

Task Description 1 :

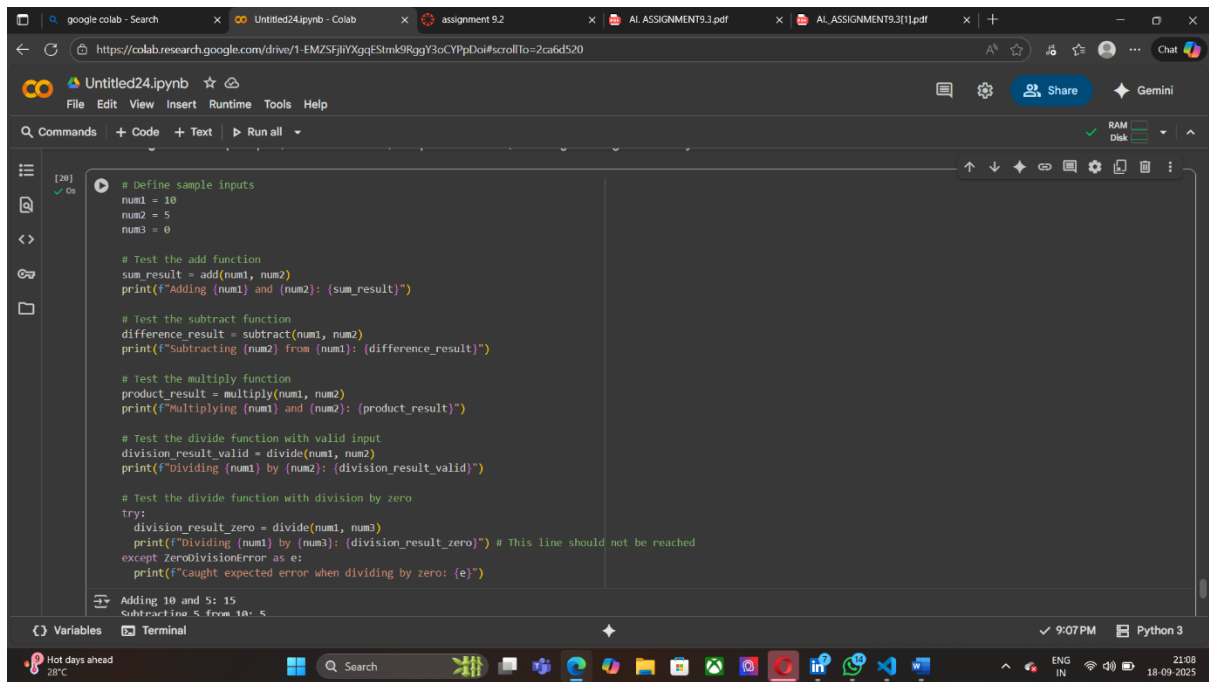
(Documentation – Google-Style Docstrings for Python Functions)

- Task: Use AI to add Google-style docstrings to all functions in a given Python script.
- Instructions:
 - o Prompt AI to generate docstrings without providing any input-output examples.
 - o Ensure each docstring includes:
 - Function description
 - Parameters with type hints
 - Return values with type hints
 - Example usage
 - o Review the generated docstrings for accuracy and formatting.
- Expected Output #1:
 - o A Python script with all functions documented using correctly formatted Google-style docstrings.

Prompt:

Add Google-style docstrings to all functions in the following Python script.

- Do not include any input-output examples.
- Each docstring should include:
 - o Function description
 - o Parameters with type hints
 - o Return values with type hints
 - o Example usage
- Ensure the docstrings are accurate and properly formatted.



```
[28] ✓ On # Define sample inputs
num1 = 10
num2 = 5
num3 = 0

# Test the add function
sum_result = add(num1, num2)
print(f"Adding {num1} and {num2}: {sum_result}")

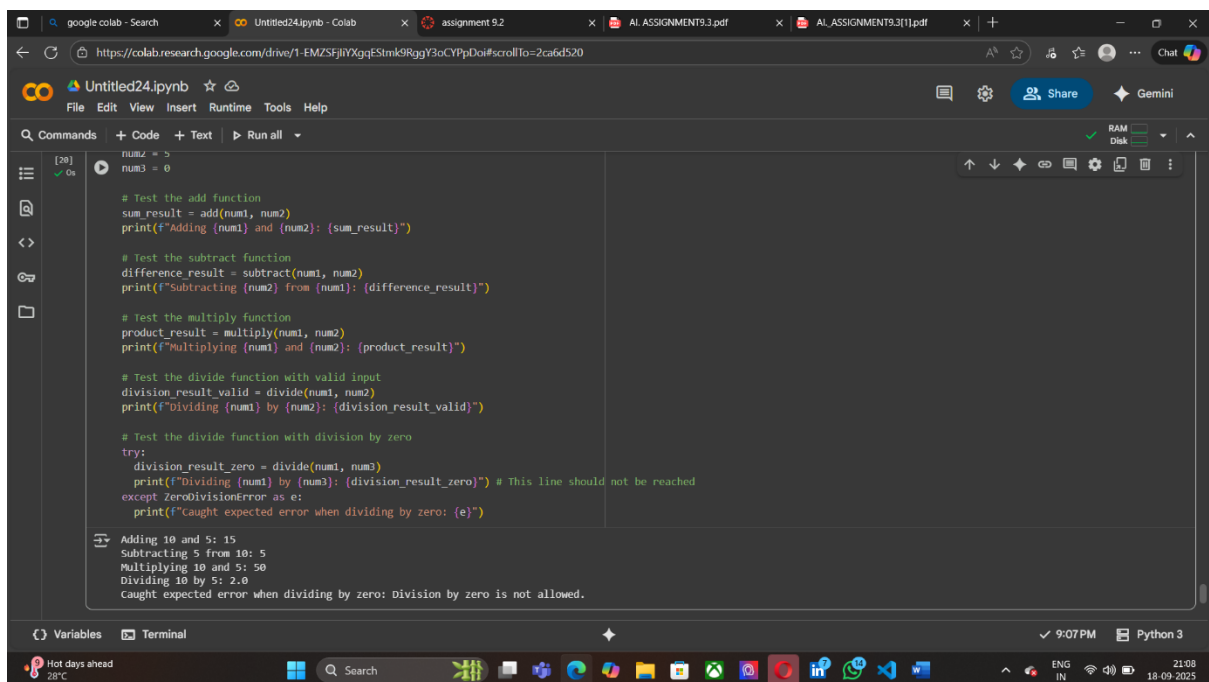
# Test the subtract function
difference_result = subtract(num1, num2)
print(f"Subtracting {num2} from {num1}: {difference_result}")

# Test the multiply function
product_result = multiply(num1, num2)
print(f"Multiplying {num1} and {num2}: {product_result}")

# Test the divide function with valid input
division_result_valid = divide(num1, num2)
print(f"Dividing {num1} by {num2}: {division_result_valid}")

# Test the divide function with division by zero
try:
    division_result_zero = divide(num1, num3)
    print(f"Dividing {num1} by {num3}: {division_result_zero}") # This line should not be reached
except ZeroDivisionError as e:
    print(f"caught expected error when dividing by zero: {e}")
```

Adding 10 and 5: 15
Subtracting 5 from 10: 5



```
num1 = 10
num2 = 5
num3 = 0

# Test the add function
sum_result = add(num1, num2)
print(f"Adding {num1} and {num2}: {sum_result}")

# Test the subtract function
difference_result = subtract(num1, num2)
print(f"Subtracting {num2} from {num1}: {difference_result}")

# Test the multiply function
product_result = multiply(num1, num2)
print(f"Multiplying {num1} and {num2}: {product_result}")

# Test the divide function with valid input
division_result_valid = divide(num1, num2)
print(f"Dividing {num1} by {num2}: {division_result_valid}")

# Test the divide function with division by zero
try:
    division_result_zero = divide(num1, num3)
    print(f"Dividing {num1} by {num3}: {division_result_zero}") # This line should not be reached
except ZeroDivisionError as e:
    print(f"caught expected error when dividing by zero: {e}")
```

Adding 10 and 5: 15
Subtracting 5 from 10: 5
Multiplying 10 and 5: 50
Dividing 10 by 5: 2.0
Caught expected error when dividing by zero: Division by zero is not allowed.

Task Description 2:

(Documentation – Inline Comments for Complex Logic)

- Task: Use AI to add meaningful inline comments to a Python program explaining only complex logic parts.

- Instructions:

- o Provide a Python script without comments to the AI.
- o Instruct AI to skip obvious syntax explanations and focus only on tricky or non-intuitive code sections.
- o Verify that comments improve code readability and maintainability.

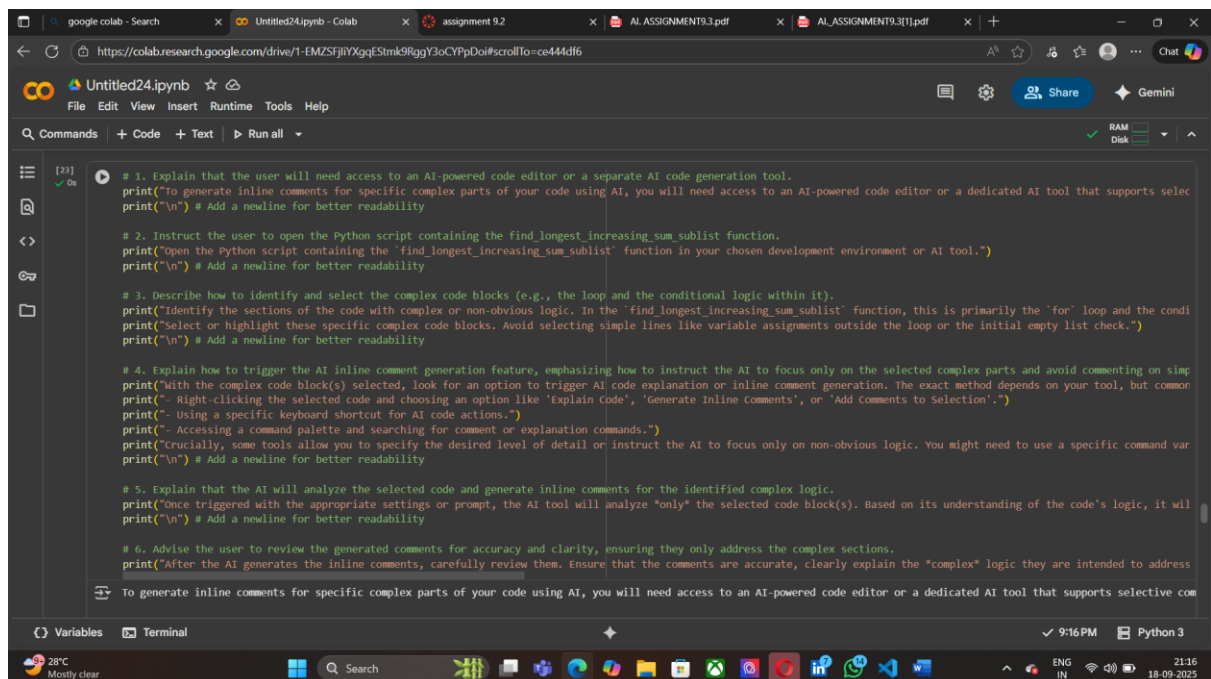
- Expected Output #2:

- o Python code with concise, context-aware inline comments for complex logic blocks.

Prompt:

Add meaningful inline comments to the following Python program, explaining only the complex or non-intuitive logic parts.

- Skip obvious syntax explanations.
- Focus on tricky or non-obvious code sections.
- Ensure comments improve code readability and maintainability.



```
[23] ✓ 0s
# 1. Explain that the user will need access to an AI-powered code editor or a separate AI code generation tool.
print("To generate inline comments for specific complex parts of your code using AI, you will need access to an AI-powered code editor or a dedicated AI tool that supports select\n") # Add a newline for better readability

# 2. Instruct the user to open the Python script containing the find_longest_increasing_sum_sublist function.
print("Open the python script containing the 'find_longest_increasing_sum_sublist' function in your chosen development environment or AI tool.")
print("\n") # Add a newline for better readability

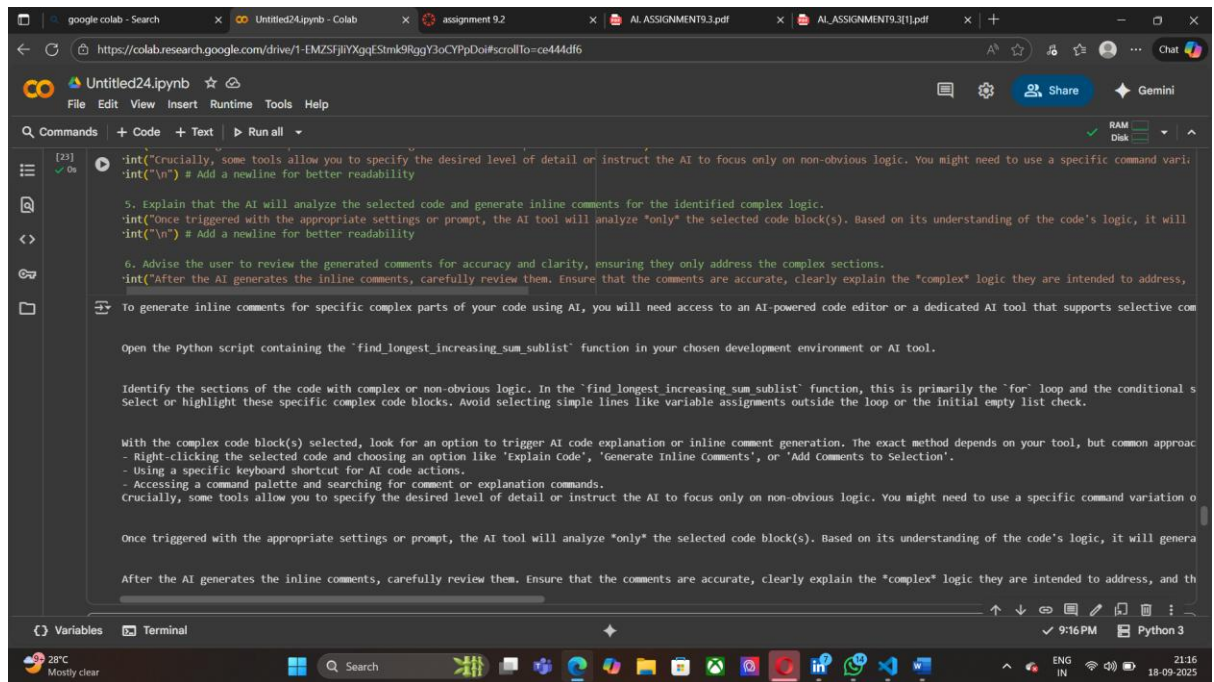
# 3. Describe how to identify and select the complex code blocks (e.g., the loop and the conditional logic within it).
print("Identify the sections of the code with complex or non-obvious logic. In the 'find_longest_increasing_sum_sublist' function, this is primarily the 'for' loop and the condi\n") # Add a newline for better readability
print("Select or highlight these specific complex code blocks. Avoid selecting simple lines like variable assignments outside the loop or the initial empty list check.")

# 4. Explain how to trigger the AI inline comment generation feature, emphasizing how to instruct the AI to focus only on the selected complex parts and avoid commenting on simple\n") # Add a newline for better readability
print("With the complex code block(s) selected, look for an option to trigger AI code explanation or inline comment generation. The exact method depends on your tool, but common\n") # Add a newline for better readability
print("- Right-clicking the selected code and choosing an option like 'Explain code', 'Generate inline comments', or 'Add comments to selection'.")
print("- Using a specific keyboard shortcut for AI code actions.")
print("- Accessing a command palette and searching for comment or explanation commands.")
print("Crucially, some tools allow you to specify the desired level of detail or instruct the AI to focus only on non-obvious logic. You might need to use a specific command var\n") # Add a newline for better readability

# 5. Explain that the AI will analyze the selected code and generate inline comments for the identified complex logic.
print("Once triggered with the appropriate settings or prompt, the AI tool will analyze *only* the selected code block(s). Based on its understanding of the code's logic, it will\n") # Add a newline for better readability

# 6. Advise the user to review the generated comments for accuracy and clarity, ensuring they only address the complex sections.
print("After the AI generates the inline comments, carefully review them. Ensure that the comments are accurate, clearly explain the 'complex' logic they are intended to address\n") # Add a newline for better readability

To generate inline comments for specific complex parts of your code using AI, you will need access to an AI-powered code editor or a dedicated AI tool that supports selective com
```



Task Description 3:

(Documentation – Module-Level Documentation)

- Task: Use AI to create a module-level docstring summarizing the purpose, dependencies, and main functions/classes of a Python file.

- Instructions:

- o Supply the entire Python file to AI.
- o Instruct AI to write a single multi-line docstring at the top of the file.
- o Ensure the docstring clearly describes functionality and usage without rewriting the entire code.

- Expected Output #3:

- o A complete, clear, and concise module-level docstring at the beginning of the file.

Prompt:

Write a single multi-line module-level docstring at the top of the following Python file.

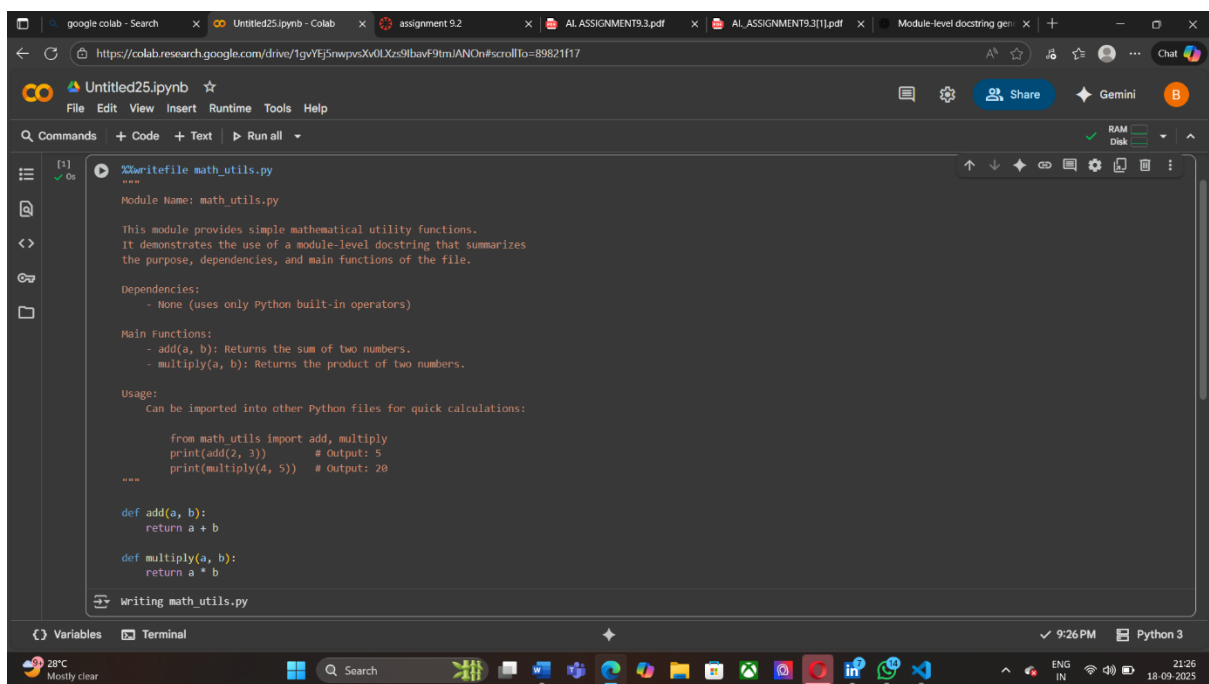
- Summarize the file's purpose, dependencies, and main functions/classes.

- Clearly describe functionality and usage without rewriting the entire code

Prompt:

Transform all existing inline comments in the following Python code into structured function docstrings using Google style.

- Move relevant details from comments into the function docstrings.
- Ensure the new docstrings keep the original meaning and improve structure.



The screenshot shows a Google Colab notebook interface. The main editor displays a Python file named 'math_utils.py'. The code includes a module-level docstring, dependencies, main functions, and usage instructions. The functions 'add' and 'multiply' are defined at the bottom.

```

writefile math_utils.py
'''
Module Name: math_utils.py

This module provides simple mathematical utility functions.
It demonstrates the use of a module-level docstring that summarizes
the purpose, dependencies, and main functions of the file.

Dependencies:
- None (uses only Python built-in operators)

Main Functions:
- add(a, b): Returns the sum of two numbers.
- multiply(a, b): Returns the product of two numbers.

Usage:
Can be imported into other Python files for quick calculations:

    from math_utils import add, multiply
    print(add(2, 3))      # Output: 5
    print(multiply(4, 5)) # Output: 20
'''

def add(a, b):
    return a + b

def multiply(a, b):
    return a * b
  
```

Task Description 4:

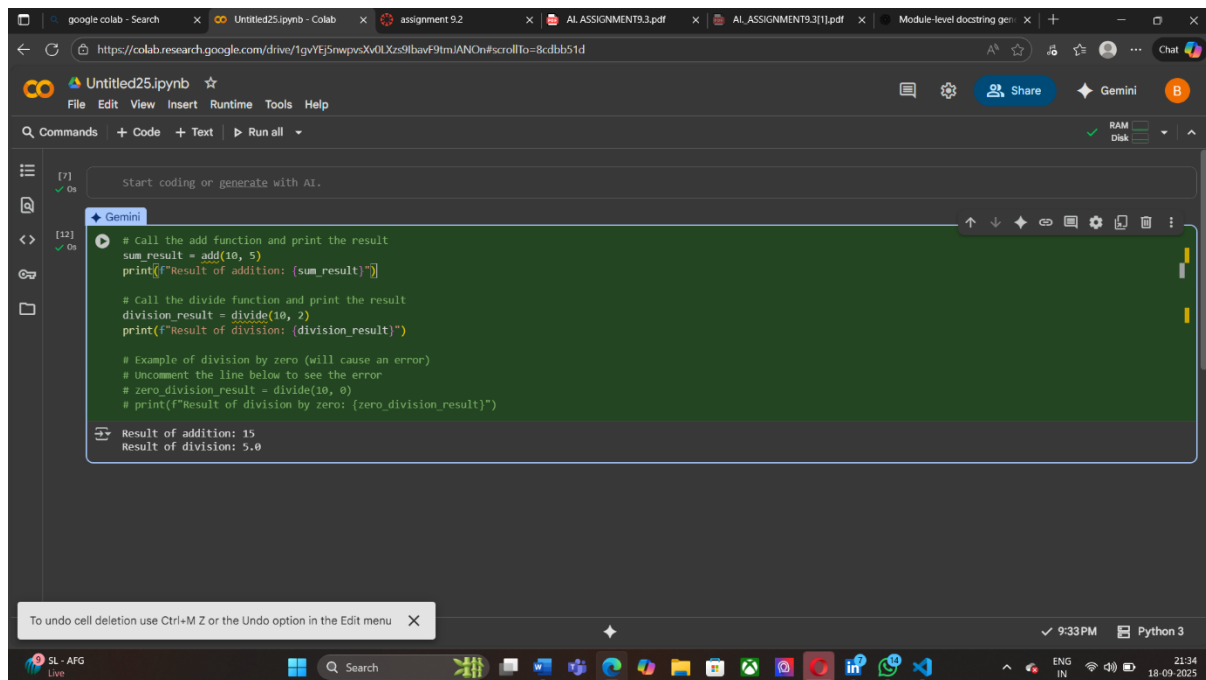
(Documentation – Convert Comments to Structured Docstrings)

- Task: Use AI to transform existing inline comments into structured function docstrings following Google style.
- Instructions:
 - o Provide AI with Python code containing inline comments.
 - o Ask AI to move relevant details from comments into function docstrings.
 - o Verify that the new docstrings keep the meaning intact while improving structure.
- Expected Output #4:
 - o Python code with comments replaced by clear, standardized docstrings.

Prompt:

Transform all existing inline comments in the following Python code into structured function docstrings using Google style.

- Move relevant details from comments into the function docstrings.
- Ensure the new docstrings keep the original meaning and improve structure.



The screenshot shows a Google Colab notebook interface. The top bar includes the Google Colab logo, a search bar, and several open tabs: 'google colab - Search', 'Untitled25.ipynb - Colab', 'assignment 9.2', 'AI_ASSIGNMENT9.3.pdf', 'AI_ASSIGNMENT9.3[1].pdf', and 'Module-level docstring gen...'. The main editor area displays a Python code cell with the following content:

```
start coding or generate with AI.

# Call the add function and print the result
sum_result = add(10, 5)
print(f"Result of addition: {sum_result}")

# Call the divide function and print the result
division_result = divide(10, 2)
print(f"Result of division: {division_result}")

# Example of division by zero (will cause an error)
# Uncomment the line below to see the error
# zero_division_result = divide(10, 0)
# print(f"Result of division by zero: {zero_division_result}")
```

Below the code cell, the output is displayed:

```
Result of addition: 15
Result of division: 5.0
```

The bottom status bar shows 'SL - AFG Live', a search bar, and system information: '9:33 PM', 'Python 3', and '18-09-2023'.

Task Description 5 :

(Documentation – Review and Correct Docstrings)

- Task: Use AI to identify and correct inaccuracies in existing docstrings.
- Instructions:
 - o Provide Python code with outdated or incorrect docstrings.
 - o Instruct AI to rewrite each docstring to match the current code behavior.
 - o Ensure corrections follow Google-style formatting.
- Expected Output #5:
 - o Python file with updated, accurate, and standardized docstrings.

Prompt:

Review the following Python code and identify any outdated or incorrect docstrings.

- Rewrite each docstring to accurately match the current code behavior.
- Ensure all docstrings follow Google-style formatting.

The screenshot shows a Google Colab notebook interface. The top bar includes the Google Colab logo, a search bar, and several open tabs: 'google colab - Search', 'Untitled25.ipynb - Colab', 'assignment 9.2', 'AI ASSIGNMENT9.3.pdf', 'AI_ASSIGNMENT9.3(1).pdf', and 'Module-level docstring gen...'. The main editor area displays a Python code cell with the following content:

```
[7] Start coding or generate with AI.
# Call the multiply function and print the result
product_result = multiply(7, 8)
print(f"Result of multiplication: {product_result}")

# Call the greet function and print the result
greeting_message = greet("Alice")
print(f"Greeting message: {greeting_message}")
```

Below the code, the output is displayed:

```
Result of multiplication: 56
Greeting message: Hello, Alice!
```

The bottom status bar shows 'Variables', 'Terminal', '9:38 PM', and 'Python 3'. The Windows taskbar is visible at the very bottom.

Task Description 6 :

(Documentation – Prompt Comparison Experiment)

- Task: Compare documentation output from a vague prompt and a detailed prompt for the same Python function.
- Instructions:
 - o Create two prompts: one simple (“Add comments to this function”) and one detailed (“Add Google-style docstrings with parameters, return types, and examples”).
 - o Use AI to process the same Python function with both prompts.
 - o Analyze and record differences in quality, accuracy, and completeness.
- Expected Output #6:
 - o A comparison table showing the results from both

Prompt:

Compare documentation output from a vague prompt and a detailed prompt for the same Python function:

- Vague prompt: “Add comments to this function.”

- Detailed prompt: “Add Google-style docstrings with parameters, return types, and examples.”
- Use AI to process the same function with both prompts.
- Analyze and record differences in quality, accuracy, and completeness in a comparison table with observations.

The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'google colab - Search', 'Untitled25.ipynb - Colab', 'assignment 9.2', 'AL ASSIGNMENT9.3.pdf', 'AL_ASSIGNMENT9.3(1).pdf', and 'Module-level docstring gen...'. The address bar shows a URL from colab.research.google.com.

The notebook content is as follows:

Define the detailed prompt

Subtask:

Create a detailed prompt asking for Google-style docstrings with specific elements (parameters, return types, examples).

Reasoning: Create a string variable `detailed_prompt` with the specified content and print it.

```
[19] ✓ 0s
detailed_prompt = """Generate a Google-style docstring for the following Python function, including the following sections:
Args: (describing parameters)
Returns: (describing the return value and its type)
Example: (providing a simple example of how to use the function)
following Python function:
"""
print(detailed_prompt)
```

Generate a Google-style docstring for the following Python function, including the following sections:
 Args: (describing parameters)
 Returns: (describing the return value and its type)
 Example: (providing a simple example of how to use the function)
 following Python function:

Generate documentation with the vague prompt

The bottom of the image shows a Windows taskbar with various application icons, a search bar, and system tray icons including the language indicator (ENG IN) and the time (21:42, 18-09-2025).