Data Structure Assignment

In [1]:
```
# 1.write a code to reverse a string
txt="Hello World"
new_txt=txt[::-1]
new_txt
```
Out[1]:
```
'dlroW olleH'
```

In [1]:
```
# 2.write code to count number of vowels in a string

str1=input("enter the sentence:")
str1_lower=str1.lower()
vowels="aeiou"
count=0

for i in str1_lower:
    if i in vowels:
        count=count+1
        print("count of the vowels in given string:",count)
```
```
enter the sentence:hello world
count of the vowels in given string: 1
count of the vowels in given string: 2
count of the vowels in given string: 3
```

In [22]:
```
#3.write a code to check if a given string is pallindrome or not

# pallindrome is a word,phrase,number,or other sequence of character that reads the same forward and backword
x1x1x1
word1="madam"
reverse=word1[::-1]

if reverse==word1:
    print("pallindrome")
else:
    print("not pallindrome")
```
```
pallindrome
```

In [21]:
```
#4.write a code to check if given strings are anagrams of each other
#anagrams are words for phrases formed by rearranging the letter of another words or phrase to produce
#a new string using all the original letters exactly once.

word1="thing"
word2="night"

if(sorted(word1)==sorted(word2)):
    print("anagrams")
else:
    print("not anagrams")
```
```
anagrams
```

In [16]:
```
#5.write a code to the occurence of a given substring within another string

def count_substring_occurrences(main_string, substring):
    # Use the count() method to find the number of non-overlapping occurrences
    return main_string.count(substring)

# Example usage
main_string = "this is a test string with a test substring test"
substring = "test"

result = count_substring_occurrences(main_string, substring)
print(f"The substring '{substring}' occurs {result} times in the main string.")
```
```
The substring 'test' occurs 3 times in the main string.
```

In [18]:
```
#6.write a code to perform basic string compression using the counts of repeated character

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

    compressed_string = []
    count = 1
    prev_char = s[0]

    for char in s[1:]:
        if char == prev_char:
            count += 1
        else:
            compressed_string.append(f"{prev_char}{count}")
            prev_char = char
            count = 1

    # Append the last set of characters
    compressed_string.append(f"{prev_char}{count}")

    # Join the list into a string and return it
    compressed_result = ''.join(compressed_string)

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

# Example usage
input_string = "aaabbcddd"
compressed = compress_string(input_string)
print(f"Compressed string: {compressed}")  # Should print "a3b2c1d3"
```
```
Compressed string: a3b2c1d3
```

In [3]:
```
#7. write a code to determine if a string has all unique characters.

def has_unique_characters(s):
    # Create a set to store unique characters
    char_set = set()

    # Iterate through each character in the string
    for char in s:
        # If the character is already in the set, return False
        if char in char_set:
            return False
        # Add the character to the set
        char_set.add(char)

    # If no duplicates were found, return True
    return True

# Test cases
print(has_unique_characters("abcdef"))  # Should return True
print(has_unique_characters("hello"))   # Should return False
print(has_unique_characters(""))        # Should return True
print(has_unique_characters("AaBbCc"))  # Should return True
```
```
True
False
True
True
```

In [2]:
```
#8. write a code to convert a given sring to uppercase or lowercase

txt="DATA ANALYST"
convert_txt=txt.lower()
convert_txt
#for uppercase
txt="data analyst"
txt_upper=txt.upper()

print("lowercase string:",convert_txt)
print("uppercase string:",txt_upper)
```
```
lowercase string: data analyst
uppercase string: DATA ANALYST
```

In [17]:
```
#9.1.write a code to count the number of words in a string

welcome=str("hello welcome home")
print(welcome)
words=welcome.split()
print(words)
number_of_Words=len(words)
print(number_of_Words)
```
```
hello welcome home
['hello', 'welcome', 'home']
3
```

In [29]:
```
#10. write a code to concatenate two strings without using the + operator

string1 = "Hello"
string2 = "World"

concatenated_string = "".join([string1, string2])
print(concatenated_string)

#another way
string1 = "Hello"
string2 = "World"

concatenated_string = "{}{}".format(string1, string2)
print(concatenated_string)
```
```
HelloWorld
HelloWorld
```

In [36]:
```
#11.implement a code to remove all occurence of a specific element from a list.

book_list=["english","maths","maths","science","hindi"]
book_list.remove("maths")
book_list
```
Out[36]:
```
['english', 'maths', 'science', 'hindi']
```

In [41]:
```
#12.implement a code to find second largest  number in a given list of integers

def second_largest(numbers):
    # Sort the list in descending order
    sorted_numbers = sorted(numbers, reverse=True)

    # Return the second element
    return sorted_numbers[1]

# Test the function
numbers = [10, 20, 30, 40, 50]
print("Second largest number:", second_largest(numbers))
```
```
Second largest number: 40
```

In [11]:
```
#13.create a code to count the occurence of each element in a list and return a dictionary with elements as keys and their counts as values.

def count_occurrences(lst):
    element_count = {}
    for item in lst:
        if item in element_count:
            element_count[item] = element_count[item]+1
        else:
            element_count[item] = 1
    return element_count

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

In [13]:
```
#14.write a code to reverse a list in place without using any built -in reverse function

def reverse_list_in_place(lst):
    left = 0
    right = len(lst) - 1

    while left < right:
        # Swap elements at left and right indices
        lst[left], lst[right] = lst[right], lst[left]
        # Move pointers towards the center
        left += 1
        right -= 1

# Example usage:
my_list = [1, 2, 3, 4, 5]
print("Original list:", my_list)
reverse_list_in_place(my_list)
print("Reversed list:", my_list)
```
```
Original list: [1, 2, 3, 4, 5]
Reversed list: [5, 4, 3, 2, 1]
```

In [4]:
```
#15. Implement a code to find and remove duplicates from a list while preserving the origin'l order of
#elements

test_list = [1, 5, 3, 6, 3, 5, 6, 1]
print ("The original list is : "
        + str(test_list))

test_list = list(set(test_list))

print ("The list after removing duplicates : "
        + str(test_list))
```
```
The original list is : [1, 5, 3, 6, 3, 5, 6, 1]
The list after removing duplicates : [1, 3, 5, 6]
```

In [10]:
```
#16 Create a code to check if a given list is sorted (either is ascending or descending) order or not.


test_list = [1, 4, 5, 8, 10]
is_sorted = all(a <= b for a, b in zip(test_list, test_list[1:]))

if is_sorted:
    print("Yes, the list is sorted.")
else:
    print("No, the list is not sorted.")
```
```
Yes, the list is sorted.
```

In [11]:
```
#17.write a code to merge two sorted list into single sorted list
def merge_sorted_lists(list1, list2):
    merged_list = []
    i = 0
    j = 0

    # Merge elements until one of the lists is exhausted
    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

    # Add remaining elements from the first list
    while i < len(list1):
        merged_list.append(list1[i])
        i += 1

    # Add remaining elements from the second list
    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]
```

In [12]:
```
#18.implement a code to find the intersection of two given lists

def intersection(list1,list2):
    set1=set(list1)
    set2=set(list2)

    intersect=set1.intersection(set2)

    return list(intersect)

list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]
intersect = intersection(list1, list2)
print("Intersection of the two lists:", intersect)
```
```
Intersection of the two lists: [3, 4, 5]
```

In [16]:
```
#19.create a code to find union of two lists without duplicate

def union(list1,list2):
    set1=set(list1)
    set2=set(list2)

    union=set1.union(set2)

    return list(union)

list1 = [1, 2, 3, 4, 5]
list2 = [8,9,10, 6, 7]
union=union(list1,list2)

print("union of two lists:",union)
```
```
union of two lists: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [17]:
```
#20.write a code to shuffle a given list randomly without using any built-in shuffle functions.

import random

def shuffle_list(arr):
    shuffled_arr = arr[:]

    # Start from the end of the list
    for i in range(len(shuffled_arr) - 1, 0, -1):
        # Generate a random index between 0 and i (inclusive)
        j = random.randint(0, i)


        shuffled_arr[i], shuffled_arr[j] = shuffled_arr[j], shuffled_arr[i]

    return shuffled_arr

# Example usage:
original_list = [1, 2, 3, 4, 5]
shuffled_list = shuffle_list(original_list)
print("Original list:", original_list)
print("Shuffled list:", shuffled_list)
```
```
Original list: [1, 2, 3, 4, 5]
Shuffled list: [5, 1, 4, 3, 2]
```

In [23]:
```
#21.Write a code that takes two tuple as input and returns a new tuple containing elements that are common to bith input tuples

def common_elements(tuple1, tuple2):
    # Convert tuples to sets
    set1 = set(tuple1)
    set2 = set(tuple2)

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

    # Convert the intersection set back to a tuple
    common_tuple = tuple(common_set)

    return common_tuple

# Example usage:
tuple1 = (1, 2, 3, 4, 5)
tuple2 = (3, 4, 5, 6, 7)
common = common_elements(tuple1, tuple2)
print("Common elements in the two tuples:", common)
```
```
Common elements in the two tuples: (3, 4, 5)
```

In [28]:
```
#22.create a code that prompts the user to enter two  sets of integers seperated by commas.then, print the intersections of these two sets
def get_set_from_input():
    user_input = input("Enter a set of integers separated by commas: ")
    # Split the input string by commas and convert it into a set of integers
    integer_set = set(map(int, user_input.split(',')))
    return integer_set

def main():
    print("Enter the first set:")
    set1 = get_set_from_input()

    print("Enter the second set:")
    set2 = get_set_from_input()

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

    print("Intersection of the two sets:", intersection_set)

if __name__ == "__main__":
    main()
```
```
Enter the first set:
Enter a set of integers separated by commas: 12,3,4,5,6
Enter the second set:
Enter a set of integers separated by commas: 2,4,6,8,10
Intersection of the two sets: {4, 6}
```

In [4]:
```
#23. write a code toconcatenate 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):
    # Use the + operator to concatenate the tuples
    return tuple1 + tuple2

# Test cases
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)

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

In [29]:
```
#24. write acode 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_two_tuples(tuple1,tuple2):
    concatenated_tuple=tuple1+tuple2
    return cocatenated_tuple
#example usage
tuple1=(10,20,30,40,50)
tuple2=(2,4,6,8,10)
concatenated_tuple=tuple1+tuple2

print("concatenated tuples are:",concatenated_tuple)
```
```
concatenated tuples are: (10, 20, 30, 40, 50, 2, 4, 6, 8, 10)
```

In [30]:
```
#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 spacified range of indices.

def slice_tuple(original_tuple, start_index, end_index):
    # Slice the original tuple based on the specified range of indices
    sliced_tuple = original_tuple[start_index:end_index + 1]
    return sliced_tuple

# Example usage:
original_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9)
start_index = 2
end_index = 5
result_tuple = slice_tuple(original_tuple, start_index, end_index)
print("Sliced tuple:", result_tuple)
```
```
Sliced tuple: (3, 4, 5, 6)
```

In [32]:
```
#26.write a code that prompts the user to input two sets of character .then print the union of these two set

def get_set_from_input():
    user_input=input("Enter a set of characters separated by commas: ")

    char_set = set(user_input.split(','))
    return char_set

def main():
    print("Enter the first set of characters:")
    set1 = get_set_from_input()

    print("Enter the second set of characters:")
    set2 = get_set_from_input()

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

    print("Union of the two sets:", union_set)

if __name__ == "__main__":
    main()
```
```
Enter the first set of characters:
Enter a set of characters separated by commas: 2,4,6
Enter the second set of characters:
Enter a set of characters separated by commas: 1,2,3
Union of the two sets: {'3', '6', '1', '2', '4'}
```

In [2]:
```
#27.& Develop a code that takes a tuple of integers as input . The function should return maximum and minimum vlues from the tuple using tuple unpacking


def find_max_min(numbers):
    if not numbers:
        return None, None  # Return None if the tuple is empty

    # Initialize min and max with the first element of the tuple
    min_val, max_val = numbers[0], numbers[0]

    for num in numbers[1:]:
        if num < min_val:
            min_val = num
        if num > max_val:
            max_val = num

    return max_val, min_val

# Example usage:
numbers = (3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5)
max_value, min_value = find_max_min(numbers)
print(f"Max value: {max_value}, Min value: {min_value}")
```
```
Max value: 9, Min value: 1
```

In [5]:
```
#28. Create a code that defines two sets of integers.then print the union, intersection and difference of these rwo sets


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

# Calculate the union of the sets
union_set = set1.union(set2)
print("Union:", union_set)

# Calculate the intersection of the sets
intersection_set = set1.intersection(set2)
print("Intersection:", intersection_set)

# Calculate the difference of the sets
difference_set1 = set1.difference(set2)
difference_set2 = set2.difference(set1)
print("Difference (set1 - set2):", difference_set1)
print("Difference (set2 - set1):", difference_set2)
```
```
Union: {1, 2, 3, 4, 5, 6, 7, 8}
Intersection: {4, 5}
Difference (set1 - set2): {1, 2, 3}
Difference (set2 - set1): {8, 6, 7}
```

In [6]:
```
#29. write a code that takes  a tuple and an element as input. the function should return the count of occurences of given elements in the tuple

def count_occurrences(tup, element):
    # Use the count() method of tuple to count the occurrences of the element
    return tup.count(element)

# Test cases
tup = (1, 2, 3, 4, 2, 2, 5, 6)
element = 2

result = count_occurrences(tup, element)
print(f"The element {element} occurs {result} times in the tuple.")
```
```
The element 2 occurs 3 times in the tuple.
```

In [7]:
```
#30.develop a code that prompts the user to input two sets of strings. Then, print the aymmetric difference of these two sets.

# Function to get a set of strings from user input
def get_set_from_input(prompt):
    input_string = input(prompt)
    return set(input_string.split())

# Prompt the user to input two sets of strings
set1 = get_set_from_input("Enter the first set of strings (separated by spaces): ")
set2 = get_set_from_input("Enter the second set of strings (separated by spaces): ")

# Calculate the asymmetric difference (symmetric difference) of the sets
asymmetric_difference = set1.symmetric_difference(set2)

# Print the asymmetric difference
print("Asymmetric Difference:", asymmetric_difference)
```
```
Enter the first set of strings (separated by spaces): Pwskills is a data science course
Enter the second set of strings (separated by spaces): I have enrolled for data analytics course
Asymmetric Difference: {'analytics', 'a', 'enrolled', 'I', 'is', 'Pwskills', 'for', 'have', 'science'}
```

In [9]:
```
#31. write a code that takes a list of words as imput 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_frequencies(word_list):
    # Create an empty dictionary to store word frequencies
    freq_dict = {}

    # Iterate through each word in the list
    for word in word_list:
        # If the word is already in the dictionary, increment its count
        if word in freq_dict:
            freq_dict[word] += 1
        # If the word is not in the dictionary, add it with a count of 1
        else:
            freq_dict[word] = 1

    # Return the dictionary with word frequencies
    return freq_dict

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

In [10]:
```
#32.write a code that takes two dictionaries as input and merges them into a single dictionary .
#if these are common keys, the values should be added together.

def merge_dictionaries(dict1, dict2):
    # Create a new dictionary to store the merged result
    merged_dict = dict1.copy()

    # Iterate through each key-value pair in the second dictionary
    for key, value in dict2.items():
        # If the key is already in the merged dictionary, add the values
        if key in merged_dict:
            merged_dict[key] += value
        # If the key is not in the merged dictionary, add the key-value pair
        else:
            merged_dict[key] = value

    # Return the merged dictionary
    return merged_dict

# Example usage
dict1 = {'a': 1, 'b': 2, 'c': 3}
dict2 = {'b': 3, 'c': 4, 'd': 5}
result = merge_dictionaries(dict1, dict2)
print(result)  # Should print {'a': 1, 'b': 5, 'c': 7, 'd': 5}
```
```
{'a': 1, 'b': 5, 'c': 7, 'd': 5}
```

In [11]:
```
#33.write a code to access value in a nested dictionary. The function should take the dictionary and list pf keys as input.
# ad return the corresponding values .if any of the key do not exist in the dictionary .the function should return none.

def get_nested_value(nested_dict, keys):
    current_value = nested_dict

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

    return current_value

# Example usage
nested_dict = {
    'a': {
        'b': {
            'c': 42
        }
    },
    'x': {
        'y': {
            'z': 99
        }
    }
}

keys = ['a', 'b', 'c']
result = get_nested_value(nested_dict, keys)
print(result)  # Should print 42

keys = ['x', 'y', 'z']
result = get_nested_value(nested_dict, keys)
print(result)  # Should print 99

keys = ['a', 'b', 'd']
result = get_nested_value(nested_dict, keys)
print(result)  # Should print None
```
```
42
99
None
```

In [12]:
```
#34. write a code that takes a dictionary as input and returns sorted version of it based on the values.you can choose whether to sort in ascending or descendi

def sort_dictionary_by_values(input_dict, ascending=True):
    # Sort the dictionary by its values
    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}

# Sort in ascending order
sorted_dict_asc = sort_dictionary_by_values(input_dict, ascending=True)
print("Ascending order:", sorted_dict_asc)  # Should print {'b': 1, 'c': 2, 'a': 3, 'd': 5}

# Sort in descending order
sorted_dict_desc = sort_dictionary_by_values(input_dict, ascending=False)
print("Descending order:", sorted_dict_desc)  # Should print {'d': 5, 'a': 3, 'c': 2, 'b': 1}
```
```
Ascending order: {'b': 1, 'c': 2, 'a': 3, 'd': 5}
Descending order: {'d': 5, 'a': 3, 'c': 2, 'b': 1}
```

In [13]:
```
#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 sort_dictionary_by_values(input_dict, ascending=True):
    # Sort the dictionary by its values
    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}

# Sort in ascending order
sorted_dict_asc = sort_dictionary_by_values(input_dict, ascending=True)
print("Ascending order:", sorted_dict_asc)  # Should print {'b': 1, 'c': 2, 'a': 3, 'd': 5}

# Sort in descending order
sorted_dict_desc = sort_dictionary_by_values(input_dict, ascending=False)
print("Descending order:", sorted_dict_desc)  # Should print {'d': 5, 'a': 3, 'c': 2, 'b': 1}
```
```
Ascending order: {'b': 1, 'c': 2, 'a': 3, 'd': 5}
Descending order: {'d': 5, 'a': 3, 'c': 2, 'b': 1}
```

In [ ]: