DATA STRUCTURES ASSIGNMENT

```python
#Write a code to reverse a string
def reverse_string(s):
    reversed_str = ""
    for char in s:
        reversed_str = char + reversed_str
    return reversed_str
string = "Hello, World!"
reversed_string = reverse_string(string)
print(reversed_string)
#Write a code to count the number of vowels in a string
def count_vowels(s):
    vowels = "aeiouAEIOU"  # Define vowels in both lowercase and uppercase
    count = 0
    for char in s:
        if char in vowels:
            count += 1
    return count
string = "Hello, World!"
vowel_count = count_vowels(string)
print(f"Number of vowels: {vowel_count}")
#Write a code to check if a given string is a palindrome or not
def is_palindrome(s):
    # Remove spaces and convert to lowercase for uniform comparison
    s = s.replace(" ", "").lower()
    # Check if the string is equal to its reverse
    return s == s[::-1]
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")
#Write a code to check if two given strings are anagrams of each other
def are_anagrams(str1, str2):
    # Remove spaces and convert to lowercase
    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()
    # Check if sorted characters of both strings are equal
    return sorted(str1) == sorted(str2)
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")
```

```python
#Write a code to find all occurrences of a given substring within another
string
def find_all_occurrences(main_string, substring):
    occurrences = []
    start = 0
    while True:
        start = main_string.find(substring, start)
        if start == -1:
            break
        occurrences.append(start)
        start += 1  # Move past the last found substring
    return occurrences
main_string = "This is a test string. Let's test this string with a test."
substring = "test"
positions = find_all_occurrences(main_string, substring)
print(f"Occurrences of '{substring}' found at positions: {positions}")
#Write a code to perform basic string compression using the counts of repeated
characters
def compress_string(s):
    compressed = []
    count = 1
    # Iterate over the string
    for i in range(1, len(s)):
        if s[i] == s[i - 1]:
            count += 1
        else:
            compressed.append(s[i - 1] + str(count))
            count = 1
    # Append the last set of characters
    compressed.append(s[-1] + str(count))
    # Join the compressed parts into a single string
    compressed_string = ''.join(compressed)
    # Return the original string if compression doesn't reduce size
    return compressed_string if len(compressed_string) < len(s) else s
string = "aaabbccccdaa"
compressed_string = compress_string(string)
print(f"Compressed string: {compressed_string}")
#Write a code to determine if a string has all unique characters
def has_unique_characters(s):
    # Using a set to track characters we've seen
    char_set = set()
    for char in s:
        if char in char_set:
            return False  # Duplicate character found
        char_set.add(char)
    return True  # No duplicates found
string = "abcdefg"
```

```python
if has_unique_characters(string):
    print(f"'{string}' has all unique characters")
else:
    print(f"'{string}' does not have all unique characters")
#Write a code to convert a given 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
string = "pw skills"
uppercase_string = convert_case(string, to_upper=True)
lowercase_string = convert_case(string, to_upper=False)
print(f"Uppercase: {uppercase_string}")
print(f"Lowercase: {lowercase_string}")
#Write a code to count the number of words in a string
def count_words(s):
    # Split the string into words based on spaces
    words = s.split()
    return len(words)
string = "PW skills is best learning platform."
word_count = count_words(string)
print(f"Number of words: {word_count}")
#Write a code to concatenate two strings without using the + operator
def concatenate_strings(str1, str2):
    return ''.join([str1, str2])
string1 = "PW "
string2 = "Skills"
concatenated_string = concatenate_strings(string1, string2)
print(f"Concatenated string: {concatenated_string}")
#Write a code to remove all occurrences of a specific element from a list
def remove_occurrences(lst, element):
    # Use list comprehension to create a new list without the specified
element
    return [item for item in lst if item != element]
my_list = [1, 2, 3, 4, 2, 5, 2]
element_to_remove = 2
updated_list = remove_occurrences(my_list, element_to_remove)
print(f"Updated list: {updated_list}")
#Implement a code to find the second largest number in a given list of
integers
def second_largest(numbers):
    if len(numbers) < 2:
        return None  # Not enough elements for a second largest

    first = second = float('-inf')  # Initialize to negative infinity
```

```
    for number in numbers:
        if number > first:
            second = first  # Update second largest
            first = number  # Update largest
        elif first > number > second:
            second = number  # Update second largest

    return second if second != float('-inf') else None

num_list = [3, 5, 1, 4, 2, 5]
result = second_largest(num_list)
if result is not None:
    print(f"The second largest number is: {result}")
else:
    print("There is no second largest number.")
#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):
    occurrences = {}
    for item in lst:
        if item in occurrences:
            occurrences[item] += 1  # Increment count if item already exists
        else:
            occurrences[item] = 1  # Initialize count for new item
    return occurrences
my_list = [1, 2, 2, 3, 1, 4, 5, 3, 2]
result = count_occurrences(my_list)
print(f"Occurrences: {result}")
#Write a code to reverse a list in-place without using any built-in reverse
functions
def reverse_list(lst):
    left = 0
    right = len(lst) - 1

    while left < right:
        # Swap the elements at left and right indices
        lst[left], lst[right] = lst[right], lst[left]
        # Move the pointers towards the center
        left += 1
        right -= 1
my_list = [1, 2, 3, 4, 5]
reverse_list(my_list)
print(f"Reversed list: {my_list}")
#Write a code to merge two sorted lists into a single sorted list
def merge_sorted_lists(list1, list2):
    merged_list = []
    i, j = 0, 0
```

```
    # Iterate through both lists and append the smaller element to the merged
list
    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 there are remaining elements in list1, add them
    while i < len(list1):
        merged_list.append(list1[i])
        i += 1

    # If there are remaining elements in list2, add them
    while j < len(list2):
        merged_list.append(list2[j])
        j += 1

    return merged_list
sorted_list1 = [1, 3, 5, 7]
sorted_list2 = [2, 4, 6, 8]
result = merge_sorted_lists(sorted_list1, sorted_list2)
print(f"Merged sorted list: {result}")
#Implement a code to find the intersection of two given lists
def list_intersection(list1, list2):
    # Use a set for faster lookups
    set2 = set(list2)
    intersection = []

    for item in list1:
        if item in set2:
            intersection.append(item)

    return intersection
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
result = list_intersection(list1, list2)
print(f"Intersection of the two lists: {result}")
#Implement a code to find and remove duplicates from a list while preserving
the original order of elements
def remove_duplicates(lst):
    seen = set()  # Create a set to keep track of seen elements
    unique_list = []  # This will store the final list without duplicates
```

```
    for item in lst:
        if item not in seen:
            seen.add(item)  # Add to seen set
            unique_list.append(item)  # Append to unique list

    return unique_list
my_list = [1, 2, 3, 2, 4, 1, 5, 3]
result = remove_duplicates(my_list)
print(f"List after removing duplicates: {result}")
#Create a code to check if a given list is sorted (either in ascending or
descending order) or not
def is_sorted(lst):
    if len(lst) < 2:  # A list with 0 or 1 element is considered sorted
        return True

    ascending = True
    descending = True

    for i in range(1, len(lst)):
        if lst[i] < lst[i - 1]:
            ascending = False
        elif lst[i] > lst[i - 1]:
            descending = False

    return ascending or descending

my_list1 = [1, 2, 3, 4, 5]  # Ascending
my_list2 = [5, 4, 3, 2, 1]  # Descending
my_list3 = [1, 3, 2, 4]     # Unsorted

print(f"Is the first list sorted? {is_sorted(my_list1)}")  # True
print(f"Is the second list sorted? {is_sorted(my_list2)}") # True
print(f"Is the third list sorted? {is_sorted(my_list3)}")  # False
#Create a code to find the union of two lists without duplicates
def union_of_lists(list1, list2):
    # Use a set to automatically handle duplicates
    union_set = set(list1) | set(list2)
    return list(union_set)
list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
result = union_of_lists(list1, list2)
print(f"Union of the two lists: {result}")
#Write a code to shuffle a given list randomly without using any built-in
shuffle functions
import random
```

```python
def shuffle_list(lst):
    # Create a copy of the original list to shuffle
    shuffled = lst[:]
    n = len(shuffled)

    # Implementing the Fisher-Yates shuffle algorithm
    for i in range(n - 1, 0, -1):
        j = random.randint(0, i)  # Pick a random index from 0 to i
        # Swap the current element with the element at the random index
        shuffled[i], shuffled[j] = shuffled[j], shuffled[i]

    return shuffled
my_list = [1, 2, 3, 4, 5]
shuffled_list = shuffle_list(my_list)
print(f"Original list: {my_list}")
print(f"Shuffled list: {shuffled_list}")
#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):
    # Convert tuples to sets to find the intersection
    common_set = set(tuple1) & set(tuple2)
    # Convert the set back to a tuple
    return tuple(common_set)
tuple1 = (1, 2, 3, 4, 5)
tuple2 = (4, 5, 6, 7, 8)
result = common_elements(tuple1, tuple2)
print(f"Common elements: {result}")
#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 get_set_from_input(prompt):
    # Take user input, split by commas, and convert to a set of integers
    return set(map(int, input(prompt).split(',')))

def main():
    # Prompt the user to enter two sets of integers
    set1 = get_set_from_input("Enter the first set of integers (separated by
commas): ")
    set2 = get_set_from_input("Enter the second set of integers (separated by
commas): ")

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

    # Print the intersection
    print(f"Intersection of the two sets: {intersection}")
```

```
# Run the program
main()
#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):
    # Concatenate the two tuples
    return tuple1 + tuple2

tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
result = concatenate_tuples(tuple1, tuple2)
print(f"Concatenated tuple: {result}")
#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_set_from_input(prompt):
    # Take user input, split by commas, and convert to a set of strings
    return set(input(prompt).split(','))

def main():
    # Prompt the user to enter two sets of strings
    set1 = get_set_from_input("Enter the first set of strings (separated by
commas): ")
    set2 = get_set_from_input("Enter the second set of strings (separated by
commas): ")

    # Find the difference between the two sets
    difference = set1 - set2

    # Print the elements present in the first set but not in the second
    print(f"Elements in the first set but not in the second set:
{difference}")

# Run the program
main()
#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 slice_tuple(original_tuple, start_index, end_index):
    # Return a new tuple containing elements from the specified range
    return original_tuple[start_index:end_index]
my_tuple = (10, 20, 30, 40, 50, 60)
start = 1  # Starting index (inclusive)
end = 4    # Ending index (exclusive)
result = slice_tuple(my_tuple, start, end)
print(f"Original tuple: {my_tuple}")
print(f"New tuple containing elements from index {start} to {end}: {result}")
```

```
#Write a code that prompts the user to input two sets of characters. Then,
print the union of these two sets
def get_set_from_input(prompt):
    # Take user input, split by commas, and convert to a set of characters
    return set(input(prompt).replace(" ", "").split(','))

def main():
    # Prompt the user to enter two sets of characters
    set1 = get_set_from_input("Enter the first set of characters (separated by
commas): ")
    set2 = get_set_from_input("Enter the second set of characters (separated
by commas): ")

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

    # Print the union
    print(f"Union of the two sets: {union}")

# Run the program
main()
#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 get_min_max(input_tuple):
    # Using the built-in min and max functions to find min and max
    min_value = min(input_tuple)
    max_value = max(input_tuple)

    # Return both values as a tuple
    return min_value, max_value

my_tuple = (10, 20, 5, 30, 15)
min_value, max_value = get_min_max(my_tuple)

print(f"Original tuple: {my_tuple}")
print(f"Minimum value: {min_value}, Maximum value: {max_value}")
#Create a code that defines two sets of integers. Then, print the union,
intersection, and difference of these two sets
def main():
    # Define two sets of integers
    set1 = {1, 2, 3, 4, 5}
    set2 = {4, 5, 6, 7, 8}

    # Calculate the union, intersection, and difference
    union = set1 | set2
    intersection = set1 & set2
```

```
    difference = set1 - set2

    # Print the results
    print(f"Set 1: {set1}")
    print(f"Set 2: {set2}")
    print(f"Union of the two sets: {union}")
    print(f"Intersection of the two sets: {intersection}")
    print(f"Difference of Set 1 and Set 2 (Set 1 - Set 2): {difference}")

# Run the program
main()
#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(input_tuple, element):
    # Use the count method of the tuple to find occurrences of the element
    return input_tuple.count(element)
my_tuple = (1, 2, 3, 1, 4, 1, 5)
element_to_count = 1
count = count_occurrences(my_tuple, element_to_count)

print(f"Original tuple: {my_tuple}")
print(f"Count of {element_to_count} in the tuple: {count}")
#Develop a code that prompts the user to input two sets of strings. Then,
print the symmetric difference of these two sets
def get_set_from_input(prompt):
    # Take user input, split by commas, and convert to a set of strings
    return set(input(prompt).replace(" ", "").split(','))

def main():
    # Prompt the user to enter two sets of strings
    set1 = get_set_from_input("Enter the first set of strings (separated by
commas): ")
    set2 = get_set_from_input("Enter the second set of strings (separated by
commas): ")

    # Calculate the symmetric difference of the two sets
    symmetric_difference = set1 ^ set2

    # Print the symmetric difference
    print(f"Symmetric difference of the two sets: {symmetric_difference}")

# Run the program
main()
#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
```

```python
def word_frequencies(words):
    frequency_dict = {}

    for word in words:
        # Increment the count for each word in the dictionary
        if word in frequency_dict:
            frequency_dict[word] += 1
        else:
            frequency_dict[word] = 1

    return frequency_dict

input_words = input("Enter a list of words separated by spaces: ").split()
result = word_frequencies(input_words)

print("Word frequencies:")
for word, count in result.items():
    print(f"{word}: {count}")
#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):
    # Create a new dictionary to store the merged result
    merged_dict = dict1.copy()  # Start with the first dictionary

    for key, value in dict2.items():
        # If the key exists in both dictionaries, add the values
        if key in merged_dict:
            merged_dict[key] += value
        else:
            # Otherwise, just add the new key-value pair
            merged_dict[key] = value

    return merged_dict

dict1 = {'a': 1, 'b': 2, 'c': 3}
dict2 = {'b': 3, 'c': 4, 'd': 5}

result = merge_dictionaries(dict1, dict2)

print("Merged dictionary:", result)
#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(nested_dict, keys):
```

```
    # Initialize the current level of the dictionary
    current_value = nested_dict

    # Iterate through each key in the list of keys
    for key in keys:
        # Check if the key exists in the current level
        if key in current_value:
            current_value = current_value[key]  # Move to the next level
        else:
            return None  # Return None if the key does not exist

    return current_value  # Return the final value if all keys are valid

nested_dictionary = {
    'a': {
        'b': {
            'c': 10,
            'd': 20
        },
        'e': 30
    },
    'f': 40
}

keys_to_access = ['a', 'b', 'c']
result = get_nested_value(nested_dictionary, keys_to_access)

print(f"Value accessed: {result}")  # Output: Value accessed: 10
#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(input_dict, descending=False):
    # Sort the dictionary by values and return a new sorted dictionary
    sorted_dict = dict(sorted(input_dict.items(), key=lambda item: item[1],
reverse=descending))
    return sorted_dict
my_dict = {'apple': 5, 'banana': 2, 'cherry': 7, 'date': 3}

# Sort in ascending order
sorted_asc = sort_dict_by_values(my_dict, descending=False)
print("Sorted dictionary (ascending):", sorted_asc)

# Sort in descending order
sorted_desc = sort_dict_by_values(my_dict, descending=True)
print("Sorted dictionary (descending):", sorted_desc)
```

```python
#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(input_dict):
    inverted_dict = {}

    for key, value in input_dict.items():
        # If the value is already a key in the inverted dictionary
        if value in inverted_dict:
            inverted_dict[value].append(key)  # Append the key to the list
        else:
            inverted_dict[value] = [key]  # Create a new list with the key

    return inverted_dict

my_dict = {'a': 1, 'b': 2, 'c': 1, 'd': 3}

inverted = invert_dictionary(my_dict)
print("Original dictionary:", my_dict)
print("Inverted dictionary:", inverted)
```