Q.1 Write a code to reverse a string

```
# Function to reverse a string
def reverse_string(s):
return s[::-1]

# Example usage string =
"Hello, World!"
reversed_string = reverse_string(string)
print("Original:", string)
print("Reversed:", reversed_string)
Original: Hello, World!
Reversed: !dlroW ,olleH
```

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

```
# Function to count vowels in a string
def count_vowels(s):
    vowels = "aeiouAEIOU"
count = 0     for char in
s:         if char in
vowels:
            count += 1
return count

# Example usage string = "Hello,
World!" vowel_count =
count_vowels(string)
print("String:", string)
print("Number of vowels:", vowel_count)
String: Hello, World!
Number of vowels: 3
```

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

```
# Function to check if a string is a palindrome
def is_palindrome(s):
    # Remove spaces and convert to lowercase for uniformity
s = s.replace(" ", "").lower()
    # Check if the string is equal to its reverse
return s == s[::-1]

# Example usage
string = "A man a plan a canal Panama" if
is_palindrome(string):
print(f'"{string}" is a palindrome!')
else:     print(f'"{string}" is not a palindrome.')

"A man a plan a canal Panama" is a palindrome!
```

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

```
# Function to check if two strings are anagrams
def are_anagrams(str1, str2):
    # Remove spaces, convert to lowercase, and sort the characters
return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace("
", "").lower())

# Example usage
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!
```

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

```
# Function to find all occurrences of a substring def
find_substring_occurrences(main_string, substring):
occurrences = []     start = 0     while start <
len(main_string):          # Find the index of the
substring          index = main_string.find(substring,
start)          if index == -1:  # No more occurrences
break
        occurrences.append(index)
        start = index + 1  # Move to the next character
return occurrences

# Example usage
main_string = "abracadabra"
substring = "abra"
occurrences = find_substring_occurrences(main_string, substring)

print(f'Substring "{substring}" found at positions: {occurrences}')
```

```
Substring "abra" found at positions: [0, 7]
```

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

```
# Function to compress a string
def compress_string(s):
    compressed = []
count = 1

    for i in range(1, len(s)):         if s[i] == s[i - 1]:  # Check
if the current character matches the previous one             count +=
1         else:             compressed.append(s[i - 1] + str(count))
# Add character and its count
            count = 1  # Reset the count

    # Add the last character and its count
if s:         compressed.append(s[-1] +
str(count))

    # Join the compressed parts
    compressed_string = "".join(compressed)

    # Return the compressed string only if it's shorter than the original
    return compressed_string if len(compressed_string) < len(s) else s

# Example usage string
= "aaabbbcccaaa"
compressed = compress_string(string)
print("Original:", string)
print("Compressed:", compressed)
Original: aaabbbcccaaa
Compressed: a3b3c3a3
```

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

```
# Function to check if a string has all unique characters def
has_unique_characters(s):     # Use a set to track characters
seen_characters = set()     for char in s:         if char in
seen_characters:  # If character is already in the set, it's not
unique             return False         seen_characters.add(char)
return True

# Example usage string =
"abcdefg" if
has_unique_characters(string):
```

```
    print(f'The string "{string}" has all unique characters.') else:
print(f'The string "{string}" does not have all unique characters.')

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

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

```
# Function to convert a string to uppercase or lowercase
def convert_case(s, to_upper=True):     if to_upper:
return s.upper()  # Convert to uppercase     else:
return s.lower()  # Convert to lowercase

# Example usage string =
"Hello, World!"
uppercase = convert_case(string, to_upper=True)
lowercase = convert_case(string, to_upper=False)

print("Original:", string)
print("Uppercase:", uppercase)
print("Lowercase:", lowercase)

Original: Hello, World!
Uppercase: HELLO, WORLD!
Lowercase: hello, world!
```

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

```
# Function to count the number of words in a string
def count_words(s):
    # Split the string by whitespace and count the resulting words
words = s.split()     return len(words)

# Example usage
string = "Hello, world! Welcome to Python programming."
word_count = count_words(string)

print("String:", string)
print("Number of words:", word_count)
String: Hello, world! Welcome to Python programming.
Number of words: 6
```

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

```
# Function to concatenate two strings
def concatenate_strings(str1, str2):
return "{}{}".format(str1, str2)

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

print("Concatenated String:", result)

Concatenated String: Hello, World!
```

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

```
# Function to remove all occurrences of a specific element from a list
def remove_element(lst, element):     return [item for item in lst if
item != element]

# Example usage
my_list = [1, 2, 3, 4, 2, 5, 2]
element_to_remove = 2
updated_list = remove_element(my_list, element_to_remove)

print("Original List:", my_list)
print("Updated List:", updated_list)
Original List: [1, 2, 3, 4, 2, 5, 2]
Updated List: [1, 3, 4, 5]
```

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

```
# Function to find the second largest number in a list def
second_largest(lst):
    # Remove duplicates by converting the list to a set
unique_lst = list(set(lst))     if len(unique_lst) < 2:
return None  # Return None if there is no second largest number
    unique_lst.sort()  # Sort the list
    return unique_lst[-2]   # The second largest is the second last
element

# Example usage
numbers = [10, 20, 4, 45, 99, 20]
second_largest_number = second_largest(numbers)

print("List of numbers:", numbers) if
second_largest_number is not None:
```

```
    print("Second largest number:", second_largest_number)
else:     print("There is no second largest number.")
List of numbers: [10, 20, 4, 45, 99, 20]
Second largest number: 45
```

Q.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

```
# Function to count occurrences of each element in a list
def count_occurrences(lst):     occurrence_dict = {}
for item in lst:
        if item in occurrence_dict:
            occurrence_dict[item] += 1  # Increment the count if item
already exists         else:             occurrence_dict[item] = 1  #
Initialize the count if item is encountered for the first time
return occurrence_dict

# Example usage
my_list = [1, 2, 2, 3, 3, 3, 4, 5, 1, 4]
occurrences = count_occurrences(my_list)

print("List:", my_list)
print("Occurrences:", occurrences)
List: [1, 2, 2, 3, 3, 3, 4, 5, 1, 4]

Occurrences: {1: 2, 2: 2, 3: 3, 4: 2, 5: 1}
```

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

```
# Function to reverse a list in-place
def reverse_list(lst):
    left, right = 0, len(lst) - 1
while left < right:
        # Swap the elements at left and right indices
lst[left], lst[right] = lst[right], lst[left]
left += 1         right -= 1

# Example usage my_list =
[1, 2, 3, 4, 5]
reverse_list(my_list)

print("Reversed List:", my_list)
Reversed List: [5, 4, 3, 2, 1]
```

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

```
# Function to remove duplicates from a list while preserving the order
def remove_duplicates(lst):     seen = set()  # Set to track seen
elements
    result = []   # List to store the result with no duplicates
        for item in lst:         if item not in seen:
result.append(item)  # Add the item to the result if not seen
            seen.add(item)         # Add the item to the 'seen' set

    return result

# Example usage
my_list = [1, 2, 2, 3, 4, 3, 5, 6, 5]
unique_list = remove_duplicates(my_list)

print("Original List:", my_list)
print("List without duplicates:", unique_list)
```

```
Original List: [1, 2, 2, 3, 4, 3, 5, 6, 5]
List without duplicates: [1, 2, 3, 4, 5, 6]
```

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

```
# Function to check if a list is sorted (ascending or descending)
def is_sorted(lst):     if all(lst[i] <= lst[i + 1] for i in
range(len(lst) - 1)):
        return "The list is sorted in ascending order."     elif
all(lst[i] >= lst[i + 1] for i in range(len(lst) - 1)):
return "The list is sorted in descending order."     else:
return "The list is not sorted."

# Example usage my_list =
[1, 2, 3, 4, 5] result =
is_sorted(my_list)
print(result)

my_list2 = [5, 4, 3, 2, 1]
result2 = is_sorted(my_list2)
print(result2)

my_list3 = [1, 3, 2, 4, 5]
result3 = is_sorted(my_list3)
print(result3)
```

```
The list is sorted in ascending order.
The list is sorted in descending order.
The list is not sorted.
```

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

```
# Function to merge two sorted lists
def merge_sorted_lists(list1, list2):
merged_list = []     i, j = 0, 0

    # Traverse both lists and merge them in sorted order
while i < len(list1) and j < len(list2):         if
list1[i] < list2[j]:
merged_list.append(list1[i])             i += 1
else:             merged_list.append(list2[j])
j += 1

    # If any elements remain in list1
while i < len(list1):
merged_list.append(list1[i])
i += 1

    # If any elements remain in list2
while j < len(list2):
merged_list.append(list2[j])
j += 1

    return merged_list

# Example usage
list1 = [1, 3, 5, 7]
list2 = [2, 4, 6, 8]
merged_list = merge_sorted_lists(list1, list2)

print("Merged Sorted List:", merged_list)

Merged Sorted List: [1, 2, 3, 4, 5, 6, 7, 8]
```

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

```
# Function to find the intersection of two lists
def intersection_of_lists(list1, list2):
    # Convert both lists to sets and find the intersection
return list(set(list1) & set(list2))

# Example usage

list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
intersection = intersection_of_lists(list1, list2)

print("Intersection of lists:", intersection)

Intersection of lists: [4, 5]
```

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

```
# Function to find the union of two lists without duplicates
def union_of_lists(list1, list2):
    # Convert both lists to sets to remove duplicates and find the
union
    return list(set(list1) | set(list2))

# Example usage list1 =
[1, 2, 3, 4, 5] list2 =
[4, 5, 6, 7, 8]
union = union_of_lists(list1, list2)

print("Union of lists:", union)
```

```
Union of lists: [1, 2, 3, 4, 5, 6, 7, 8]
```

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

```
import random

# Function to shuffle a list randomly
def shuffle_list(lst):
    # Iterate through the list and swap each element with a randomly
selected one     for i in range(len(lst)):
        # Randomly select an index greater than or equal to i
j = random.randint(i, len(lst) - 1)          # Swap the
elements at indices i and j          lst[i], lst[j] = lst[j],
lst[i]

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

print("Shuffled List:", my_list)
```

```
Shuffled List: [2, 1, 4, 3, 5]
```

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

```
# Function to find common elements between two tuples
def common_elements(tuple1, tuple2):
    # Find the intersection of both tuples and return it as a new
tuple
    return tuple(set(tuple1) & set(tuple2))

# Example usage tuple1 =
(1, 2, 3, 4, 5) tuple2 =
(4, 5, 6, 7, 8)
common_tuple = common_elements(tuple1, tuple2)

print("Common elements:", common_tuple)
```

```
Common elements: (4, 5)
```

Q.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

```
# Function to find and print the intersection of two sets
def find_intersection():
    # Prompt the user to enter the first set of integers
    set1_input = input("Enter the first set of integers (separated by
commas): ")
    set1 = set(map(int, set1_input.split(',')))  # Convert input string
to a set of integers

    # Prompt the user to enter the second set of integers
    set2_input = input("Enter the second set of integers (separated by
commas): ")
    set2 = set(map(int, set2_input.split(',')))  # Convert input string
to a set of integers

    # Find the intersection of the two sets
intersection = set1 & set2

    # Print the intersection
    print("The intersection of the two sets is:", intersection)

# Call the function to prompt the user and print the result
find_intersection()
```

```
The intersection of the two sets is: {3, 4}
```

Q.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.

```
# Function to concatenate two tuples
def concatenate_tuples(tuple1, tuple2):
return tuple1 + tuple2
```

```
# Example usage
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)
```

Q.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

```
# Function to find and print elements present in the first set but not
in the second def find_difference():
    # Prompt the user to enter the first set of strings
    set1_input = input("Enter the first set of strings (separated by
commas): ")
    set1 = set(set1_input.split(','))  # Convert input string to a set
of strings

    # Prompt the user to enter the second set of strings
    set2_input = input("Enter the second set of strings (separated by
commas): ")
    set2 = set(set2_input.split(','))  # Convert input string to a set
of strings

    # Find elements in set1 but not in set2
difference = set1 - set2

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

# Call the function to prompt the user and print the result
find_difference()
Elements present in the first set but not in the second set:
{'ORANGE', 'APPLE'}
```

Q.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

```
# Function to extract elements from a tuple within the specified
range of indices def extract_range_from_tuple(tpl, start_idx,
end_idx):
    # Return a new tuple with elements within the specified range
return tpl[start_idx:end_idx]

# Example usage
```

```
input_tuple = tuple(input("Enter the tuple elements separated by
commas: ").split(','))
start_index = int(input("Enter the starting index: "))
end_index = int(input("Enter the ending index: "))

# Get the result
result_tuple = extract_range_from_tuple(input_tuple, start_index,
end_index)

print("New Tuple with elements in the specified range:", result_tuple)
```

```
New Tuple with elements in the specified range: ('2', '3', '4', '5')
```

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

```
# Function to find and print the union of two sets of characters
def find_union():
    # Prompt the user to enter the first set of characters
    set1_input = input("Enter the first set of characters (separated by
commas): ")
    set1 = set(set1_input.split(','))  # Convert input string to a set
of characters

    # Prompt the user to enter the second set of characters
    set2_input = input("Enter the second set of characters (separated by
commas): ")
    set2 = set(set2_input.split(','))  # Convert input string to a set
of characters

    # Find the union of the two sets
union = set1 | set2  # Union operation

    # Print the union
    print("The union of the two sets is:", union)

# Call the function to prompt the user and print the result
find_union()
```

```
The union of the two sets is: {'D', 'C', 'A', 'B', 'E', 'F'}
```

Q.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

```
# Function to find the maximum and minimum values from a tuple using
tuple unpacking def
find_max_min(tpl):
    # Unpacking the tuple to get the max and min values
max_val, min_val = max(tpl), min(tpl)     return
max_val, min_val
```

```
# Example usage
input_tuple = tuple(map(int, input("Enter integers separated by
commas: ").split(',')))

# Get the max and min values
max_value, min_value = find_max_min(input_tuple)

print(f"The maximum value is: {max_value}")
print(f"The minimum value is: {min_value}")

The maximum value is: 34
The minimum value is: 2
```

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

```
# Function to print union, intersection, and difference of two sets
def set_operations():
    # Define two sets of integers
set1 = {1, 2, 3, 4, 5}     set2 =
{4, 5, 6, 7, 8}

    # Union of set1 and set2
union = set1 | set2

    # Intersection of set1 and set2
intersection = set1 & set2

    # Difference of set1 and set2
difference = set1 - set2

    # Print the results
    print("Union of the sets:", union)
    print("Intersection of the sets:", intersection)
print("Difference of set1 and set2:", difference)

# Call the function to perform the set operations
set_operations()
Union of the sets: {1, 2, 3, 4, 5, 6, 7, 8}
Intersection of the sets: {4, 5}

Difference of set1 and set2: {1, 2, 3}
```

Q.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

```
# Function to count occurrences of an element in a tuple
def count_occurrences(tpl, element):
    return tpl.count(element)
```

```
# Example usage input_tuple = tuple(input("Enter tuple elements
separated by commas:
").split(','))
element = input("Enter the element to count: ")

# Get the count of occurrences
count = count_occurrences(input_tuple, element)

print(f"The element '{element}' appears {count} times in the tuple.")
```

```
The element '1' appears 3 times in the tuple.
```

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

```
# Function to find and print the symmetric difference of two sets of
strings def
find_symmetric_difference():
    # Prompt the user to enter the first set of strings
    set1_input = input("Enter the first set of strings (separated by
commas): ")
    set1 = set(set1_input.split(','))  # Convert input string to a set
of strings

    # Prompt the user to enter the second set of strings
    set2_input = input("Enter the second set of strings (separated by
commas): ")
    set2 = set(set2_input.split(','))  # Convert input string to a set
of strings

    # Find the symmetric difference of the two sets
    symmetric_difference = set1 ^ set2  # Symmetric difference operation

    # Print the symmetric difference
    print("The symmetric difference of the two sets is:",
symmetric_difference)

# Call the function to prompt the user and print the result
find_symmetric_difference()
```

```
The symmetric difference of the two sets is: {'KIWI', 'APPLE'}
```

Q.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

```
# Function to count the frequency of words in a list and return a
dictionary
```

```
def word_frequencies():
    # Prompt the user to enter a list of words     words_input =
input("Enter a list of words (separated by spaces):
")
    words_list = words_input.split()  # Split the input string into a
list of words

    # Initialize an empty dictionary to store the word frequencies
word_count = {}

    # Loop through the list of words and count the frequencies
for word in words_list:         if word in word_count:
word_count[word] += 1  # Increment the count if the word is already
in the dictionary         else:             word_count[word] = 1  #
Add the word to the dictionary with a count of 1

    # Return the dictionary of word frequencies
return word_count

# Get the word frequencies
word_count_dict = word_frequencies()

# Print the result
print("Word Frequencies:", word_count_dict)
Word Frequencies: {'APPLE': 2, 'BANANA': 2, 'ORANGE': 1, 'KIWI': 1,
'JACKFRUIT': 1}
```

Q.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

```
# Function to merge two dictionaries and add values for common keys def
merge_dictionaries(dict1, dict2):     merged_dict = dict1.copy()  #
Make a copy of the first dictionary

    # Loop through the second dictionary and add values to the merged
dictionary     for key, value in dict2.items():         if key in
merged_dict:             merged_dict[key] += value  # Add the value if
the key is already present         else:             merged_dict[key]
= value  # Add the key-value pair if the key is not present

    return merged_dict
```

```
# Example usage
dict1_input = input("Enter the first dictionary (in the format
key1:value1,key2:value2,...): ")
dict2_input = input("Enter the second dictionary (in the format
key1:value1,key2:value2,...): ")

# Convert input strings to dictionaries
dict1 = dict(item.split(':') for item in dict1_input.split(','))
dict2 = dict(item.split(':') for item in dict2_input.split(','))

# Convert string values to integers
dict1 = {key: int(value) for key, value in dict1.items()}
dict2 = {key: int(value) for key, value in dict2.items()}

# Merge the dictionaries
merged_dict = merge_dictionaries(dict1, dict2)

# Print the merged dictionary
print("Merged Dictionary:", merged_dict)
```

```
Merged Dictionary: {'A': 1, 'B': 5, 'C': 4, 'D': 5}
```

Q.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

```
# Function to access a value in a nested dictionary using a list of
keys def access_nested_value(nested_dict,
keys):     current_dict = nested_dict

    # Iterate over the keys
for key in keys:
        # Check if the current key exists in the dictionary         if
key in current_dict:             current_dict = current_dict[key]  #
Move to the next level         else:             return None  # Return
None if the key doesn't exist

    return current_dict  # Return the final value after following all
keys

# Example usage
nested_dict = {
'a': {
        'b': {
            'c': 10
        }
    },
    'x': {
```

```
        'y': {
            'z': 20
        }
    }
}

# Input keys as a list keys = input("Enter the list of keys
(separated by commas): ").split(',')

# Get the corresponding value
value = access_nested_value(nested_dict, keys)

# Print the result if value is not None:     print(f"The
value for the given keys is: {value}") else:     print("One
or more keys do not exist in the dictionary.")
```

```
The value for the given keys is: 10
```

Q.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

```
# Function to sort a dictionary based on its values
def sort_dict_by_value(input_dict, ascending=True):
    # Sort the dictionary by value using sorted() and a lambda
function     sorted_dict = dict(sorted(input_dict.items(),
key=lambda item: item[1], reverse=not ascending))     return
sorted_dict

# Example usage
input_dict = {
'a': 3,
    'b': 1,
    'c': 2,
    'd': 5
}

# Get the sorting order from the user
order = input("Enter 'asc' for ascending or 'desc' for descending
order: ").strip().lower()

# Determine if sorting is ascending or descending
ascending = True if order == 'asc' else False

# Get the sorted dictionary
sorted_dict = sort_dict_by_value(input_dict, ascending)
```

```
# Print the sorted dictionary print(f"Sorted dictionary:
{sorted_dict}")

Sorted dictionary: {'b': 1, 'c': 2, 'a': 3, 'd': 5}
```

Q.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.

```
# Function to invert a dictionary, swapping keys and values
def invert_dict(input_dict):     inverted_dict = {}

    # Iterate through each key-value pair in the original dictionary
for key, value in input_dict.items():
        # If the value is already a key in the inverted dictionary,
append the key to the list         if value in inverted_dict:
inverted_dict[value].append(key)         else:
            # If the value is not yet a key, create a new list with
the current key
            inverted_dict[value] = [key]

    return inverted_dict

# Example usage
input_dict = {
'a': 1,
    'b': 2,
    'c': 1,
    'd': 3,
    'e': 2
}

# Invert the dictionary
inverted_dict = invert_dict(input_dict)

# Print the inverted dictionary
print("Inverted Dictionary:", inverted_dict)
Inverted Dictionary: {1: ['a', 'c'], 2: ['b', 'e'], 3: ['d']}
```