1. Write a Python program to Extract Unique values dictionary values?

ANS: def extract\_unique\_values(dictionary):

unique\_values = set()

for values\_list in dictionary.values():

unique\_values.update(set(values\_list))

return list(unique\_values)

# Test the function

if \_\_name\_\_ == "\_\_main\_\_":

my\_dictionary = {

'key1': [1, 2, 3],

'key2': [2, 3, 4],

'key3': [3, 4, 5],

'key4': [4, 5, 6]

}

unique\_values\_list = extract\_unique\_values(my\_dictionary)

print("Unique values in dictionary:", unique\_values\_list)

1. Write a Python program to find the sum of all items in a dictionary?

ANS: def sum\_all\_items\_in\_dictionary(dictionary):

total\_sum = 0

for value in dictionary.values():

if isinstance(value, (int, float)):

total\_sum += value

elif isinstance(value, list) or isinstance(value, tuple):

total\_sum += sum(value)

return total\_sum

# Test the function

if \_\_name\_\_ == "\_\_main\_\_":

my\_dictionary = {

'key1': 10,

'key2': [1, 2, 3],

'key3': 5.5,

'key4': (10, 20)

}

result = sum\_all\_items\_in\_dictionary(my\_dictionary)

print("Sum of all items in the dictionary:", result)

1. Write a Python program to Merging two Dictionaries?

ANS: To merge two dictionaries in Python, you can use the `update()` method or the dictionary unpacking (`\*\*`) feature. Both methods allow you to combine the key-value pairs of two dictionaries into one. Here's a Python program that demonstrates both approaches:

Using `update()` method:

def merge\_dictionaries(dict1, dict2):

merged\_dict = dict1.copy()

merged\_dict.update(dict2)

return merged\_dict

# Test the function

if \_\_name\_\_ == "\_\_main\_\_":

dict1 = {'a': 1, 'b': 2}

dict2 = {'c': 3, 'd': 4}

merged\_dict = merge\_dictionaries(dict1, dict2)

print("Merged dictionary using update():", merged\_dict)

Using dictionary unpacking (`\*\*`):

def merge\_dictionaries(dict1, dict2):

merged\_dict = {\*\*dict1, \*\*dict2}

return merged\_dict

# Test the function

if \_\_name\_\_ == "\_\_main\_\_":

dict1 = {'a': 1, 'b': 2}

dict2 = {'c': 3, 'd': 4}

merged\_dict = merge\_dictionaries(dict1, dict2)

print("Merged dictionary using dictionary unpacking:", merged\_dict)

Both approaches will produce the same result. In this example, we have two dictionaries `dict1` and `dict2`, and we merge them into a new dictionary called `merged\_dict`. The resulting merged dictionary will contain all the key-value pairs from both `dict1` and `dict2`. If there are overlapping keys, the values from `dict2` will overwrite the values from `dict1`.

1. Write a Python program to convert key-values list to flat dictionary?

ANS: To convert a list of key-value pairs into a flat dictionary, you can iterate through the list and add each key-value pair to the dictionary. Here's a Python program to achieve this:

def list\_to\_flat\_dictionary(key\_value\_list):

flat\_dict = {}

for key, value in key\_value\_list:

flat\_dict[key] = value

return flat\_dict

# Test the function

if \_\_name\_\_ == "\_\_main\_\_":

key\_value\_list = [('a', 1), ('b', 2), ('c', 3), ('d', 4)]

flat\_dict = list\_to\_flat\_dictionary(key\_value\_list)

print("Flat dictionary:", flat\_dict)

In this program, the `list\_to\_flat\_dictionary` function takes a list of key-value pairs as input and converts it into a flat dictionary. It iterates through each pair and assigns the key as the dictionary key and the value as the corresponding value. The final result is a flat dictionary containing all the key-value pairs from the list.

Output:

Flat dictionary: {'a': 1, 'b': 2, 'c': 3, 'd': 4}

1. Write a Python program to insertion at the beginning in OrderedDict?

ANS: The `collections` module provides the `OrderedDict` data structure, which is a dictionary subclass that remembers the order in which items were inserted. To insert an item at the beginning of an `OrderedDict`, you can use the `OrderedDict.move\_to\_end()` method. Here's a Python program to demonstrate insertion at the beginning in an `OrderedDict`:

from collections import OrderedDict

def insert\_at\_beginning(ordered\_dict, key, value):

# Move the existing key to the end (last position) to make room for the new key-value pair

ordered\_dict.move\_to\_end(key, last=False)

# Insert the new key-value pair at the beginning

ordered\_dict[key] = value

# Test the program

if \_\_name\_\_ == "\_\_main\_\_":

ordered\_dict = OrderedDict([('a', 1), ('b', 2), ('c', 3)])

print("Original OrderedDict:", ordered\_dict)

insert\_at\_beginning(ordered\_dict, 'd', 4)

print("OrderedDict after insertion at the beginning:", ordered\_dict)

In this program, we define the `insert\_at\_beginning` function, which takes an `OrderedDict`, a `key`, and a `value` as input. The function first moves the existing key to the end of the `OrderedDict` using the `move\_to\_end()` method with `last=False`. Then, it inserts the new key-value pair at the beginning, and since `OrderedDict` maintains the order of insertion, the new pair will appear at the beginning.

Output:

Original OrderedDict: OrderedDict([('a', 1), ('b', 2), ('c', 3)])

OrderedDict after insertion at the beginning: OrderedDict([('d', 4), ('a', 1), ('b', 2), ('c', 3)])

1. Write a Python program to check order of character in string using OrderedDict()?

ANS: To check the order of characters in a string using `OrderedDict`, you can iterate through both the input string and the `OrderedDict` to verify if the characters appear in the same order. Here's a Python program that demonstrates how to do this:

from collections import OrderedDict

def check\_order\_of\_characters(input\_string, pattern):

# Create an OrderedDict to store the occurrence of characters in the input string

char\_occurrence = OrderedDict.fromkeys(input\_string, 0)

# Count the occurrence of characters in the input string

for char in input\_string:

char\_occurrence[char] += 1

# Iterate through the pattern and check if the characters appear in the same order in the input string

index = 0

for char in pattern:

if char in char\_occurrence and char\_occurrence[char] > 0:

char\_occurrence[char] -= 1

index += 1

else:

return False

# If the index reaches the length of the pattern, it means all characters were found in order

return index == len(pattern)

# Test the program

if \_\_name\_\_ == "\_\_main\_\_":

input\_string = "hello world"

pattern = "hlo"

result = check\_order\_of\_characters(input\_string, pattern)

if result:

print("The characters in the pattern appear in order in the input string.")

else:

print("The characters in the pattern do not appear in order in the input string.")

In this program, the `check\_order\_of\_characters` function takes two strings as input: `input\_string`, which is the original string, and `pattern`, which represents the pattern of characters to check for order. It uses an `OrderedDict` called `char\_occurrence` to count the occurrence of each character in the input string. Then, it iterates through the `pattern` and checks if the characters appear in the same order in the input string. If they do, the function returns `True`, indicating that the characters in the pattern appear in order in the input string; otherwise, it returns `False`.

1. Write a Python program to sort Python Dictionaries by Key or Value?

ANS: To sort Python dictionaries by key or value, you can use the `sorted()` function along with lambda functions or the `itemgetter()` function from the `operator` module. Both methods allow you to specify the sorting criteria.

Here's a Python program that demonstrates sorting dictionaries by key and value:

# Using lambda functions to sort by key and value

def sort\_dict\_by\_key(dictionary):

return {k: v for k, v in sorted(dictionary.items(), key=lambda item: item[0])}

def sort\_dict\_by\_value(dictionary):

return {k: v for k, v in sorted(dictionary.items(), key=lambda item: item[1])}

# Using itemgetter() to sort by key and value

from operator import itemgetter

def sort\_dict\_by\_key\_itemgetter(dictionary):

return {k: v for k, v in sorted(dictionary.items(), key=itemgetter(0))}

def sort\_dict\_by\_value\_itemgetter(dictionary):

return {k: v for k, v in sorted(dictionary.items(), key=itemgetter(1))}

# Test the program

if \_\_name\_\_ == "\_\_main\_\_":

my\_dict = {'apple': 5, 'banana': 2, 'orange': 8, 'kiwi': 3}

sorted\_by\_key = sort\_dict\_by\_key(my\_dict)

print("Sorted by key:", sorted\_by\_key)

sorted\_by\_value = sort\_dict\_by\_value(my\_dict)

print("Sorted by value:", sorted\_by\_value)

# Using itemgetter()

sorted\_by\_key\_itemgetter = sort\_dict\_by\_key\_itemgetter(my\_dict)

print("Sorted by key using itemgetter:", sorted\_by\_key\_itemgetter)

sorted\_by\_value\_itemgetter = sort\_dict\_by\_value\_itemgetter(my\_dict)

print("Sorted by value using itemgetter:", sorted\_by\_value\_itemgetter)

In this program, we define four functions to sort a dictionary by key and value using both lambda functions and `itemgetter()`:

i). `sort\_dict\_by\_key`: Sorts the dictionary by key using a lambda function for the sorting criteria.

ii). `sort\_dict\_by\_value`: Sorts the dictionary by value using a lambda function for the sorting criteria.

iii). `sort\_dict\_by\_key\_itemgetter`: Sorts the dictionary by key using `itemgetter(0)` as the sorting criteria.

iv). `sort\_dict\_by\_value\_itemgetter`: Sorts the dictionary by value using `itemgetter(1)` as the sorting criteria.

The program demonstrates both approaches and prints the sorted dictionaries for both key and value sorting.