# AI-ASSISTED CODING

**NAME : ANJALI. JEELUKA**

**HT.NO : 2403A510F0**

**BATCH :05**

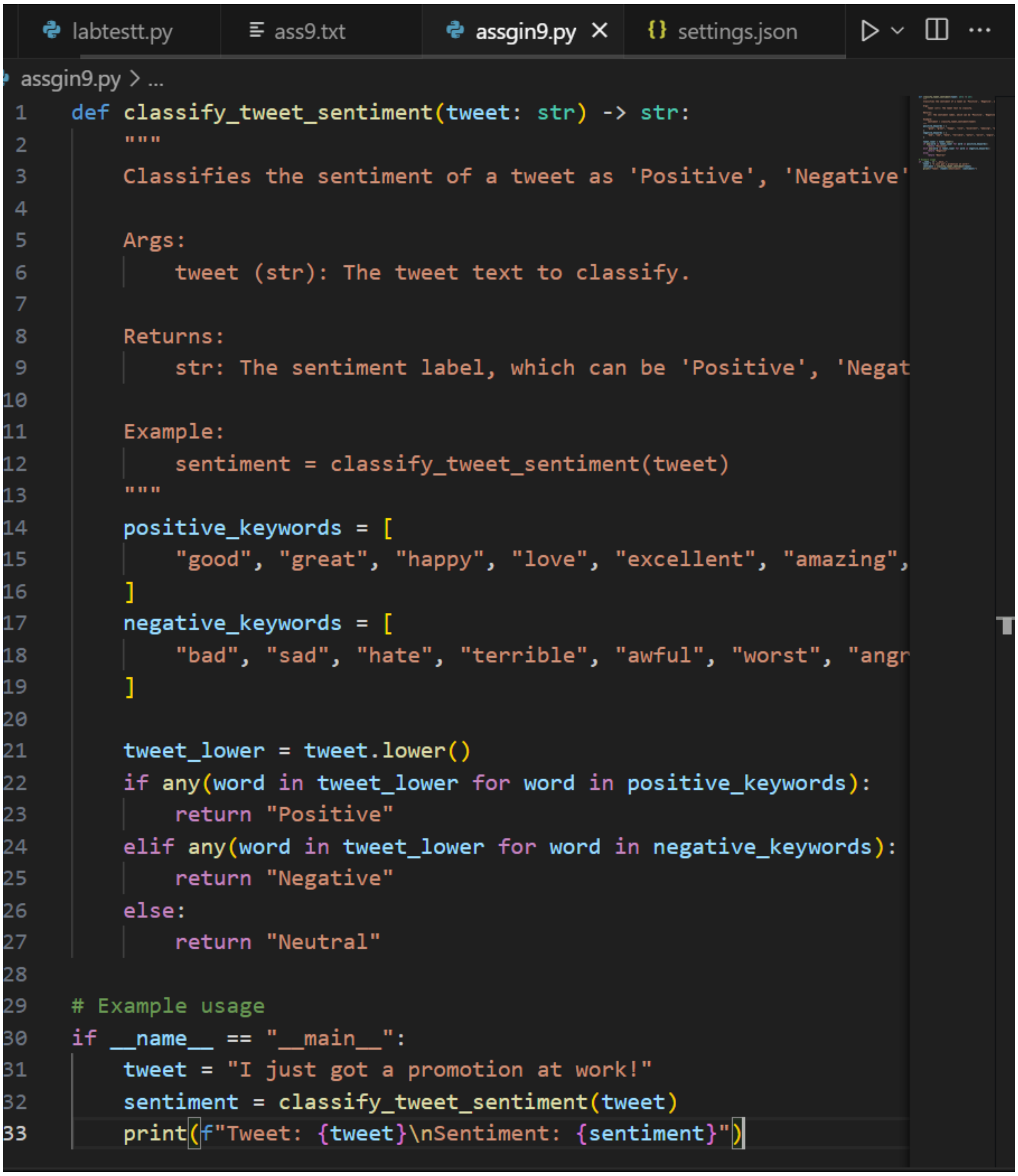
**DEPT:CSE**

**TASK-1:**

- ❖ **PROMPT :**

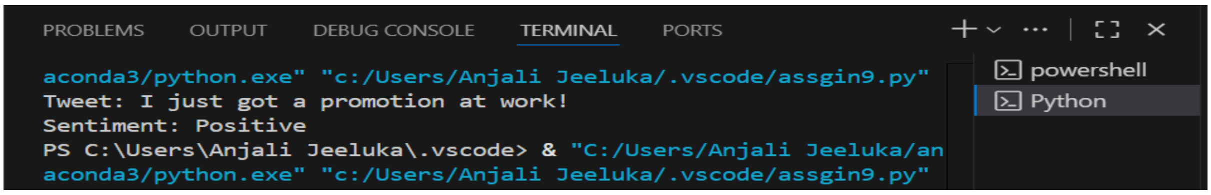
  Write Google-style docstrings for all the functions in a given Python script. Each docstring should include a brief description of the function, the parameters with their type hints, the return values with type hints, and an example usage section. The docstrings must follow the exact Google-style formatting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

- ❖ **CODE :**



```python
def classify_tweet_sentiment(tweet: str) -> str:
    """
    Classifies the sentiment of a tweet as 'Positive', 'Negative'

    Args:
        tweet (str): The tweet text to classify.

    Returns:
        str: The sentiment label, which can be 'Positive', 'Negat

    Example:
        sentiment = classify_tweet_sentiment(tweet)
    """
    positive_keywords = [
        "good", "great", "happy", "love", "excellent", "amazing",
    ]
    negative_keywords = [
        "bad", "sad", "hate", "terrible", "awful", "worst", "angr
    ]

    tweet_lower = tweet.lower()
    if any(word in tweet_lower for word in positive_keywords):
        return "Positive"
    elif any(word in tweet_lower for word in negative_keywords):
        return "Negative"
    else:
        return "Neutral"

# Example usage
if __name__ == "__main__":
    tweet = "I just got a promotion at work!"
    sentiment = classify_tweet_sentiment(tweet)
    print(f"Tweet: {tweet}\nSentiment: {sentiment}")
```

## OUTPUT :



```
aconda3/python.exe" "c:/Users/Anjali Jeeluka/.vscode/assgin9.py"
Tweet: I just got a promotion at work!
Sentiment: Positive
PS C:\Users\Anjali Jeeluka\.vscode> & "C:/Users/Anjali Jeeluka/an
aconda3/python.exe" "c:/Users/Anjali Jeeluka/.vscode/assgin9.py"
```

- ➢ **OBSERVATION:**

The code defines a simple sentiment classifier that labels tweets as Positive, Negative, or Neutral by checking for predefined keywords. It works case-insensitively by converting tweets to lowercase and uses list membership checks for classification. While straightforward and easy to read, it may misclassify tweets if context or sarcasm is involved, since it relies only on keyword matching.

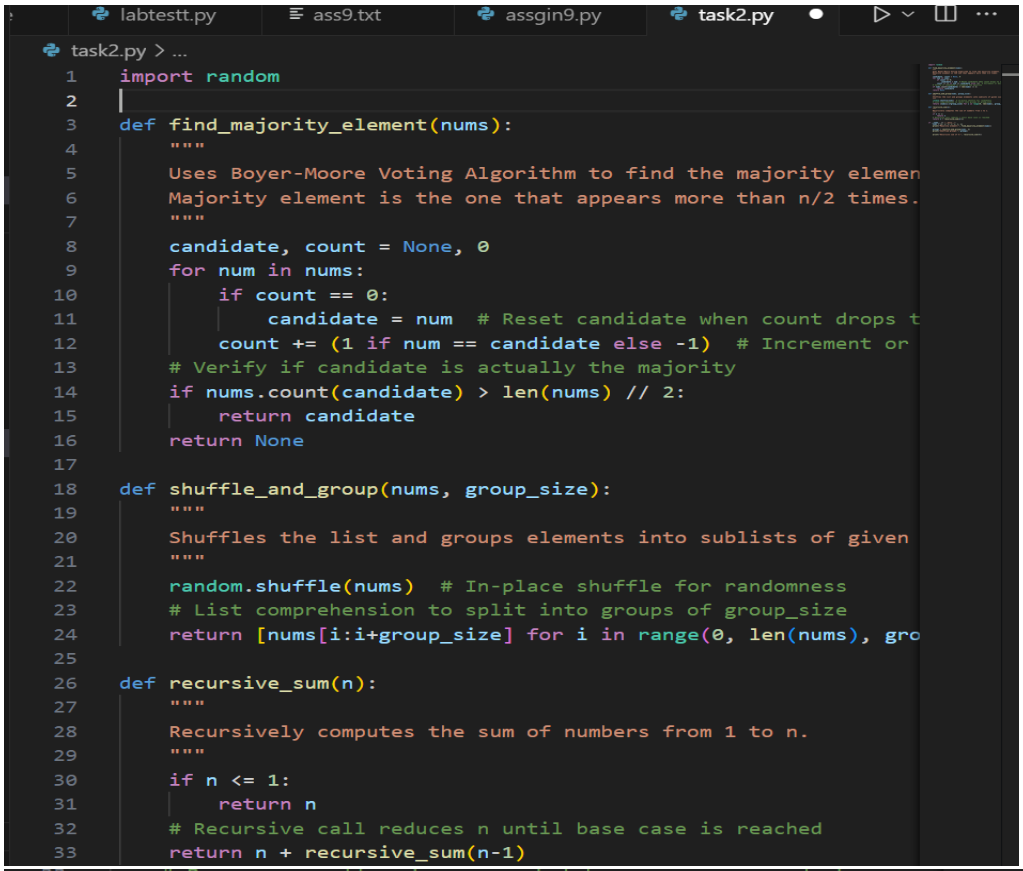
**TASK-2 :**

- ➢ **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.

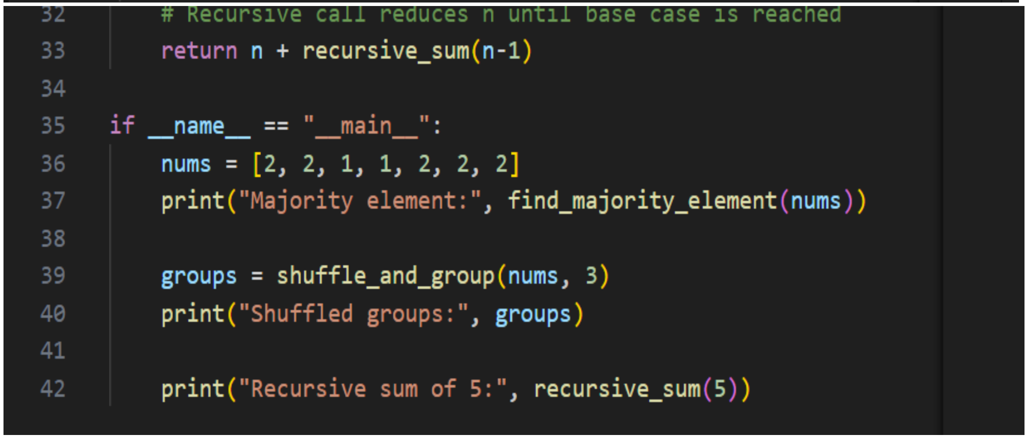
Expected Output:

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

- ➢ **CODE :**



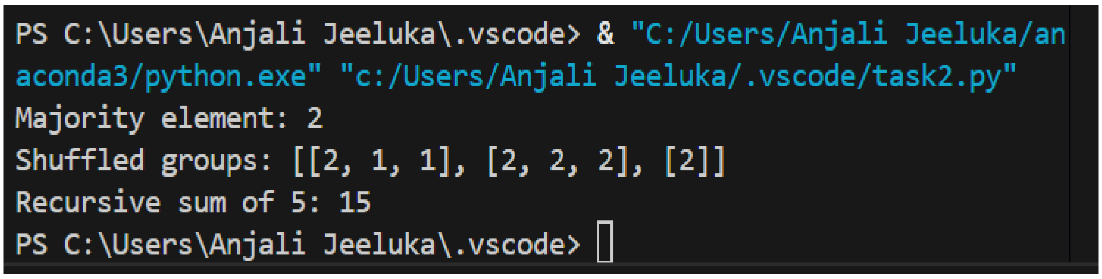
```python
import random

def find_majority_element(nums):
    """
    Uses Boyer-Moore Voting Algorithm to find the majority elemen
    Majority element is the one that appears more than n/2 times.
    """
    candidate, count = None, 0
    for num in nums:
        if count == 0:
            candidate = num  # Reset candidate when count drops t
        count += (1 if num == candidate else -1)  # Increment or
    # Verify if candidate is actually the majority
    if nums.count(candidate) > len(nums) // 2:
        return candidate
    return None

def shuffle_and_group(nums, group_size):
    """
    Shuffles the list and groups elements into sublists of given
    """
    random.shuffle(nums)  # In-place shuffle for randomness
    # List comprehension to split into groups of group_size
    return [nums[i:i+group_size] for i in range(0, len(nums), gro

def recursive_sum(n):
    """
    Recursively computes the sum of numbers from 1 to n.
    """
    if n <= 1:
        return n
    # Recursive call reduces n until base case is reached
    return n + recursive_sum(n-1)
```



```python
    # Recursive call reduces n until base case is reached
    return n + recursive_sum(n-1)
```

```python

if __name__ == "__main__":
    nums = [2, 2, 1, 1, 2, 2, 2]
    print("Majority element:", find_majority_element(nums))

    groups = shuffle_and_group(nums, 3)
    print("Shuffled groups:", groups)

    print("Recursive sum of 5:", recursive_sum(5))
```

## ➢ OUTPUT :



```
PS C:\Users\Anjali Jeeluka\.vscode> & "C:/Users/Anjali Jeeluka/an
aconda3/python.exe" "c:/Users/Anjali Jeeluka/.vscode/task2.py"
Majority element: 2
Shuffled groups: [[2, 1, 1], [2, 2, 2], [2]]
Recursive sum of 5: 15
PS C:\Users\Anjali Jeeluka\.vscode> 
```

## ➢ OBSERVATION:

- Uses Boyer-Moore Voting Algorithm to efficiently find the majority element in a list.
- Randomly shuffles a list and splits it into groups of a given size.
- Recursively sums numbers from 1 to n (less efficient for large n).
- Demonstrates each function with example data in the main block.
- Code is modular, readable, and includes concise comments for complex logic.

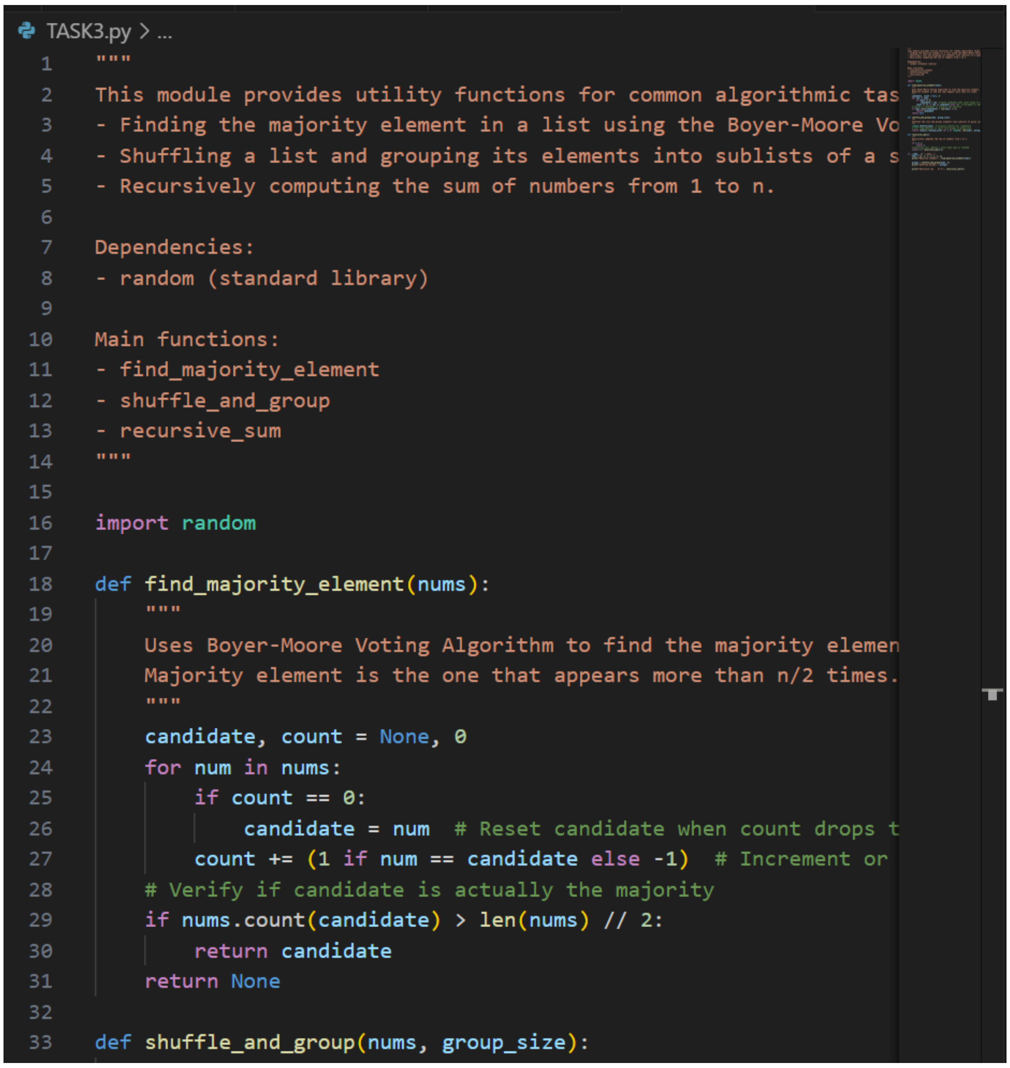
**TASK-3:**

- **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.

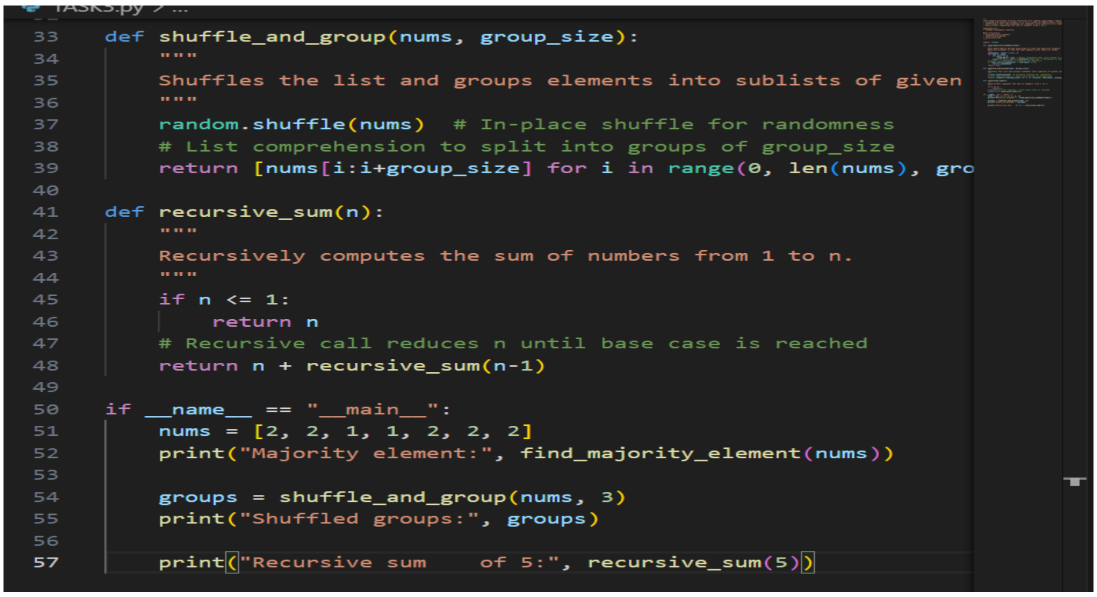
  Expected Output #3:
  A complete, clear, and concise module-level docstring placed at the beginning of the file.

➢ **CODE :**



TASK3.py > ...

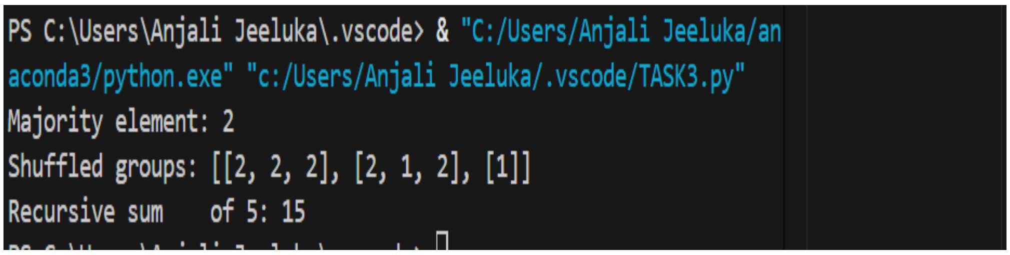
```python
"""
This module provides utility functions for common algorithmic tas
- Finding the majority element in a list using the Boyer-Moore Vo
- Shuffling a list and grouping its elements into sublists of a s
- Recursively computing the sum of numbers from 1 to n.

Dependencies:
- random (standard library)

Main functions:
- find_majority_element
- shuffle_and_group
- recursive_sum
"""

import random

def find_majority_element(nums):
    """
    Uses Boyer-Moore Voting Algorithm to find the majority elemen
    Majority element is the one that appears more than n/2 times.
    """
    candidate, count = None, 0
    for num in nums:
        if count == 0:
            candidate = num  # Reset candidate when count drops t
        count += (1 if num == candidate else -1)  # Increment or
    # Verify if candidate is actually the majority
    if nums.count(candidate) > len(nums) // 2:
        return candidate
    return None

def shuffle_and_group(nums, group_size):
```



TASK3.py > ...

```python
def shuffle_and_group(nums, group_size):
    """
    Shuffles the list and groups elements into sublists of given
    """
    random.shuffle(nums)  # In-place shuffle for randomness
    # List comprehension to split into groups of group_size
    return [nums[i:i+group_size] for i in range(0, len(nums), gro

def recursive_sum(n):
    """
    Recursively computes the sum of numbers from 1 to n.
    """
    if n <= 1:
        return n
    # Recursive call reduces n until base case is reached
    return n + recursive_sum(n-1)

if __name__ == "__main__":
    nums = [2, 2, 1, 1, 2, 2, 2]
    print("Majority element:", find_majority_element(nums))

    groups = shuffle_and_group(nums, 3)
    print("Shuffled groups:", groups)

    print("Recursive sum    of 5:", recursive_sum(5))
```

➢ OUTPUT:



```
PS C:\Users\Anjali Jeeluka\.vscode> & "C:/Users/Anjali Jeeluka/an
aconda3/python.exe" "c:/Users/Anjali Jeeluka/.vscode/TASK3.py"
Majority element: 2
Shuffled groups: [[2, 2, 2], [2, 1, 2], [1]]
Recursive sum   of 5: 15
```

➢ OBSERVATION:

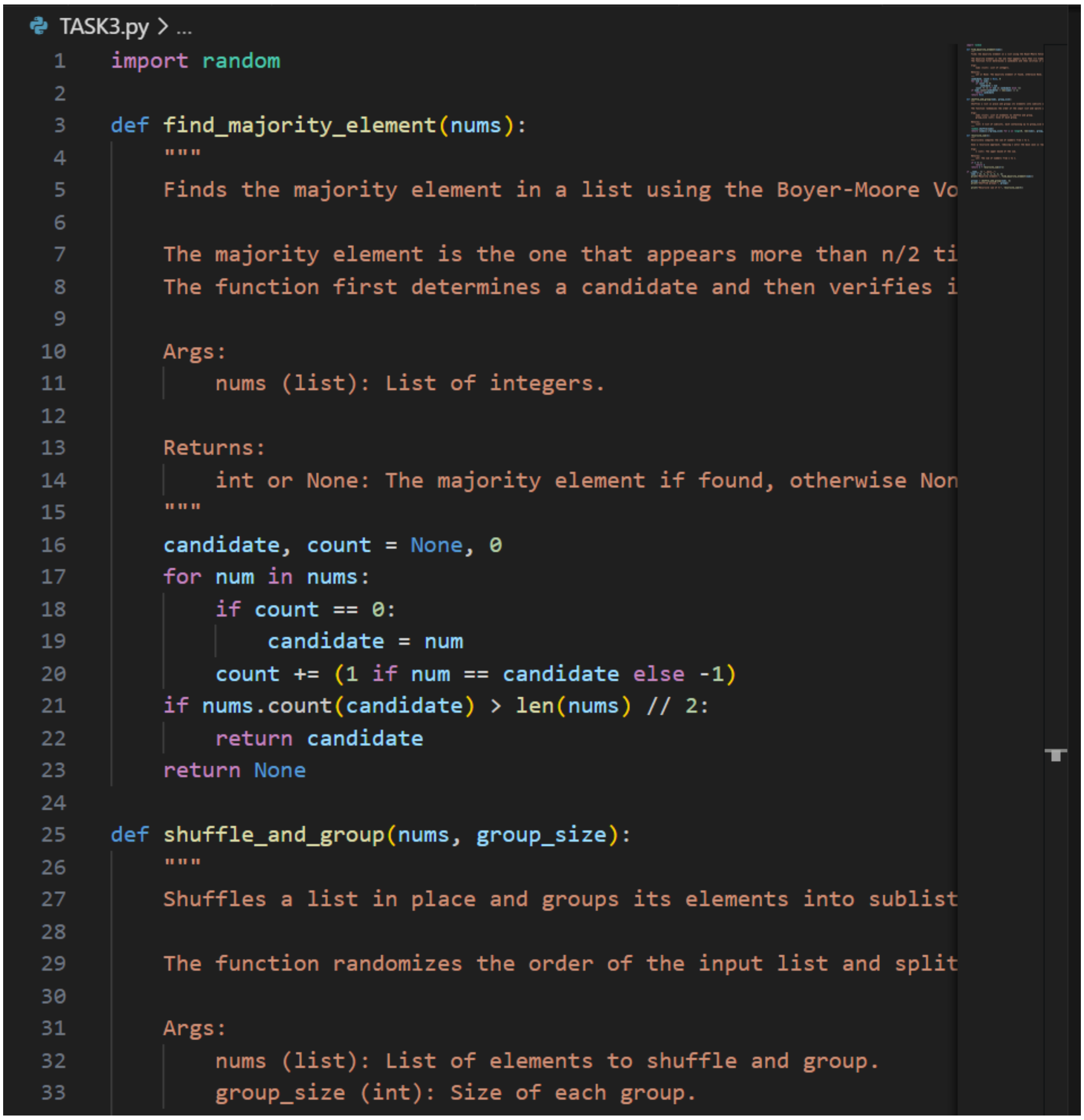
- Provides functions for majority element detection, shuffling/grouping, and recursive sum.
- Uses Boyer-Moore Voting Algorithm for efficient majority search.
- Shuffles list in place and groups by specified size.

Recursively sums numbers from 1 to n.

- Code is modular, clear, and uses only the standard random library.

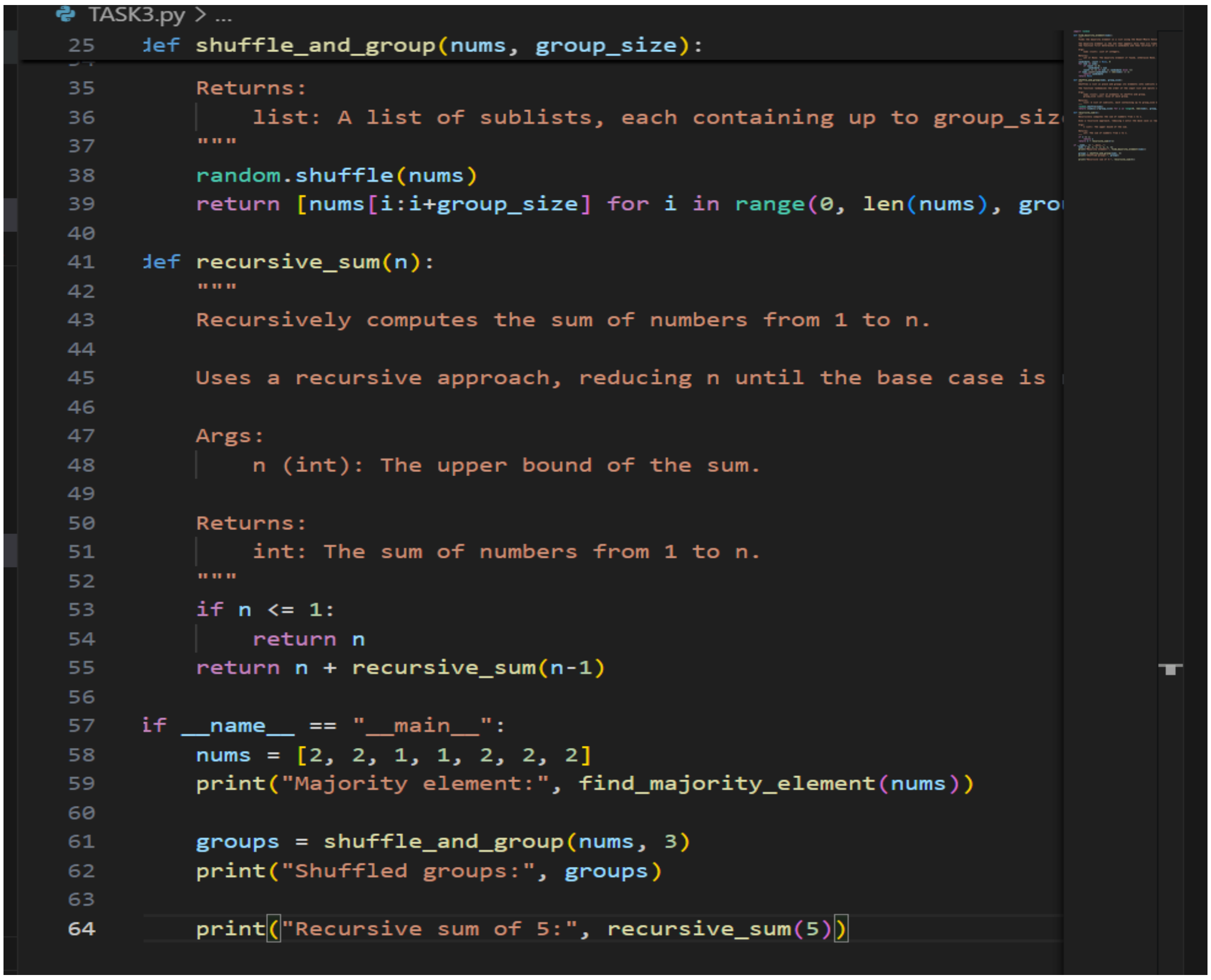
**TASK-4:**

- **PROMPT:**
  How can you take the following Python code with inline comments and convert those comments into structured Google-style function docstrings? Move all relevant details from the comments into the docstrings while keeping their meaning intact. The final output should be the same Python code, but with comments removed and replaced by clear, standardized Google-style docstrings.
- **CODE:**



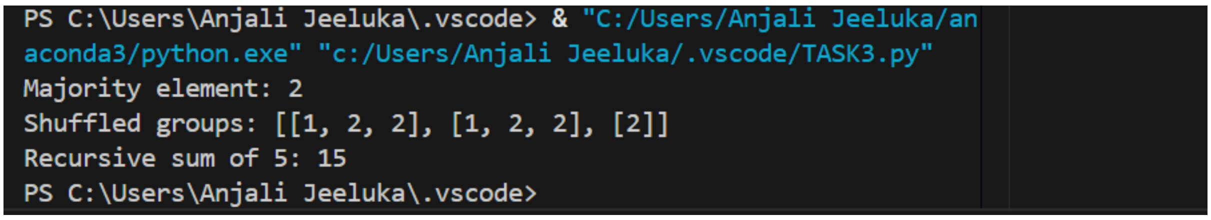
TASK3.py > ...

```python
import random

def find_majority_element(nums):
    """
    Finds the majority element in a list using the Boyer-Moore Vo

    The majority element is the one that appears more than n/2 ti
    The function first determines a candidate and then verifies i

    Args:
        nums (list): List of integers.

    Returns:
        int or None: The majority element if found, otherwise Non
    """
    candidate, count = None, 0
    for num in nums:
        if count == 0:
            candidate = num
        count += (1 if num == candidate else -1)
    if nums.count(candidate) > len(nums) // 2:
        return candidate
    return None

def shuffle_and_group(nums, group_size):
    """
    Shuffles a list in place and groups its elements into sublist

    The function randomizes the order of the input list and split

    Args:
        nums (list): List of elements to shuffle and group.
        group_size (int): Size of each group.
```



```
TASK3.py > ...
def shuffle_and_group(nums, group_size):
    Returns:
        list: A list of sublists, each containing up to group_siz
    """
    random.shuffle(nums)
    return [nums[i:i+group_size] for i in range(0, len(nums), gro

def recursive_sum(n):
    """
    Recursively computes the sum of numbers from 1 to n.

    Uses a recursive approach, reducing n until the base case is

    Args:
        n (int): The upper bound of the sum.

    Returns:
        int: The sum of numbers from 1 to n.
    """
    if n <= 1:
        return n
    return n + recursive_sum(n-1)

if __name__ == "__main__":
    nums = [2, 2, 1, 1, 2, 2, 2]
    print("Majority element:", find_majority_element(nums))

    groups = shuffle_and_group(nums, 3)
    print("Shuffled groups:", groups)

    print("Recursive sum of 5:", recursive_sum(5))
```

- **OUTPUT:**



```
PS C:\Users\Anjali Jeeluka\.vscode> & "C:/Users/Anjali Jeeluka/an
aconda3/python.exe" "c:/Users/Anjali Jeeluka/.vscode/TASK3.py"
Majority element: 2
Shuffled groups: [[1, 2, 2], [1, 2, 2], [2]]
Recursive sum of 5: 15
PS C:\Users\Anjali Jeeluka\.vscode>
```

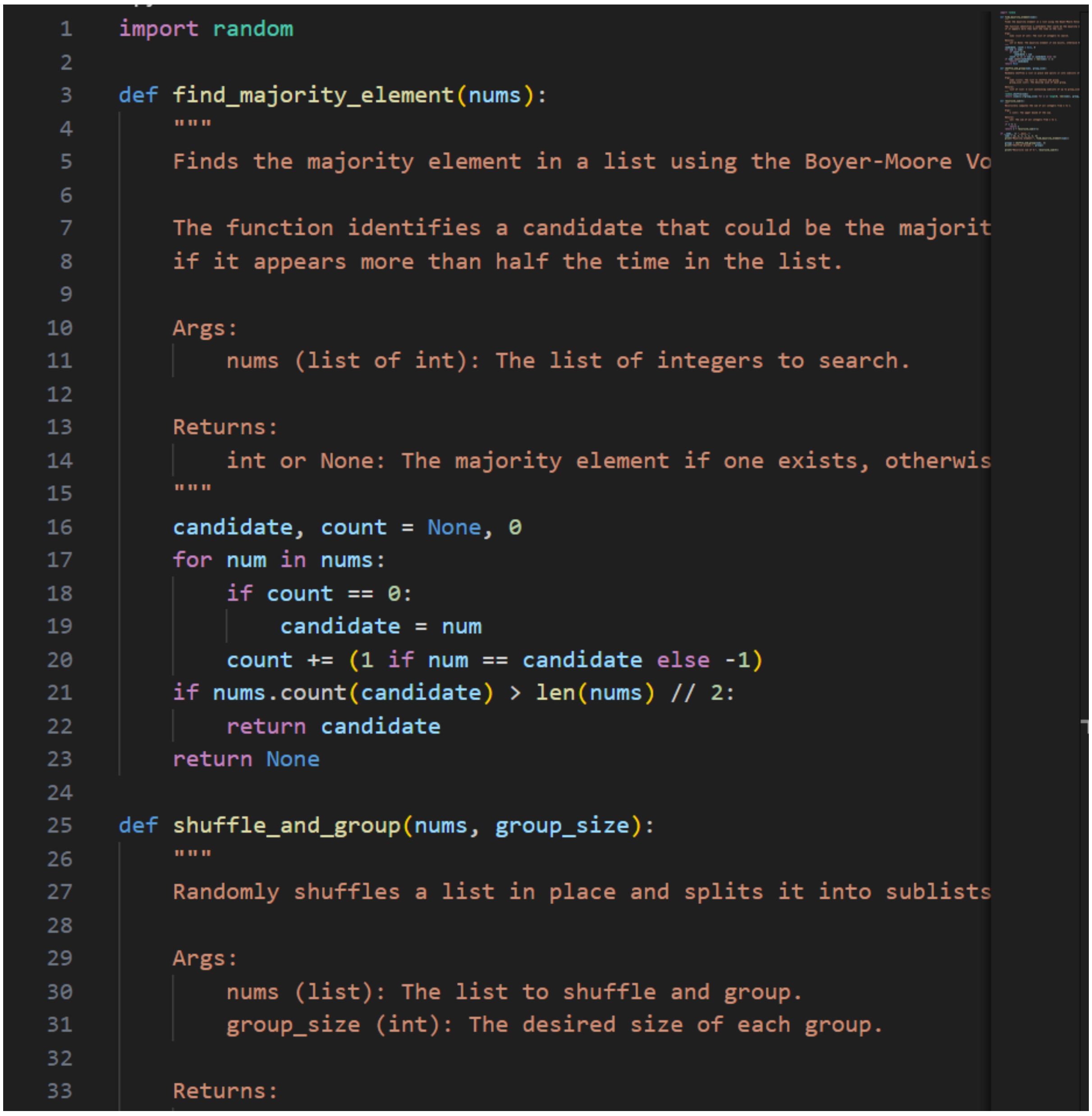
## OBSERVATION:

- Implements majority element detection, shuffling/grouping, and recursive sum functions.
- Uses efficient Boyer-Moore algorithm for majority search.
- Shuffles and groups list elements by specified size.
- Recursively sums numbers from 1 to n.
- Functions are well-documented with Google-style docstrings.

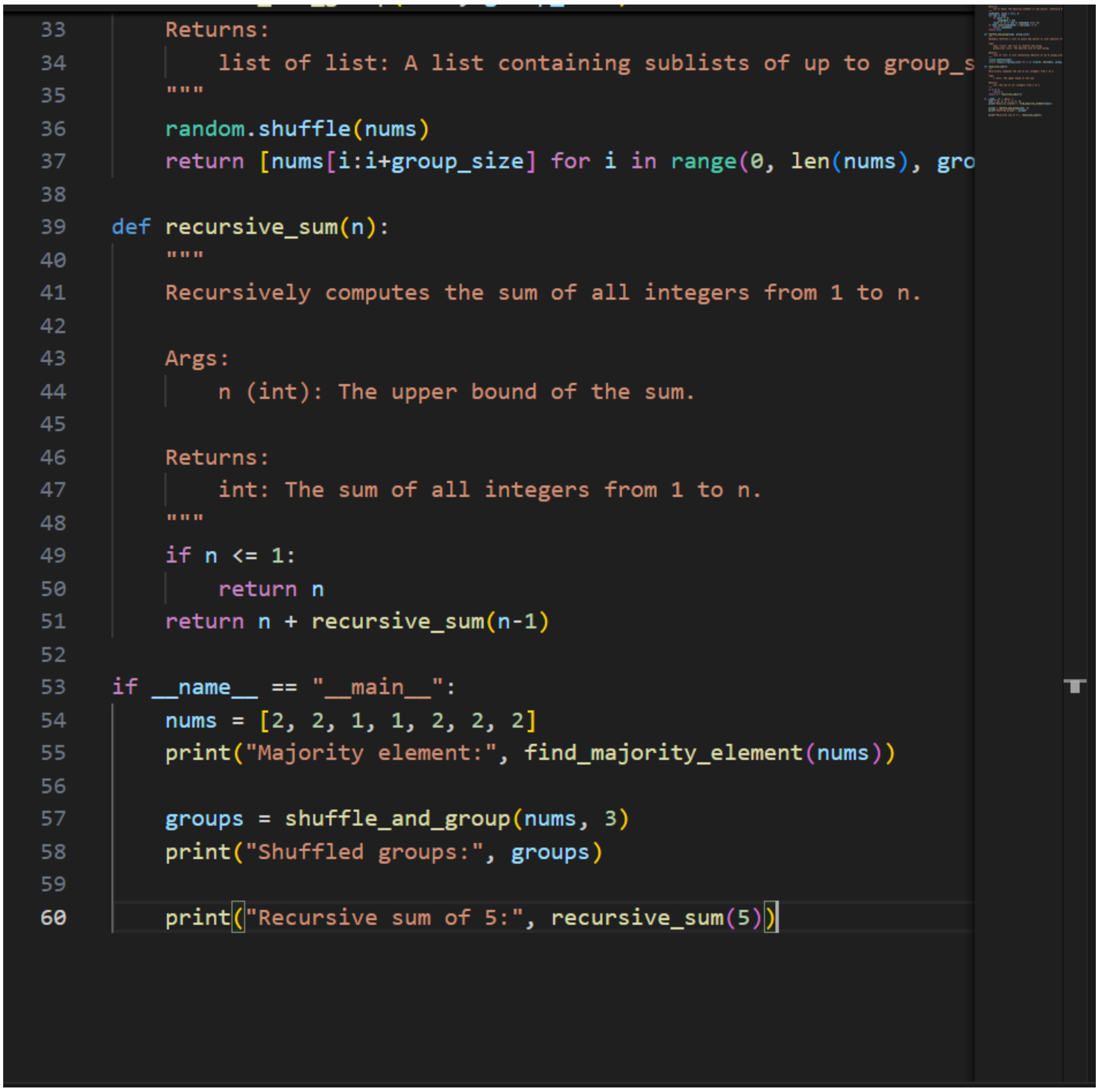
- Code is modular, clear, and uses only the standard random library.

**TASK-5:**

- **PROMPT:**
  How can you review the following Python code, which contains outdated or inaccurate docstrings, and rewrite them so that they accurately describe the current behavior of each function? Make sure the updated docstrings follow the Google-style format. The output should be the same Python file, but with corrected, accurate, and standardized docstrings.
- **CODE :**

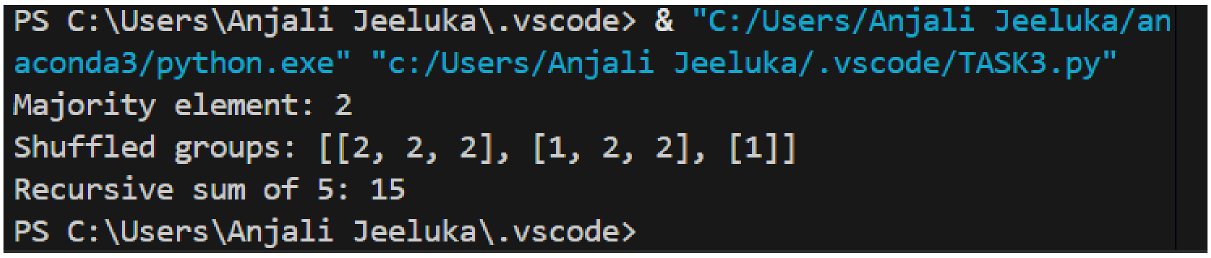


```python
import random

def find_majority_element(nums):
    """
    Finds the majority element in a list using the Boyer-Moore Vo

    The function identifies a candidate that could be the majorit
    if it appears more than half the time in the list.

    Args:
        nums (list of int): The list of integers to search.

    Returns:
        int or None: The majority element if one exists, otherwis
    """
    candidate, count = None, 0
    for num in nums:
        if count == 0:
            candidate = num
        count += (1 if num == candidate else -1)
    if nums.count(candidate) > len(nums) // 2:
        return candidate
    return None

def shuffle_and_group(nums, group_size):
    """
    Randomly shuffles a list in place and splits it into sublists

    Args:
        nums (list): The list to shuffle and group.
        group_size (int): The desired size of each group.

    Returns:
```



```
    Returns:
        list of list: A list containing sublists of up to group_s
    """
    random.shuffle(nums)
    return [nums[i:i+group_size] for i in range(0, len(nums), gro

def recursive_sum(n):
    """
    Recursively computes the sum of all integers from 1 to n.

    Args:
        n (int): The upper bound of the sum.

    Returns:
        int: The sum of all integers from 1 to n.
    """
    if n <= 1:
        return n
    return n + recursive_sum(n-1)

if __name__ == "__main__":
    nums = [2, 2, 1, 1, 2, 2, 2]
    print("Majority element:", find_majority_element(nums))

    groups = shuffle_and_group(nums, 3)
    print("Shuffled groups:", groups)

    print("Recursive sum of 5:", recursive_sum(5))
```

➢ **OUTPUT :**



```
PS C:\Users\Anjali Jeeluka\.vscode> & "C:/Users/Anjali Jeeluka/an
aconda3/python.exe" "c:/Users/Anjali Jeeluka/.vscode/TASK3.py"
Majority element: 2
Shuffled groups: [[2, 2, 2], [1, 2, 2], [1]]
Recursive sum of 5: 15
PS C:\Users\Anjali Jeeluka\.vscode>
```

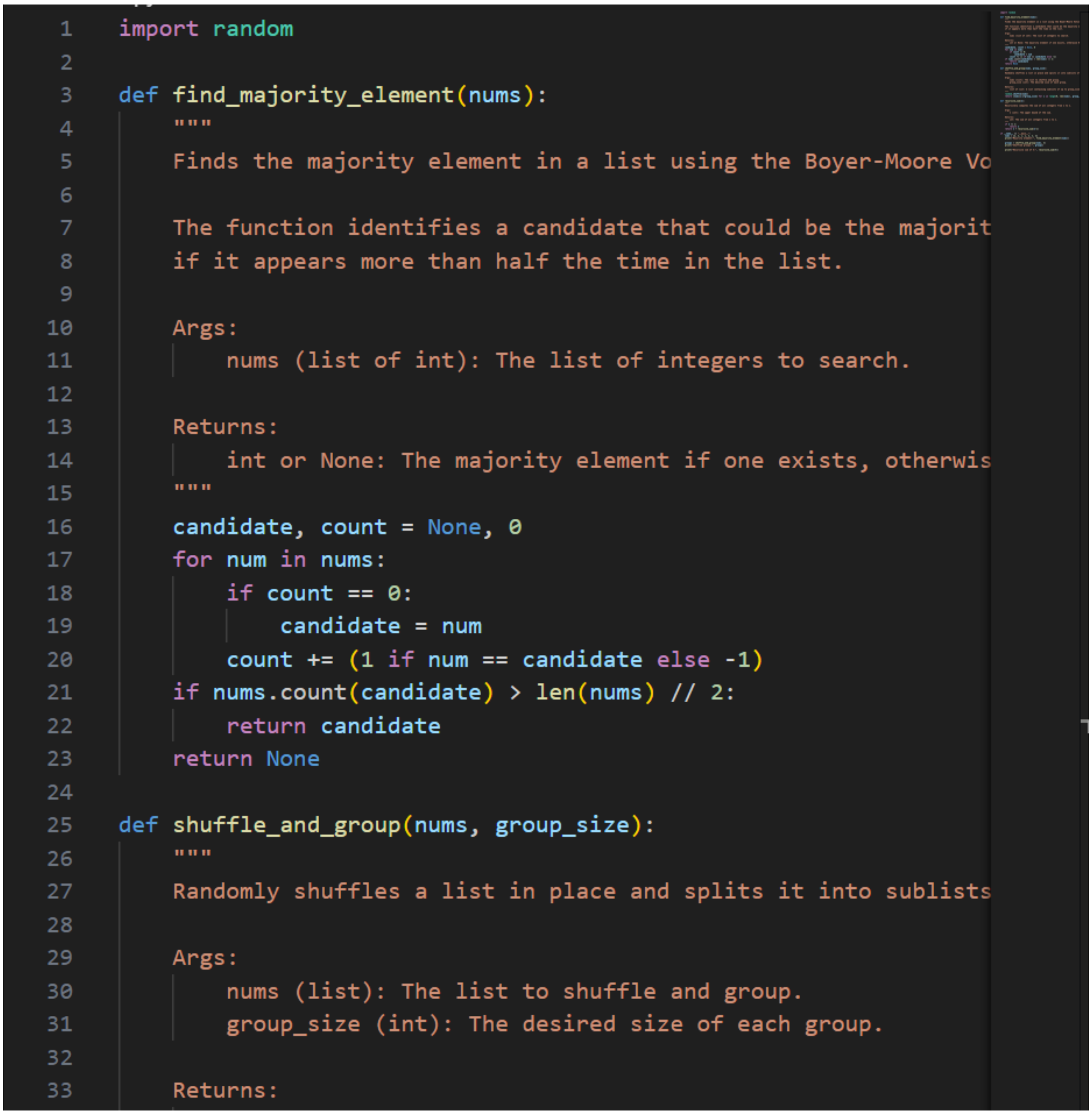
➢ **OBSERVATION :**

- Implements functions for majority element detection, shuffling/grouping, and recursive sum.
- Uses efficient algorithms and clear, Google-style docstrings.
- Demonstrates each function with example data.

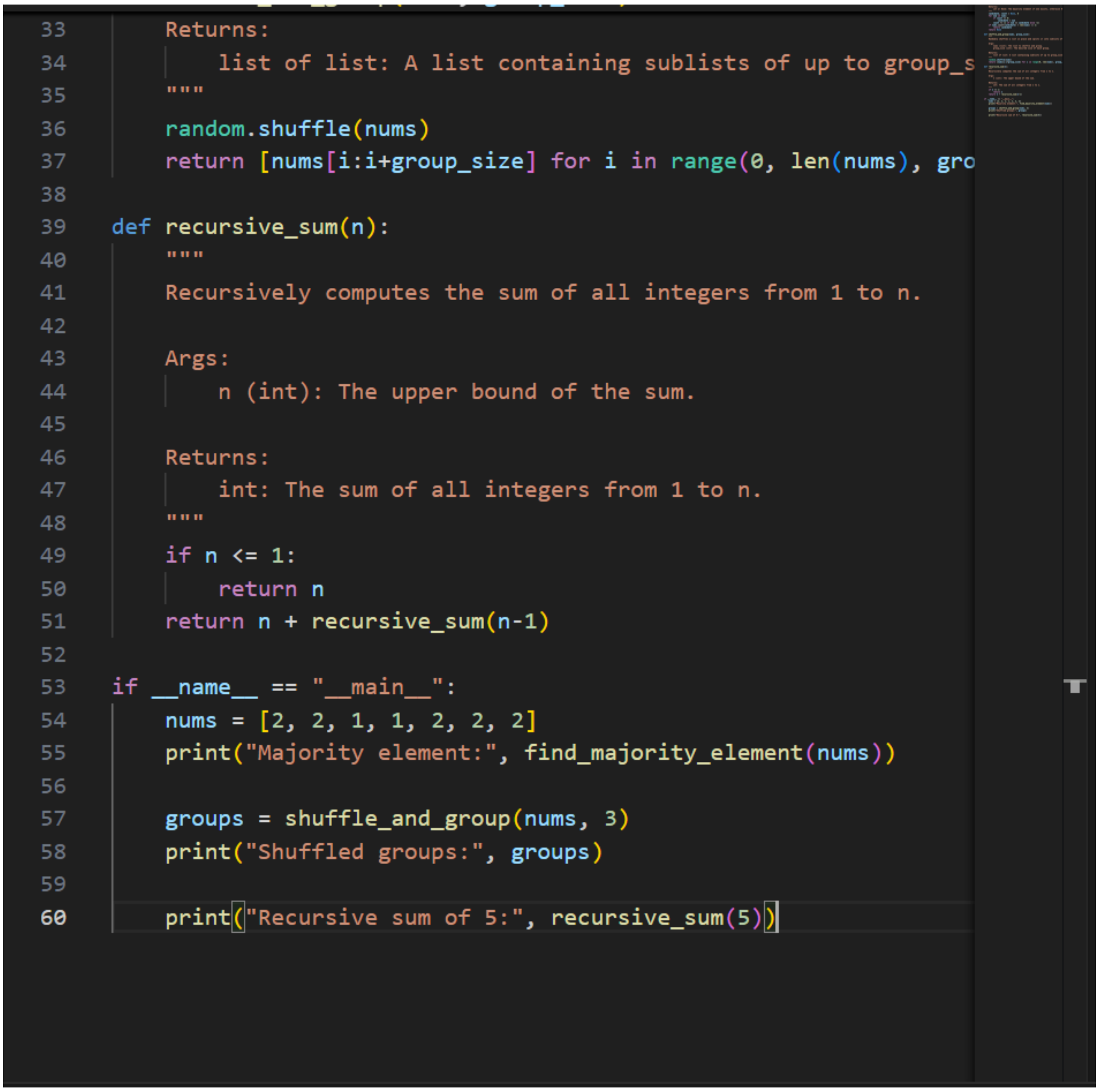
- Code is modular, readable, and only depends on the standard random library.

**TASK-6:**

- **PROMPT:**
  How can you review the following Python code, which contains outdated or inaccurate docstrings, and rewrite them so that they accurately describe the current behavior of each function? Make sure the updated docstrings follow the Google-style format. The output should be the same Python file, but with corrected, accurate, and standardized docstrings.
- **CODE:**

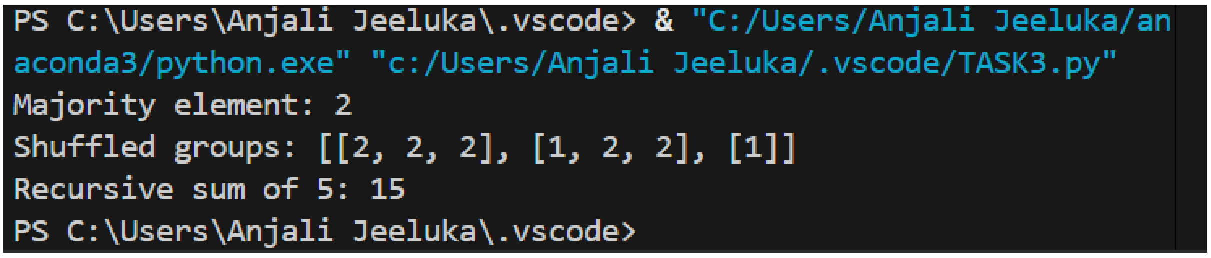


```python
import random

def find_majority_element(nums):
    """
    Finds the majority element in a list using the Boyer-Moore Vo

    The function identifies a candidate that could be the majorit
    if it appears more than half the time in the list.

    Args:
        nums (list of int): The list of integers to search.

    Returns:
        int or None: The majority element if one exists, otherwis
    """
    candidate, count = None, 0
    for num in nums:
        if count == 0:
            candidate = num
        count += (1 if num == candidate else -1)
    if nums.count(candidate) > len(nums) // 2:
        return candidate
    return None

def shuffle_and_group(nums, group_size):
    """
    Randomly shuffles a list in place and splits it into sublists

    Args:
        nums (list): The list to shuffle and group.
        group_size (int): The desired size of each group.

    Returns:
```



```
    Returns:
        list of list: A list containing sublists of up to group_s
    """
    random.shuffle(nums)
    return [nums[i:i+group_size] for i in range(0, len(nums), gro

def recursive_sum(n):
    """
    Recursively computes the sum of all integers from 1 to n.

    Args:
        n (int): The upper bound of the sum.

    Returns:
        int: The sum of all integers from 1 to n.
    """
    if n <= 1:
        return n
    return n + recursive_sum(n-1)

if __name__ == "__main__":
    nums = [2, 2, 1, 1, 2, 2, 2]
    print("Majority element:", find_majority_element(nums))

    groups = shuffle_and_group(nums, 3)
    print("Shuffled groups:", groups)

    print("Recursive sum of 5:", recursive_sum(5))
```

➢ **OUTPUT :**



```
PS C:\Users\Anjali Jeeluka\.vscode> & "C:/Users/Anjali Jeeluka/an
aconda3/python.exe" "c:/Users/Anjali Jeeluka/.vscode/TASK3.py"
Majority element: 2
Shuffled groups: [[2, 2, 2], [1, 2, 2], [1]]
Recursive sum of 5: 15
PS C:\Users\Anjali Jeeluka\.vscode>
```

➢ OBSERVATION:

- Implements functions for majority element detection, shuffling/grouping, and recursive sum.
- Uses efficient algorithms and clear, Google-style docstrings.
- Demonstrates each function with example data.

- Code is modular, readable, and only depends on the standard random library.