AI ASSISTED CODING

LAB:9.2

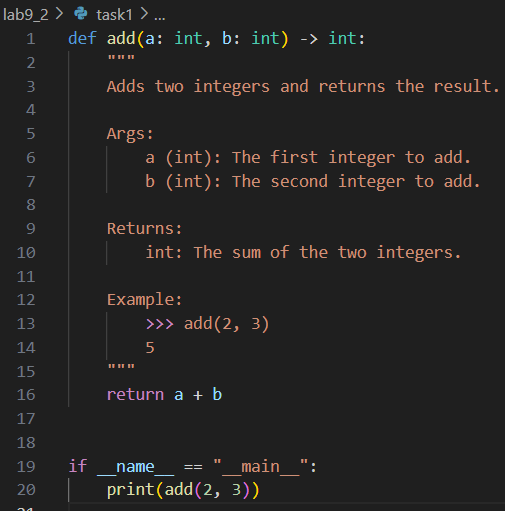
ROLLNO:2403A52096

BATCH:04

TASK1:

PROMPT: Add Google-style docstrings (with description, parameter types, return type, and example usage) to all functions in the Python script.

CODE:



OUTPUT:



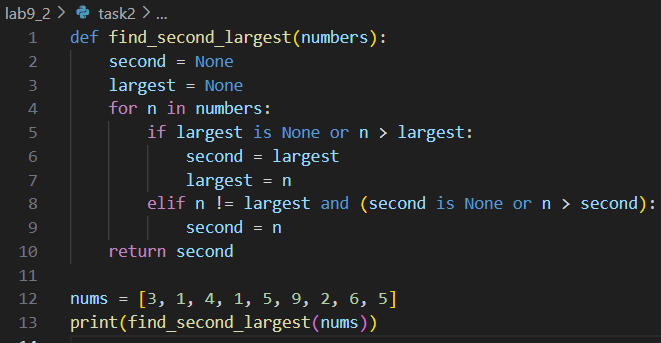
OBSERVATION:

The Python script defines a single function, add, which takes two integers and returns their sum. The function is documented with a clear Google-style docstring, including a description, parameter types, return type, and example usage. When the script is run directly, it prints the result of add(2, 3), which is 5. The code is simple, well-documented, and demonstrates best practices for function documentation.

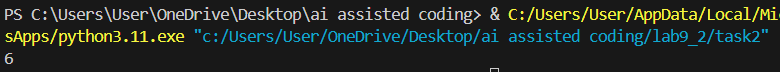
TASK2:

PROMPT: Add concise inline comments to the following Python code, explaining only complex or non-intuitive logic. Skip obvious syntax and focus on tricky parts to improve readability and maintainability.

CODE:



OUTPUT:



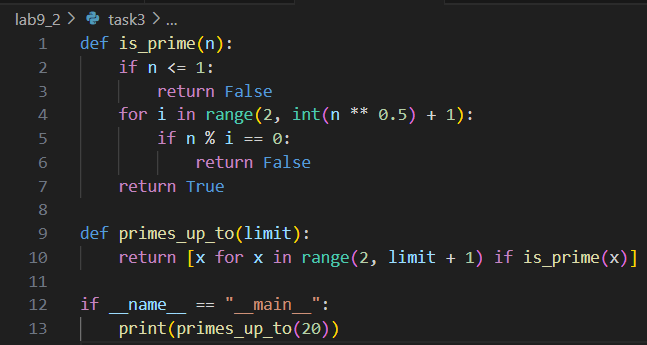
OBSERVATION:

The script defines a function to find the second largest number in a list. The logic for updating the second and largest variables within the loop is non-trivial and could benefit from inline comments. Currently, the code is clear for experienced programmers but may be confusing for beginners without comments explaining the update conditions for second and largest. Adding targeted comments to these sections would improve readability and maintainability.

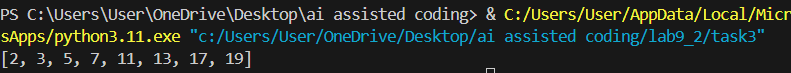
TASK3:

PROMPT: Add a clear, multi-line module-level docstring at the top of this Python file, summarizing its purpose, dependencies, and main functions/classes. Do not rewrite the code.

CODE:



OUTPUT:



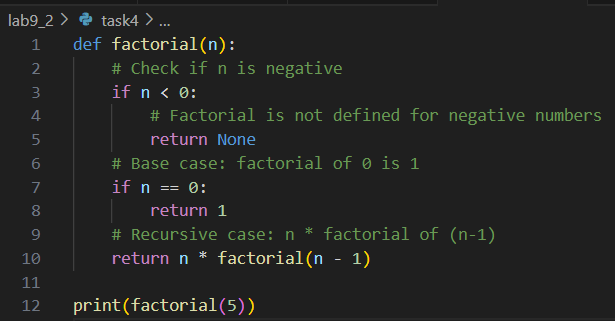
OBSERVATION:

The script  provides two main functions: is\_prime, which checks if a number is prime, and primes\_up\_to, which generates a list of all prime numbers up to a given limit. The script does not use any external dependencies and includes a main block that prints all primes up to 20. A module-level docstring summarizing these points would improve clarity and provide context for users of the module.

TASK4:

PROMPT: Convert the inline comments in this Python code into structured Google-style docstrings for each function, preserving all important details and improving documentation clarity.

CODE:



OUTPUT:



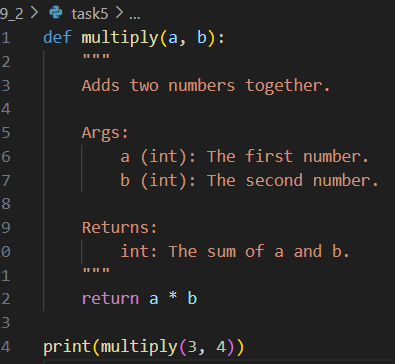
OBSERVATION:

The script defines a factorial function with several inline comments explaining negative input handling, the base case, and the recursive case. These comments provide useful information about the function's logic. Converting these inline comments into a structured Google-style docstring would improve the documentation by centralizing all relevant details at the top of the function, making the code cleaner and easier to maintain.

TASK5:

PROMPT: Review and correct all docstrings in this Python code to ensure they accurately describe the current function behavior, using clear and standardized Google-style formatting.

CODE:



OUTPUT:



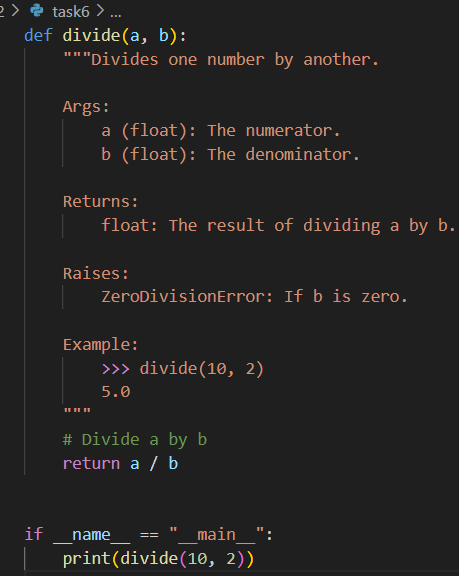
OBSERVATION:

The function multiply has a docstring that incorrectly states it "adds two numbers together," while the function actually multiplies its arguments. The docstring also incorrectly describes the return value as the sum instead of the product. This docstring should be updated to accurately reflect the function's behavior, following Google-style formatting.

TASK6:

PROMPT:Add a Google-style docstring to this function, including a description, parameter types, return type, raised exceptions, and an example usage.

CODE:



OUTPUT:



OBSERVATION:

The detailed prompt ("Add Google-style docstring...") produced a comprehensive docstring with a clear description, parameter and return type annotations, exception information, and an example. The detailed prompt leads to much higher quality, accuracy, and completeness in documentation, making the code easier to understand and maintain.