```python
# Q1 - Write a code to reverse a string.

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


string = "Daksh Patel"
print(reverse_string(string))
```

```
letaP hskaD
```

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

def count_vowels(s):
    vowels = "aeiouAEIOU"
    count = 0
    for char in s:
        if char in vowels:
            count += 1
    return count


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

```
3
```

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

def is_palindrome(s):
    s = s.lower().replace(" ", "")
    return s == s[::-1]


string = "A man a plan a canal Panama"
print(is_palindrome(string))
```

```
True
```

```python
#4. Write a code to check if two given strings are anagrams of each
other.
def are_anagrams(s1, s2):
    return sorted(s1) == sorted(s2)


string1 = "listen"
string2 = "silent"
print(are_anagrams(string1, string2))
```

```
True
```

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

def find_all_occurrences(string, substring):
    start = 0
    while True:
        start = string.find(substring, start)
        if start == -1:
            break
        yield start
        start += 1


string = "banana"
substring = "ana"
occurrences = list(find_all_occurrences(string, substring))
print(occurrences)
```

```
[1, 3]
```

```python
#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
    previous_char = s[0]

    for i in range(1, len(s)):
        if s[i] == previous_char:
            count += 1
        else:
            compressed.append(previous_char + str(count))
            previous_char = s[i]
            count = 1
    compressed.append(previous_char + str(count))

    compressed_string = ''.join(compressed)
    return compressed_string if len(compressed_string) < len(s) else s


string = "aabcccccaaa"
print(compress_string(string))
```

```
a2b1c5a3
```

```python
#7. Write a code to determine if a string has all unique characters.
def has_unique_characters(s):
    return len(s) == len(set(s))
```

```
string = "abcdefg"
print(has_unique_characters(string))

string = "hello"
print(has_unique_characters(string))
```

```
True
False
```

```
#8. Write a code to convert a given string to uppercase or lowercase.
def to_uppercase(s):
    return s.upper()

def to_lowercase(s):
    return s.lower()


string = "Hello, World!"
print(to_uppercase(string))
print(to_lowercase(string))
```

```
HELLO, WORLD!
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)


string = "Hello, this is an example string."
print(count_words(string))
```

```
6
```

```
#10. Write a code to concatenate two strings without using the +
operator.
def concatenate_strings(s1, s2):
    return "{}{}".format(s1, s2)


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

```
Hello, World!
```

```
#11. Write a code to remove all occurrences of a specific element from
a list.
def remove_element(lst, element):
```

```
    return [x for x in lst if x != element]


lst = [1, 2, 3, 2, 4, 2, 5]
element = 2
print(remove_element(lst, element))
```

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

```
#12.Implement a code to find the second largest number in a given list
of integers.
def second_largest(lst):
    first = second = float('-inf')
    for number in lst:
        if number > first:
            second = first
            first = number
        elif number > second and number != first:
            second = number
    return second


lst = [10, 20, 4, 45, 99]
print(second_largest(lst))
```

```
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 element in lst:
        counts[element] = counts.get(element, 0) + 1
    return counts


lst = [1, 2, 2, 3, 4, 4, 4, 5]
print(count_occurrences(lst))
```

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

```
#14. Write a code to reverse a list in-place without using any built-
in reverse functions.
def reverse_list(lst):
    start = 0
    end = len(lst) - 1
    while start < end:
        lst[start], lst[end] = lst[end], lst[start]
        start += 1
        end -= 1
```

```
    return lst


lst = [1, 2, 3, 4, 5]
print(reverse_list(lst))
```

```
[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()
    unique_list = []
    for item in lst:
        if item not in seen:
            unique_list.append(item)
            seen.add(item)
    return unique_list


original_list = [1, 2, 2, 3, 4, 4, 5]
print(remove_duplicates(original_list))
```

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

```
#16. Create a code to check if a given list is sorted (either in
ascending or descending order) or not.
def is_sorted(lst):
    ascending = all(lst[i] <= lst[i + 1] for i in range(len(lst) - 1))
    descending = all(lst[i] >= lst[i + 1] for i in range(len(lst) -
1))
    return ascending or descending


list1 = [1, 2, 3, 4, 5]
list2 = [5, 4, 3, 2, 1]
list3 = [1, 3, 2, 4, 5]
print(is_sorted(list1))
print(is_sorted(list2))
print(is_sorted(list3))
```

```
True
True
False
```

```
#17. Write a code to merge two sorted lists into a single sorted list.
def merge_sorted_lists(lst1, lst2):
    merged_list = []
    i = j = 0
    while i < len(lst1) and j < len(lst2):
```

```
        if lst1[i] < lst2[j]:
            merged_list.append(lst1[i])
            i += 1
        else:
            merged_list.append(lst2[j])
            j += 1
    merged_list.extend(lst1[i:])
    merged_list.extend(lst2[j:])
    return merged_list


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

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

```
#18. Implement a code to find the intersection of two given lists.
def intersection(lst1, lst2):
    return list(set(lst1) & set(lst2))


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

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

```
#19. Create a code to find the union of two lists without duplicates.
def union(lst1, lst2):
    return list(set(lst1) | set(lst2))


list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]
print(union(list1, list2))
```

```
[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 shuffle_list(lst):
    shuffled = lst[:]
    for i in range(len(shuffled)):
        j = random.randint(0, len(shuffled) - 1)
        shuffled[i], shuffled[j] = shuffled[j], shuffled[i]
    return shuffled
```

```
original_list = [1, 2, 3, 4, 5]
print(shuffle_list(original_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 get_integer_set(prompt):
    return set(map(int, input(prompt).split(',')))

def main():
    set1 = get_integer_set("Enter the first set of integers")
    set2 = get_integer_set("Enter the second set of integers")

    intersection_set = set1 & set2
    print("The intersection of the two sets is:", intersection_set)

if __name__ == "__main__":
    main()
```

```
Enter the first set of integers12345
Enter the second set of integers56789
The intersection of the two sets is: set()
```

```
'''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 concatenate_tuples(tuple1, tuple2):
    return tuple1 + tuple2


tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
result = concatenate_tuples(tuple1, tuple2)
print(result)
```

```
(1, 2, 3, 4, 5, 6)
```

```
'''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):
    return tuple1 + tuple2


tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
result = concatenate_tuples(tuple1, tuple2)
print(result)
```

```
(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_input_set(prompt):

    return set(input(prompt).strip().split(','))

def main():

    set1 = get_input_set("Enter the first set of strings  ")


    set2 = get_input_set("Enter the second set of strings ")


    difference = set1 - set2


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

if __name__ == "__main__":
    main()
```

```
Enter the first set of strings  daksh
Enter the second set of strings patel
Elements present in the first set but not in the second set: {'daksh'}
```

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

def main():
    set1 = get_char_set("Enter the first set of characters")
    set2 = get_char_set("Enter the second set of characters")

    union_set = set1 | set2
    print("Union of the two sets:", union_set)

if __name__ == "__main__":
    main()
```

```
Enter the first set of characterssdsgs
Enter the second set of charactersgdf
Union of the two sets: {'sdsgs', 'gdf'}
```

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

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

def main():
    set1 = get_char_set("Enter the first set of characters")
    set2 = get_char_set("Enter the second set of characters")

    union_set = set1 | set2
    print("Union of the two sets:", union_set)

if __name__ == "__main__":
    main()
```

```
Enter the first set of characterssad
Enter the second set of charactersfdsat
Union of the two sets: {'fdsat', 'sad'}
```

```
'''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(tpl):
    return min(tpl), max(tpl)


tuple_of_integers = (10, 20, 30, 40, 5)
minimum, maximum = find_min_max(tuple_of_integers)
print("Minimum:", minimum)
print("Maximum:", maximum)
```

```
Minimum: 5
Maximum: 40
```

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

def main():
    set1 = {1, 2, 3, 4, 5}
    set2 = {4, 5, 6, 7, 8}

    union_set = set1 | set2
    intersection_set = set1 & set2
    difference_set = set1 - set2
```

```
    print("Union:", union_set)
    print("Intersection:", intersection_set)
    print("Difference:", difference_set)

if __name__ == "__main__":
    main()
```

```
Union: {1, 2, 3, 4, 5, 6, 7, 8}
Intersection: {4, 5}
Difference: {1, 2, 3}
```

```
'''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, element):
    return tpl.count(element)


tuple_of_integers = (1, 2, 2, 3, 4, 2, 5)
element = 2
count = count_occurrences(tuple_of_integers, element)
print(f"The element {element} occurs {count} times.")
```

```
The element 2 occurs 3 times.
```

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

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

def main():
    set1 = get_string_set("Enter the first set of strings")
    set2 = get_string_set("Enter the second set of strings")

    symmetric_difference_set = set1 ^ set2
    print("Symmetric difference of the two sets:",
symmetric_difference_set)

if __name__ == "__main__":
    main()
```

```
Enter the first set of stringsdaksh
Enter the second set of stringspatel
Symmetric difference of the two sets: {'patel', 'daksh'}
```

```
'''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 word_frequency(words):
    frequency = {}
    for word in words:
        if word in frequency:
            frequency[word] += 1
        else:
            frequency[word] = 1
    return frequency


words_list = ["apple", "banana", "apple", "orange", "banana", "apple"]
print(word_frequency(words_list))
```

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


dict1 = {'a': 1, 'b': 2, 'c': 3}
dict2 = {'b': 3, 'c': 4, 'd': 5}
print(merge_dictionaries(dict1, dict2))
```

```
{'a': 1, 'b': 5, 'c': 7, 'd': 5}
```

```
'''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 get_nested_value(d, keys):


    current_dict = d
```

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

    return current_dict

nested_dict = {
    'a': {
        'b': {
            'c': 42
        }
    }
}

keys = ['a', 'b', 'c']
print(get_nested_value(nested_dict, keys))

keys = ['a', 'b', 'd']
print(get_nested_value(nested_dict, keys))
```

```
42
None
```

```
'''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_dict_by_values(d, ascending=True):
    sorted_dict = sorted(d.items(), key=lambda item: item[1],
reverse=not ascending)
    return dict(sorted_dict)

example_dict = {'a': 3, 'b': 1, 'c': 2}
sorted_dict_asc = sort_dict_by_values(example_dict, ascending=True)
sorted_dict_desc = sort_dict_by_values(example_dict, ascending=False)

print("Sorted dictionary in ascending order:", sorted_dict_asc)

print("Sorted dictionary in descending order:", sorted_dict_desc)
```

```
Sorted dictionary in ascending order: {'b': 1, 'c': 2, 'a': 3}
Sorted dictionary in descending order: {'a': 3, 'c': 2, 'b': 1}
```

```python
''' 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_dict(d):
    inverted = {}
    for key, value in d.items():
        if value in inverted:
            inverted[value].append(key)
        else:
            inverted[value] = [key]
    return inverted


original_dict = {'a': 1, 'b': 2, 'c': 1, 'd': 3, 'e': 2}
inverted_dict = invert_dict(original_dict)
print(inverted_dict)
```

```
{1: ['a', 'c'], 2: ['b', 'e'], 3: ['d']}
```