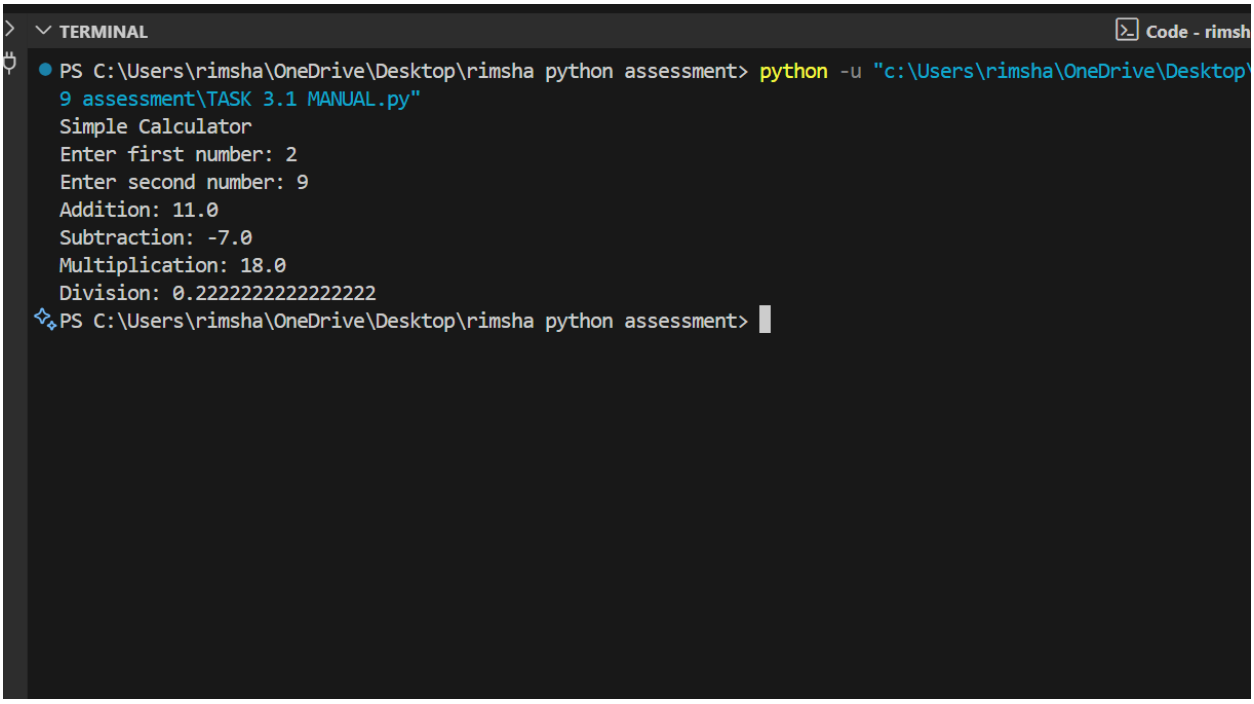




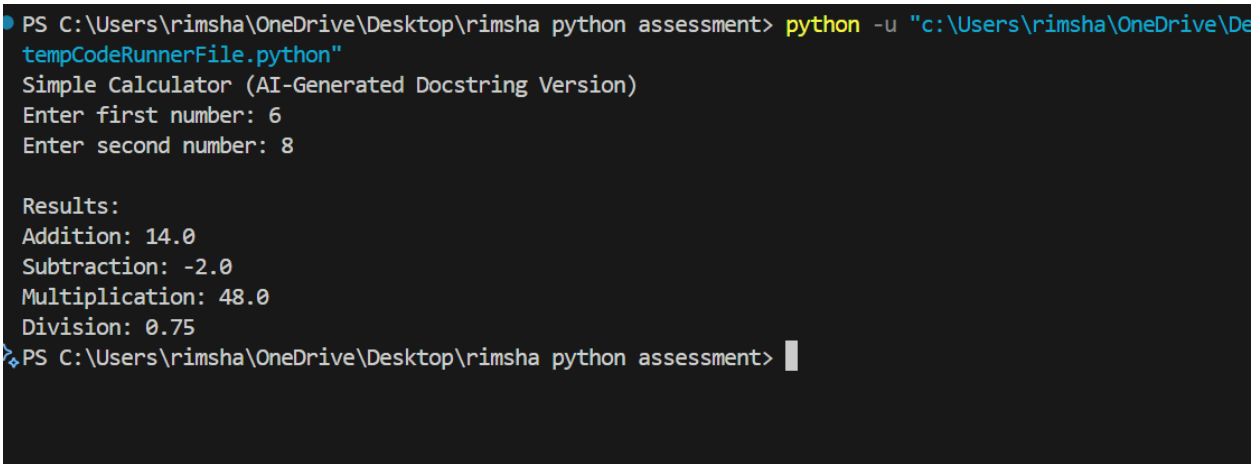
Expected Output#3: Students learn structured documentation for multi-function scripts

Practical output:

Manual docstring in Google Style:



AI-Generated Docstring Version:



📊 Comparison: Manual vs AI-Generated Docstrings

Aspect	Manual NumPy-Style (Student)	AI-Generated (Copilot/Cursor AI)
Structure	Detailed NumPy style with `Parameters`, `Returns`, `Example`	Short single-line description
Module Docstring	Explains purpose and structure	Briefly summarizes functionality
Detail Level	Highly descriptive and formatted	Concise and to the point
Clarity	Excellent for learners and documentation	Suitable for professional quick reference
Examples Included	✅ Yes	❌ No