1.Write a code to reverse a string.

```
def reverse_string(s):
    return s[::-1]

# Example
string = "Hello, World!"
reversed_string = reverse_string(string)
print(reversed_string)
```

```
!dlroW ,olleH
```

2.Write a code to count the number of vowels in a string.

```
def count_vowels(s):
    vowels = "aeiouAEIOU"
    return sum(1 for char in s if char in vowels)

# Example
string = "Hello, World!"
vowel_count = count_vowels(string)
print(vowel_count)
```

```
3
```

3.Write a code to check if a given string is a palindrome or not.

```
def is_palindrome(s):

    cleaned = ''.join(char.lower() for char in s if char.isalnum())

    return cleaned == cleaned[::-1]

# Example
string = "A man, a plan, a canal, Panama!"
if is_palindrome(string):
    print("The string is a palindrome.")
else:
    print("The string is not a palindrome.")
```

```
The string is a palindrome.
```

4. Write a code to check if two given strings are anagrams of each other.

```
def are_anagrams(str1, str2):
    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()
    if len(str1) != len(str2):
        return False
```

```
    char_count = {}

    for char in str1:
        char_count[char] = char_count.get(char, 0) + 1

    for char in str2:
        if char in char_count:
            char_count[char] -= 1
        else:
            return False
    return all(count == 0 for count in char_count.values())

# Example
string1 = "Listen"
string2 = "Silent"
if are_anagrams(string1, string2):
    print(f"'{string1}' and '{string2}' are anagrams.")
else:
    print(f"'{string1}' and '{string2}' are not anagrams.")
```

```
'Listen' and 'Silent' are anagrams.
```

5. Write a code to find all occurrences of a given substring within another string.

```
def find_all_occurrences(text, substring):
    return [i for i in range(len(text)) if text.startswith(substring,
i)]

# Example
text = "This is a test. This test is only a test."
substring = "test"
positions = find_all_occurrences(text, substring)
print("Occurrences at positions:", positions)
```

```
Occurrences at positions: [10, 21, 36]
```

6.Write a code to perform basic string compression using the counts of repeated characters.

```
def compress_string(s):
    if not s:
        return ""

    compressed = []
    count = 1

    for i in range(1, len(s)):
        if s[i] == s[i - 1]:
            count += 1
        else:
```

```
            compressed.append(s[i - 1] + (str(count) if count > 1 else
''))
            count = 1

    compressed.append(s[-1] + (str(count) if count > 1 else ''))
    compressed_string = ''.join(compressed)

    return compressed_string if len(compressed_string) < len(s) else s

# Example
input_string = "aabcccccaaa"
compressed = compress_string(input_string)
print(f"Original: {input_string}")
print(f"Compressed: {compressed}")
```

```
Original: aabcccccaaa
Compressed: a2bc5a3
```

7. Write a code to determine if a string has all unique characters.

```
def has_unique_characters(s):
    return len(s) == len(set(s))

# Example
input_string = "abcdefg"
if has_unique_characters(input_string):
    print(f"The string '{input_string}' has all unique characters.")
else:
    print(f"The string '{input_string}' does not have all unique
characters.")
```

```
The string 'abcdefg' has all unique characters.
```

8.Write a code to convert a given string to uppercase or lowercase.

```
# Convert to uppercase
original_string = "Hello, World!"
uppercase_string = original_string.upper()
print("Uppercase:", uppercase_string)

# Convert to lowercase
lowercase_string = original_string.lower()
print("Lowercase:", lowercase_string)
```

```
Uppercase: HELLO, WORLD!
Lowercase: hello, world!
```

9.Write a code to count the number of words in a string.

```
def count_words(s) :
    words = s.split()
    return len(words)
```

```
# Example
input_string = "Hello, how are you today?"
word_count = count_words(input_string)
print(f"The number of words in the string is: {word_count}")
```

```
The number of words in the string is: 5
```

10.Write a code to concatenate two strings without using the + operator.

```
def concatenate_strings(str1, str2):
    return ''.join([str1, str2])

# Example
string1 = "Hello, "
string2 = "World!"
result = concatenate_strings(string1, string2)
print(result)
```

```
Hello, World!
```

11.Write a code to remove all occurrences of a specific element from a list.

```
def remove_occurrences(lst, element):
    return [item for item in lst if item != element]

# Example:
my_list = [1, 2, 3, 4, 2, 5, 2, 6]
element_to_remove = 2
new_list = remove_occurrences(my_list, element_to_remove)
print(new_list)
```

```
[1, 3, 4, 5, 6]
```

12.Implement a code to find the second largest number in a given list of integers.

```
def second_largest(lst):
    if len(lst) < 2:
        return None

    unique_nums = list(set(lst))
    unique_nums.sort(reverse=True)

    return unique_nums[1] if len(unique_nums) > 1 else None

# Example:
numbers = [10, 20, 4, 45, 99, 99, 45]
print(second_largest(numbers))
```

```
45
```

13.Create a code to count the occurrences of each element in a list and return a dictionary with elements as keys and their counts as values.

```
def count_occurrences(lst):
    counts = {}
    for item in lst:
        counts[item] = counts.get(item, 0) + 1
    return counts

# Example:
my_list = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
result = count_occurrences(my_list)
print(result)
```

```
{1: 1, 2: 2, 3: 3, 4: 4}
```

14.Write a code to reverse a list in-place without using any built-in reverse functions.

```
def reverse_list(lst):
    left, right = 0, len(lst) - 1

    while left < right:
        lst[left], lst[right] = lst[right], lst[left]
        left += 1
        right -= 1

# Example:
my_list = [1, 2, 3, 4, 5]
reverse_list(my_list)
print(my_list)
```

```
[5, 4, 3, 2, 1]
```

15.Implement a code to find and remove duplicates from a list while preserving the original order of elements.

```
def remove_duplicates(lst):
    seen = set()
    result = []

    for item in lst:
        if item not in seen:
            seen.add(item)
            result.append(item)

    return result
```

```
# Example:
my_list = [1, 2, 3, 2, 4, 3, 5, 6, 5]
new_list = remove_duplicates(my_list)
print(new_list)
```

```
[1, 2, 3, 4, 5, 6]
```

16.Create a code to check if a given list is sorted (either in ascending or descending order) or not.

```
def is_sorted(lst):
    if lst == sorted(lst):
        return "Ascending"
    elif lst == sorted(lst, reverse=True):
        return "Descending"
    else:
        return "Not sorted"

# Example:
print(is_sorted([1, 2, 3, 4, 5]))
print(is_sorted([5, 4, 3, 2, 1]))
print(is_sorted([1, 3, 2, 4, 5]))
```

```
Ascending
Descending
Not sorted
```

17. Write a code to merge two sorted lists into a single sorted list.

```
def merge_sorted_lists(lst1, lst2):
    merged = []
    i, j = 0, 0
    while i < len(lst1) and j < len(lst2):
        if lst1[i] < lst2[j]:
            merged.append(lst1[i])
            i += 1
        else:
            merged.append(lst2[j])
            j += 1

    merged.extend(lst1[i:])
    merged.extend(lst2[j:])

    return merged

# Example:
list1 = [1, 3, 5, 7]
list2 = [2, 4, 6, 8]
result = merge_sorted_lists(list1, list2)
print(result)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

18.Implement a code to find the intersection of two given lists.

```
def list_intersection(lst1, lst2):
    return list(set(lst1) & set(lst2))

# Example:
list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]
result = list_intersection(list1, list2)
print(result)
```

```
[3, 4, 5]
```

19.Create a code to find the union of two lists without duplicates.

```
def list_union(lst1, lst2):
    return list(set(lst1) | set(lst2))

# Example:
list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]
result = list_union(list1, list2)
print(result)
```

```
[1, 2, 3, 4, 5, 6, 7]
```

20.Write a code to shuffle a given list randomly without using any built-in shuffle functions.

```
import random

def custom_shuffle(lst):
    n = len(lst)
    for i in range(n - 1, 0, -1):
        j = random.randint(0, i)
        lst[i], lst[j] = lst[j], lst[i]

# Example:
my_list = [1, 2, 3, 4, 5]
custom_shuffle(my_list)
print(my_list)
```

```
[3, 2, 1, 5, 4]
```

21.Write a code that takes two tuples as input and returns a new tuple containing elements that are
common to both input tuples.

```
def common_elements(tuple1, tuple2):
    return tuple(set(tuple1) & set(tuple2))
```

```
# Example:
tuple1 = (1, 2, 3, 4, 5)
tuple2 = (3, 4, 5, 6, 7)
result = common_elements(tuple1, tuple2)
print(result)
```

```
(3, 4, 5)
```

22.Create a code that prompts the user to enter two sets of integers separated by commas. Then, print the intersection of these two sets.

```
def count_occurrences(tpl: tuple, element) -> int:
    """
    Returns the count of occurrences of an element in a tuple.

    :param tpl: The input tuple
    :param element: The element to count occurrences of
    :return: The number of times the element appears in the tuple
    """
    return tpl.count(element)

# Example usage
tuple_data = (1, 2, 3, 4, 1, 2, 1, 5)
element_to_count = 1
print(count_occurrences(tuple_data, element_to_count))  # Output: 3

# Code to find symmetric difference of two sets of strings
def symmetric_difference():
    set1 = set(input("Enter first set of strings (comma-separated):
").split(','))
    set2 = set(input("Enter second set of strings (comma-separated):
").split(','))

    sym_diff = set1.symmetric_difference(set2)
    print("Symmetric Difference:", sym_diff)

# Example
symmetric_difference()

# Code to count word frequencies
def word_frequencies(word_list):
    """
    Returns a dictionary with unique words as keys and their
frequencies as values.

    :param word_list: List of words
    :return: Dictionary with word frequencies
    """
    frequency_dict = {}
```

```
    for word in word_list:
        frequency_dict[word] = frequency_dict.get(word, 0) + 1
    return frequency_dict

# Example
words = ["apple", "banana", "apple", "orange", "banana", "banana"]
print(word_frequencies(words))

# Code to merge two dictionaries
def merge_dictionaries(dict1, dict2):
    """
    Merges two dictionaries, adding values for common keys.

    :param dict1: First dictionary
    :param dict2: Second dictionary
    :return: Merged dictionary with summed values for common keys
    """
    merged_dict = dict1.copy()
    for key, value in dict2.items():
        merged_dict[key] = merged_dict.get(key, 0) + value
    return merged_dict

# Example
dict_a = {"a": 1, "b": 2, "c": 3}
dict_b = {"b": 3, "c": 4, "d": 5}
print(merge_dictionaries(dict_a, dict_b))

# Code to access a value in a nested dictionary
def get_nested_value(nested_dict, keys):
    """
    Retrieves a value from a nested dictionary using a list of keys.

    :param nested_dict: The nested dictionary
    :param keys: List of keys to access the desired value
    :return: The corresponding value or None if any key is missing
    """
    current = nested_dict
    for key in keys:
        if isinstance(current, dict) and key in current:
            current = current[key]
        else:
            return None
    return current

# Example
nested_dict_example = {"a": {"b": {"c": 42}}}
keys_to_access = ["a", "b", "c"]
print(get_nested_value(nested_dict_example, keys_to_access))  #
Output: 42
```

```python
# Code to sort a dictionary by values
def sort_dictionary_by_value(input_dict, ascending=True):
    """
    Returns a sorted dictionary based on values.

    :param input_dict: Dictionary to sort
    :param ascending: Boolean flag to determine sort order (True for
ascending, False for descending)
    :return: Sorted dictionary
    """
    return dict(sorted(input_dict.items(), key=lambda item: item[1],
reverse=not ascending))

# Example
unsorted_dict = {"a": 3, "b": 1, "c": 2}
print(sort_dictionary_by_value(unsorted_dict, ascending=True))

# Code to invert a dictionary
def invert_dictionary(input_dict):
    """
    Inverts a dictionary, swapping keys and values.
    If multiple keys have the same value, store them as a list.

    :param input_dict: Dictionary to invert
    :return: Inverted dictionary with lists for duplicate values
    """
    inverted_dict = {}
    for key, value in input_dict.items():
        if value in inverted_dict:
            inverted_dict[value].append(key)
        else:
            inverted_dict[value] = [key]
    return inverted_dict

# Example
original_dict = {"a": 1, "b": 2, "c": 1, "d": 3}
print(invert_dictionary(original_dict))

# Code to find the intersection of two sets of integers
def intersection_of_sets():
    """
    Prompts the user to enter two sets of integers and prints their
intersection.
    """
    set1 = set(map(int, input("Enter first set of integers (comma-
separated): ").split(',')))
    set2 = set(map(int, input("Enter second set of integers (comma-
separated): ").split(',')))

    intersection = set1.intersection(set2)
```

```
    print("Intersection:", intersection)

# Example
intersection_of_sets()
```

23. Write a code to concatenate two tuples. The function should take two tuples as input and return a new tuple containing elements from both input tuples.

```
def concatenate_tuples(tuple1, tuple2):
    """Concatenates two tuples and returns the result."""
    return tuple1 + tuple2

# Example:
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
result = concatenate_tuples(tuple1, tuple2)
print("Concatenated tuple:", result)
```

```
Concatenated tuple: (1, 2, 3, 4, 5, 6)
```

24. Develop a code that prompts the user to input two sets of strings. Then, print the elements that are present in the first set but not in the second set.

```
def get_set_from_input(prompt):
    return set(input(prompt).split(','))

# Get input from the user
set1 = get_set_from_input("Enter the first set of strings, separated by commas: ")
set2 = get_set_from_input("Enter the second set of strings, separated by commas: ")

# Compute the difference
difference = set1 - set2

# Print the result
print("Elements present in the first set but not in the second set:", difference)
```

25. Create a code that takes a tuple and two integers as input. The function should return a new tuple containing elements from the original tuple within the specified range of indices.

```
def slice_tuple(input_tuple, start, end):
    """Returns a new tuple containing elements within the specified index range."""
    return input_tuple[start:end]
```

```
# Example:
tuple_data = (10, 20, 30, 40, 50, 60)
start_index = int(input("Enter the start index: "))
end_index = int(input("Enter the end index: "))

result = slice_tuple(tuple_data, start_index, end_index)
print("Sliced tuple:", result)
```

26. Write a code that prompts the user to input two sets of characters. Then, print the union of these two sets.

```
def get_set_from_input(prompt):
    return set(input(prompt).replace(" ", "").split(','))

# Get input from the user
set1 = get_set_from_input("Enter the first set of characters,
separated by commas: ")
set2 = get_set_from_input("Enter the second set of characters,
separated by commas: ")

# Compute the union
union_set = set1 | set2

# Print the result
print("Union of the two sets:", union_set)
```

27.Develop a code that takes a tuple of integers as input. The function should return the maximum and minimum values from the tuple using tuple unpacking.

```
def min_max_tuple(input_tuple):
    """Returns the minimum and maximum values from the tuple using
tuple unpacking."""
    min_value, max_value = min(input_tuple), max(input_tuple)
    return min_value, max_value

# Example:
tuple_data = tuple(map(int, input("Enter integers separated by commas:
").split(',')))
min_val, max_val = min_max_tuple(tuple_data)
print("Minimum value:", min_val)
print("Maximum value:", max_val)
```

28.Create a code that defines two sets of integers. Then, print the union, intersection, and difference of these two sets.

```
# Define two sets of integers
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
```

```
# Compute union, intersection, and difference
union_set = set1 | set2
intersection_set = set1 & set2
difference_set = set1 - set2

# Print the results
print("Union:", union_set)
print("Intersection:", intersection_set)
print("Difference (set1 - set2):", difference_set)
```

29.Write a code that takes a tuple and an element as input. The function should return the count of occurrences of the given element in the tuple.

```
def count_occurrences(tpl: tuple, element) -> int:
    """
    Returns the count of occurrences of an element in a tuple.

    :param tpl: The input tuple
    :param element: The element to count occurrences of
    :return: The number of times the element appears in the tuple
    """
    return tpl.count(element)

# Example:
tuple_data = (1, 2, 3, 4, 1, 2, 1, 5)
element_to_count = 1
print(count_occurrences(tuple_data, element_to_count))
```

```
3
```

30.Develop a code that prompts the user to input two sets of strings. Then, print the symmetric difference of these two sets.

```
def count_occurrences(tpl: tuple, element) -> int:
    """
    Returns the count of occurrences of an element in a tuple.

    :param tpl: The input tuple
    :param element: The element to count occurrences of
    :return: The number of times the element appears in the tuple
    """
    return tpl.count(element)

# Example
tuple_data = (1, 2, 3, 4, 1, 2, 1, 5)
element_to_count = 1
print(count_occurrences(tuple_data, element_to_count))

# Code to find symmetric difference of two sets of strings
```

```
def symmetric_difference():
    set1 = set(input("Enter first set of strings (comma-separated):
").split(','))
    set2 = set(input("Enter second set of strings (comma-separated):
").split(','))

    sym_diff = set1.symmetric_difference(set2)
    print("Symmetric Difference:", sym_diff)

# Example
symmetric_difference()
```

3

31.Write a code that takes a list of words as input and returns a dictionary where the keys are unique words
and the values are the frequencies of those words in the input list.

```
def count_occurrences(tpl: tuple, element) -> int:
    """
    Returns the count of occurrences of an element in a tuple.

    :param tpl: The input tuple
    :param element: The element to count occurrences of
    :return: The number of times the element appears in the tuple
    """
    return tpl.count(element)

# Example usage
tuple_data = (1, 2, 3, 4, 1, 2, 1, 5)
element_to_count = 1
print(count_occurrences(tuple_data, element_to_count))  # Output: 3

# Code to find symmetric difference of two sets of strings
def symmetric_difference():
    set1 = set(input("Enter first set of strings (comma-separated):
").split(','))
    set2 = set(input("Enter second set of strings (comma-separated):
").split(','))

    sym_diff = set1.symmetric_difference(set2)
    print("Symmetric Difference:", sym_diff)

# Example
symmetric_difference()

# Code to count word frequencies
def word_frequencies(word_list):
    """
    Returns a dictionary with unique words as keys and their
frequencies as values.
```

```
    :param word_list: List of words
    :return: Dictionary with word frequencies
    """
    frequency_dict = {}
    for word in word_list:
        frequency_dict[word] = frequency_dict.get(word, 0) + 1
    return frequency_dict

# Example
words = ["apple", "banana", "apple", "orange", "banana", "banana"]
print(word_frequencies(words))
```

32.Write a code that takes two dictionaries as input and merges them into a single dictionary. If there are common keys, the values should be added together.

```
def count_occurrences(tpl: tuple, element) -> int:
    """
    Returns the count of occurrences of an element in a tuple.

    :param tpl: The input tuple
    :param element: The element to count occurrences of
    :return: The number of times the element appears in the tuple
    """
    return tpl.count(element)

# Example usage
tuple_data = (1, 2, 3, 4, 1, 2, 1, 5)
element_to_count = 1
print(count_occurrences(tuple_data, element_to_count))  # Output: 3

# Code to find symmetric difference of two sets of strings
def symmetric_difference():
    set1 = set(input("Enter first set of strings (comma-separated):
").split(','))
    set2 = set(input("Enter second set of strings (comma-separated):
").split(','))

    sym_diff = set1.symmetric_difference(set2)
    print("Symmetric Difference:", sym_diff)

# Example
symmetric_difference()

# Code to count word frequencies
def word_frequencies(word_list):
    """
    Returns a dictionary with unique words as keys and their
frequencies as values.
```

```
    :param word_list: List of words
    :return: Dictionary with word frequencies
    """
    frequency_dict = {}
    for word in word_list:
        frequency_dict[word] = frequency_dict.get(word, 0) + 1
    return frequency_dict

# Example
words = ["apple", "banana", "apple", "orange", "banana", "banana"]
print(word_frequencies(words))

# Code to merge two dictionaries
def merge_dictionaries(dict1, dict2):
    """
    Merges two dictionaries, adding values for common keys.

    :param dict1: First dictionary
    :param dict2: Second dictionary
    :return: Merged dictionary with summed values for common keys
    """
    merged_dict = dict1.copy()
    for key, value in dict2.items():
        merged_dict[key] = merged_dict.get(key, 0) + value
    return merged_dict

# Example
dict_a = {"a": 1, "b": 2, "c": 3}
dict_b = {"b": 3, "c": 4, "d": 5}
print(merge_dictionaries(dict_a, dict_b))
```

33.Write a code to access a value in a nested dictionary. The function should take the dictionary and a list of keys as input, and return the corresponding value. If any of the keys do not exist in the dictionary, the function should return None.

```
def count_occurrences(tpl: tuple, element) -> int:
    """
    Returns the count of occurrences of an element in a tuple.

    :param tpl: The input tuple
    :param element: The element to count occurrences of
    :return: The number of times the element appears in the tuple
    """
    return tpl.count(element)

# Example
tuple_data = (1, 2, 3, 4, 1, 2, 1, 5)
element_to_count = 1
```

```
print(count_occurrences(tuple_data, element_to_count))

# Code to find symmetric difference of two sets of strings
def symmetric_difference():
    set1 = set(input("Enter first set of strings (comma-separated):
").split(','))
    set2 = set(input("Enter second set of strings (comma-separated):
").split(','))

    sym_diff = set1.symmetric_difference(set2)
    print("Symmetric Difference:", sym_diff)

# Example
symmetric_difference()

# Code to count word frequencies
def word_frequencies(word_list):
    """
    Returns a dictionary with unique words as keys and their
frequencies as values.

    :param word_list: List of words
    :return: Dictionary with word frequencies
    """
    frequency_dict = {}
    for word in word_list:
        frequency_dict[word] = frequency_dict.get(word, 0) + 1
    return frequency
    #Example
words = ["apple", "banana", "apple", "orange", "banana", "banana"]
print(word_frequencies(words))

# Code to merge two dictionaries
def merge_dictionaries(dict1, dict2):
    """
    Merges two dictionaries, adding values for common keys.

    :param dict1: First dictionary
    :param dict2: Second dictionary
    :return: Merged dictionary with summed values for common keys
    """
    merged_dict = dict1.copy()
    for key, value in dict2.items():
        merged_dict[key] = merged_dict.get(key, 0) + value
    return merged_dict

# Example
dict_a = {"a": 1, "b": 2, "c": 3}
dict_b = {"b": 3, "c": 4, "d": 5}
print(merge_dictionaries(dict_a, dict_b))
```

```python
# Code to access a value in a nested dictionary
def get_nested_value(nested_dict, keys):
    """
    Retrieves a value from a nested dictionary using a list of keys.

    :param nested_dict: The nested dictionary
    :param keys: List of keys to access the desired value
    :return: The corresponding value or None if any key is missing
    """
    current = nested_dict
    for key in keys:
        if isinstance(current, dict) and key in current:
            current = current[key]
        else:
            return None
    return current

# Example
nested_dict_example = {"a": {"b": {"c": 42}}}
keys_to_access = ["a", "b", "c"]
print(get_nested_value(nested_dict_example, keys_to_access))
```

34. Write a code that takes a dictionary as input and returns a sorted version of it based on the values. You can choose whether to sort in ascending or descending order.

```python
def count_occurrences(tpl: tuple, element) -> int:
    """
    Returns the count of occurrences of an element in a tuple.

    :param tpl: The input tuple
    :param element: The element to count occurrences of
    :return: The number of times the element appears in the tuple
    """
    return tpl.count(element)

# Example
tuple_data = (1, 2, 3, 4, 1, 2, 1, 5)
element_to_count = 1
print(count_occurrences(tuple_data, element_to_count))  # Output: 3

# Code to find symmetric difference of two sets of strings
def symmetric_difference():
    set1 = set(input("Enter first set of strings (comma-separated): ").split(','))
    set2 = set(input("Enter second set of strings (comma-separated): ").split(','))

    sym_diff = set1.symmetric_difference(set2)
```

```
    print("Symmetric Difference:", sym_diff)

# Example
symmetric_difference()

# Code to count word frequencies
def word_frequencies(word_list):
    """
    Returns a dictionary with unique words as keys and their
frequencies as values.

    :param word_list: List of words
    :return: Dictionary with word frequencies
    """
    frequency_dict = {}
    for word in word_list:
        frequency_dict[word] = frequency_dict.get(word, 0) + 1
    return frequency_dict

# Example
words = ["apple", "banana", "apple", "orange", "banana", "banana"]
print(word_frequencies(words))

# Code to merge two dictionaries
def merge_dictionaries(dict1, dict2):
    """
    Merges two dictionaries, adding values for common keys.

    :param dict1: First dictionary
    :param dict2: Second dictionary
    :return: Merged dictionary with summed values for common keys
    """
    merged_dict = dict1.copy()
    for key, value in dict2.items():
        merged_dict[key] = merged_dict.get(key, 0) + value
    return merged_dict

# Example
dict_a = {"a": 1, "b": 2, "c": 3}
dict_b = {"b": 3, "c": 4, "d": 5}
print(merge_dictionaries(dict_a, dict_b))

# Code to access a value in a nested dictionary
def get_nested_value(nested_dict, keys):
    """
    Retrieves a value from a nested dictionary using a list of keys.

    :param nested_dict: The nested dictionary
    :param keys: List of keys to access the desired value
    :return: The corresponding value or None if any key is missing
```

```
    """
    current = nested_dict
    for key in keys:
        if isinstance(current, dict) and key in current:
            current = current[key]
        else:
            return None
    return current

# Example
nested_dict_example = {"a": {"b": {"c": 42}}}
keys_to_access = ["a", "b", "c"]
print(get_nested_value(nested_dict_example, keys_to_access))

# Code to sort a dictionary by values
def sort_dictionary_by_value(input_dict, ascending=True):
    """
    Returns a sorted dictionary based on values.

    :param input_dict: Dictionary to sort
    :param ascending: Boolean flag to determine sort order (True for
ascending, False for descending)
    :return: Sorted dictionary
    """
    return dict(sorted(input_dict.items(), key=lambda item: item[1],
reverse=not ascending))

# Example
unsorted_dict = {"a": 3, "b": 1, "c": 2}
print(sort_dictionary_by_value(unsorted_dict, ascending=True))
```

35.Write a code that inverts a dictionary, swapping keys and values. Ensure that the inverted dictionary
correctly handles cases where multiple keys have the same value by storing the keys as a list in the
inverted dictionary.

```
def count_occurrences(tpl: tuple, element) -> int:
    """
    Returns the count of occurrences of an element in a tuple.

    :param tpl: The input tuple
    :param element: The element to count occurrences of
    :return: The number of times the element appears in the tuple
    """
    return tpl.count(element)

# Example usage
tuple_data = (1, 2, 3, 4, 1, 2, 1, 5)
element_to_count = 1
```

```
print(count_occurrences(tuple_data, element_to_count))  # Output: 3

# Code to find symmetric difference of two sets of strings
def symmetric_difference():
    set1 = set(input("Enter first set of strings (comma-separated):
").split(','))
    set2 = set(input("Enter second set of strings (comma-separated):
").split(','))

    sym_diff = set1.symmetric_difference(set2)
    print("Symmetric Difference:", sym_diff)

# Example usage
symmetric_difference()

# Code to count word frequencies
def word_frequencies(word_list):
    """
    Returns a dictionary with unique words as keys and their
frequencies as values.

    :param word_list: List of words
    :return: Dictionary with word frequencies
    """
    frequency_dict = {}
    for word in word_list:
        frequency_dict[word] = frequency_dict.get(word, 0) + 1
    return frequency_dict

# Example usage
words = ["apple", "banana", "apple", "orange", "banana", "banana"]
print(word_frequencies(words))  # Output: {'apple': 2, 'banana': 3,
'orange': 1}

# Code to merge two dictionaries
def merge_dictionaries(dict1, dict2):
    """
    Merges two dictionaries, adding values for common keys.

    :param dict1: First dictionary
    :param dict2: Second dictionary
    :return: Merged dictionary with summed values for common keys
    """
    merged_dict = dict1.copy()
    for key, value in dict2.items():
        merged_dict[key] = merged_dict.get(key, 0) + value
    return merged_dict

# Example usage
dict_a = {"a": 1, "b": 2, "c": 3}
```

```
dict_b = {"b": 3, "c": 4, "d": 5}
print(merge_dictionaries(dict_a, dict_b))  # Output: {'a': 1, 'b': 5,
'c': 7, 'd': 5}

# Code to access a value in a nested dictionary
def get_nested_value(nested_dict, keys):
    """
    Retrieves a value from a nested dictionary using a list of keys.

    :param nested_dict: The nested dictionary
    :param keys: List of keys to access the desired value
    :return: The corresponding value or None if any key is missing
    """
    current = nested_dict
    for key in keys:
        if isinstance(current, dict) and key in current:
            current = current[key]
        else:
            return None
    return current

# Example usage
nested_dict_example = {"a": {"b": {"c": 42}}}
keys_to_access = ["a", "b", "c"]
print(get_nested_value(nested_dict_example, keys_to_access))

# Code to sort a dictionary by values
def sort_dictionary_by_value(input_dict, ascending=True):
    """
    Returns a sorted dictionary based on values.

    :param input_dict: Dictionary to sort
    :param ascending: Boolean flag to determine sort order (True for
ascending, False for descending)
    :return: Sorted dictionary
    """
    return dict(sorted(input_dict.items(), key=lambda item: item[1],
reverse=not ascending))

# Example
unsorted_dict = {"a": 3, "b": 1, "c": 2}
print(sort_dictionary_by_value(unsorted_dict, ascending=True))

# Code to invert a dictionary
def invert_dictionary(input_dict):
    """
    Inverts a dictionary, swapping keys and values.
    If multiple keys have the same value, store them as a list.

    :param input_dict: Dictionary to invert
```

```
    :return: Inverted dictionary with lists for duplicate values
    """
    inverted_dict = {}
    for key, value in input_dict.items():
        if value in inverted_dict:
            inverted_dict[value].append(key)
        else:
            inverted_dict[value] = [key]
    return inverted_dict

# Example
original_dict = {"a": 1, "b": 2, "c": 1, "d": 3}
print(invert_dictionary(original_dict))
```