# Write a code to reverse a string.

string = "hello world"

reversed\_string = string[::-1]

print(reversed\_string)  # Output: "dlrow olleh"

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

1. def count\_vowels(string):
2. """Counts the number of vowels in a string.
3. Args:
4. string: The input string.
5. Returns:
6. The number of vowels in the string.
7. """
8. vowels = "aeiouAEIOU"
9. count = 0
10. for char in string:
11. if char in vowels:
12. count += 1
14. return count
15. # Example usage:
16. string = "hello world"
17. vowel\_count = count\_vowels(string)
18. print("Number of vowels:", vowel\_count)  # Output: Number of vowels: 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

# 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]

# 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 = "aabcccccaaa"

compressed\_string = compress\_string(string)

print("Compressed string:", compressed\_string)  # Output: Compressed string: a2b1c5a3

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

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

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

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)

# 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

# . 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

# . 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.

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.

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']}