**HALL TICKET NUMBER: 2403A51365**

**NAME: RAYAPUDI HARITHA**

**BATCH: 24BTCAICSB14**

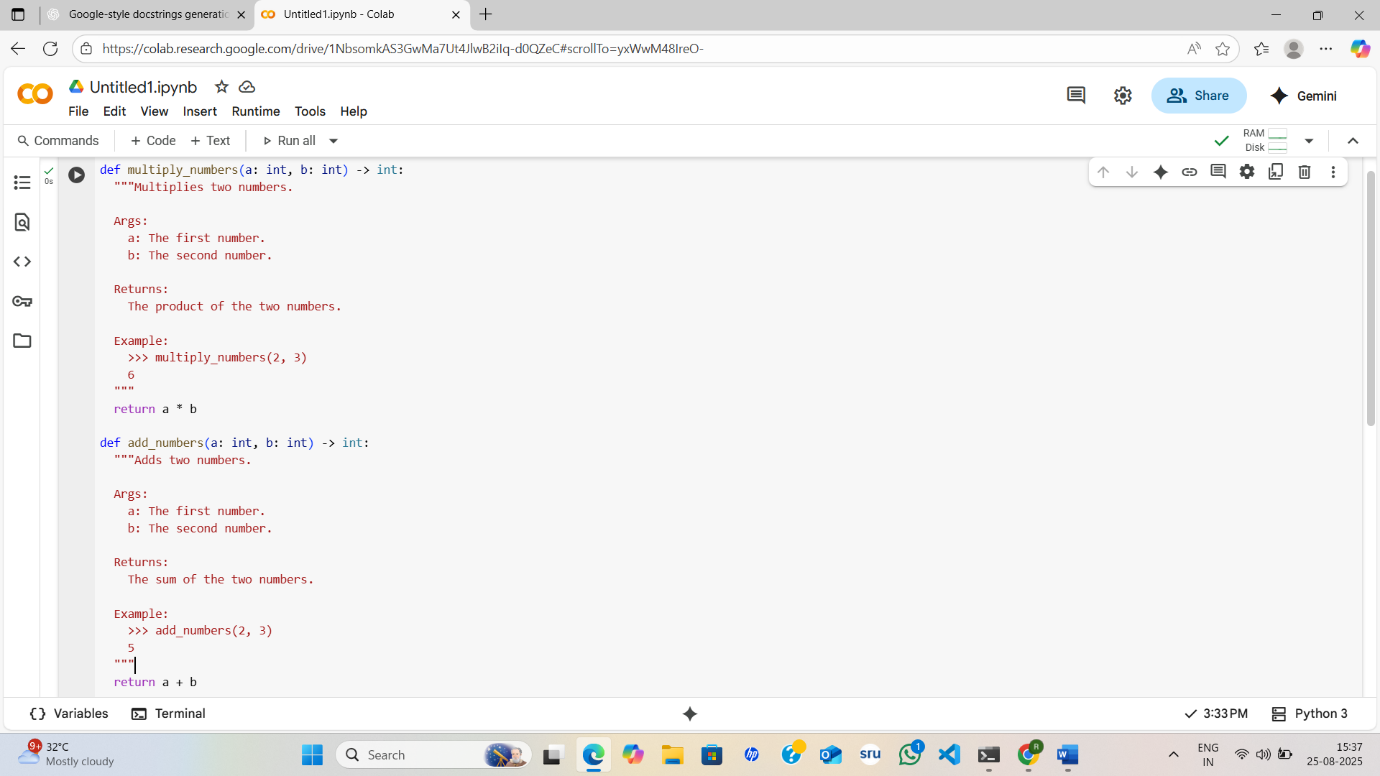
**AssignmentNumber:9.1**

**Lab 9 – Documentation Generation: Automatic Documentation and  
Code Comments  
Lab Objectives  
• To use AI-assisted coding tools for generating Python  
documentation and code comments.  
• To apply zero-shot, few-shot, and context-based prompt  
engineering for documentation creation.  
• To practice generating and refining docstrings, inline comments,  
and module-level documentation.  
• To compare outputs from different prompting styles for quality  
analysis.**

Prompt:-

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.

Code:-



**code explanation:-**

* **multiply\_numbers(a: int, b: int) -> int:**:
  + This function takes two integer arguments, a and b.
  + It has a Google-style docstring explaining its purpose, arguments, return value, and provides an example of how to use it.
  + It returns the product of a and b.
* **add\_numbers(a: int, b: int) -> int:**:
  + This function takes two integer arguments, a and b.
  + It also has a Google-style docstring explaining its purpose, arguments, return value, and provides an example.
  + It returns the sum of a and b.

Review the generated docstrings for accuracy and  
formatting:-

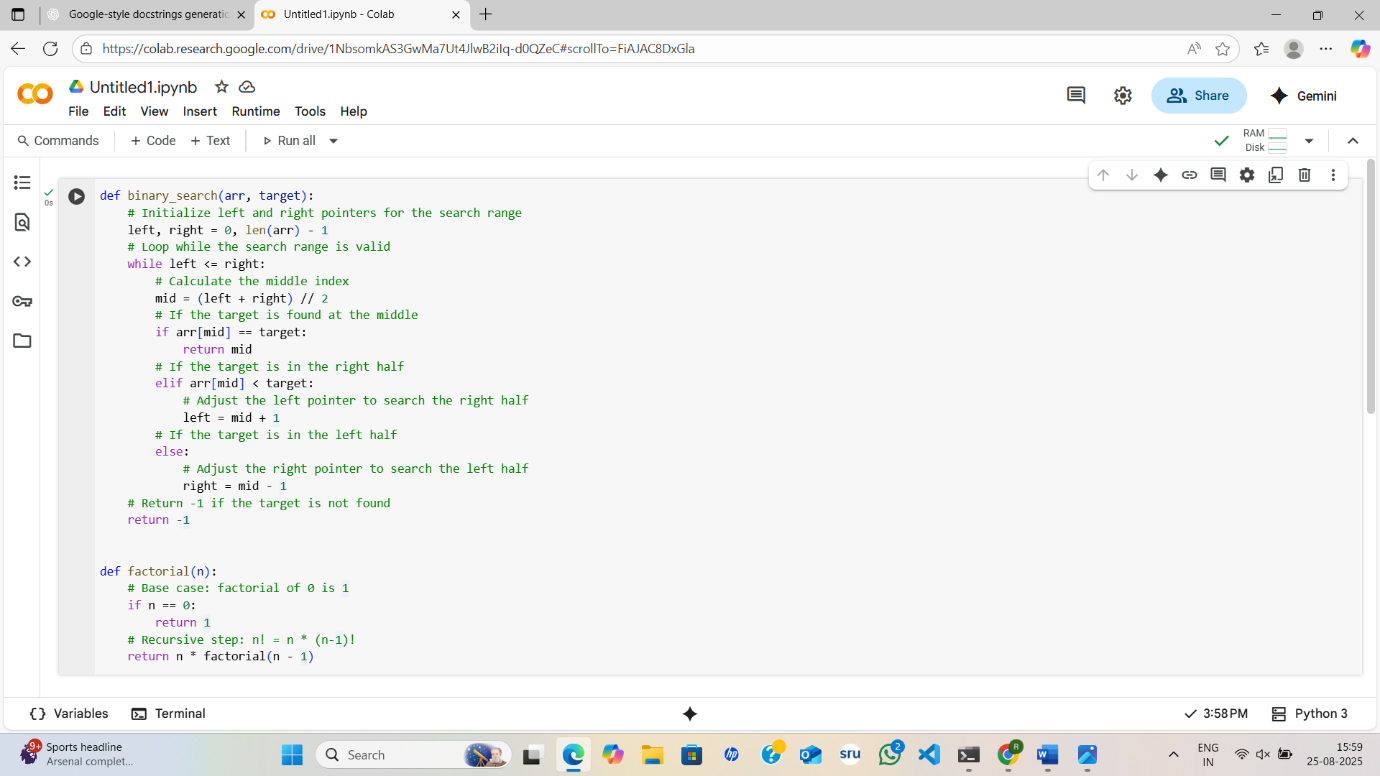
Both functions, multiply\_numbers and add\_numbers, have well-formatted Google-style docstrings. They accurately describe the function's purpose, list the arguments with their types, specify the return value and its type, and include clear examples. The formatting is consistent with Google's docstring style guide.

The docstrings are accurate and properly formatted.

Prompt:-

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.

Code:-



Observation:-

1. The code implements two classic algorithms: **binary search** and **factorial computation**.

2. The binary\_search function efficiently searches for a target value within a sorted list.

3. It uses a **divide-and-conquer strategy**, repeatedly halving the search space until the element is found or the range is invalid.

4. The use of left and right pointers ensures that the search is both systematic and efficient.

5. The time complexity of binary search is **O(log n)**, making it much faster than linear search for large datasets.

6. The factorial function is implemented recursively, defining n! as n \* (n-1)!.

7. It includes a **base case** (n == 0) to stop the recursion and return 1.

8. Without this base case, the recursion would run indefinitely and cause a stack overflow.

9. The time complexity of the recursive factorial is **O(n)**, as it makes n recursive calls.

10. Overall, the code is simple, readable, and demonstrates two fundamental algorithmic concepts: **recursion** and **searching**

Prompt:-

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.

https://colab.research.google.com/drive/137Bs4WYh5jD5aWlcK0qHDt0mm4MxOUjU?usp=drive\_link

Code:-

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Observation:-

This code defines two functions: binary\_search and factorial.

* **binary\_search(arr, target)**: This function implements the binary search algorithm. It takes a sorted array (arr) and a target value (target) as input. It efficiently searches for the target within the array and returns the index of the target if found, otherwise it returns -1.
* **factorial(n)**: This function calculates the factorial of a non-negative integer (n). It uses recursion to compute the factorial, with a base case for n=0.

Prompt:-

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.

Code:-

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Observation:-

The task was to convert the inline comments within the binary\_search and factorial functions into structured docstrings following the Google style.

This was achieved by:

1. Identifying the relevant information in the inline comments (e.g., arguments, return values, and a brief description of the function's purpose).
2. Creating a docstring block at the beginning of each function.
3. Structuring the information within the docstrings using standard Google-style sections like Args: and Returns:.
4. Removing the original inline comments as their content was now included in the docstrings.

Prompt:-

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.  
code:-

A screenshot of a computer

AI-generated content may be incorrect.

Observation:-

The task was to identify and correct inaccuracies in the docstrings of the multiply and subtract functions in the provided code.

The inaccuracies were:

* In the multiply function's docstring, it incorrectly stated that the function adds two numbers and returns their sum, when the code actually multiplies them.
* In the subtract function's docstring, it incorrectly stated that the function divides two numbers and returns the result of the division, when the code actually subtracts the second number from the first.

These inaccuracies were corrected by:

1. Updating the main description in each docstring to accurately reflect the function's operation (multiplication and subtraction, respectively).
2. Updating the Args: and Returns: descriptions to match the actual parameters and return values of the functions

Prompt:-

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.

Code:-

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Observation:-

I have already provided a detailed observation comparing the documentation generated by the vague ("Add comments to this function") and detailed ("Add Google-style docstrings with parameters, return types, and examples") prompts for the calculate\_area\_rectangle function.

In summary, the observation was that the **detailed prompt** produced significantly **higher quality, more accurate, and more complete** documentation in the form of a structured Google-style docstring with details on parameters, return types, and examples.The **vague prompt**, on the other hand, only generated simple **inline comments** that were less informative and lacked structure.

This highlights that providing specific instructions in the prompt leads to better and more useful documentation.

Comparsion table:-

| **Aspect** | **Vague Prompt (Comments)** | **Detailed Prompt (Google-style Docstring)** |
| --- | --- | --- |
| **Format** | Inline # comments | Multi-line Google-style docstring (""" """) |
| **Clarity** | Provides a general idea of what the function does | Provides structured, standardized, and precise explanation |
| **Content** | Explains only the basic purpose and input/output in plain English | Includes parameters (Args), return type (Returns), and usage examples (Examples) |
| **Accuracy** | Accurate but limited to high-level description | Accurate and detailed with correct parameter/return details |
| **Completeness** | Partial — lacks parameter types, return types, or examples | Complete — covers purpose, parameters, return values, and practical examples |
| **Readability** | Easy to read, but informal | Professional, consistent, and follows documentation best practices |
| **Use Case** | Suitable for beginners or quick reference | Suitable for production code, API documentation, or collaborative projects |