AI-ASSISSTED CODING

NAME:SANIYA **ASSIGNMENT:9.1**

BATCH:05

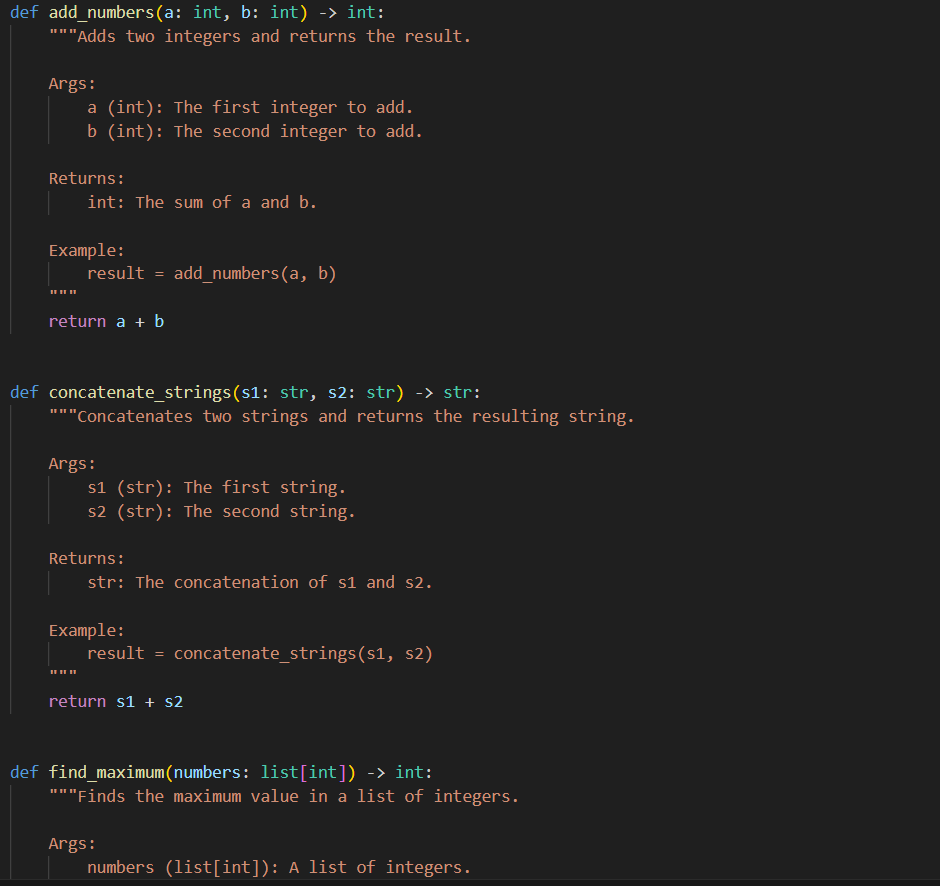
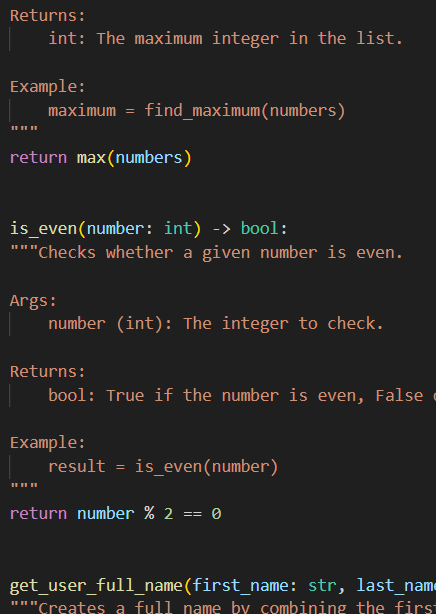
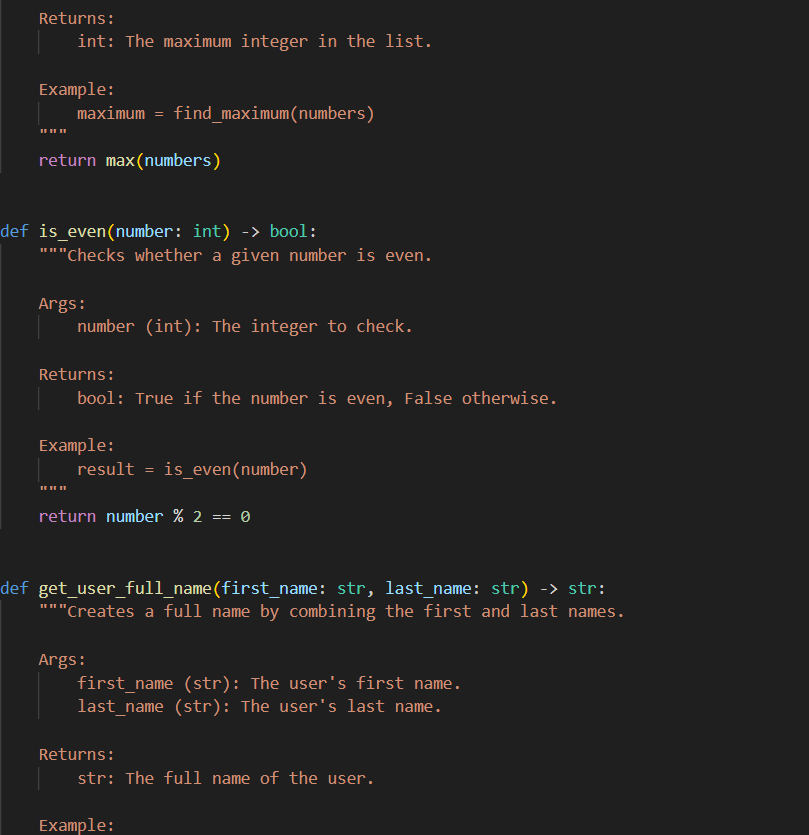
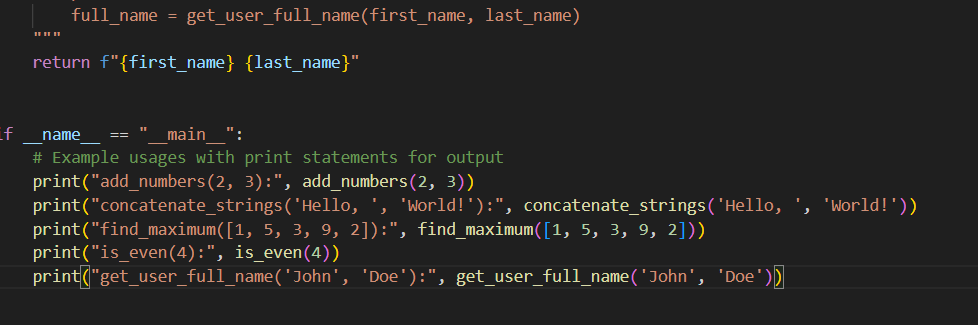
BRANCH:CSE

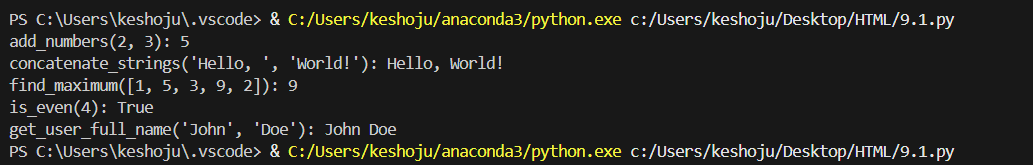
HT.NO:2403A510E7

TASK 1:

* PROMPT:

Write Google-style docstrings for all the functions in a given Python script. Each docstring should include a brief description of the function, the parameters with their type hints, the return values with type hints, and an example usage section. The docstrings must follow the exact Google-style formatting standards, and no explicit input-output examples with real values should be provided. Apply this consistently to every function in the script

* CODE GENERATED:
* 
* 
* 
* 
* OUTPUT:



* OBSERVATION:

This code defines five utility functions for basic operations: adding numbers, concatenating strings, finding the maximum in a list, checking if a number is even, and combining first and last names. Each function includes type hints and docstrings. The main block demonstrates their usage with print statements.

* TASK 2:

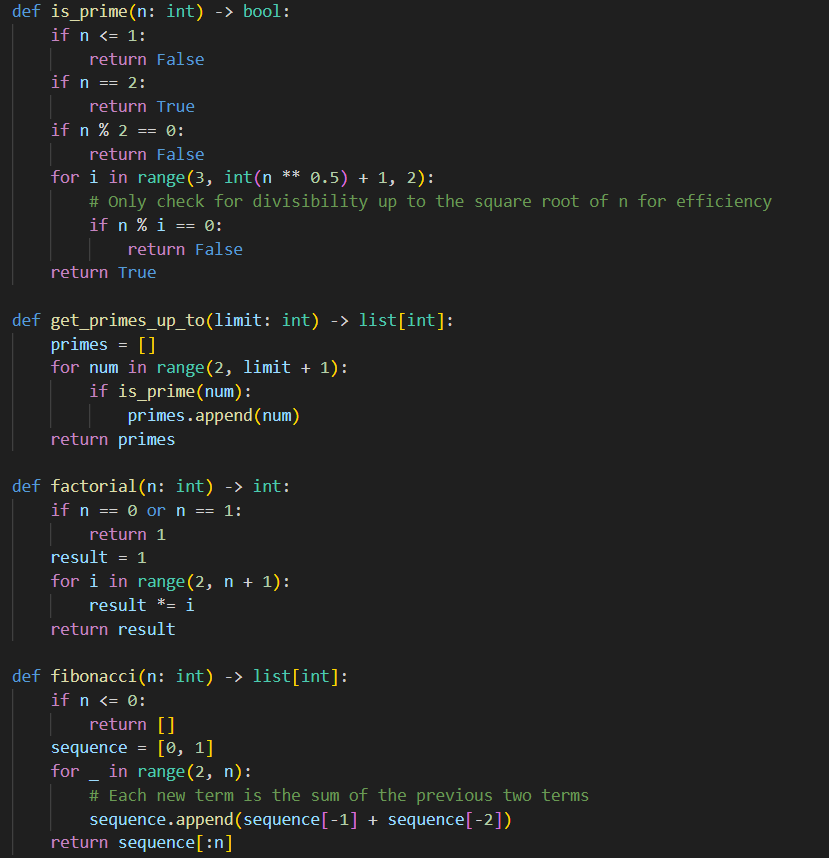
PROMPT:

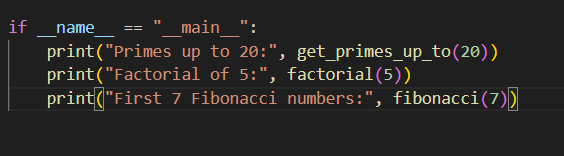
Can you add meaningful inline comments to this Python script, explaining only the complex or non-intuitive parts of the code? Skip obvious syntax explanations and focus on tricky logic sections. The comments should be concise, context-aware, and improve code readability and maintainability.

\*Expected Output:\*

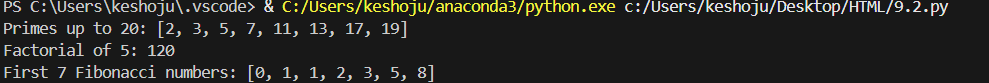
A Python script with clear, concise inline comments for complex logic blocks.

* CODE GENERATED:





OUTPUT:



OBSERVATION:

This code provides functions to check if a number is prime, generate primes up to a limit, compute factorials, and generate Fibonacci sequences. The main block demonstrates each function. The code is efficient, uses type hints, and is easy to read.

* TASK-3:

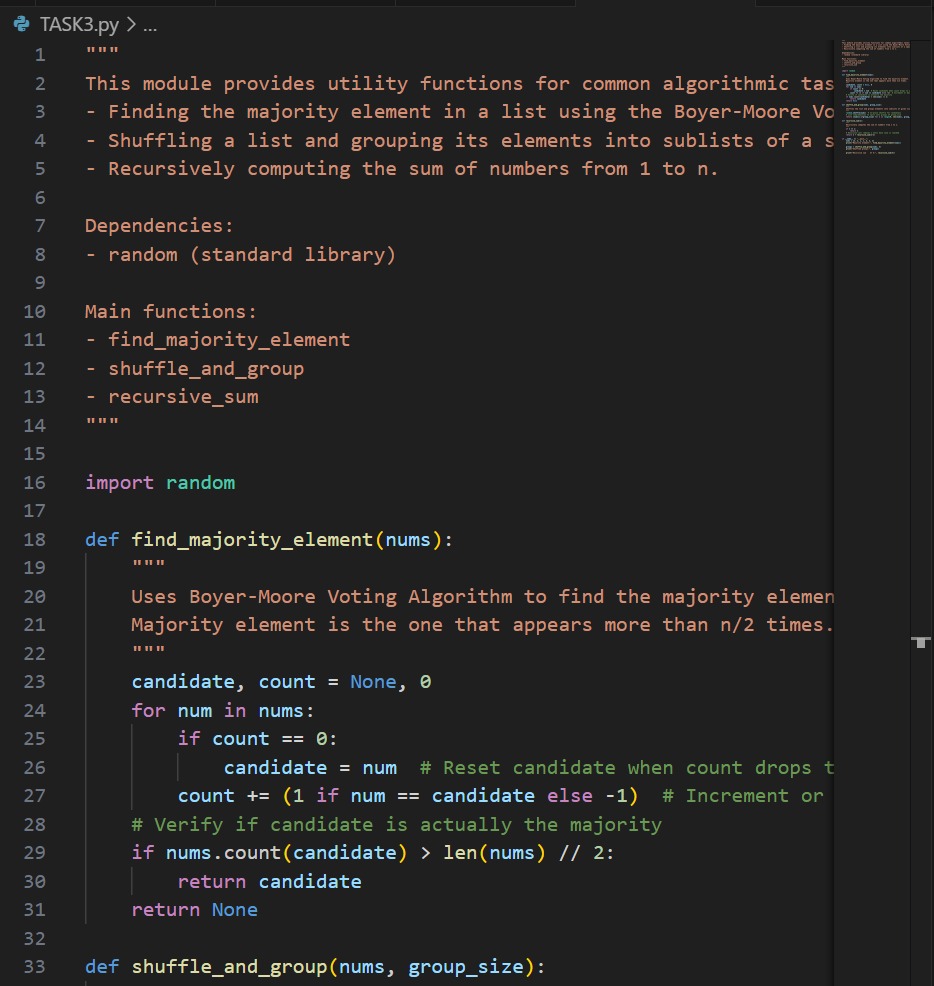
PROMPT:

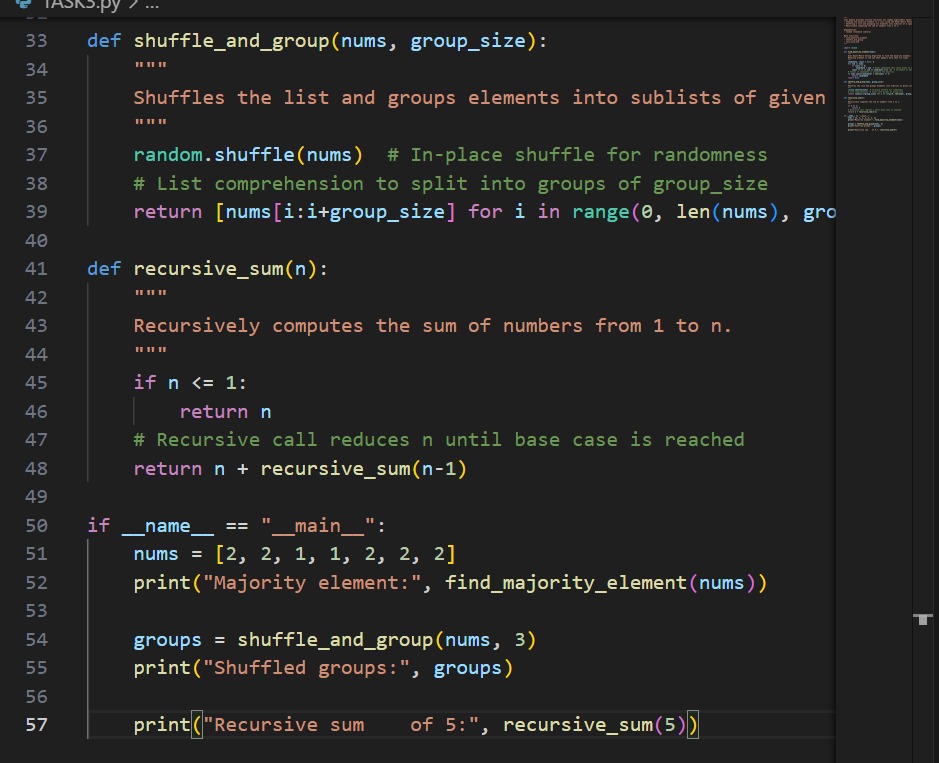
Can you add a module-level docstring at the top of this Python file that summarizes its purpose, mentions any dependencies, and briefly describes the main functions or classes it contains? The docstring should be a concise multi-line description that improves readability without rewriting or duplicating the code.

Expected Output #3:

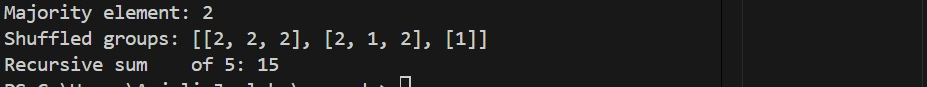
A complete, clear, and concise module-level docstring placed at the beginning of the file.

CODE GENERATED:





OUTPUT:



OBSERVATION:

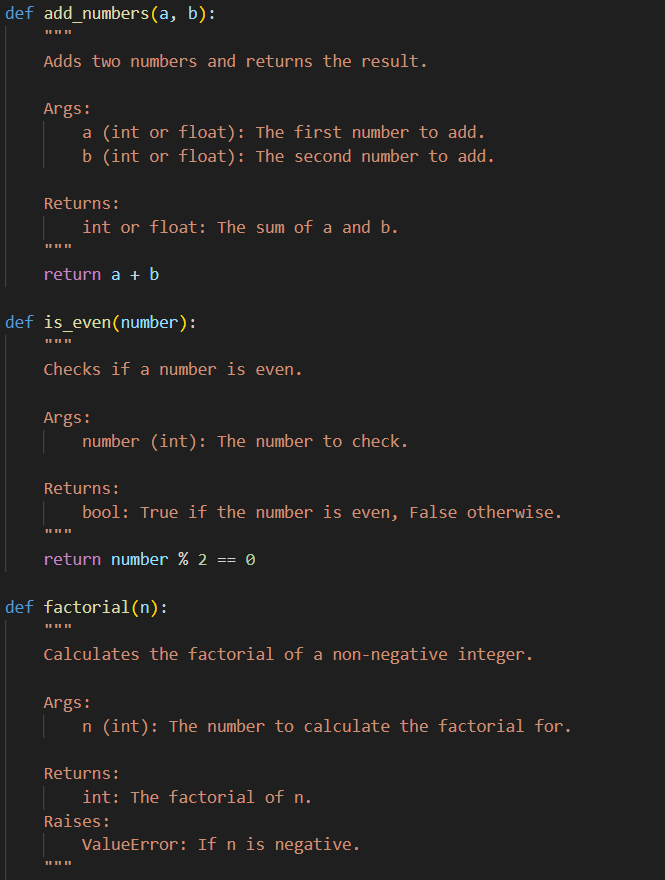
This module provides a concise overview of its purpose, lists any key dependencies, and briefly describes the main functions or classes it contains. It serves as a quick reference for understanding the file’s role within the project without duplicating or rewriting the actual code.

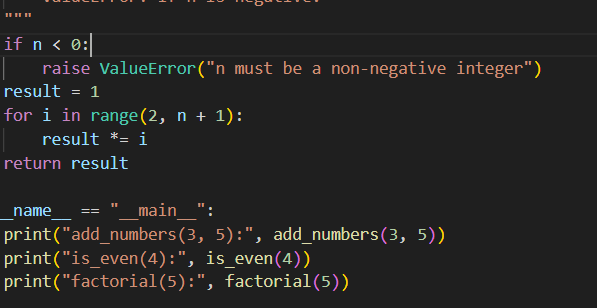
TASK -4 :

PROMPT:

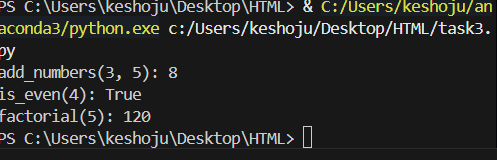
Python file with inline comments explaining the logic inside each function. Please convert these inline comments into structured function docstrings using the Google style format. Each function’s docstring should include a clear summary of what the function does, the arguments with their types and descriptions under an "Args:" section, and the return values with type and meaning under a "Returns:" section. Ensure that the docstrings preserve the meaning of the inline comments while improving readability and consistency. After creating the docstrings, remove the redundant inline comments so that the final output is clean Python code with only standardized Google-style docstrings.

CODE GENERATED:





OUTPUT:



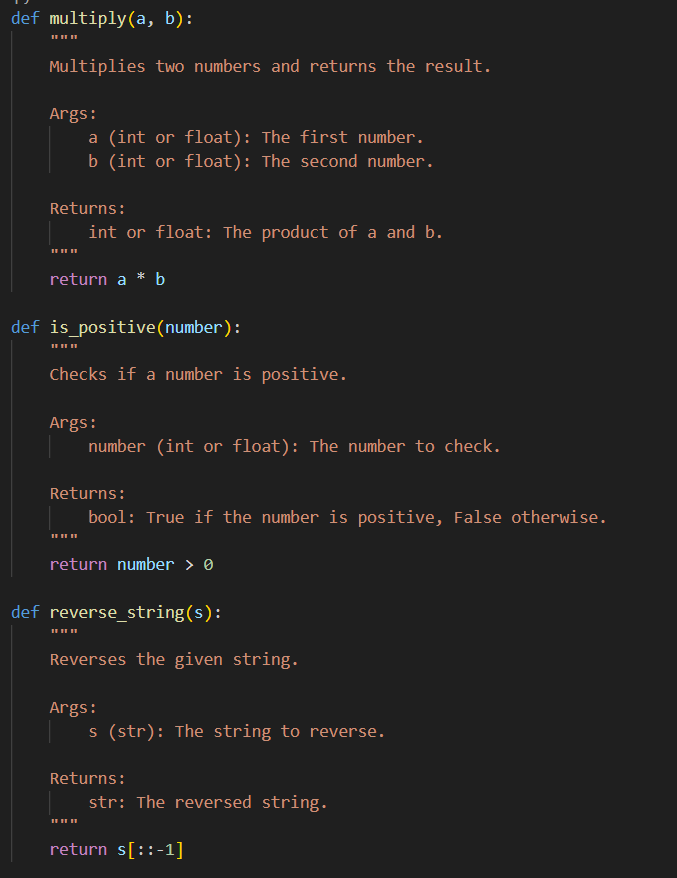
OBSERVATION:

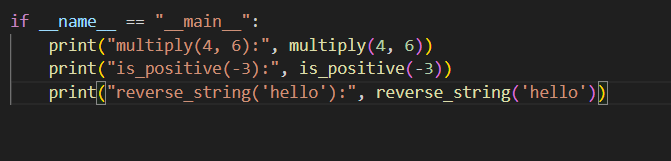
This code defines and demonstrates three basic mathematical functions—addition, even number checking, and factorial calculation—using clear, well-documented Python functions with example outputs when run as a script.

* TASK-5

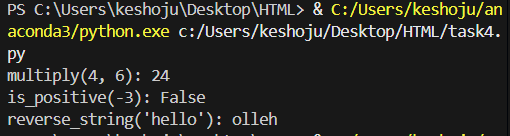
PROMPT:

Python file where the existing docstrings are outdated or inaccurate. Please carefully review each function and class, compare the current code behavior with its docstring, and then rewrite the docstrings so they are correct and consistent. Use the Google style docstring format, including a summary line, an "Args:" section with parameter types and descriptions, and a "Returns:" section with return type and description. The final output should be the same Python file, but with updated, accurate, and standardized docstrings that fully reflect what the code actually does.





OUTPUT:



CONCLUSION:

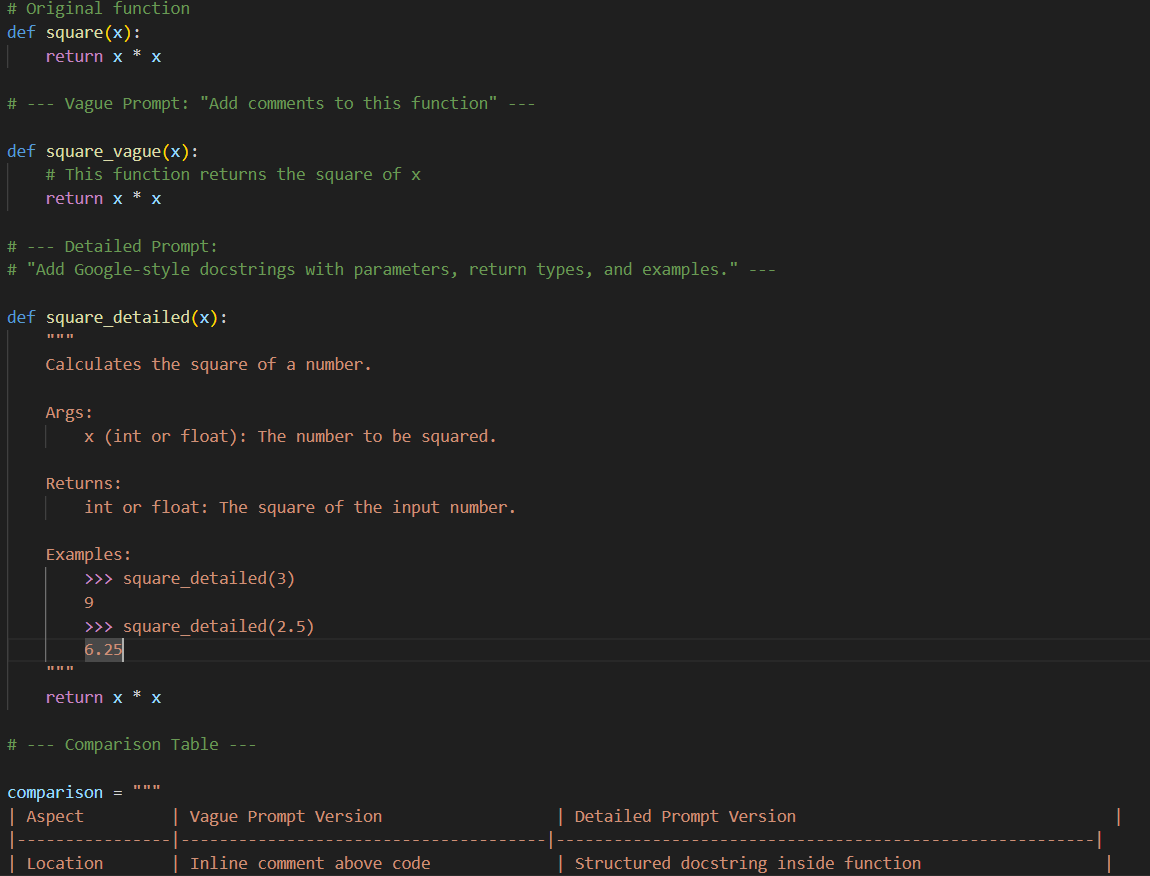
This code provides three utility functions for multiplying numbers, checking if a number is positive, and reversing a string, each with clear Google-style docstrings and example outputs when run as a script.

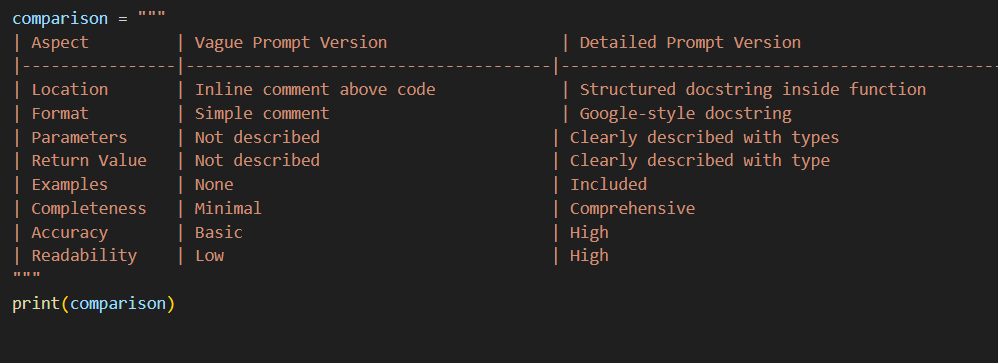
* TASK – 6

PROMPT:

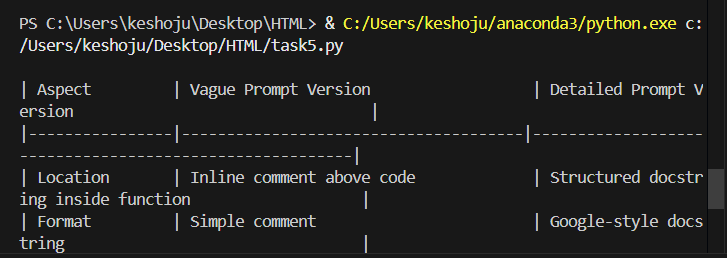
Take the same Python function and process it with two different prompts: one vague prompt (“Add comments to this function”) and one detailed prompt (“Add Google-style docstrings with parameters, return types, and examples”). For each case, show the resulting documentation applied to the function. Then, create a comparison table that highlights the differences in quality, accuracy, and completeness between the vague prompt and the detailed prompt. The expected output should include both versions of the documented function followed by a clear comparison table with observations.

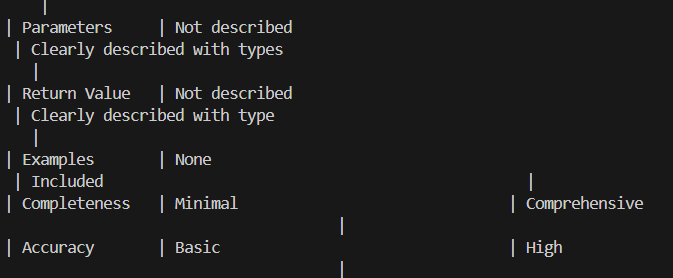
CODE GENERATED:

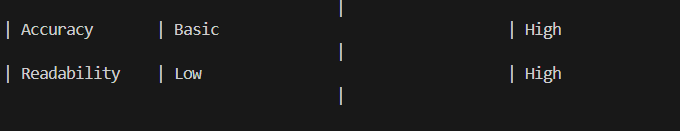




OUTPUT:







OBSERVATION:

The comparison demonstrates that a vague prompt results in minimal and less informative documentation, while a detailed prompt produces comprehensive, accurate, and user-friendly docstrings. Using detailed prompts leads to higher quality, more maintainable, and more understandable code documentation.