1. Write a code to reverse a string.

```
string = "hello world"
reversed_string = string[::-1]
print(reversed_string)  # Output: "dlrow olleh"
```

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

```
3.
4. def count_vowels(string):
5.   """Counts the number of vowels in a string.
6.
7.   Args:
8.     string: The input string.
9.
10.  Returns:
11.    The number of vowels in the string.
12.  """
13.
14.  vowels = "aeiouAEIOU"
15.  count = 0
16.  for char in string:
17.    if char in vowels:
18.      count += 1
19.
20.  return count
21.
22.# Example usage:
23.string = "hello world"
24.vowel_count = count_vowels(string)
25.print("Number of vowels:", vowel_count)  # Output: Number of vowels: 3
26.
```

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

```
def is_palindrome(string):
  """Checks if a given string is a palindrome.

  Args:
    string: The input string.
```

```
  Returns:
    True if the string is a palindrome, False otherwise.
  """

  # Remove non-alphanumeric characters and convert to lowercase
  cleaned_string = ''.join(char for char in string if char.isalnum()).lower()

  # Compare the cleaned string with its reversed version
  return cleaned_string == cleaned_string[::-1]

# Example usage:
string = "racecar"
is_palindrome_result = is_palindrome(string)
print("Is the string a palindrome?", is_palindrome_result)  # Output: Is the
string a palindrome? True
```

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

```
def are_anagrams(string1, string2):
    """Checks if two strings are anagrams of each other.

    Args:
        string1: The first input string.
        string2: The second input string.

    Returns:
        True if the strings are anagrams, False otherwise.
    """

    # Check if the strings have the same length
    if len(string1) != len(string2):
        return False

    # Convert both strings to lowercase and sort them
    string1 = sorted(string1.lower())
    string2 = sorted(string2.lower())

    # Compare the sorted strings
```

```
    return string1 == string2

# Example usage:
string1 = "listen"
string2 = "silent"
are_anagrams_result = are_anagrams(string1, string2)
print("Are the strings anagrams?", are_anagrams_result)  # Output: Are the
strings anagrams? True
```

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

```
def find_all_occurrences(string, substring):
  """Finds all occurrences of a substring within a string.

  Args:
    string: The input string.
    substring: The substring to search for.

  Returns:
    A list of indices where the substring occurs in the input string.
  """

  indices = []
  start_index = 0

  while True:
    index = string.find(substring, start_index)
    if index == -1:
      break
    indices.append(index)
    start_index = index + 1

  return indices

# Example usage:
string = "hello world, hello again"
substring = "hello"
occurrences = find_all_occurrences(string, substring)
print("Occurrences of 'hello':", occurrences)  # Output: Occurrences of 'hello':
[0, 13]
```

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

```
def compress_string(string):
  """Compresses a string by counting consecutive repeated characters.

  Args:
    string: The input string.

  Returns:
    The compressed string.
  """

  compressed_string = ""
  count = 1

  for i in range(1, len(string)):
    if string[i] == string[i - 1]:
      count += 1
    else:
      compressed_string += string[i - 1] + str(count)
      count = 1

  compressed_string += string[-1] + str(count)

  return compressed_string

# Example usage:
string = "aabccccccaaa"
compressed_string = compress_string(string)
print("Compressed string:", compressed_string)  # Output: Compressed string:
a2b1c5a3
```

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

```
```

```python
def has_unique_characters(string):
  """Checks if a string has all unique characters.

  Args:
    string: The input string.

  Returns:
    True if the string has all unique characters, False otherwise.
  """

  # Create a set to store unique characters
  unique_chars = set()

  # Iterate through each character in the string
  for char in string:
    # If the character is already in the set, it's not unique
    if char in unique_chars:
      return False
    # Add the character to the set
    unique_chars.add(char)

  # If no duplicates were found, the string has all unique characters
  return True

# Example usage:
string = "hello"
has_unique = has_unique_characters(string)
print("Does the string have all unique characters?", has_unique)  # Output: Does
the string have all unique characters? False
```

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

```python
def convert_case(string, case):
  """Converts a given string to uppercase or lowercase.

  Args:
    string: The input string.
    case: A string indicating the desired case ('upper' or 'lower').

  Returns:
```

```
    The converted string.
  """

  if case == 'upper':
    return string.upper()
  elif case == 'lower':
    return string.lower()
  else:
    return "Invalid case. Please specify 'upper' or 'lower'."

# Example usage:
string = "Hello, World!"
uppercase_string = convert_case(string, "upper")
lowercase_string = convert_case(string, "lower")

print("Uppercase string:", uppercase_string)
print("Lowercase string:", lowercase_string)
```

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

```
def count_words(string):
  """Counts the number of words in a string.

  Args:
    string: The input string.

  Returns:
    The number of words in the string.
  """

  words = string.split()
  return len(words)

# Example usage:
string = "This is a sample sentence."
word_count = count_words(string)
print("Number of words:", word_count)  # Output: Number of words: 4
```

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

```
def concatenate_strings(string1, string2):
  """Concatenates two strings without using the + operator.

  Args:
    string1: The first input string.
    string2: The second input string.

  Returns:
    The concatenated string.
  """

  # Create a list containing both strings
  strings_list = [string1, string2]

  # Join the list elements into a single string
```

```
  concatenated_string = ''.join(strings_list)

  return concatenated_string

# Example usage:
string1 = "hello"
string2 = " world"
concatenated_string = concatenate_strings(string1, string2)
print("Concatenated string:", concatenated_string)  # Output: Concatenated
string: hello world
```

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

```
def remove_all_occurrences(my_list, element):
  """Removes all occurrences of a specific element from a list.

  Args:
    my_list: The input list.
    element: The element to remove.

  Returns:
    A new list with all occurrences of the element removed.
  """

  new_list = []
  for item in my_list:
    if item != element:
      new_list.append(item)
  return new_list

# Example usage:
my_list = [1, 2, 3, 2, 4, 1]
element_to_remove = 1
filtered_list = remove_all_occurrences(my_list, element_to_remove)
print(filtered_list)  # Output: [2, 3, 2, 4]
```

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

```
def find_second_largest(numbers):
    # Remove duplicates by converting the list to a set, then back to a sorted
list
    unique_numbers = list(set(numbers))

    # Sort the list in ascending order
    unique_numbers.sort()

    # The second largest number will be at the second-to-last index
    if len(unique_numbers) < 2:
        return None  # If the list doesn't have at least two unique numbers
    else:
        return unique_numbers[-2]

# Example usage
numbers = [10, 20, 4, 45, 99, 99, 20]
second_largest = find_second_largest(numbers)
print("The second largest number is:", second_largest)
```

## 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.

```
from collections import Counter

def count_occurrences(my_list):
  """Counts occurrences of elements in a list.

  Args:
    my_list: The input list.

  Returns:
    A dictionary of element counts.
  """
```

```
  return Counter(my_list)

# Example usage:
my_list = [1, 2, 3, 2, 4, 1]
occurrence_dict = count_occurrences(my_list)
print(occurrence_dict)  # Output: Counter({1: 2, 2: 2, 3: 1, 4: 1})
```

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

```
def reverse_in_place(lst):
  """Reverses a list in-place without using built-in reverse functions.

  Args:
    lst: The input list.

  Modifies:
    The input list is modified in-place.
  """

  left = 0
  right = len(lst) - 1

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

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

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

```python
def remove_duplicates_preserve_order(lst):
  """Removes duplicates from a list while preserving the original order.

  Args:
    lst: The input list.

  Returns:
    A new list with duplicates removed.
  """

  seen = set()
  result = []
  for element in lst:
    if element not in seen:
      seen.add(element)
      result.append(element)
  return result

# Example usage:
my_list = [1, 2, 3, 2, 4, 1]
filtered_list = remove_duplicates_preserve_order(my_list)
print(filtered_list)  # Output: [1, 2, 3, 4]
remove_duplicates_preserve_order(list)
```

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

```python
def check_sorted(lst):
    """Checks if a list is sorted (either in ascending or descending order).

    Args:
        lst: The input list.

    Returns:
        "List is sorted in ascending order" if the list is sorted in ascending
order.
        "List is sorted in descending order" if the list is sorted in descending
order.
        "List is not sorted" if the list is not sorted.
    """
```

```
    if lst == sorted(lst):
        return "List is sorted in ascending order"
    elif lst == sorted(lst, reverse=True):
        return "List is sorted in descending order"
    else:
        return "List is not sorted"

# Example usage:
my_list = [1, 2, 3, 4, 5]
result = check_sorted(my_list)
print(result)  # Output: List is sorted in ascending order
```

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

```
def merge_sorted_lists(list1, list2):
    return sorted(list1 + list2)

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

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

```
def find_intersection(list1, list2):
    # Convert both lists to sets and find their intersection
    return list(set(list1) & set(list2))

# Example usage
list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
result = find_intersection(list1, list2)
print(result)
```

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

```
def unio(lis1,lis2):
  result = []
  sett = (set(lis1) | set(lis2))
  for i in sett:
    if i not in result:
      result.append(i)
  return result

# Example usage:
list1 = [1, 2, 3]
list2 = [2, 3, 4]
unio(list1,list2)

#output [1,2,3,4]
```

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

```
import random

def shuffle_list(lst):
    for i in range(len(lst)):
        # Pick a random index to swap with
        j = random.randint(0, len(lst) - 1)
        # Swap the elements at indices i and j
        lst[i], lst[j] = lst[j], lst[i]

# Example usage
numbers = [1, 2, 3, 4, 5]
shuffle_list(numbers)
```

```
print(numbers)
```

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

## 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 find_intersection(set1, set2):
  """Finds the intersection of two sets of integers.

  Args:
    set1: A string representing a set of integers separated by commas.
    set2: A string representing a set of integers separated by commas.

  Returns:
    A list containing the intersection of the two sets.
  """

  # Convert the strings to sets of integers
  set1 = set(map(int, set1.split(',')))
  set2 = set(map(int, set2.split(',')))

  # Find the intersection of the sets
```

```
  intersection = list(set1 & set2)

  return intersection

# Prompt the user to enter two sets of integers
set1_str = input("Enter the first set of integers (separated by commas): ")
set2_str = input("Enter the second set of integers (separated by commas): ")

# Find the intersection and print the result
intersection = find_intersection(set1_str, set2_str)
print("Intersection:", intersection)
```

## 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.

  Args:
    tuple1: The first input tuple.
    tuple2: The second input tuple.

  Returns:
    A new tuple containing elements from both input tuples.
  """

  # Combine the tuples using the + operator
  concatenated_tuple = tuple1 + tuple2

  return concatenated_tuple

# Example usage:
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
concatenated_tuple = concatenate_tuples(tuple1, tuple2)
print(concatenated_tuple)  # Output: (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.

```
  # Use the set difference operator to find the elements in set1 that are not in
set2
  elements_in_first_set_only = set1 - set2

  return list(elements_in_first_set_only)

# Prompt the user to enter two sets of strings
set1_str = input("Enter the first set of strings (separated by commas): ")
set2_str = input("Enter the second set of strings (separated by commas): ")

# Convert the strings to sets
set1 = set(set1_str.split(','))
set2 = set(set2_str.split(','))

# Find the elements in the first set but not in the second set
elements_in_first_set_only = find_elements_in_first_set_but_not_in_second(set1,
set2)

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

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 tuple_slice(original_tuple, start_idx, end_idx):
    # Return a new tuple that contains elements from start_idx to end_idx
    return original_tuple[start_idx:end_idx]

# Get user input for the tuple and indices
```

```
user_tuple = tuple(input("Enter the elements of the tuple, separated by spaces:
").split())
start_idx = int(input("Enter the starting index: "))
end_idx = int(input("Enter the ending index: "))

# Call the function and print the result
sliced_tuple = tuple_slice(user_tuple, start_idx, end_idx)
print("The sliced tuple is:", sliced_tuple)
```

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

```
def uni():
# Get user input for two sets of characters
  set1 = set(input("Enter the characters for the first set, separated by spaces:
").split())
  set2 = set(input("Enter the characters for the second set, separated by spaces:
").split())

# Find the union of the two sets
  union_set = set1.union(set2)

# Print the result
  print("The union of the two sets is:")
  print(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 find_min_max(numbers):
    # Use tuple unpacking to get the minimum and maximum values
    min_value, max_value = min(numbers), max(numbers)
```

```
    return min_value, max_value

# Get user input for the tuple of integers
user_tuple = tuple(map(int, input("Enter the integers for the tuple, separated by
spaces: ").split()))

# Call the function and unpack the returned values
min_value, max_value = find_min_max(user_tuple)

# Print the minimum and maximum values
print(f"Minimum value: {min_value}")
print(f"Maximum value: {max_value}")
```

## 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}

# Find the union of the two sets
union_set = set1.union(set2)

# Find the intersection of the two sets
intersection_set = set1.intersection(set2)

# Find the difference of the two sets (elements in set1 but not in set2)
difference_set = set1.difference(set2)

# Print the results
print("Union of the two sets:", union_set)
print("Intersection of the two sets:", intersection_set)
print("Difference of the two sets (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 occurance(tuple , element):
  """Counts the occurrences of an element in a tuple.

  Args:
    tuple: The input tuple.
    element: The element to count.

  Returns:
    The number of occurrences of the element in the tuple.
  """

  return tuple.count(element)
t1 = (1,2,2,3,3,44,4,5,67,3,2,2,1,1,1,1,1)
t2 = 1
occurance(t1,t2)
```

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

```
def find_symmetric_difference(set1, set2):
  """Finds the symmetric difference of two sets.

  Args:
    set1: The first set.
    set2: The second set.

  Returns:
    A list containing the elements that are present in either set but not in
both.
  """

  # Use the symmetric difference operator to find the elements that are
present in either set but not in both
```

```
  symmetric_difference = set1 ^ set2

  return list(symmetric_difference)

# Prompt the user to enter two sets of strings
set1_str = input("Enter the first set of strings (separated by commas): ")
set2_str = input("Enter the second set of strings (separated by commas): ")

# Convert the strings to sets
set1 = set(set1_str.split(','))
set2 = set(set2_str.split(','))

# Find the symmetric difference and print the result
symmetric_difference = find_symmetric_difference(set1, set2)
print("Symmetric difference:", symmetric_difference)
```

## 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.

```
from collections import Counter

def count_word_frequencies(words):
  """Counts the frequencies of words in a list.

  Args:
    words: The input list of words.

  Returns:
    A dictionary where the keys are unique words and the values are their
frequencies.
  """

  # Use Counter to count the occurrences of each word
  word_counts = Counter(words)
```

```
  return word_counts

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

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

  Args:
    dict1: The first dictionary.
    dict2: The second dictionary.

  Returns:
    A new dictionary containing the merged key-value pairs.
  """

  merged_dict = dict1.copy()
  for key, value in dict2.items():
    if key in merged_dict:
      merged_dict[key] += value
    else:
      merged_dict[key] = value
  return merged_dict

# Example usage:
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
merged_dict = merge_dictionaries(dict1, dict2)
```

```
print(merged_dict)  # Output: {'a': 1, 'b': 5, 'c': 4}
```

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 merge_dictionaries(dict1, dict2):
  """Merges two dictionaries, adding values for common keys.

  Args:
    dict1: The first dictionary.
    dict2: The second dictionary.

  Returns:
    A new dictionary containing the merged key-value pairs.
  """

  merged_dict = dict1.copy()
  for key, value in dict2.items():
    if key in merged_dict:
      merged_dict[key] += value
    else:
      merged_dict[key] = value
  return merged_dict

# Example usage:
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
merged_dict = merge_dictionaries(dict1, dict2)
print(merged_dict)  # Output: {'a': 1, 'b': 5, 'c': 4}
```

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.

```
def sort_dictionary_by_values(dictionary, ascending=True):
  """Sorts a dictionary by its values.

  Args:
    dictionary: The input dictionary.
    ascending: A boolean indicating whether to sort in ascending or
descending order.

  Returns:
    A list of tuples, where each tuple contains a key-value pair from the
sorted dictionary.
  """

  sorted_items = sorted(dictionary.items(), key=lambda item: item[1],
reverse=not ascending)
  return sorted_items

# Example usage:
my_dict = {'a': 3, 'b': 1, 'c': 2}
sorted_dict = sort_dictionary_by_values(my_dict, ascending=True)
print(sorted_dict)  # Output: [('b', 1), ('c', 2), ('a', 3)]
```

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 invert_dictionary(dictionary):
  """Inverts a dictionary, swapping keys and values.

  Args:
    dictionary: The input dictionary.

  Returns:
    A new dictionary with the keys and values swapped.
  """
```

```
  inverted_dict = {}
  for key, value in dictionary.items():
    if value in inverted_dict:
      inverted_dict[value].append(key)
    else:
      inverted_dict[value] = [key]
  return inverted_dict

# Example usage:
my_dict = {'a': 1, 'b': 2, 'c': 1}
inverted_dict = invert_dictionary(my_dict)
print(inverted_dict)  # Output: {1: ['a', 'c'], 2: ['b']}
```