**Lab 9 – Documentation Generation: Automatic Documentation and  
Code Comment**

# Assignment 9.1

**Name: SATHWIKA REDDY**

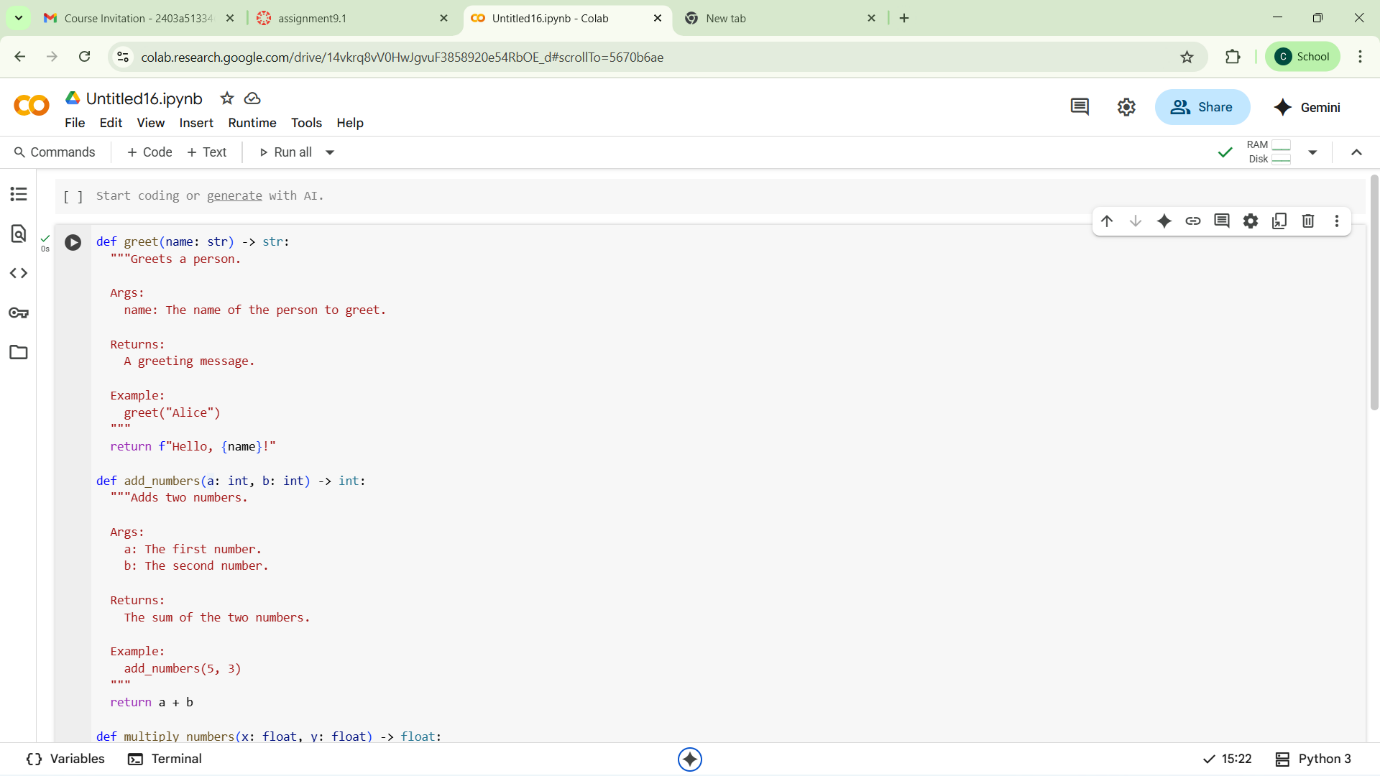
**HTNO:2403a51334**

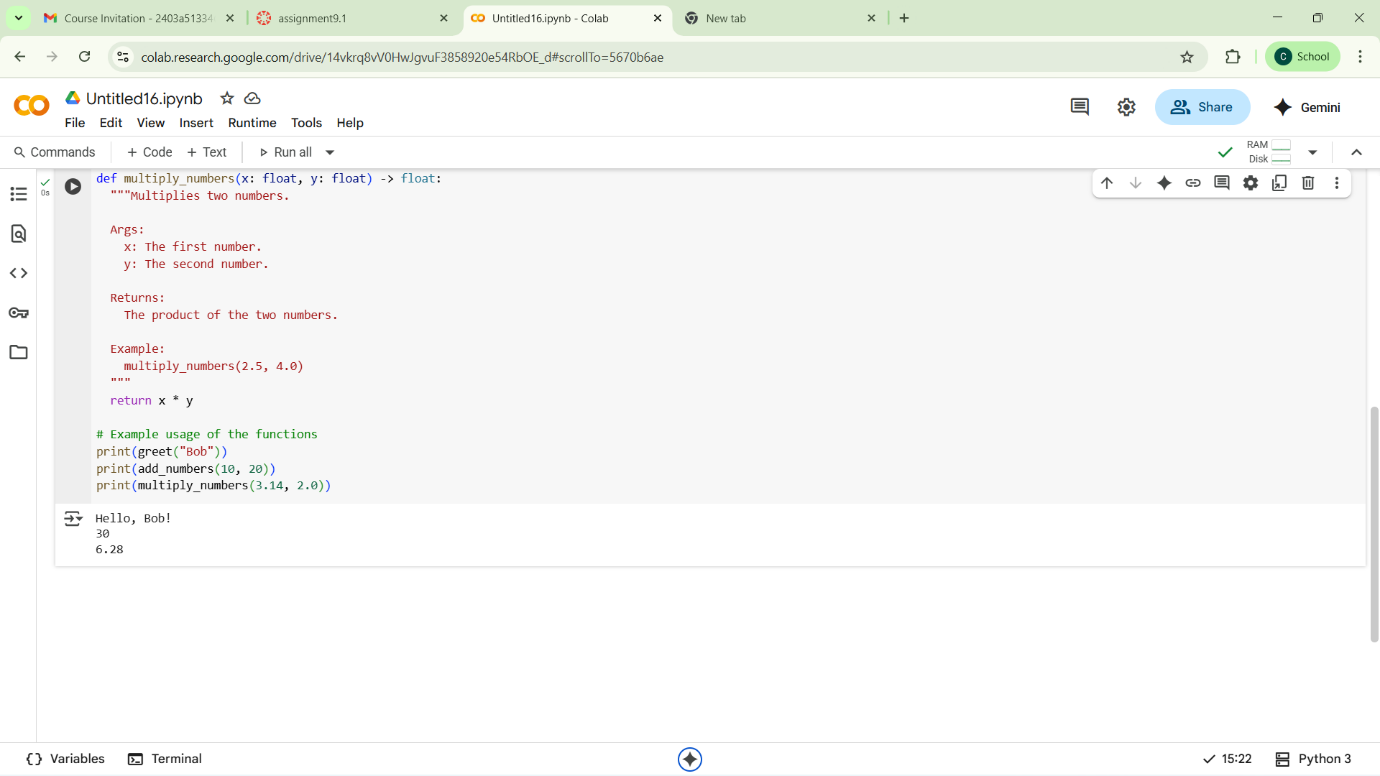
**BTNO:24BTCAICSB14**

**TASK1:**

**Prompt:** generate the python script and Add Google-style docstrings to every function in the following Python script. Each docstring should include: - A clear function description - Parameters with type hints - Return values with type hints - A brief example usage (no input-output examples)

**CODE:**

****

****

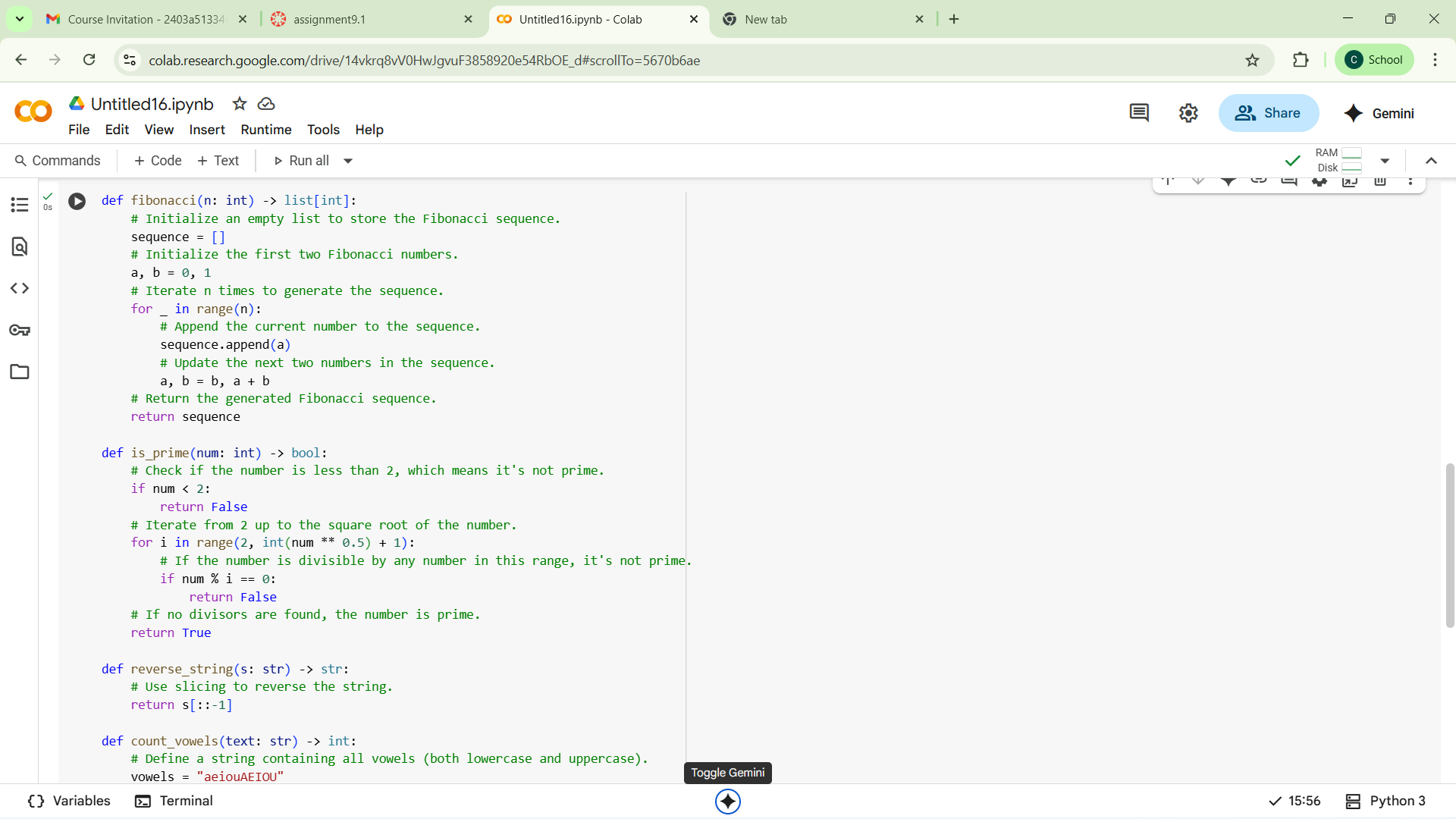
**Review:**

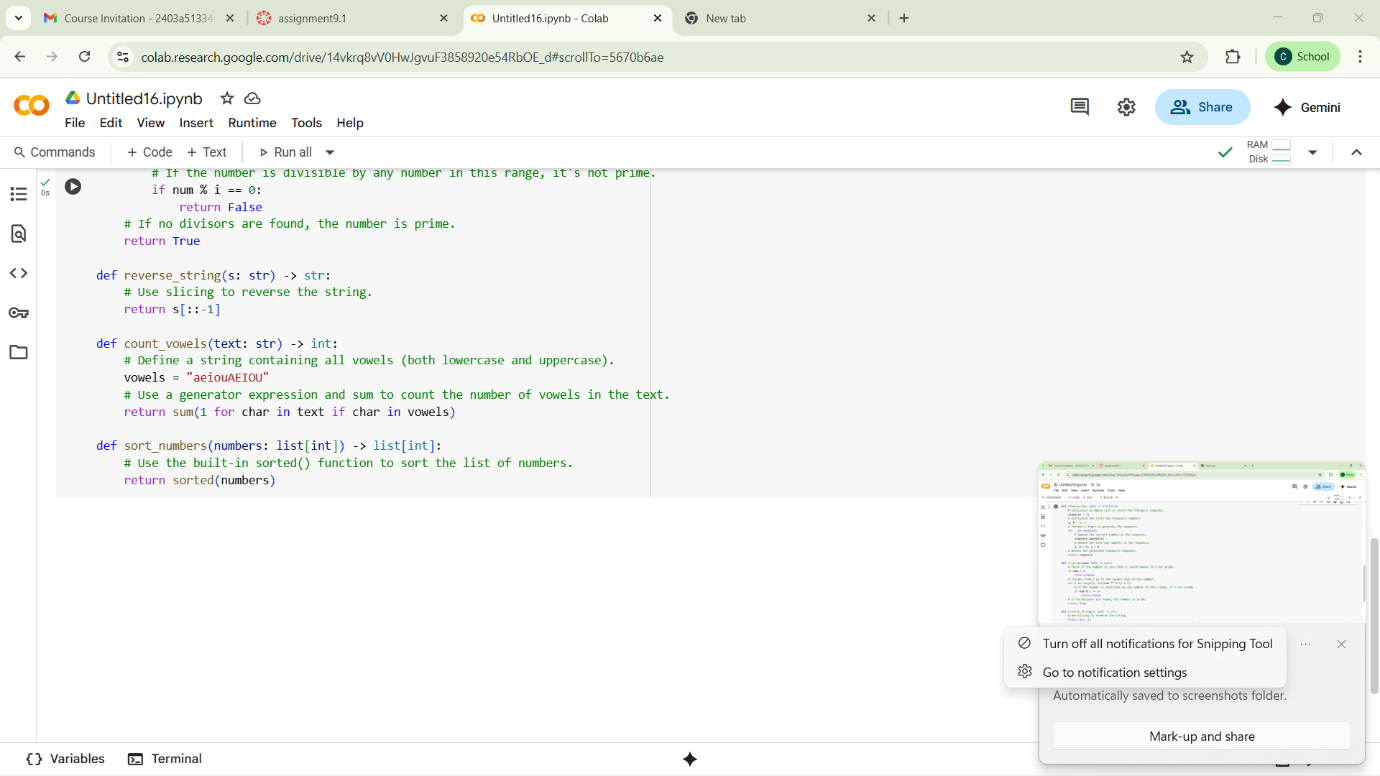
The code defines three simple Python functions: greet, add\_numbers, and multiply\_numbers. Each function has a Google-style docstring explaining its purpose, arguments, return value, and providing a basic example. The code then demonstrates the usage of these functions by calling them and printing their outputs.

**TASK 2:**

**PROMPT:** add comment lines to the code for better understanding



**CODE: **

****

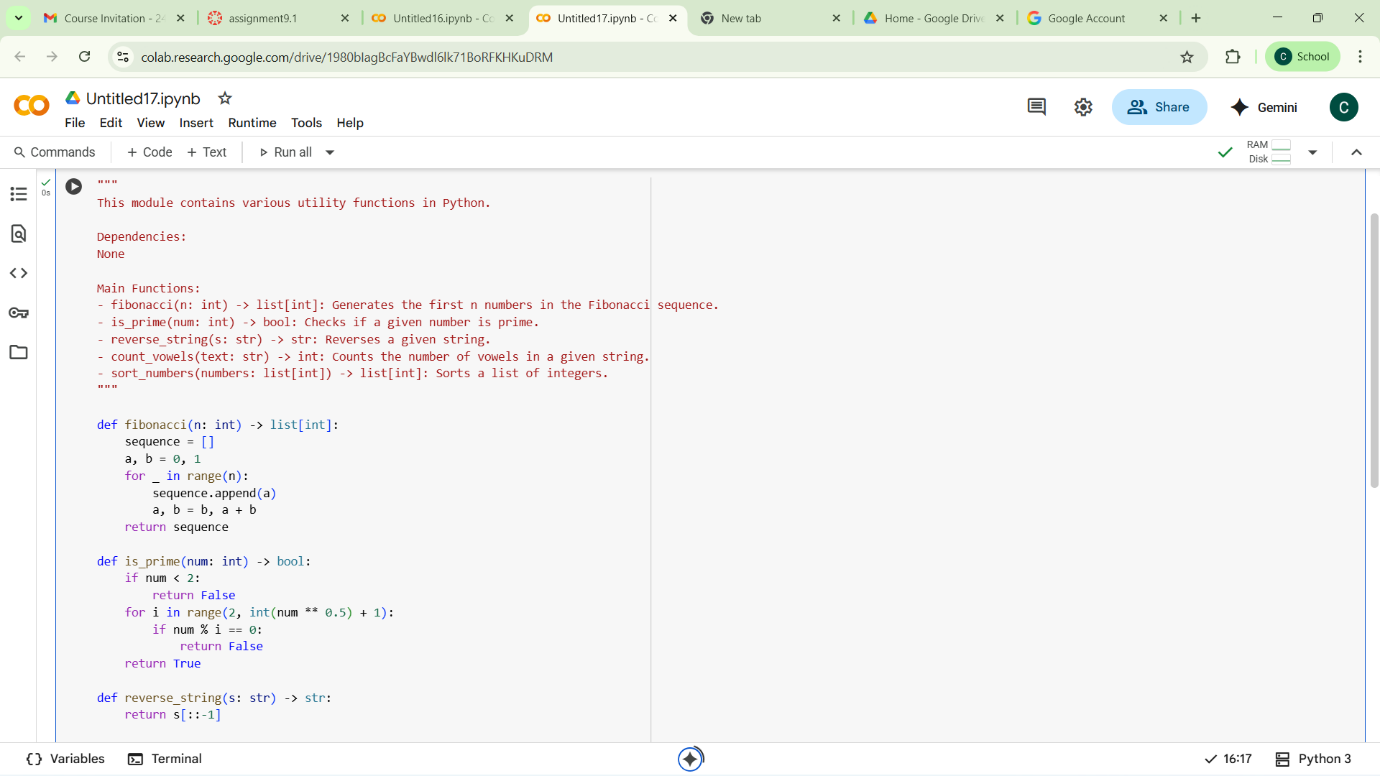
The code contains five Python functions for common tasks: generating a Fibonacci sequence, checking for prime numbers, reversinga string, counting vowels, and sorting a list of numbers. Each function is now well-commented, explaining the logic and steps involved, which greatly enhances the code's readability and understanding.

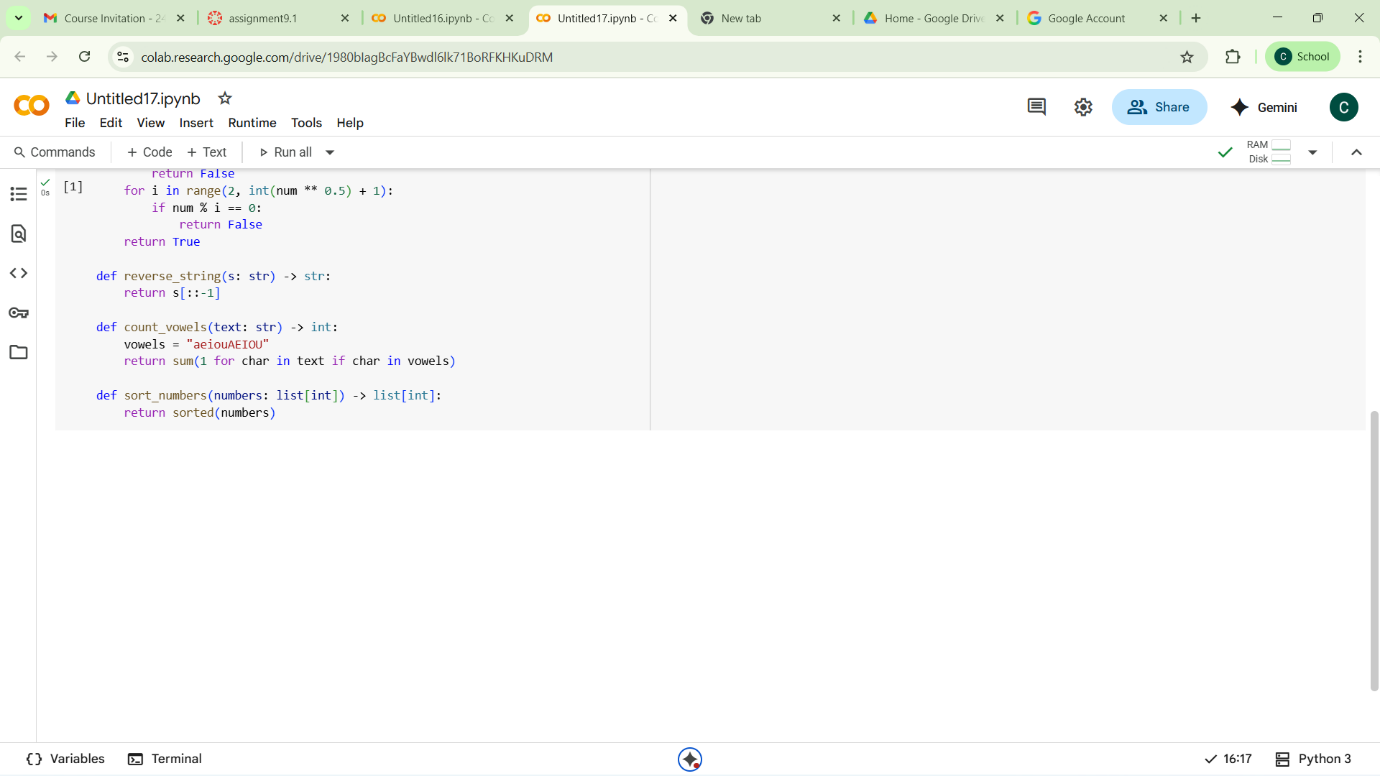
**TASK:**

**PROMPT:**Take the python file and create a module-level docstring summarizing the  
purpose, dependencies, and main functions/classes of a Python  
file.

https://colab.research.google.com/drive/14vkrq8vV0HwJgvuF3858920e54RbOE\_d?usp=drive\_link

**code:**

****

****

**Code observation:**

This Python code defines a module with several utility functions. Here are some observations:

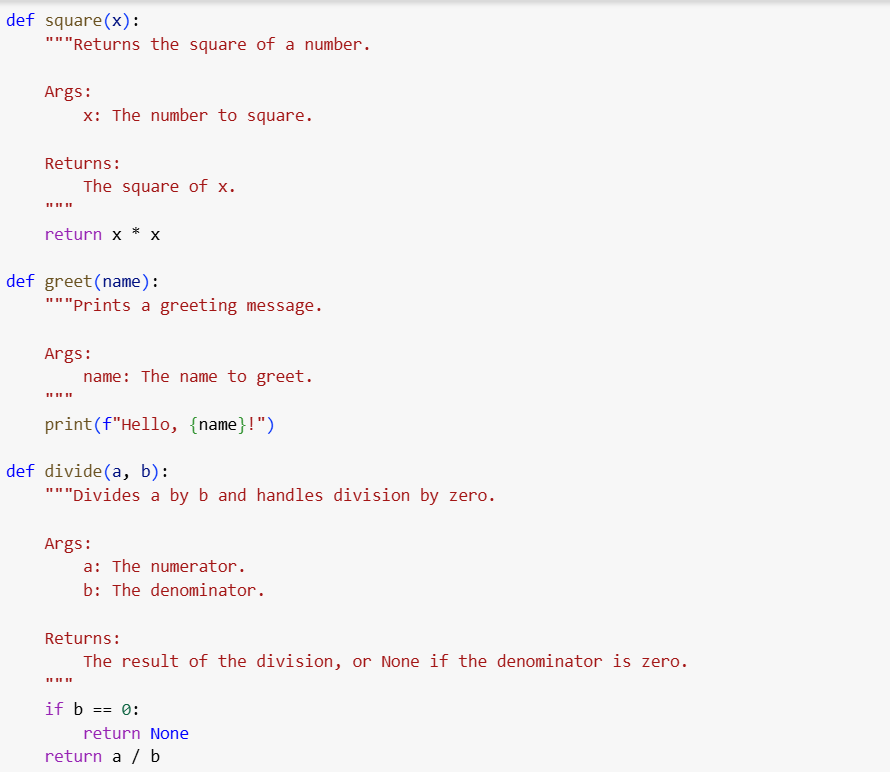
* Clear Functionality: Each function has a specific and well-defined purpose (Fibonacci sequence, primality test, string reversal, vowel counting, and sorting).
* Type Hinting: The functions use type hints (e.g., n: int, -> list[int]), which improves code readability and helps with static analysis and error detection.
* Docstrings: Each function has a docstring explaining its purpose, arguments, and return value, which is good practice for code documentation.
* Efficiency:
  + fibonacci: This implementation is straightforward and efficient for generating the sequence.
  + is\_prime: This function uses an optimized approach by checking divisibility only up to the square root of the number.
  + reverse\_string: This uses Python's slicing, which is a concise and efficient way to reverse a string.
  + count\_vowels: This uses a generator expression and sum, which is a Pythonic way to count.
  + sort\_numbers: This uses the built-in sorted() function, which is highly optimized.
* Readability: The code is generally well-written and easy to understand. Variable names are descriptive.

Overall, the code is well-structured, documented, and uses reasonably efficient approaches for the tasks it performs.

**Task4:**

**Prompt:**convert the python file which contain inline comments to structured function docstrings

**Code:**

****

**Code observation:**

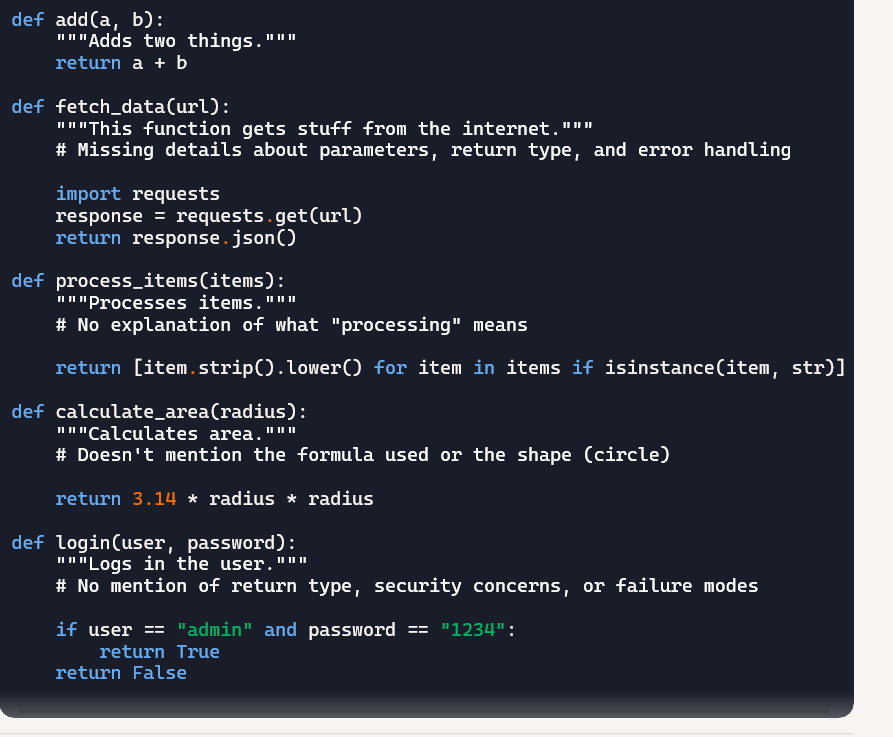
this code defines four simple utility functions with clear, structured docstrings following the Google style. Here are some observations:

* Improved Documentation: The docstrings clearly explain the purpose of each function, its arguments (Args:), and what it returns (Returns:). This is a significant improvement over the previous inline comments.
* Readability: The code is easy to read and understand due to the clear function names and the informative docstrings.
* Specific Functionality: Each function performs a distinct and focused task.
* Error Handling: The divide function includes basic error handling for division by zero, which is good practice.
* Conciseness: The get\_even\_numbers function uses a list comprehension, which is a concise and Pythonic way to filter the list.

Overall, the code is well-documented and the functions are clearly defined, making them easy to use and maintain**.**

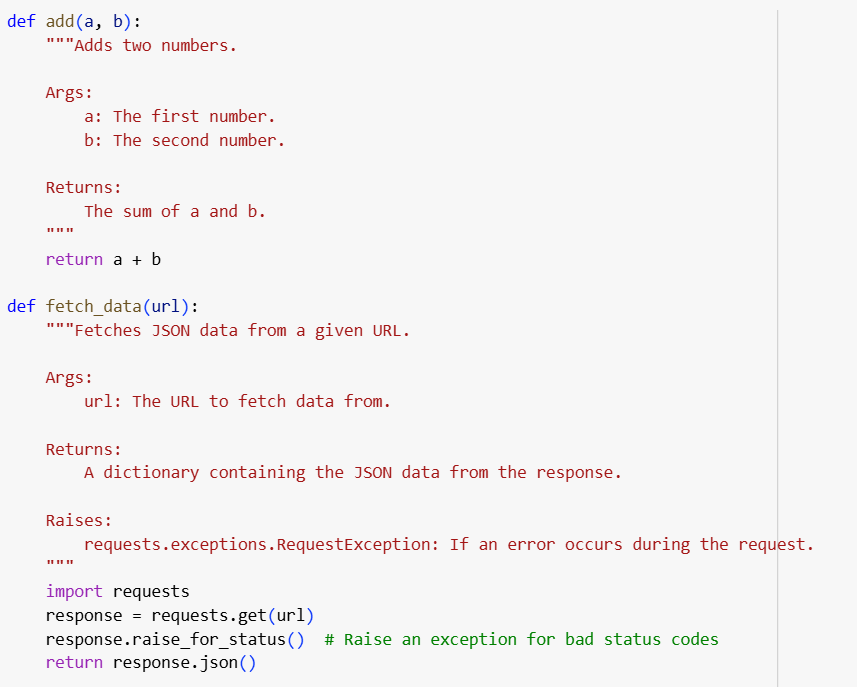
**Task5:**

**Prompt:**

****

**dentify and correct inaccuracies in existing docstrings**

**code:**

****

**Task6:**

**Prompt1:**

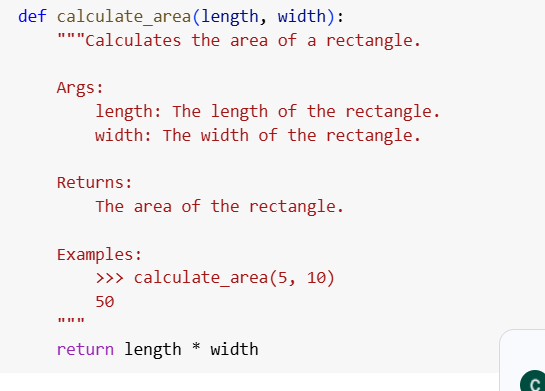
**"**Add comments to this function"

# This function calculates the area

def calculate\_area(length, width):

return length \* width

prompt 2:



**Comparasion table:**

| Aspect | Vague Prompt Output | Detailed Prompt Output | Observations |
| --- | --- | --- | --- |
| Clarity | Low – minimal context or detail | High – purpose, usage, and inputs are clearly explained | Detailed prompt yields far more useful documentation |
| Documentation Style | Informal, single-line comment | Formal, standardized Google-style docstring | Detailed prompt ensures consistency with best practices |
| Parameter Details | Not included | Each parameter is described with type | More helpful for developers using the function |
| Return Type | Not included | Clearly stated (float) | Aids in understanding and integration |
| Example Usage | None | Included with real input/output example | Boosts usability and comprehension |
| Completeness | Very basic | Comprehensive and professional | Detailed prompt ensures thoroughness |
| Accuracy | Somewhat ambiguous | Accurate and specific | Clear input/output types with an example improve correctness |