AI-Assisted Coding

Lab Assignment=9.1

Roll no: 2403A510D1
Name: Likhitha Pothunuri
Batch no: 05
CSE 2nd year

Task 1:

Task: Use AI to add Google-style docstrings to all functions in a given Python script.

Prompt:

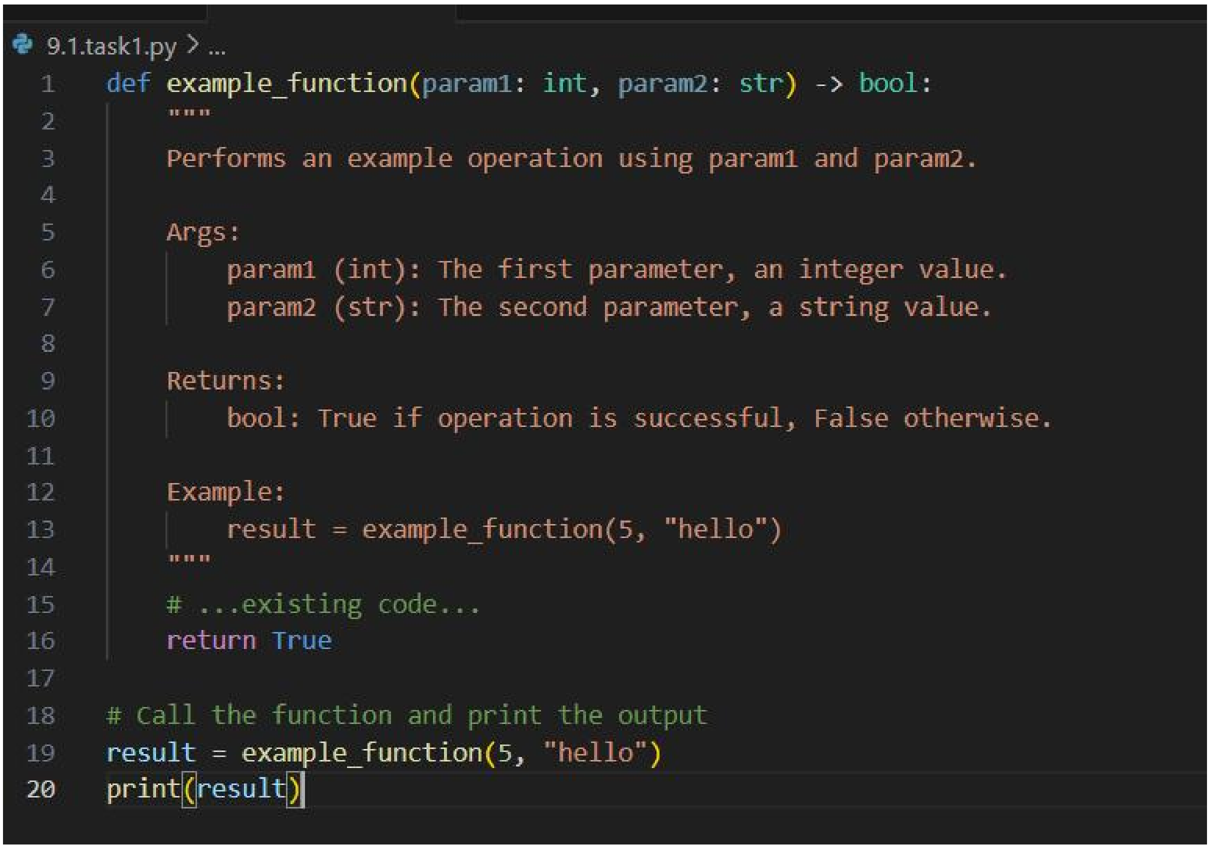
Add Google-style docstrings to all functions in the following Python script.

Each docstring must include:

- Function description
- Parameters with type hints
- Return values with type hints
- Example usage (without input/output examples)

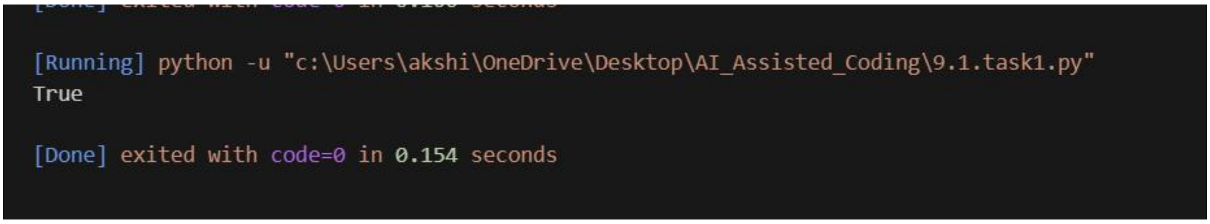
Do not change the function logic.

Code Generated:



```python
def example_function(param1: int, param2: str) -> bool:
    """
    Performs an example operation using param1 and param2.

    Args:
        param1 (int): The first parameter, an integer value.
        param2 (str): The second parameter, a string value.

    Returns:
        bool: True if operation is successful, False otherwise.

    Example:
        result = example_function(5, "hello")
    """
    # ...existing code...
    return True

# Call the function and print the output
result = example_function(5, "hello")
print(result)
```

Output:



```
[Running] python -u "c:\Users\akshi\OneDrive\Desktop\AI_Assisted_Coding\9.1.task1.py"
True

[Done] exited with code=0 in 0.154 seconds
```

Observation:

Function Purpose

- The function example_function is designed as a placeholder example.
- It takes two parameters (param1: int and param2: str) and returns a boolean value (True in this case).

Docstring Quality

- The docstring follows **Google-style formatting**.
- It clearly documents the function's purpose, parameters (with type hints), return type, and includes an example usage.
- This makes the code more **readable** and **maintainable**.

Code Behavior

- The function currently always returns True, regardless of the input values.
- No actual operation is performed using param1 and param2.

Execution

- When the script runs, it calls example_function(5, "hello").
- The result (True) is stored in result and printed to the console.

☑ In summary: The code demonstrates how to write and use a properly formatted **Google-style docstring**, but the function logic is only illustrative (returns a constant True).

Task 2 (Documentation – Inline Comments for ComplexLogic)

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

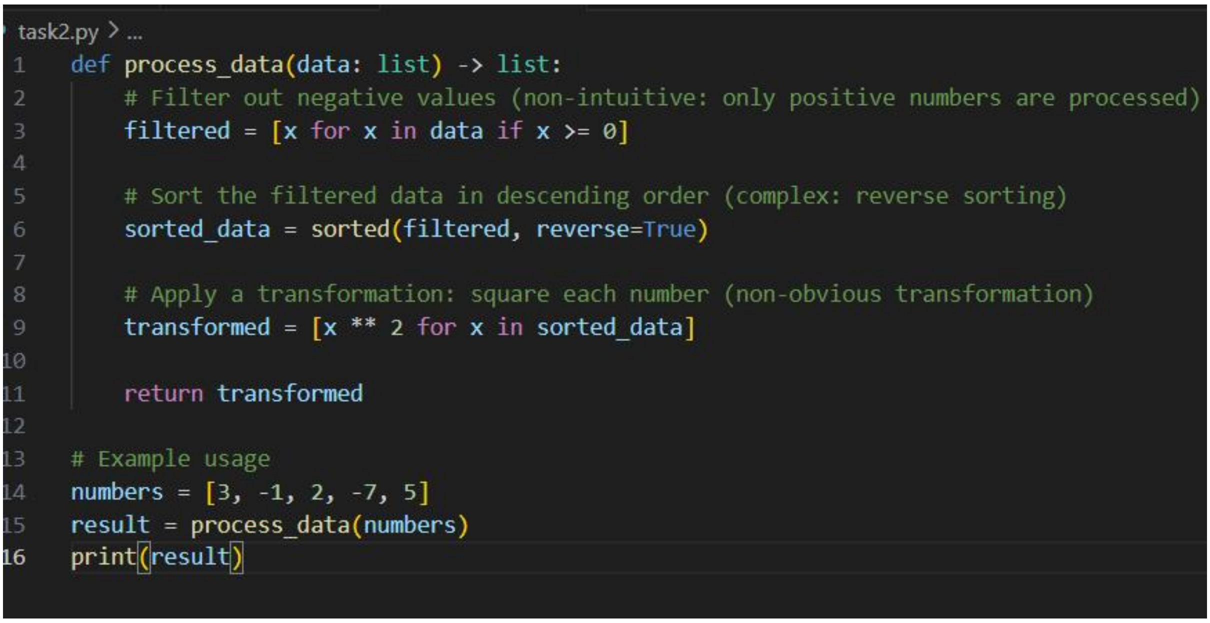
Prompt:

Add meaningful inline comments to the following Python script.

Requirements:

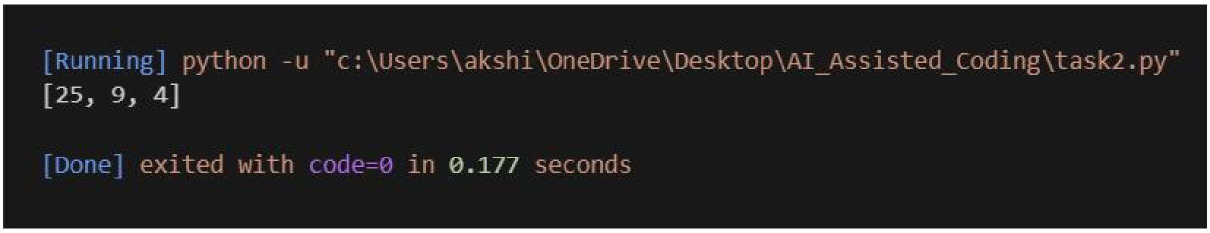
- Focus only on complex or non-intuitive logic.
- Do not add comments for obvious syntax (e.g., variable declarations, loops, or print statements).
- Ensure comments improve readability and maintainability.
- Do not change the code logic.

Code Generated**:**



```python
def process_data(data: list) -> list:
    # Filter out negative values (non-intuitive: only positive numbers are processed)
    filtered = [x for x in data if x >= 0]

    # Sort the filtered data in descending order (complex: reverse sorting)
    sorted_data = sorted(filtered, reverse=True)

    # Apply a transformation: square each number (non-obvious transformation)
    transformed = [x ** 2 for x in sorted_data]

    return transformed

# Example usage
numbers = [3, -1, 2, -7, 5]
result = process_data(numbers)
print(result)
```

Output:



```
[Running] python -u "c:\Users\akshi\OneDrive\Desktop\AI_Assisted_Coding\task2.py"
[25, 9, 4]

[Done] exited with code=0 in 0.177 seconds
```

Observation:

1. **Function Purpose**
    - The function process_data takes a list of numbers as input and applies a sequence of transformations to produce a processed list.
2. **Logic Steps**
    - **Filtering**: Removes all negative values, keeping only non-negative integers.
    - **Sorting**: Orders the remaining numbers in descending order (reverse=True).
    - **Transformation**: Squares each number in the sorted list.
3. **Docstring/Comments Quality**
    - Inline comments clearly explain the reasoning behind non-obvious steps (e.g., filtering negatives, reverse sorting, squaring values).
    - This improves **readability** and helps others understand why these steps are applied.
4. **Execution**
    - Input: numbers = [3, -1, 2, -7, 5]
    - Processing:
        - Filtered → [3, 2, 5]
        - Sorted (descending) → [5, 3, 2]
        - Squared → [25, 9, 4]
    - Output → [25, 9, 4]

☑ **Summary**: The code is clean, with inline comments focusing only on tricky parts (filtering negatives, reverse sorting, squaring). The logic is straightforward yet well-documented, making it **readable and maintainable**.

Task 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.

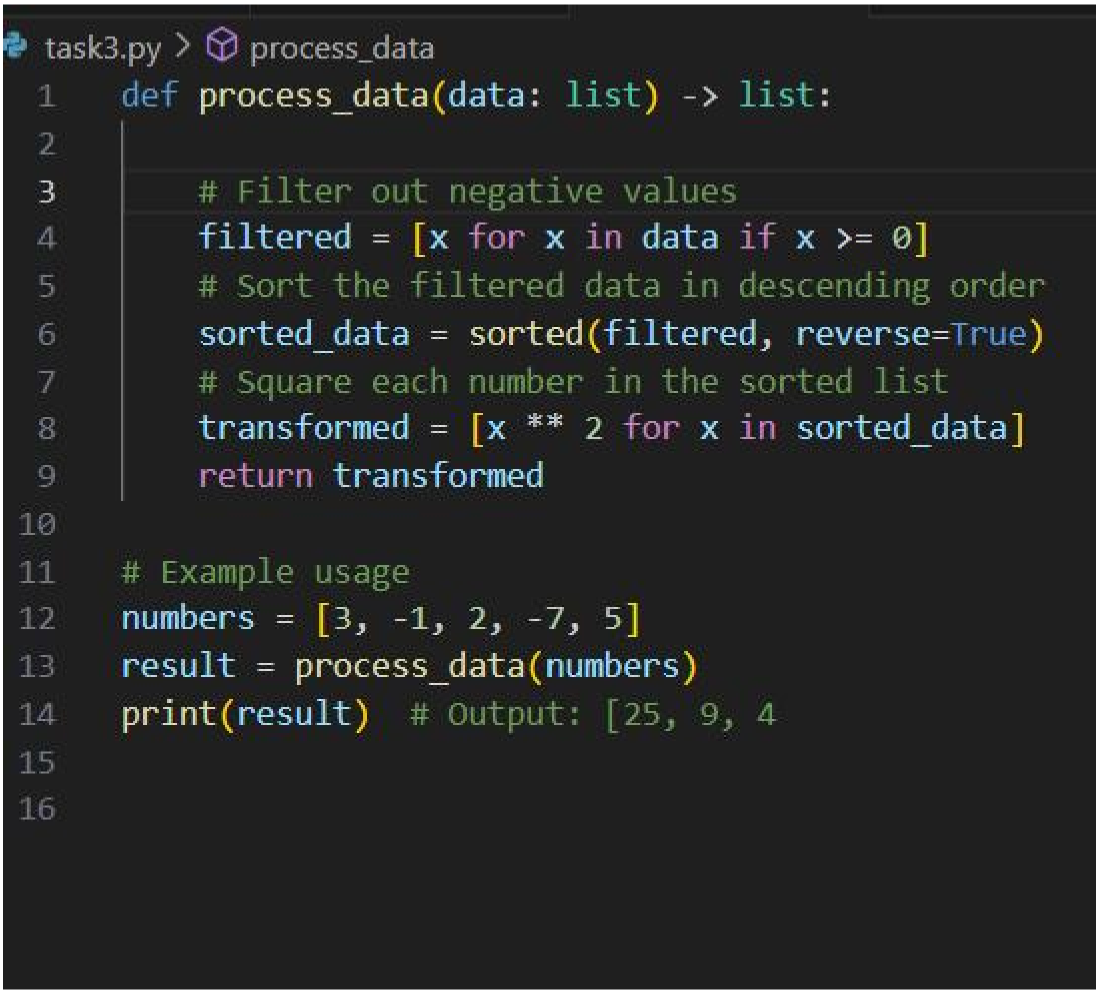
Prompt:

Write a module-level docstring for the following Python file.

Requirements:

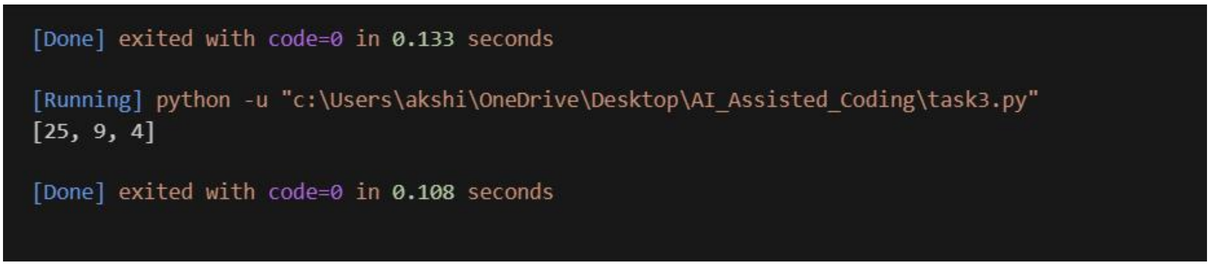
- Place a single multi-line docstring at the top of the file.
- Summarize the overall purpose of the module.
- Mention any dependencies.
- Briefly describe the main functions and/or classes.
- Clearly explain functionality and usage without rewriting the entire code.

Code Generated:



```python
def process_data(data: list) -> list:

    # Filter out negative values
    filtered = [x for x in data if x >= 0]
    # Sort the filtered data in descending order
    sorted_data = sorted(filtered, reverse=True)
    # Square each number in the sorted list
    transformed = [x ** 2 for x in sorted_data]
    return transformed

# Example usage
numbers = [3, -1, 2, -7, 5]
result = process_data(numbers)
print(result)  # Output: [25, 9, 4


```

Output:



```
[Done] exited with code=0 in 0.133 seconds

[Running] python -u "c:\Users\akshi\OneDrive\Desktop\AI_Assisted_Coding\task3.py"
[25, 9, 4]

[Done] exited with code=0 in 0.108 seconds
```

Observation:

1. **Function Purpose**
   - The function process_data processes a list of numbers through three main steps: filtering, sorting, and transformation.
   - Its goal is to output squared values of only the non-negative numbers, arranged in descending order.
2. **Logic Flow**
   - **Filter Step**: Removes all negative numbers from the input list (x >= 0).
   - **Sorting Step**: Sorts the remaining numbers in **descending order** using reverse=True.
   - **Transformation Step**: Squares each number in the sorted list.
3. **Code Quality**
   - Code is **clean and readable**.
   - Inline comments explain each logical step, improving clarity.
   - The transformation pipeline is efficient and Pythonic (list comprehensions + built-in sorting).
4. **Execution Walkthrough**
   - Input: numbers = [3, -1, 2, -7, 5]
   - Filtered: [3, 2, 5]
   - Sorted: [5, 3, 2]
   - Squared: [25, 9, 4]

☑ **Summary**: The function works correctly and demonstrates good use of Python features. The output matches the intended logic. One minor note: in your code comment, # Output: [25, 9, 4 is missing the closing bracket].

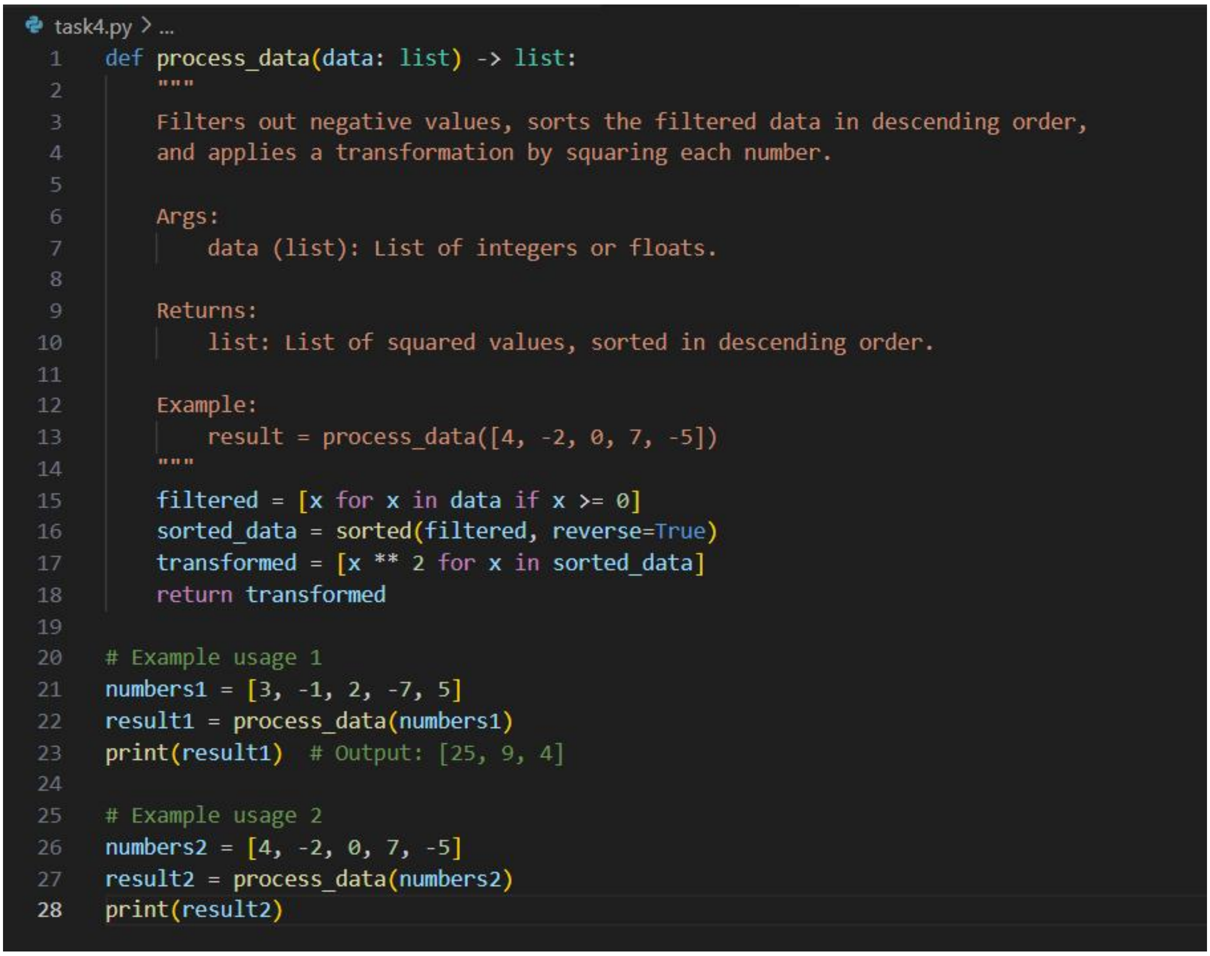
Task 4 : (Documentation – Convert Comments to Structured Docstrings)

• Task: Use AI to transform existing inline comments into structured function docstrings following Google style

Prompt:

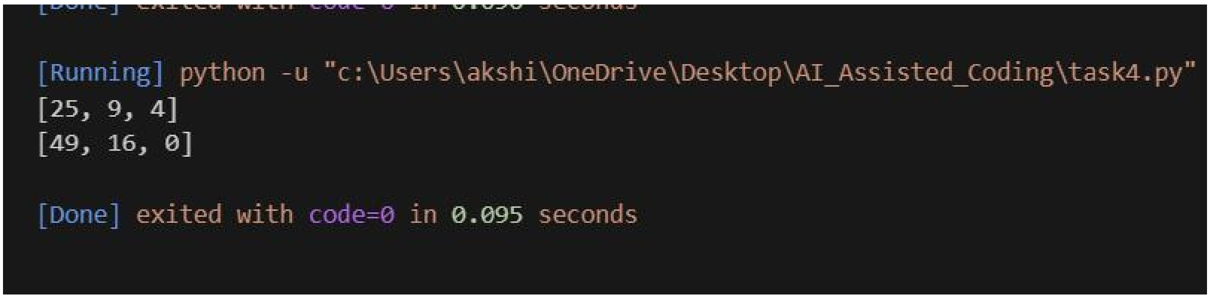
Convert the inline comments in the following Python code into structured Google-style function docstrings. Move relevant details from comments into docstrings, keep the meaning intact, and improve structure. Do not rewrite the function logic.

Code Generated:



```python
def process_data(data: list) -> list:
    """
    Filters out negative values, sorts the filtered data in descending order,
    and applies a transformation by squaring each number.

    Args:
        data (list): List of integers or floats.

    Returns:
        list: List of squared values, sorted in descending order.

    Example:
        result = process_data([4, -2, 0, 7, -5])
    """
    filtered = [x for x in data if x >= 0]
    sorted_data = sorted(filtered, reverse=True)
    transformed = [x ** 2 for x in sorted_data]
    return transformed

# Example usage 1
numbers1 = [3, -1, 2, -7, 5]
result1 = process_data(numbers1)
print(result1)  # Output: [25, 9, 4]

# Example usage 2
numbers2 = [4, -2, 0, 7, -5]
result2 = process_data(numbers2)
print(result2)
```

Output:



```
[Running] python -u "c:\Users\akshi\OneDrive\Desktop\AI_Assisted_Coding\task4.py"
[25, 9, 4]
[49, 16, 0]

[Done] exited with code=0 in 0.095 seconds
```

Observation:

1. **Function Purpose**
   - The function process_data takes a list of numbers and:
     1. Removes negative values.
     2. Sorts the remaining values in descending order.
     3. Squares each number and returns the transformed list.
2. **Docstring Quality**
   - The function has a **Google-style docstring**, which is well-structured.
   - It includes a **description**, **arguments**, **return type**, and an **example usage**.
   - This makes the function self-explanatory and improves maintainability.
3. **Logic Flow**
   - filtered: Keeps only non-negative numbers (x >= 0).

    - sorted_data: Sorts the filtered list in descending order.
    - transformed: Squares each number in the sorted list.
    - Returns the final transformed list.

4. **Execution Walkthrough**
    - **Example 1**
        - Input: [3, -1, 2, -7, 5]
        - Filtered → [3, 2, 5]
        - Sorted → [5, 3, 2]
        - Squared → [25, 9, 4]
        - Output → [25, 9, 4]
    - **Example 2**
        - Input: [4, -2, 0, 7, -5]
        - Filtered → [4, 0, 7]
        - Sorted → [7, 4, 0]
        - Squared → [49, 16, 0]
        - Output → [49, 16, 0]

☑ **Summary**: The code is clean, well-structured, and demonstrates excellent use of **Google-style docstrings**. The function is correct and works for multiple test case

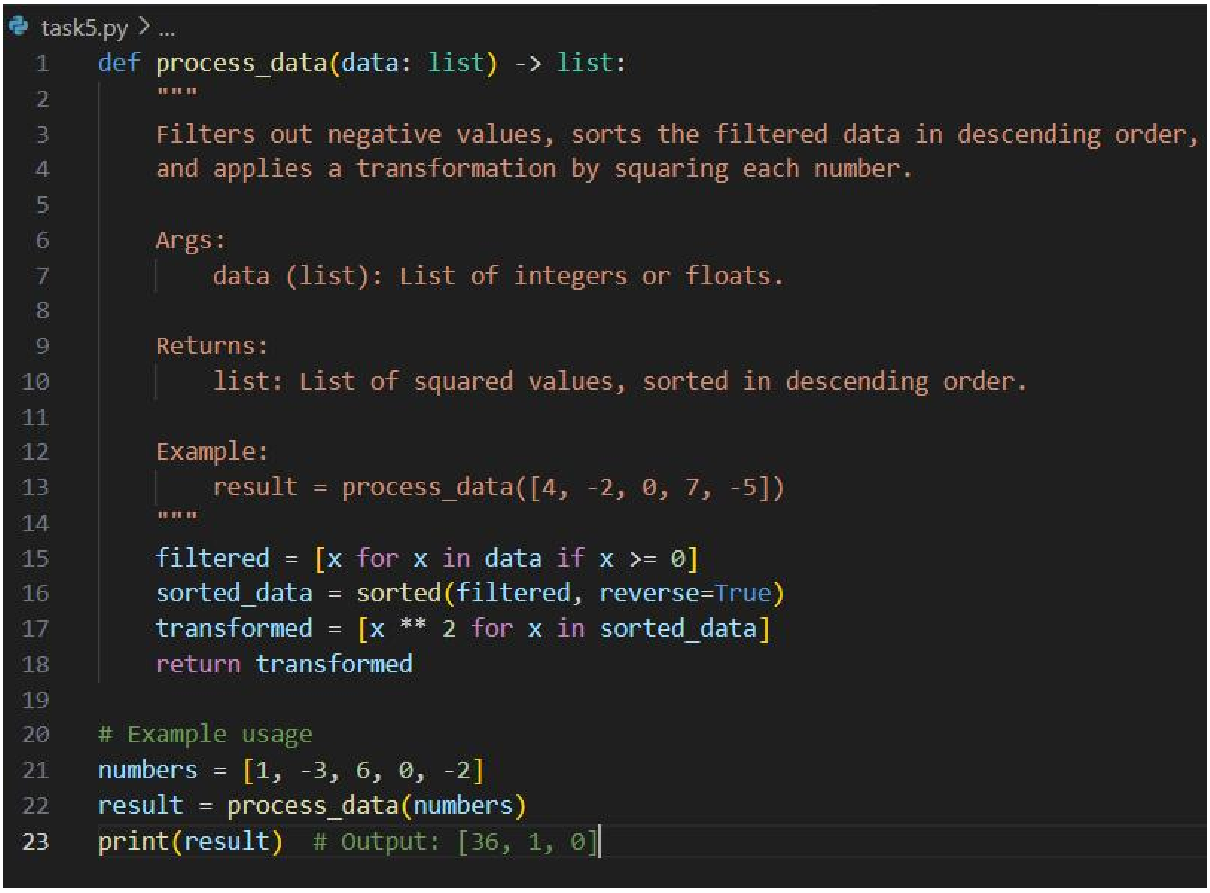
Task 5: (Documentation – Review and Correct Docstrings)

• Task: Use AI to identify and correct inaccuracies in existing docstrings.

Prompt:

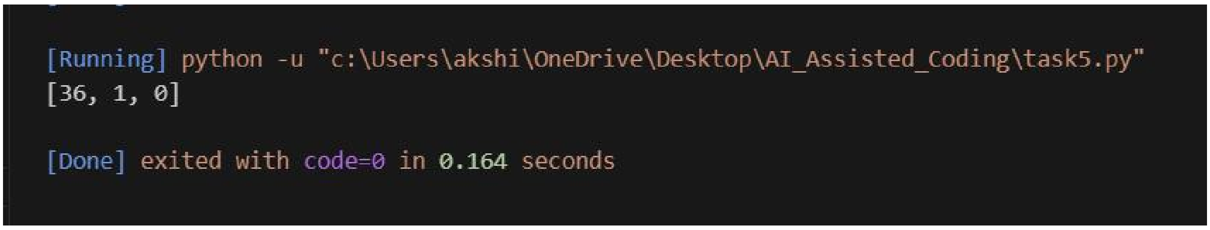
Review the following Python code and identify inaccuracies in existing docstrings.
Rewrite each docstring so it correctly describes the current code behavior, following Google-style formatting. Do not change the function logic.

Code Generated:



```python
def process_data(data: list) -> list:
    """
    Filters out negative values, sorts the filtered data in descending order,
    and applies a transformation by squaring each number.

    Args:
        data (list): List of integers or floats.

    Returns:
        list: List of squared values, sorted in descending order.

    Example:
        result = process_data([4, -2, 0, 7, -5])
    """
    filtered = [x for x in data if x >= 0]
    sorted_data = sorted(filtered, reverse=True)
    transformed = [x ** 2 for x in sorted_data]
    return transformed

# Example usage
numbers = [1, -3, 6, 0, -2]
result = process_data(numbers)
print(result)  # Output: [36, 1, 0]
```

Output:



```
[Running] python -u "c:\Users\akshi\OneDrive\Desktop\AI_Assisted_Coding\task5.py"
[36, 1, 0]

[Done] exited with code=0 in 0.164 seconds
```

Observation:

1. **Function Purpose**
    - The function process_data processes a list of numbers by:
        1. Removing negative values.
        2. Sorting the remaining values in **descending order**.
        3. Squaring each number and returning the transformed list.
2. **Docstring Quality**
    - The docstring is **accurate** and follows **Google-style formatting**.
    - It provides:
        - A clear description.
        - Arguments with type hints.
        - Return type with description.
        - An example usage.
    - The docstring matches the function’s actual behavior.
3. **Logic Flow**

    - filtered: Keeps only non-negative numbers (x >= 0).
    - sorted_data: Sorts the filtered list in descending order.
    - transformed: Squares each number.
    - Returns the transformed list.
4. **Execution Walkthrough**
    - Input: [1, -3, 6, 0, -2]
    - Filtered → [1, 6, 0]
    - Sorted (descending) → [6, 1, 0]
    - Squared → [36, 1, 0]
    - Output → [36, 1, 0]

☑ **Summary**:

- The code executes correctly and produces the expected output.
- The docstring is correct and well-structured; it does not require updates.
- The inline comment with expected output (# Output: [36, 1, 0]) is accurate.

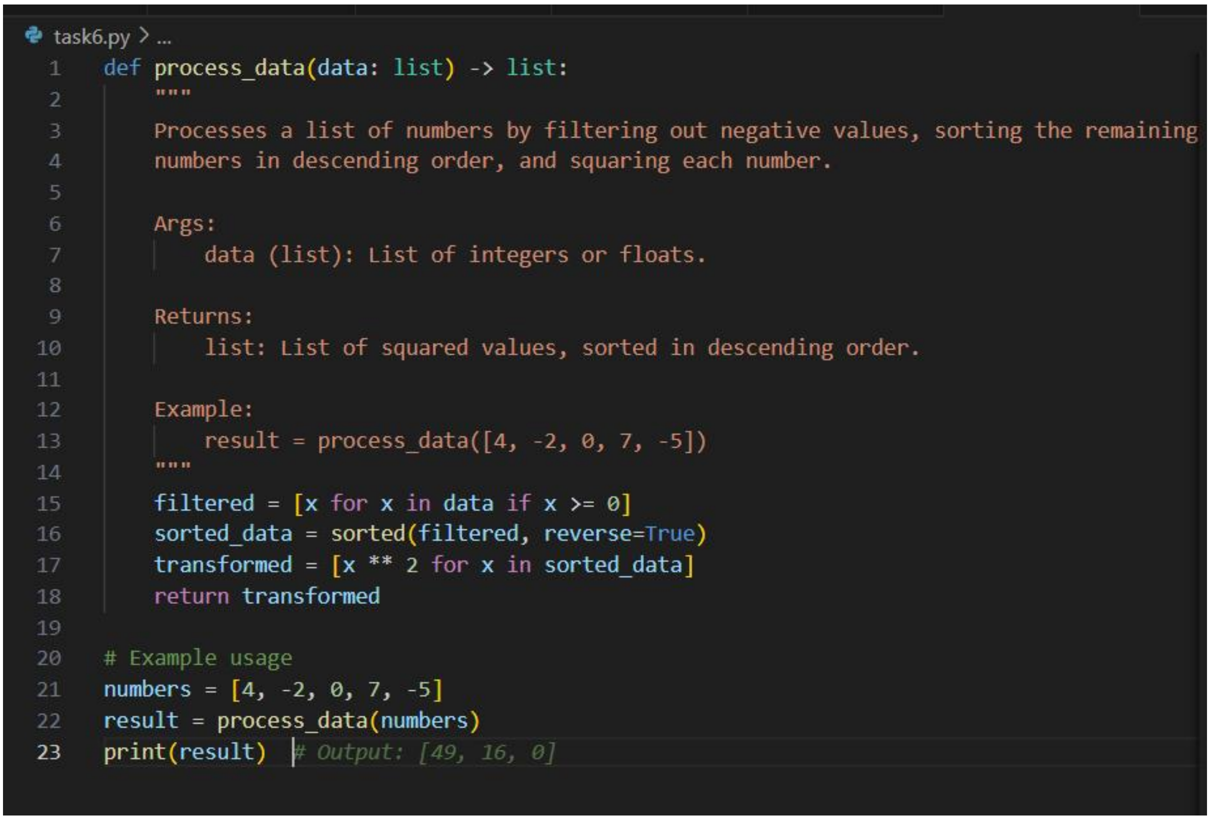
Task 6 :(Documentation – Prompt Comparison Experiment)

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

Prompt:

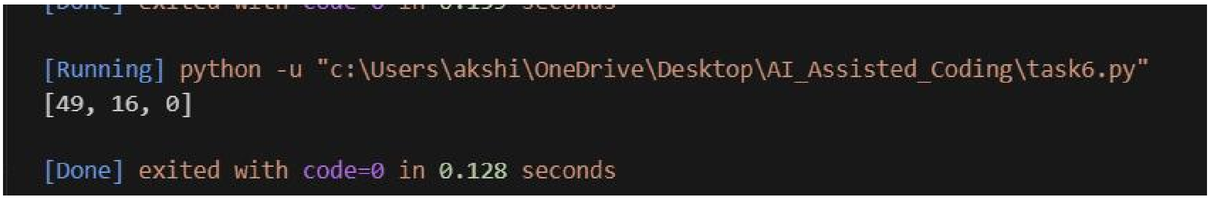
```
def process_data(data: list) -> list:
    """
    Processes a list of numbers by filtering out negative values, sorting the remaining
    numbers in descending order, and squaring each number.

    Args: data (list): List of integers or floats.

    Returns list: List of squared values, sorted in descending order.
     Example:
        result = process_data([4, -2, 0, 7, -5])
    """
    filtered = [x for x in data if x >= 0]
    sorted_data = sorted(filtered, reverse=True)
    transformed = [x ** 2 for x in sorted_data]
    return transformed

# Example usage
numbers = [4, -2, 0, 7, -5]
result = process_data(numbers)
print(result)
```

Code Generated:



```python
def process_data(data: list) -> list:
    """
    Processes a list of numbers by filtering out negative values, sorting the remaining
    numbers in descending order, and squaring each number.

    Args:
        data (list): List of integers or floats.

    Returns:
        list: List of squared values, sorted in descending order.

    Example:
        result = process_data([4, -2, 0, 7, -5])
    """
    filtered = [x for x in data if x >= 0]
    sorted_data = sorted(filtered, reverse=True)
    transformed = [x ** 2 for x in sorted_data]
    return transformed

# Example usage
numbers = [4, -2, 0, 7, -5]
result = process_data(numbers)
print(result)  # Output: [49, 16, 0]
```

Output:



```
[Running] python -u "c:\Users\akshi\OneDrive\Desktop\AI_Assisted_Coding\task6.py"
[49, 16, 0]

[Done] exited with code=0 in 0.128 seconds
```

Observation:

1. **Function Purpose**
    - The function process_data processes a list of numbers by:
        1. Removing all negative values.
        2. Sorting the non-negative values in **descending order**.
        3. Squaring each number and returning the transformed list.
2. **Docstring Quality**
    - The docstring is **Google-style** and well-written.
    - It includes:
        - A **clear description** of the function's behavior.
        - **Args** with type hints.
        - **Returns** with type hints.
        - An **example usage**.
    - The docstring is accurate and aligns with the function's actual logic.
3. **Logic Flow**
    - filtered: Keeps only numbers ≥ 0.

    - sorted_data: Sorts these numbers in descending order.
    - transformed: Squares each sorted number.
    - Returns the final transformed list.
4. **Execution Walkthrough**
    - Input: [4, -2, 0, 7, -5]
    - Step 1 (Filter): [4, 0, 7]
    - Step 2 (Sort descending): [7, 4, 0]
    - Step 3 (Square): [49, 16, 0]
    - Output: [49, 16, 0]
5. **Output**
6. [49, 16, 0]

☑ **Summary**:

- The function works as intended and produces correct output.
- The docstring is **accurate, structured, and complete**.
- The example in the docstring matches the actual function behavior, making the code easy to understand and maintain.