AI ASSISTED CODING

ASSIGNMENT-9.2

Name: Baggi praneeth

HT NO: 2403A51246

Batch No.: 11

Task – 1

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

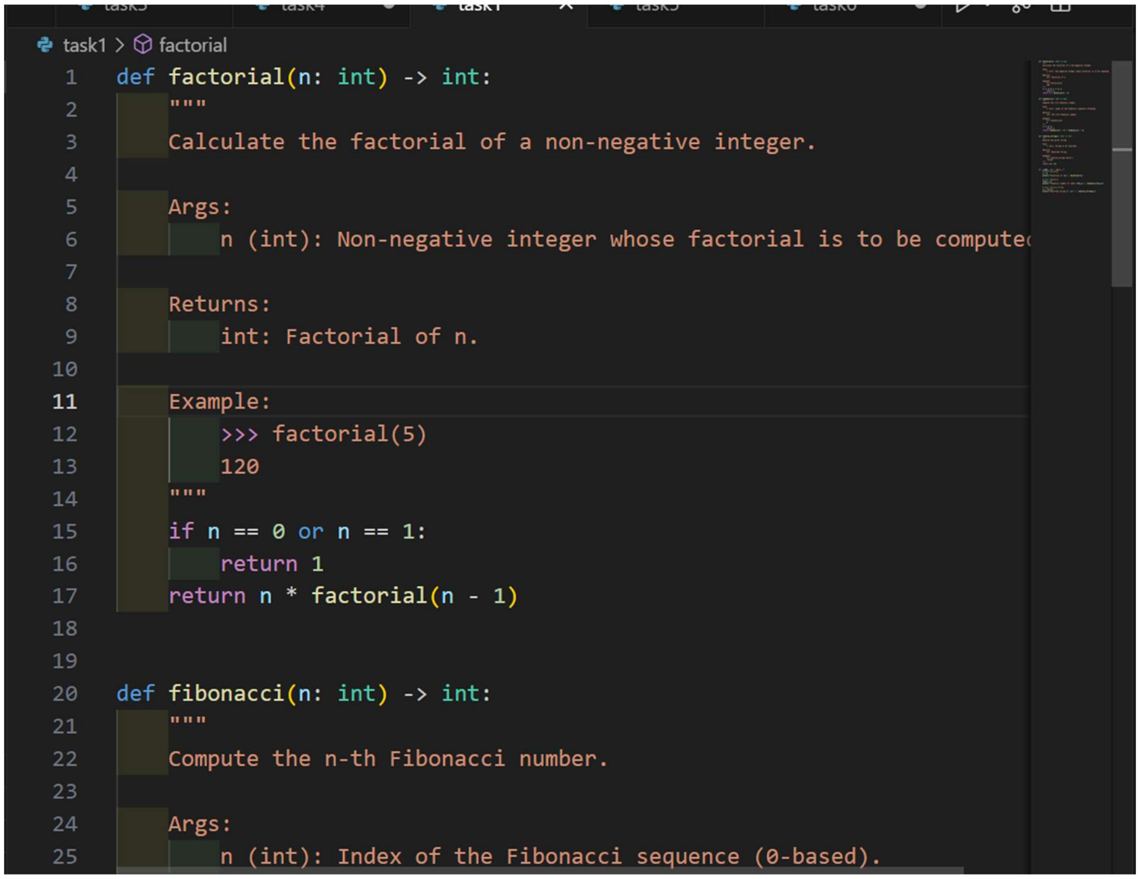
▪ Return values with type hints

▪ Example usage

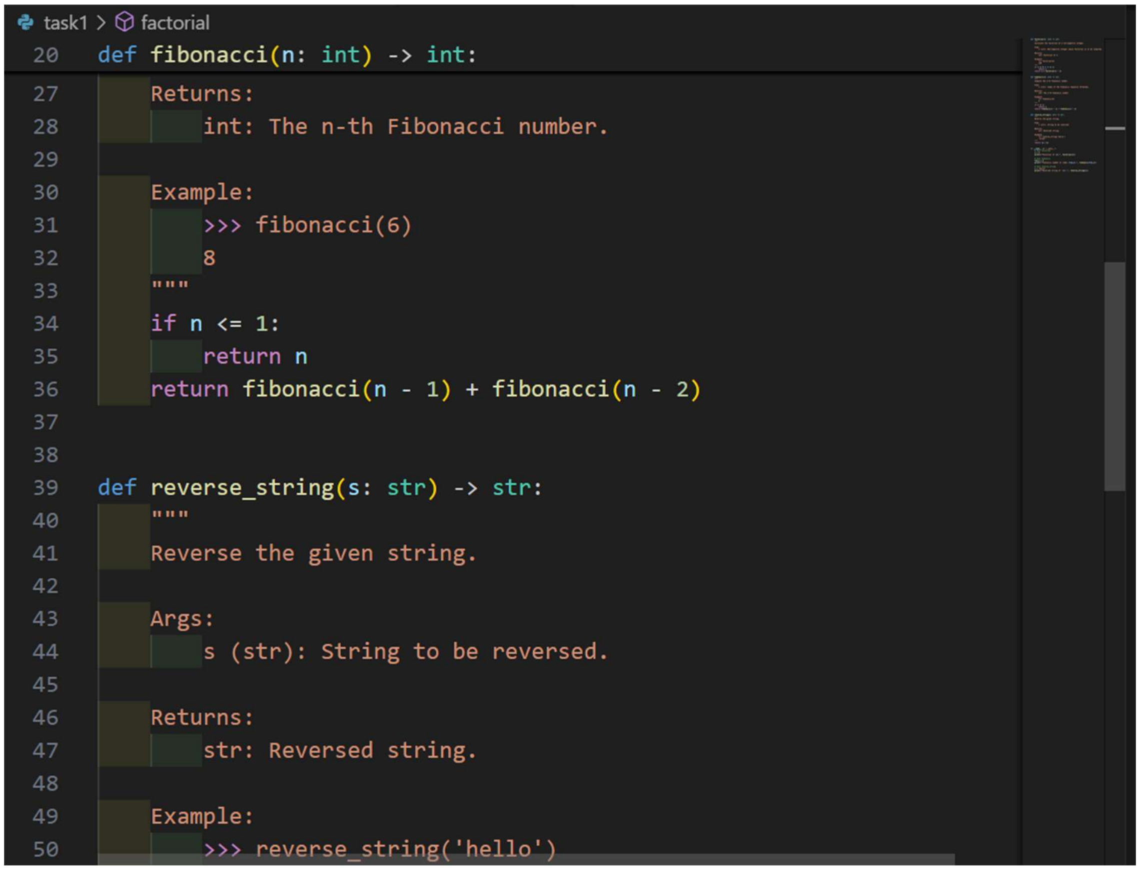
o Review the generated docstrings for accuracy and formatting.

Expected Output 1:

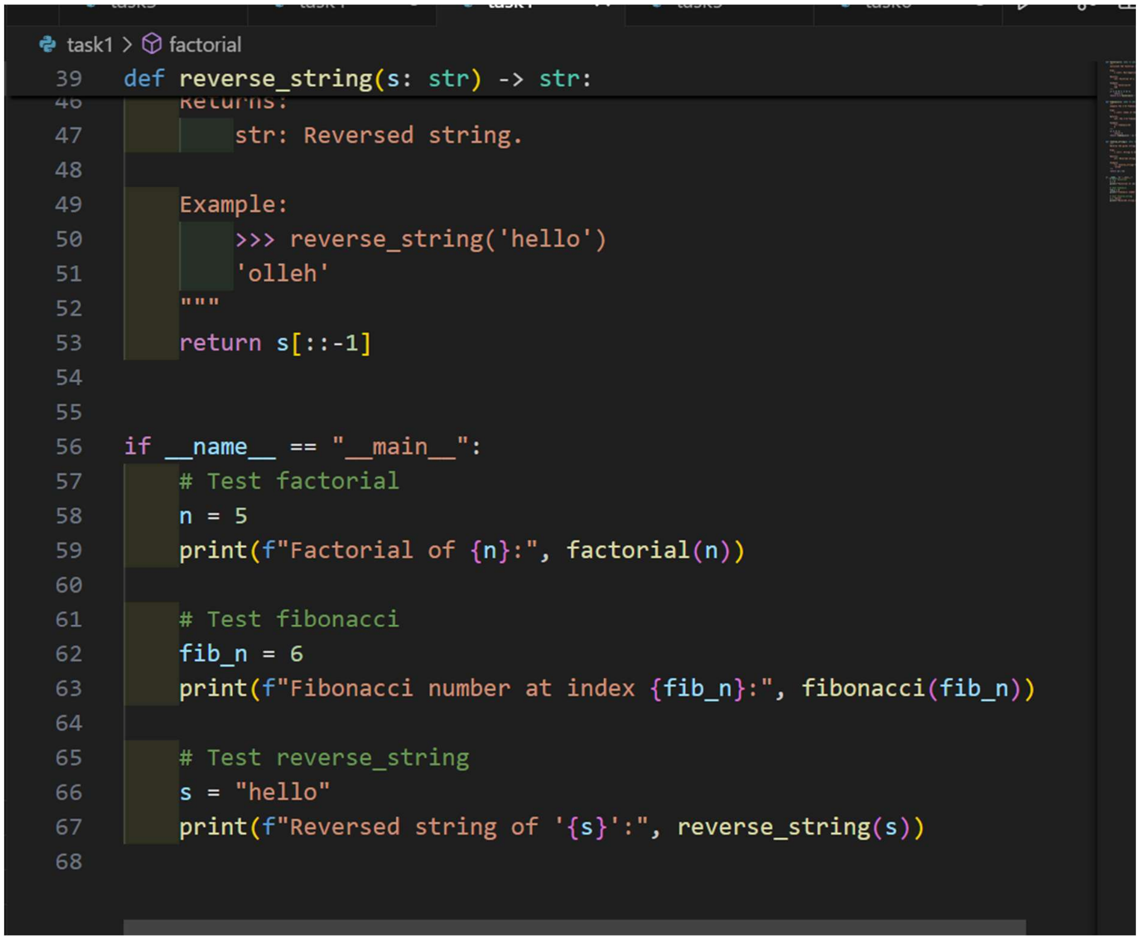
o A Python script with all functions documented using correctly formatted Google-style docstrings.



```python
def factorial(n: int) -> int:
    """
    Calculate the factorial of a non-negative integer.

    Args:
        n (int): Non-negative integer whose factorial is to be compute

    Returns:
        int: Factorial of n.

    Example:
        >>> factorial(5)
        120
    """
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)


def fibonacci(n: int) -> int:
    """
    Compute the n-th Fibonacci number.

    Args:
        n (int): Index of the Fibonacci sequence (0-based).
```

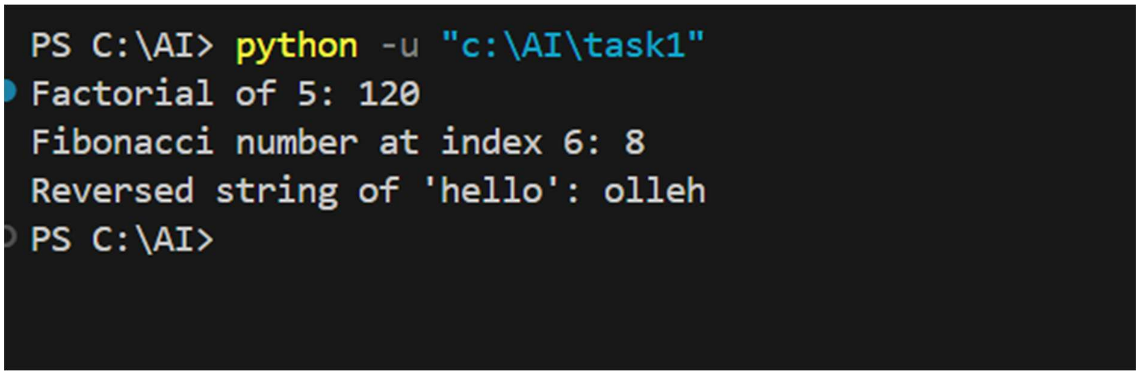


```python
def fibonacci(n: int) -> int:
    Returns:
        int: The n-th Fibonacci number.

    Example:
        >>> fibonacci(6)
        8
    """
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)


def reverse_string(s: str) -> str:
    """
    Reverse the given string.

    Args:
        s (str): String to be reversed.

    Returns:
        str: Reversed string.

    Example:
        >>> reverse_string('hello')
```



```
task1 > factorial
def reverse_string(s: str) -> str:
    Returns:
        str: Reversed string.

    Example:
        >>> reverse_string('hello')
        'olleh'
    """
    return s[::-1]


if __name__ == "__main__":
    # Test factorial
    n = 5
    print(f"Factorial of {n}:", factorial(n))

    # Test fibonacci
    fib_n = 6
    print(f"Fibonacci number at index {fib_n}:", fibonacci(fib_n))

    # Test reverse_string
    s = "hello"
    print(f"Reversed string of '{s}':", reverse_string(s))

```

OUTPUT:



```
PS C:\AI> python -u "c:\AI\task1"
Factorial of 5: 120
Fibonacci number at index 6: 8
Reversed string of 'hello': olleh
PS C:\AI>
```

Task – 2

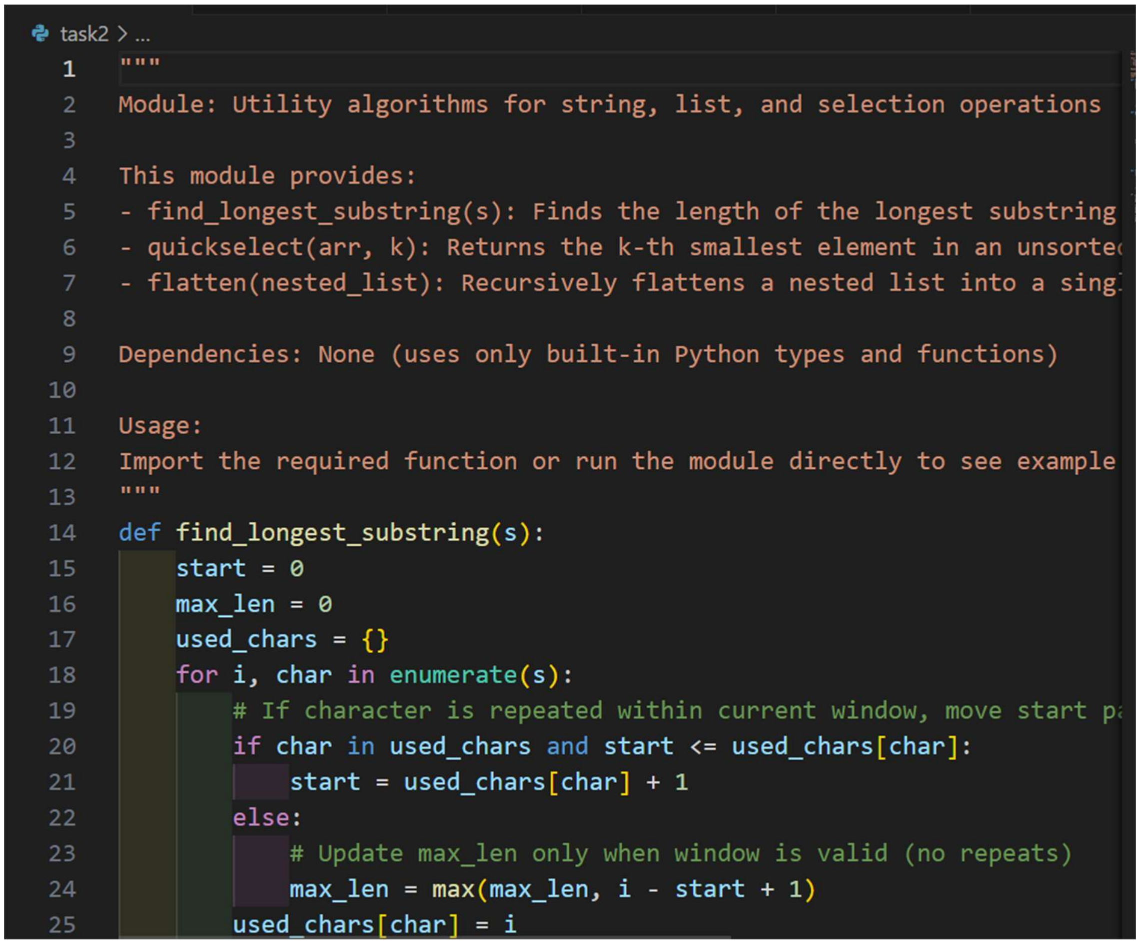
Task: Use AI to add meaningful inline comments to a Python program explaining only complex logic parts.

• Instructions:

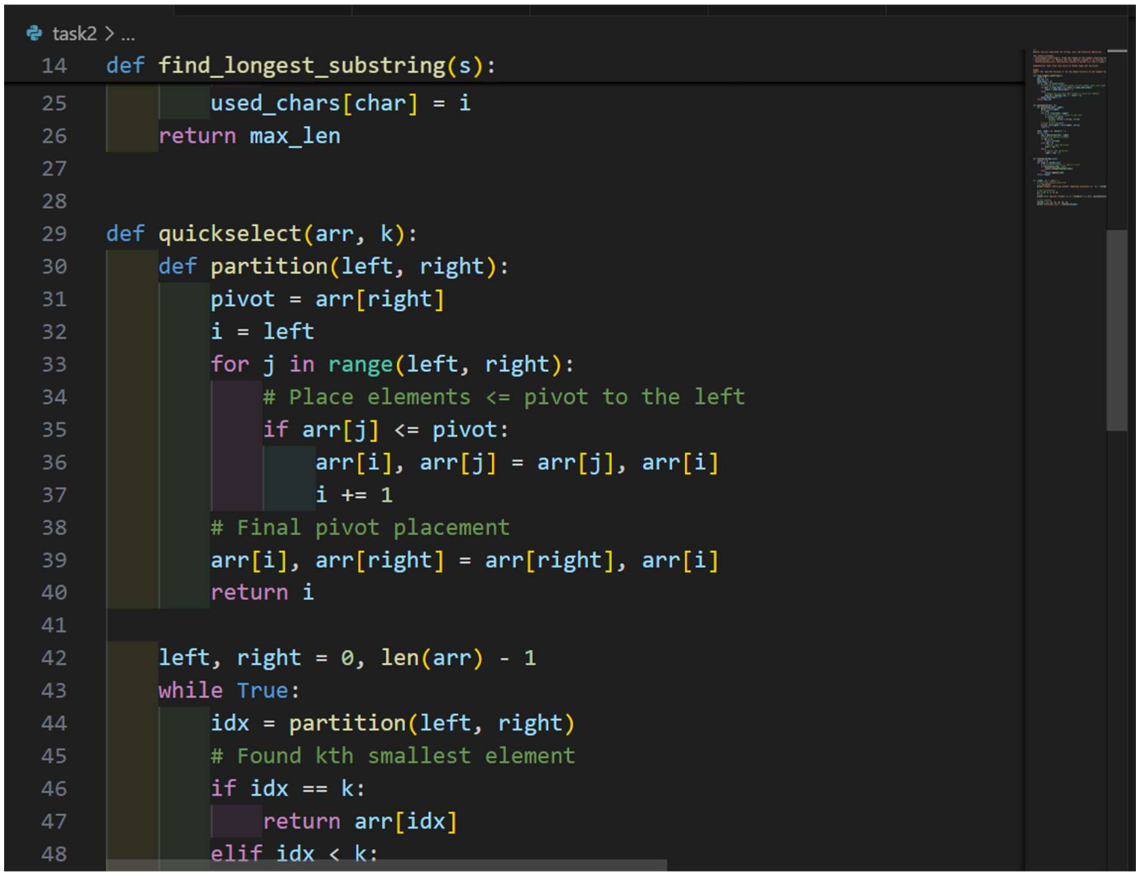
o Provide a Python script without comments to the AI.

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.

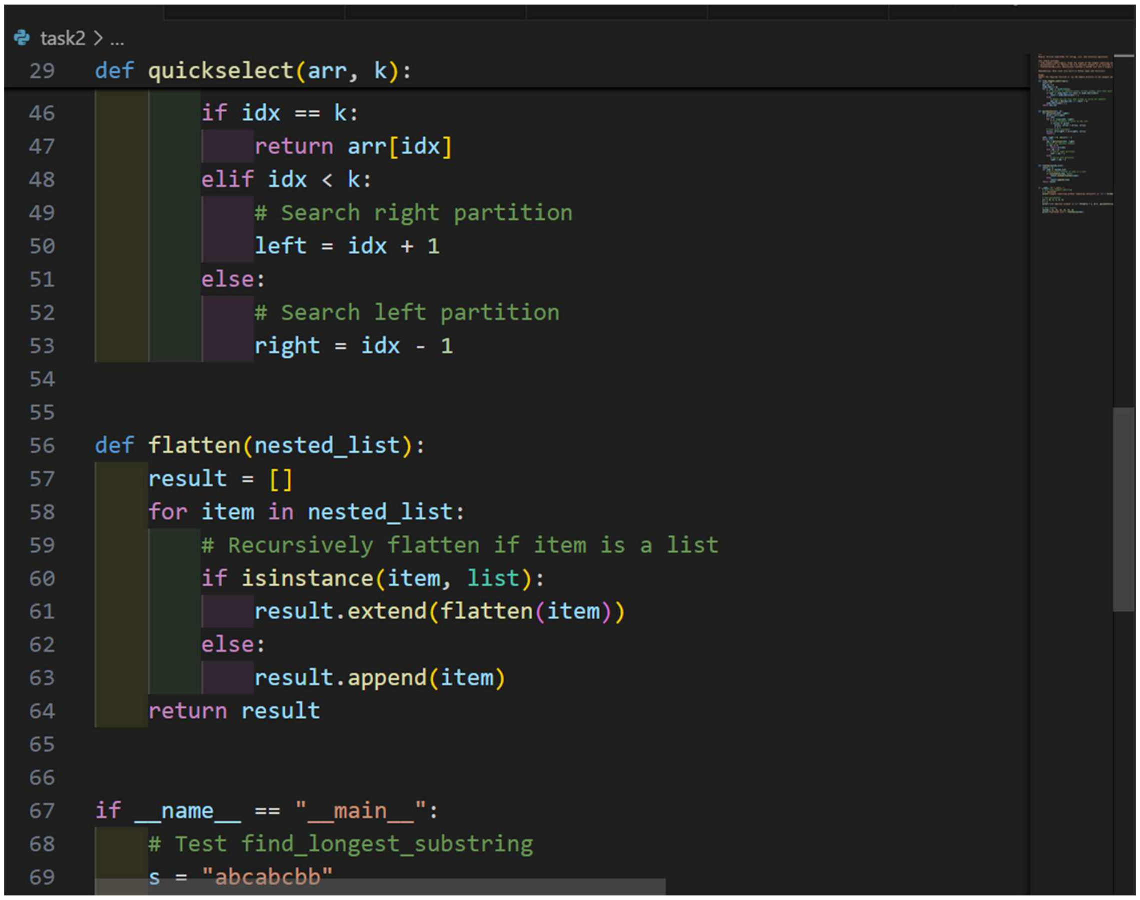
Expected Output 2:
o Python code with concise, context-aware inline comments for complex logic blocks.



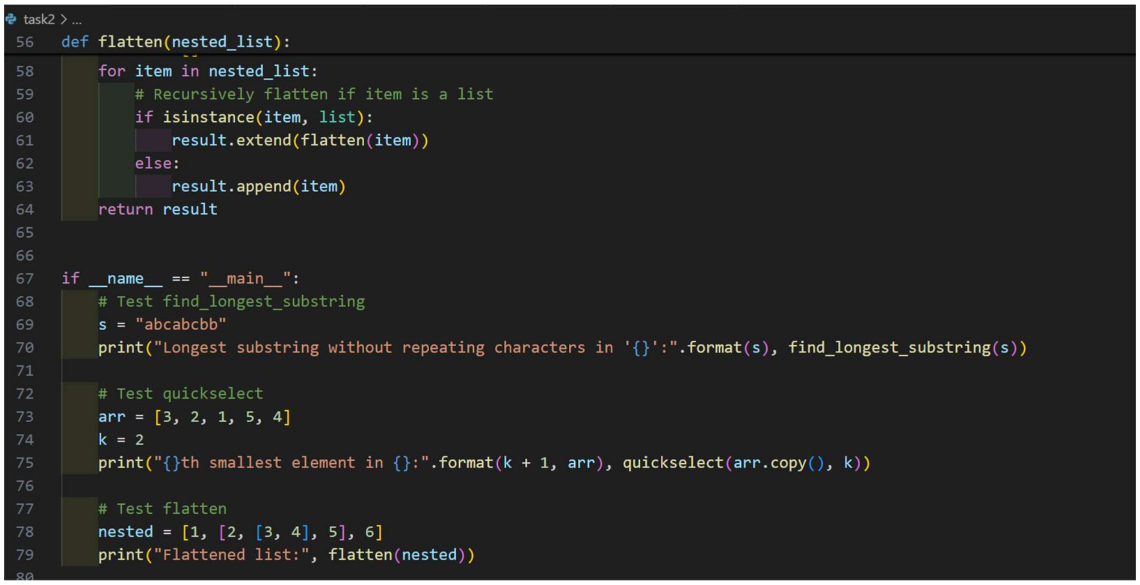
```python
"""
Module: Utility algorithms for string, list, and selection operations

This module provides:
- find_longest_substring(s): Finds the length of the longest substring
- quickselect(arr, k): Returns the k-th smallest element in an unsorte
- flatten(nested_list): Recursively flattens a nested list into a sing

Dependencies: None (uses only built-in Python types and functions)

Usage:
Import the required function or run the module directly to see example
"""
def find_longest_substring(s):
    start = 0
    max_len = 0
    used_chars = {}
    for i, char in enumerate(s):
        # If character is repeated within current window, move start pa
        if char in used_chars and start <= used_chars[char]:
            start = used_chars[char] + 1
        else:
            # Update max_len only when window is valid (no repeats)
            max_len = max(max_len, i - start + 1)
        used_chars[char] = i
```


task2 > ...

```python
def find_longest_substring(s):
            used_chars[char] = i
        return max_len


def quickselect(arr, k):
    def partition(left, right):
        pivot = arr[right]
        i = left
        for j in range(left, right):
            # Place elements <= pivot to the left
            if arr[j] <= pivot:
                arr[i], arr[j] = arr[j], arr[i]
                i += 1
        # Final pivot placement
        arr[i], arr[right] = arr[right], arr[i]
        return i

    left, right = 0, len(arr) - 1
    while True:
        idx = partition(left, right)
        # Found kth smallest element
        if idx == k:
            return arr[idx]
        elif idx < k:
```

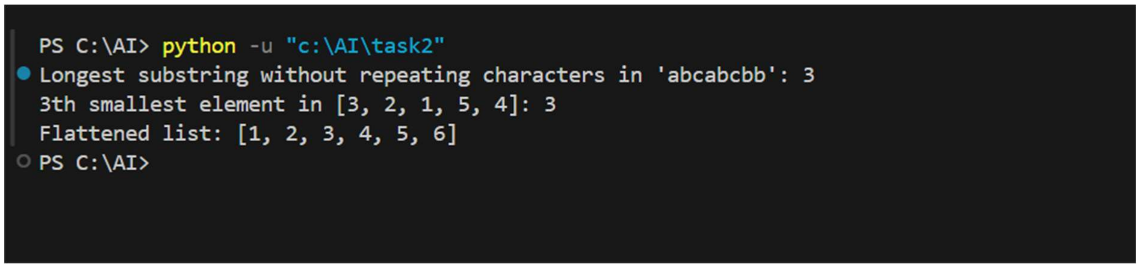

task2 > ...

```python
def quickselect(arr, k):
        if idx == k:
            return arr[idx]
        elif idx < k:
            # Search right partition
            left = idx + 1
        else:
            # Search left partition
            right = idx - 1


def flatten(nested_list):
    result = []
    for item in nested_list:
        # Recursively flatten if item is a list
        if isinstance(item, list):
            result.extend(flatten(item))
        else:
            result.append(item)
    return result


if __name__ == "__main__":
    # Test find_longest_substring
    s = "abcabcbb"
```


task2 > ...

```python
def flatten(nested_list):
    for item in nested_list:
        # Recursively flatten if item is a list
        if isinstance(item, list):
            result.extend(flatten(item))
        else:
            result.append(item)
    return result


if __name__ == "__main__":
    # Test find_longest_substring
    s = "abcabcbb"
    print("Longest substring without repeating characters in '{}':".format(s), find_longest_substring(s))

    # Test quickselect
    arr = [3, 2, 1, 5, 4]
    k = 2
    print("{}th smallest element in {}:".format(k + 1, arr), quickselect(arr.copy(), k))

    # Test flatten
    nested = [1, [2, [3, 4], 5], 6]
    print("Flattened list:", flatten(nested))
```

OUTPUT:



```
PS C:\AI> python -u "c:\AI\task2"
Longest substring without repeating characters in 'abcabcbb': 3
3th smallest element in [3, 2, 1, 5, 4]: 3
Flattened list: [1, 2, 3, 4, 5, 6]
PS C:\AI>
```

Task – 3

Task: Use AI to create a module-level docstring summarizing the purpose, dependencies, and main functions/classes of a Python file.

• Instructions:

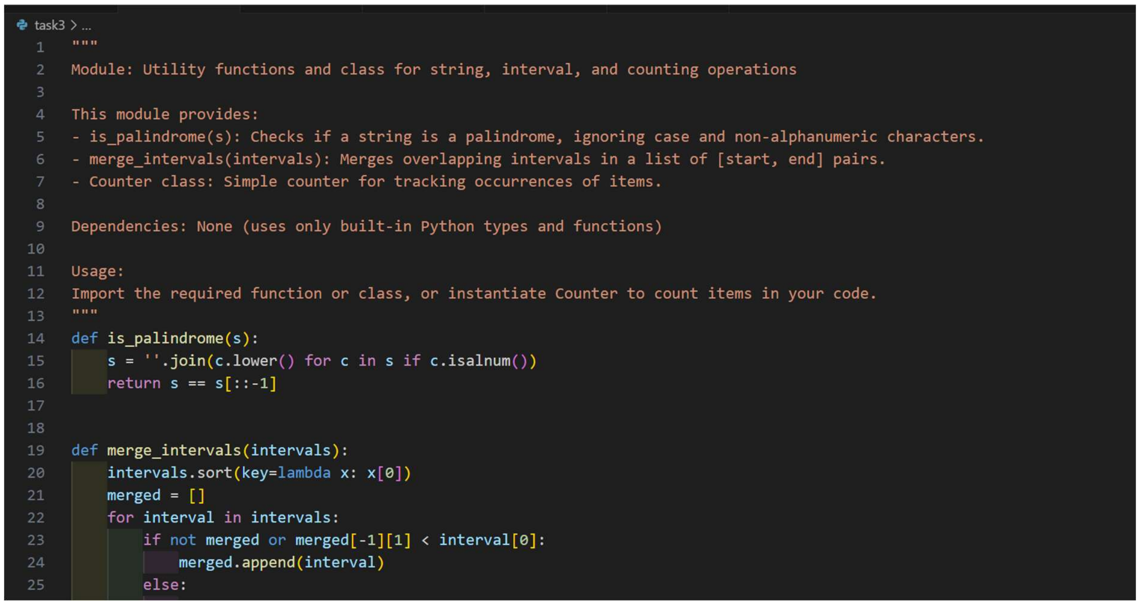
o Supply the entire Python file to AI.

o Instruct AI to write a single multi-line docstring at the top of the file.

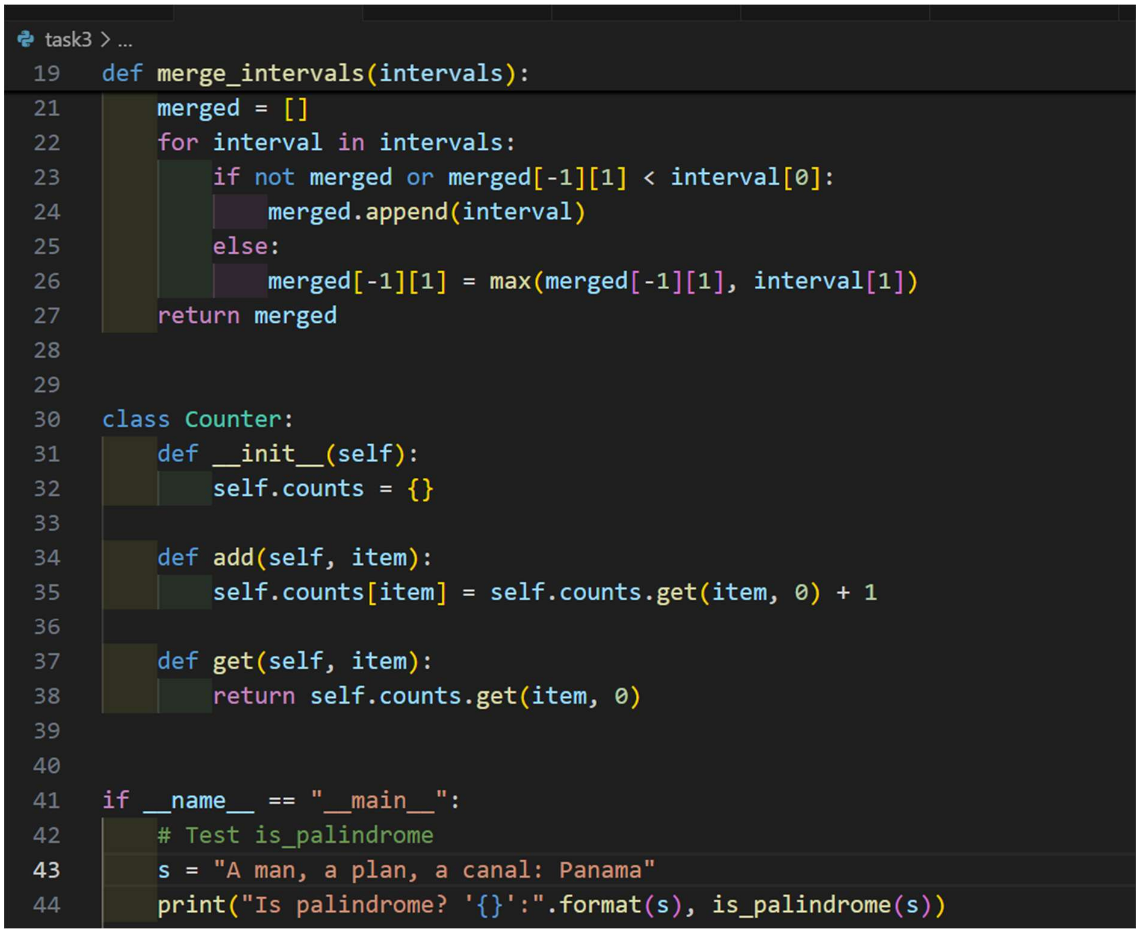
o Ensure the docstring clearly describes functionality and usage without rewriting the entire code.

Expected Output 3:

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

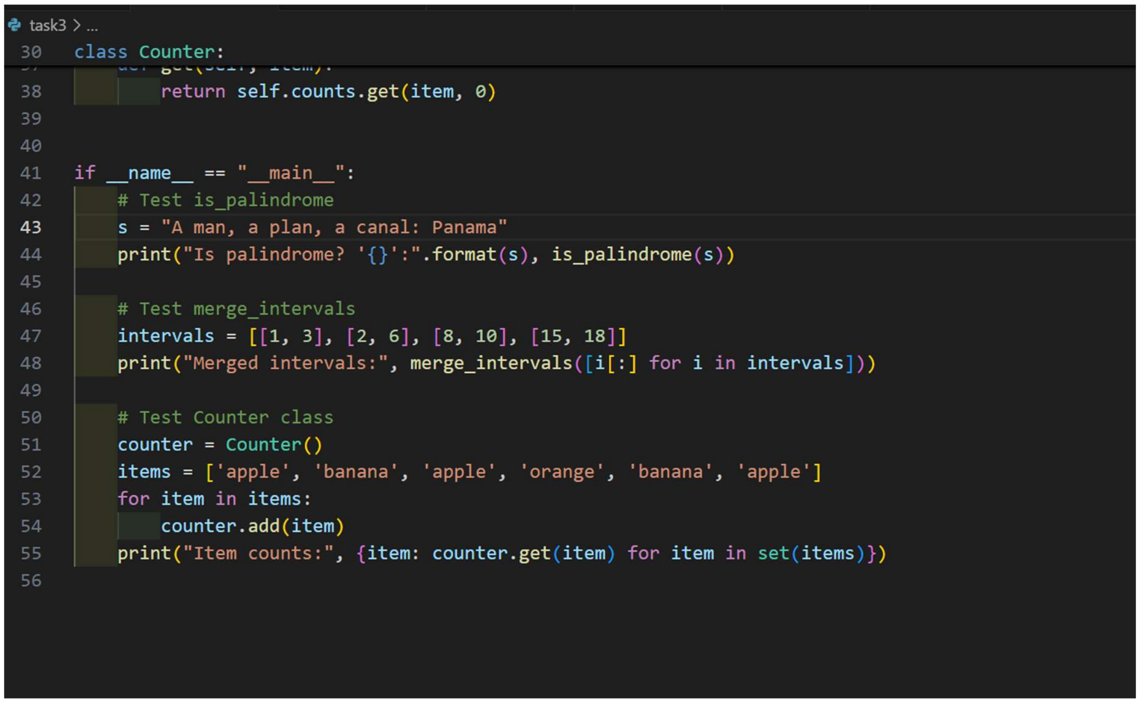


```
"""
Module: Utility functions and class for string, interval, and counting operations

This module provides:
- is_palindrome(s): Checks if a string is a palindrome, ignoring case and non-alphanumeric characters.
- merge_intervals(intervals): Merges overlapping intervals in a list of [start, end] pairs.
- Counter class: Simple counter for tracking occurrences of items.

Dependencies: None (uses only built-in Python types and functions)

Usage:
Import the required function or class, or instantiate Counter to count items in your code.
"""
def is_palindrome(s):
    s = ''.join(c.lower() for c in s if c.isalnum())
    return s == s[::-1]


def merge_intervals(intervals):
    intervals.sort(key=lambda x: x[0])
    merged = []
    for interval in intervals:
        if not merged or merged[-1][1] < interval[0]:
            merged.append(interval)
        else:
```



task3 > ...

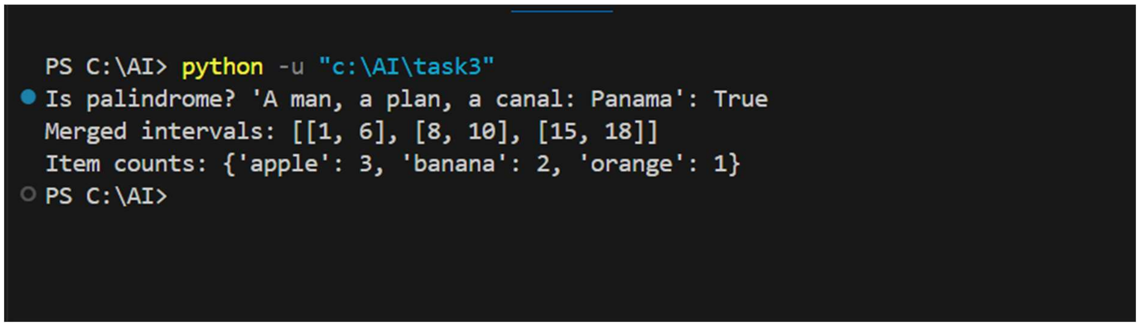
```python
def merge_intervals(intervals):
    merged = []
    for interval in intervals:
        if not merged or merged[-1][1] < interval[0]:
            merged.append(interval)
        else:
            merged[-1][1] = max(merged[-1][1], interval[1])
    return merged


class Counter:
    def __init__(self):
        self.counts = {}

    def add(self, item):
        self.counts[item] = self.counts.get(item, 0) + 1

    def get(self, item):
        return self.counts.get(item, 0)


if __name__ == "__main__":
    # Test is_palindrome
    s = "A man, a plan, a canal: Panama"
    print("Is palindrome? '{}':".format(s), is_palindrome(s))
```



task3 > ...

```python
class Counter:
        return self.counts.get(item, 0)


if __name__ == "__main__":
    # Test is_palindrome
    s = "A man, a plan, a canal: Panama"
    print("Is palindrome? '{}':".format(s), is_palindrome(s))

    # Test merge_intervals
    intervals = [[1, 3], [2, 6], [8, 10], [15, 18]]
    print("Merged intervals:", merge_intervals([i[:] for i in intervals]))

    # Test Counter class
    counter = Counter()
    items = ['apple', 'banana', 'apple', 'orange', 'banana', 'apple']
    for item in items:
        counter.add(item)
    print("Item counts:", {item: counter.get(item) for item in set(items)})
```

OUTPUT:



```
PS C:\AI> python -u "c:\AI\task3"
Is palindrome? 'A man, a plan, a canal: Panama': True
Merged intervals: [[1, 6], [8, 10], [15, 18]]
Item counts: {'apple': 3, 'banana': 2, 'orange': 1}
PS C:\AI>
```

Task – 4

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

• Instructions:

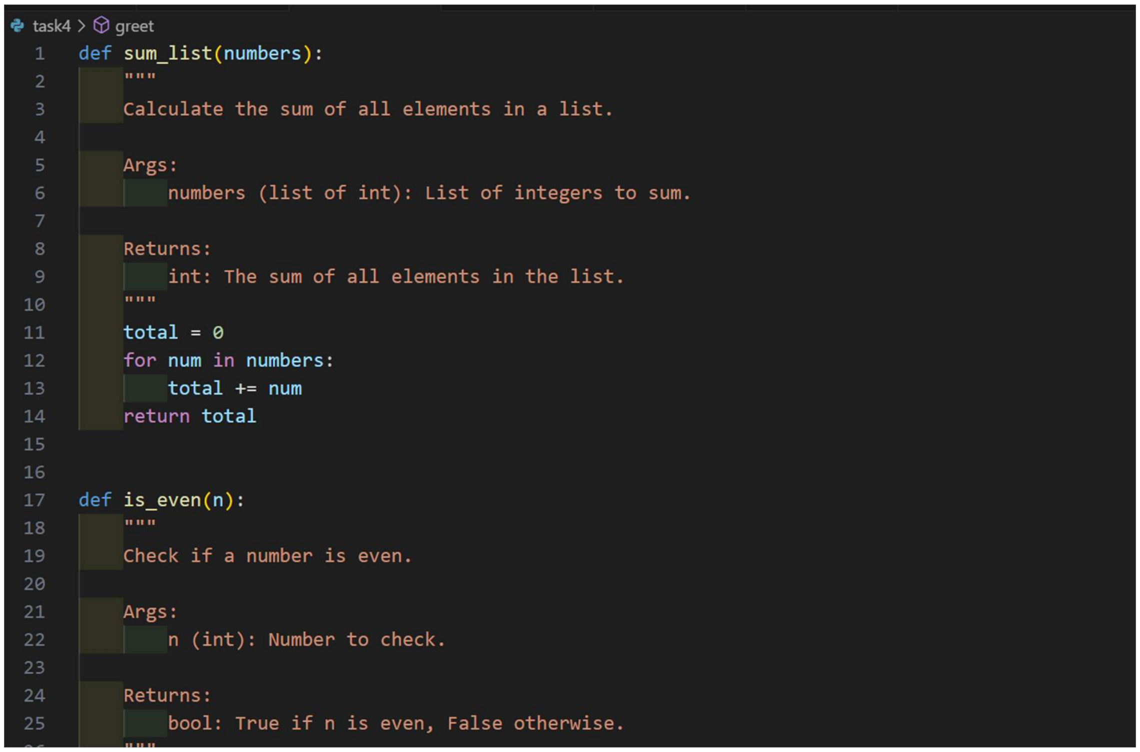
o Provide AI with Python code containing inline comments.

o Ask AI to move relevant details from comments into function docstrings.

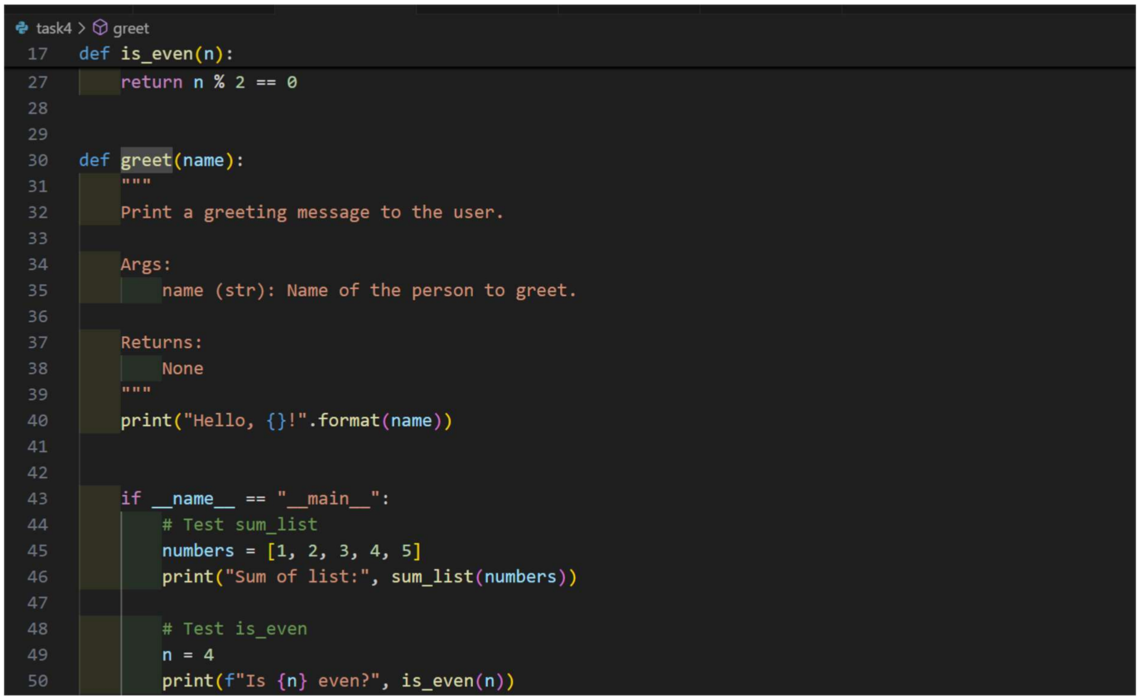
o Verify that the new docstrings keep the meaning intact while improving structure.

Expected Output 4:

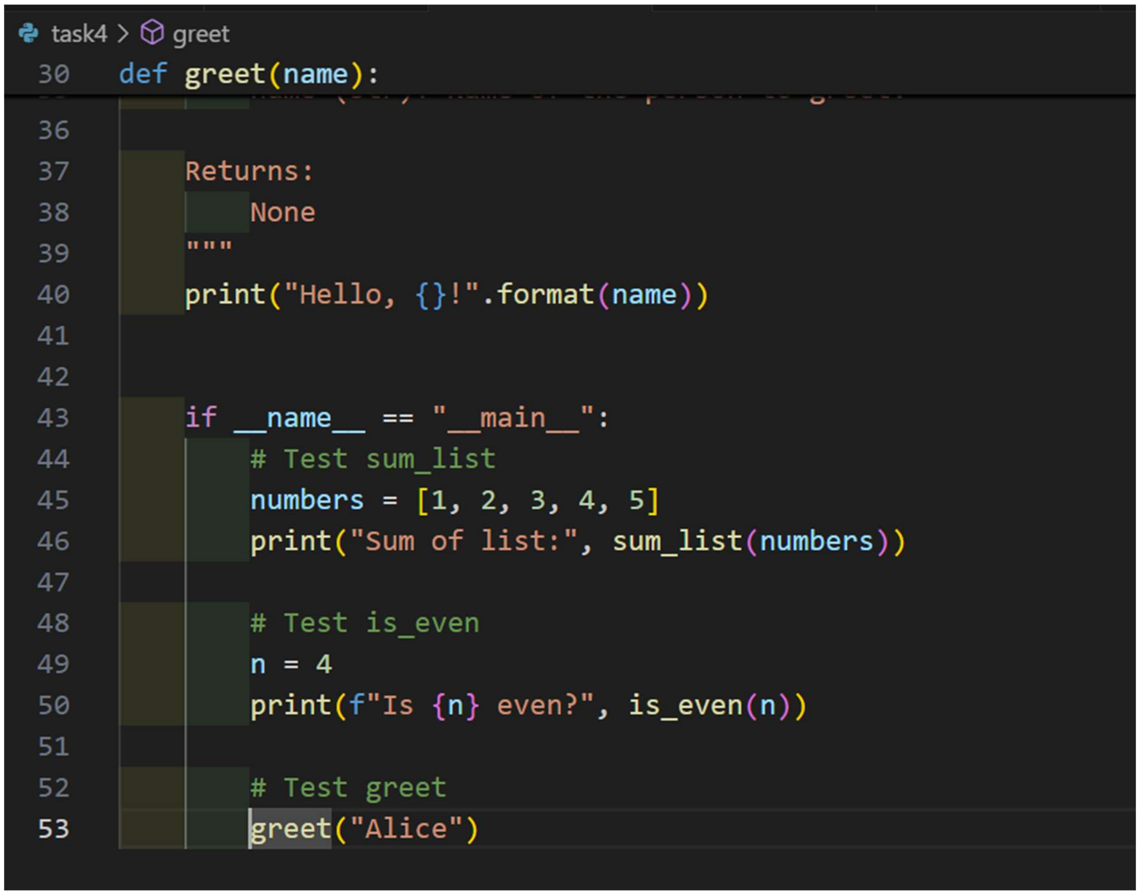
o Python code with comments replaced by clear, standardized docstrings.



```python
def sum_list(numbers):
    """
    Calculate the sum of all elements in a list.

    Args:
        numbers (list of int): List of integers to sum.

    Returns:
        int: The sum of all elements in the list.
    """
    total = 0
    for num in numbers:
        total += num
    return total


def is_even(n):
    """
    Check if a number is even.

    Args:
        n (int): Number to check.

    Returns:
        bool: True if n is even, False otherwise.
    """
```

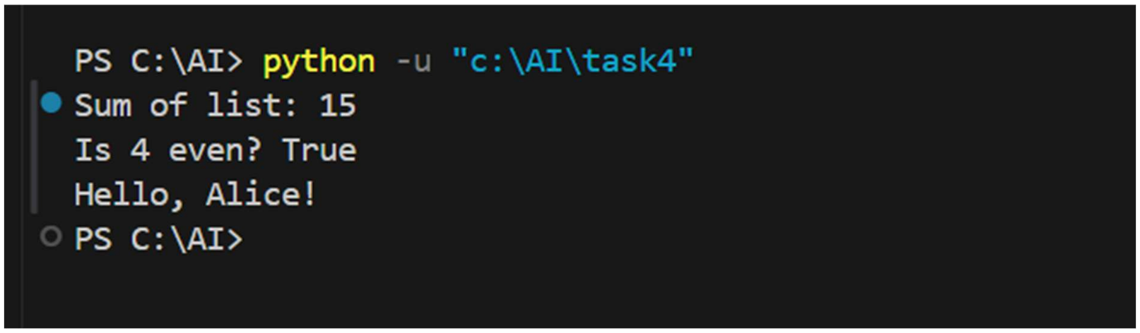


```python
def is_even(n):
    return n % 2 == 0


def greet(name):
    """
    Print a greeting message to the user.

    Args:
        name (str): Name of the person to greet.

    Returns:
        None
    """
    print("Hello, {}!".format(name))


    if __name__ == "__main__":
        # Test sum_list
        numbers = [1, 2, 3, 4, 5]
        print("Sum of list:", sum_list(numbers))

        # Test is_even
        n = 4
        print(f"Is {n} even?", is_even(n))
```



```python
def greet(name):

    Returns:
        None
    """
    print("Hello, {}!".format(name))


    if __name__ == "__main__":
        # Test sum_list
        numbers = [1, 2, 3, 4, 5]
        print("Sum of list:", sum_list(numbers))

        # Test is_even
        n = 4
        print(f"Is {n} even?", is_even(n))

        # Test greet
        greet("Alice")
```

OUTPUT:



```
PS C:\AI> python -u "c:\AI\task4"
Sum of list: 15
Is 4 even? True
Hello, Alice!
PS C:\AI>
```

Task – 5

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

• Instructions:

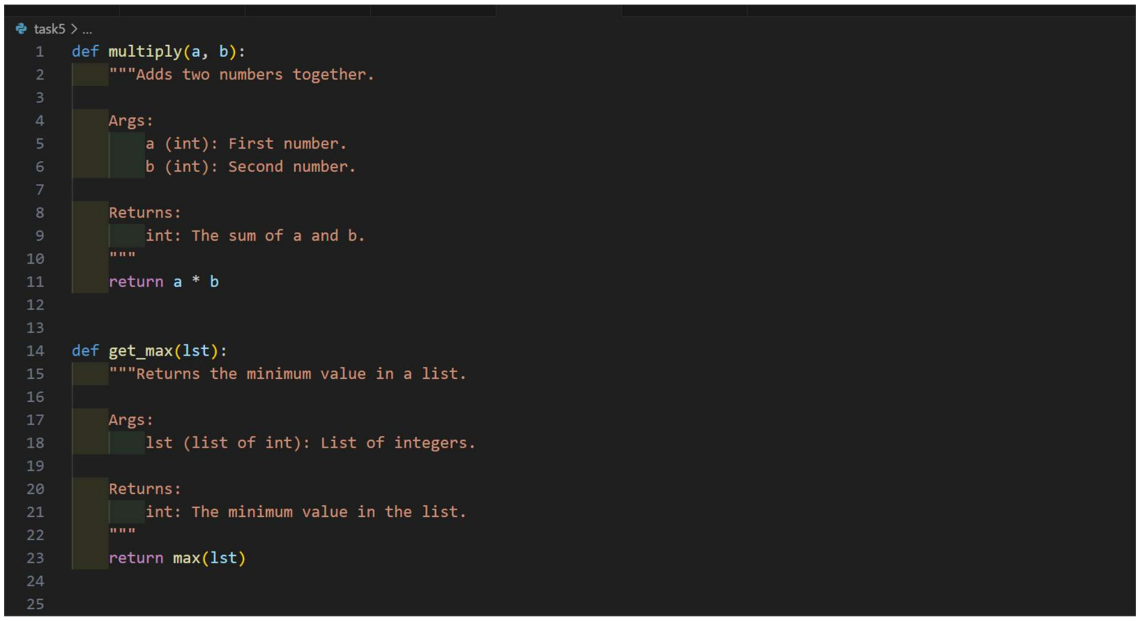
o Provide Python code with outdated or incorrect docstrings.

o Instruct AI to rewrite each docstring to match the current code behavior.

o Ensure corrections follow Google-style formatting.

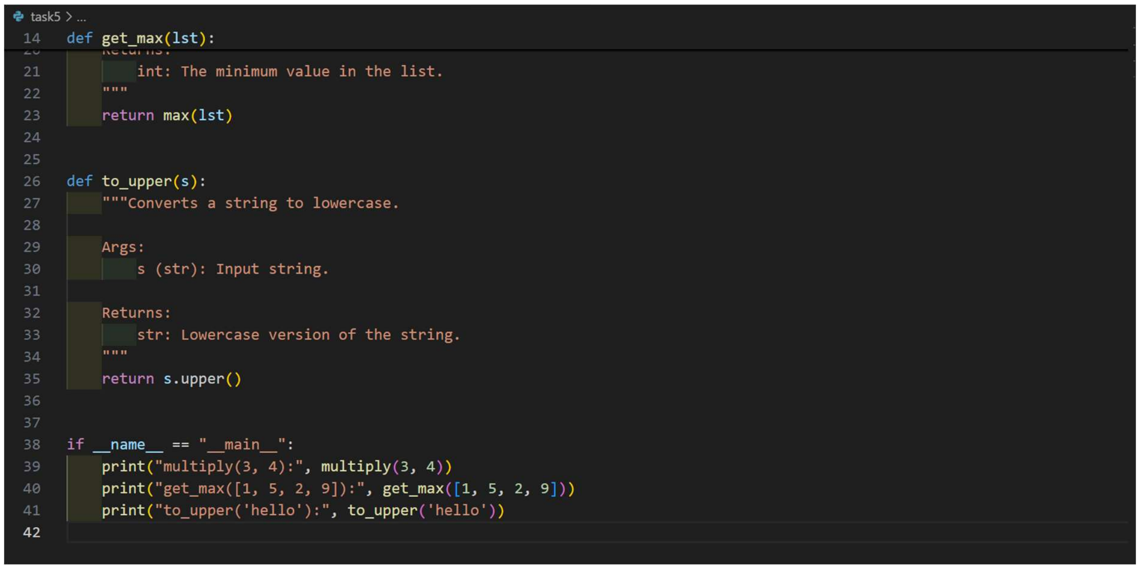
Expected Output 5:

o Python file with updated, accurate, and standardized docstrings.



task5 > ...

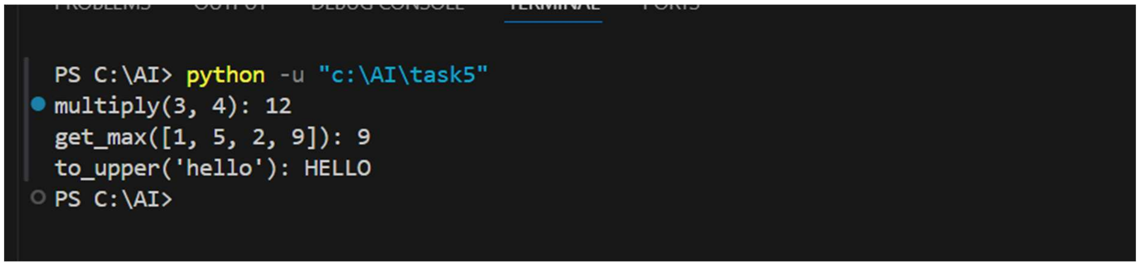
```python
def multiply(a, b):
    """Adds two numbers together.

    Args:
        a (int): First number.
        b (int): Second number.

    Returns:
        int: The sum of a and b.
    """
    return a * b


def get_max(lst):
    """Returns the minimum value in a list.

    Args:
        lst (list of int): List of integers.

    Returns:
        int: The minimum value in the list.
    """
    return max(lst)


```



task5 > ...

```python
def get_max(lst):
        int: The minimum value in the list.
    """
    return max(lst)


def to_upper(s):
    """Converts a string to lowercase.

    Args:
        s (str): Input string.

    Returns:
        str: Lowercase version of the string.
    """
    return s.upper()


if __name__ == "__main__":
    print("multiply(3, 4):", multiply(3, 4))
    print("get_max([1, 5, 2, 9]):", get_max([1, 5, 2, 9]))
    print("to_upper('hello'):", to_upper('hello'))

```

OUTPUT:


```
PS C:\AI> python -u "c:\AI\task5"
multiply(3, 4): 12
get_max([1, 5, 2, 9]): 9
to_upper('hello'): HELLO
PS C:\AI>
```