1. Write a Python script to implement an in-place shuffle of a list.

```
import random

def in_place_shuffle(lst):
    for i in range(len(lst) - 1, 0, -1):
        j = random.randint(0, i)
        lst[i], lst[j] = lst[j], lst[i]

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

```

2. How would you extract alternating elements from a list to create two new lists?

```
def extract_alternating_elements(lst):
    list1 = lst[::2]  # Extract even-indexed elements
    list2 = lst[1::2]  # Extract odd-indexed elements
    return list1, list2

# Example usage
original_list = [1, 2, 3, 4, 5, 6]
list1, list2 = extract_alternating_elements(original_list)
print("List 1:", list1)
print("List 2:", list2)

```

3. Implement a Python script to check if a list is a subsequence of another list.

```
def is_subsequence(subseq, seq):
    sub_index = 0
    for item in seq:
        if item == subseq[sub_index]:
            sub_index += 1
            if sub_index == len(subseq):
                return True
    return False

# Example usage
sequence = [1, 2, 3, 4, 5]
subsequence = [2, 4]
print("Is subsequence:", is_subsequence(subsequence, sequence))

```

4. Create a Python script to perform a zig-zag merge of two lists into a single list.

```
def zigzag_merge(list1, list2):
    merged_list = []
    while list1 and list2:
        merged_list.append(list1.pop(0))
        merged_list.append(list2.pop())
    merged_list.extend(list1)
    merged_list.extend(list2)
    return merged_list

# Example usage
list1 = [1, 3, 5]
list2 = [2, 4, 6]
print("Zig-zag merged list:", zigzag_merge(list1, list2))

```

5. Write a Python script that returns all unique combinations of elements from a list that sum to a specific value.

```
from itertools import combinations

def unique_combinations(lst, target):
    result = []
    for r in range(1, len(lst) + 1):
        for comb in combinations(lst, r):
            if sum(comb) == target:
                result.append(comb)
    return result

# Example usage
my_list = [1, 2, 3, 4, 5]
target_sum = 5
print("Unique combinations:", unique_combinations(my_list, target_sum))

```

6. Given a tuple, write a Python script to add an element at a specific index.

```
def add_element_to_tuple(tup, index, element):
    return tup[:index] + (element,) + tup[index:]

# Example usage
original_tuple = (1, 2, 3, 4, 5)
index = 2
element = 10
new_tuple = add_element_to_tuple(original_tuple, index, element)
print("New tuple:", new_tuple)

```

7. Implement a script that returns a new tuple containing elements from a given tuple after modifying every nth element.

```
def modify_nth_elements(tup, n, modification):
    return tuple(modification(element) if (i + 1) % n == 0 else element for i, element ir

# Example usage
original_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
n = 3
modification = lambda x: x * 2
new_tuple = modify_nth_elements(original_tuple, n, modification)
print("New tuple:", new_tuple)

```

8. Write a Python script to reverse the contents of a tuple without converting the tuple into a list.

```
def reverse_tuple(tup):
    return tup[::-1]

# Example usage
original_tuple = (1, 2, 3, 4, 5)
reversed_tuple = reverse_tuple(original_tuple)
print("Reversed tuple:", reversed_tuple)

```

9. Create a script to find the longest consecutive sequence of integers in a tuple.

```
def longest_consecutive_sequence(tup):
    longest_sequence = ()
    current_sequence = ()
    for num in tup:
        if not current_sequence or num == current_sequence[-1] + 1:
            current_sequence += num,
        else:
            if len(current_sequence) > len(longest_sequence):
                longest_sequence = current_sequence
            current_sequence = num,
    return longest_sequence

# Example usage
original_tuple = (1, 2, 3, 5, 6, 7, 8, 10, 11, 12)
sequence = longest_consecutive_sequence(original_tuple)
print("Longest consecutive sequence:", sequence)

```

10. Write a Python script to create a tuple of the first and last elements of every other tuple in a list of tuples.

```
def first_last_elements(tuples_list):
    result = []
    for i, tup in enumerate(tuples_list):
        if i % 2 == 0:
            result.append((tup[0], tup[-1]))
    return tuple(result)

# Example usage
list_of_tuples = [(1, 2, 3), (4, 5, 6), (7, 8, 9), (10, 11, 12)]
result_tuple = first_last_elements(list_of_tuples)
print("Result tuple:", result_tuple)

```

11. Write a Python script to extract keys from a dictionary based on a subset of values.

```
def extract_keys_by_values(dictionary, subset_values):
    return [key for key, value in dictionary.items() if value in subset_values]

# Example usage
my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
subset_values = [2, 3]
result_keys = extract_keys_by_values(my_dict, subset_values)
print("Keys:", result_keys)

```

12. Implement a script that takes a dictionary and returns a new dictionary with keys and values inverted, where values are lists of keys from the original dictionary.

```
def invert_dictionary(dictionary):
    inverted_dict = {}
    for key, value in dictionary.items():
        if value not in inverted_dict:
            inverted_dict[value] = [key]
        else:
            inverted_dict[value].append(key)
    return inverted_dict

# Example usage
my_dict = {'a': 1, 'b': 2, 'c': 1, 'd': 3}
inverted_dict = invert_dictionary(my_dict)
print("Inverted dictionary:", inverted_dict)

```

13. Create a Python script to filter a dictionary by removing entries that do not satisfy a predicate function.

```
def filter_dictionary(dictionary, predicate):
    return {key: value for key, value in dictionary.items() if predicate(key, value)}

# Example usage
my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
predicate = lambda key, value: value % 2 == 0
filtered_dict = filter_dictionary(my_dict, predicate)
print("Filtered dictionary:", filtered_dict)

```

14. Implement a dictionary comprehension that partitions dictionary values into groups based on their modulus with a given number.

```
def partition_dictionary(dictionary, divisor):
    return {i: [value for value in dictionary.values() if value % divisor == i] for i in

# Example usage
my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6}
divisor = 3
partitioned_dict = partition_dictionary(my_dict, divisor)
print("Partitioned dictionary:", partitioned_dict)

```

15. Write a Python script to recursively merge nested dictionaries.

```
def merge_nested_dicts(dicts):
    result = {}
    for d in dicts:
        for key, value in d.items():
            if isinstance(value, dict):
                result[key] = merge_nested_dicts([result.get(key, {}), value])
            else:
                result[key] = value
    return result

# Example usage
nested_dicts = [{'a': {'b': 1}}, {'a': {'c': 2}}, {'a': {'d': 3}}]
merged_dict = merge_nested_dicts(nested_dicts)
print("Merged dictionary:", merged_dict)

```

16. Implement a script to find elements that are in either of two sets but not in their intersection.

```
def symmetric_difference(set1, set2):
    return (set1 | set2) - (set1 & set2)

# Example usage
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
result = symmetric_difference(set1, set2)
print("Symmetric difference:", result)

```

17. Write a Python script that determines if a given set is a proper subset of another set without using the subset method.

```
def is_proper_subset(set1, set2):
    return set1 < set2 and set1 != set2

# Example usage
set1 = {1, 2, 3}
set2 = {1, 2, 3, 4, 5}
print("Is set1 a proper subset of set2:", is_proper_subset(set1, set2))

```

18. Create a script to generate all possible pairs (as tuples) from two sets, excluding pairs with duplicate items.

```
def generate_pairs(set1, set2):
    return {(x, y) for x in set1 for y in set2 if x != y}

# Example usage
set1 = {1, 2, 3}
set2 = {'a', 'b', 'c'}
result = generate_pairs(set1, set2)
print("Generated pairs:", result)

```

19. Implement a Python script to simulate the set difference operation using list comprehensions.

```
def set_difference(set1, set2):
    return {x for x in set1 if x not in set2}

# Example usage
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
result = set_difference(set1, set2)
print("Set difference:", result)

```

20. Write a script to find the minimum set of subsets from a given set such that each subset has an element-wise property (like being divisible by a specific number).

```
def find_minimum_subsets(input_set, property_func):
    result = []
    remaining_set = input_set.copy()
    while remaining_set:
        subset = {x for x in remaining_set if property_func(x)}
        result.append(subset)
        remaining_set -= subset
    return result

# Example usage
input_set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
property_func = lambda x: x % 2 == 0  # Example property: even numbers
minimum_subsets = find_minimum_subsets(input_set, property_func)
print("Minimum subsets:", minimum_subsets)

```

21. Write a Python script to interleave multiple lists of different lengths.

```
def interleave_lists(*lists):
    max_length = max(len(lst) for lst in lists)
    interleaved = []
    for i in range(max_length):
        for lst in lists:
            if i < len(lst):
                interleaved.append(lst[i])
    return interleaved

# Example usage
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c', 'd']
list3 = [True, False]
result = interleave_lists(list1, list2, list3)
print("Interleaved list:", result)

```

22. Implement a method to convert a list of strings into a dictionary where the keys are strings and values are their corresponding lengths.

```
def strings_to_dictionary(strings):
    return {s: len(s) for s in strings}

# Example usage
strings = ['apple', 'banana', 'cherry']
result_dict = strings_to_dictionary(strings)
print("Resulting dictionary:", result_dict)

```

23. Create a script that returns the nth Fibonacci number, where n is provided by the user, using a list to store intermediate results.

```
def fibonacci(n):
    fib = [0, 1]
    for i in range(2, n + 1):
        fib.append(fib[i - 1] + fib[i - 2])
    return fib[n]

# Example usage
n = int(input("Enter the value of n: "))
result = fibonacci(n)
print("The {}th Fibonacci number is:".format(n), result)

```

24. Develop a Python script to find all pairs in a list whose sum is equal to a given number without using extra space.

```
def find_pairs_with_sum(lst, target):
    pairs = []
    seen = set()
    for num in lst:
        complement = target - num
        if complement in seen:
            pairs.append((num, complement))
        seen.add(num)
    return pairs

# Example usage
numbers = [2, 4, 3, 5, 7, 8, 9]
target_sum = 7
result_pairs = find_pairs_with_sum(numbers, target_sum)
print("Pairs with sum {}: {}".format(target_sum, result_pairs))

```

25. Write a script to cyclically rotate a list left by k elements.

```
def rotate_left(lst, k):
    k %= len(lst)
    return lst[k:] + lst[:k]

# Example usage
my_list = [1, 2, 3, 4, 5]
k = 2
rotated_list = rotate_left(my_list, k)
print("Rotated list:", rotated_list)

```

26. Write a Python script to calculate the sum of tuple elements, where tuples are nested within a larger tuple.

```
def sum_nested_tuples(tup):
    total = 0
    for item in tup:
        if isinstance(item, tuple):
            total += sum(item)
        else:
            total += item
    return total

# Example usage
nested_tuple = ((1, 2), (3, 4), 5, (6, (7, 8)))
result = sum_nested_tuples(nested_tuple)
print("Sum of nested tuple elements:", result)

```

27. Implement a method to convert a flat list into a tuple of tuples, where each tuple contains consecutive n elements from the list.

```
def list_to_tuple_of_tuples(lst, n):
    return tuple(tuple(lst[i:i+n]) for i in range(0, len(lst), n))

# Example usage
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
n = 3
result = list_to_tuple_of_tuples(my_list, n)
print("Resulting tuple of tuples:", result)

```

28. Create a Python script that finds the tuple with the highest sum of elements in a list of tuples.

```
def tuple_with_highest_sum(tuple_list):
    return max(tuple_list, key=sum)

# Example usage
tuple_list = [(1, 2), (3, 4, 5), (6, 7, 8, 9)]
result = tuple_with_highest_sum(tuple_list)
print("Tuple with the highest sum:", result)

```

29. Develop a script to determine if two tuples contain any common elements efficiently.

```
def has_common_elements(tup1, tup2):
    return any(item in tup1 for item in tup2)

# Example usage
tuple1 = (1, 2, 3)
tuple2 = (3, 4, 5)
result = has_common_elements(tuple1, tuple2)
print("Do the tuples have common elements?", result)

```

30. Write a script that replaces every element in a tuple with the cumulative sum up to that element.

```
def cumulative_sum_tuple(tup):
    cumulative_sum = 0
    result = []
    for item in tup:
        cumulative_sum += item
        result.append(cumulative_sum)
    return tuple(result)

# Example usage
my_tuple = (1, 2, 3, 4, 5)
result = cumulative_sum_tuple(my_tuple)
print("Cumulative sum tuple:", result)

```

31. Write a Python script to count the frequency of each value in a dictionary where the values are lists.

```
def count_values_frequency(dictionary):
    frequency = {}
    for key, values in dictionary.items():
        for value in values:
            frequency[value] = frequency.get(value, 0) + 1
    return frequency

# Example usage
my_dict = {'a': [1, 2, 3], 'b': [2, 3, 4], 'c': [3, 4, 5]}
result = count_values_frequency(my_dict)
print("Frequency of values:", result)

```

32. Implement a method to create a dictionary from two lists without using the zip function.

```
def create_dictionary(keys, values):
    if len(keys) != len(values):
        raise ValueError("Length of keys and values must be equal")
    result = {}
    for i in range(len(keys)):
        result[keys[i]] = values[i]
    return result

# Example usage
keys = ['a', 'b', 'c']
values = [1, 2, 3]
result = create_dictionary(keys, values)
print("Resulting dictionary:", result)

```

33. Create a Python script to find keys with the highest value in a dictionary.

```
def keys_with_highest_value(dictionary):
    max_value = max(dictionary.values())
    return [key for key, value in dictionary.items() if value == max_value]

# Example usage
my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 3}
result = keys_with_highest_value(my_dict)
print("Keys with highest value:", result)

```

34. Develop a script that combines several dictionaries into one by summing the values of common keys.

```
def combine_dictionaries(*dicts):
    result = {}
    for d in dicts:
        for key, value in d.items():
            result[key] = result.get(key, 0) + value
    return result

# Example usage
dict1 = {'a': 1, 'b': 2, 'c': 3}
dict2 = {'b': 3, 'c': 4, 'd': 5}
dict3 = {'c': 5, 'd': 6, 'e': 7}
result = combine_dictionaries(dict1, dict2, dict3)
print("Combined dictionary:", result)

```

35. Write a Python script to sort a dictionary by the length of its values.

```
def sort_dict_by_values_length(dictionary):
    return dict(sorted(dictionary.items(), key=lambda x: len(x[1])))

# Example usage
my_dict = {'a': [1, 2, 3], 'b': [4, 5], 'c': [6, 7, 8, 9]}
result = sort_dict_by_values_length(my_dict)
print("Sorted dictionary by values length:", result)

```

36. Implement a script to find the union of three or more sets without using union method.

```
def union_sets(*sets):
    result = set()
    for s in sets:
        result |= s
    return result

# Example usage
set1 = {1, 2, 3}
set2 = {3, 4, 5}
set3 = {5, 6, 7}
result = union_sets(set1, set2, set3)
print("Union of sets:", result)

```

37. Write a Python script that checks if a set is disjoint from multiple other sets.

```
def is_disjoint(set1, *sets):
    for s in sets:
        if not set1.isdisjoint(s):
            return False
    return True

# Example usage
set1 = {1, 2, 3}
set2 = {4, 5, 6}
set3 = {7, 8, 9}
set4 = {3, 10, 11}
result = is_disjoint(set1, set2, set3)
print("Are set1 disjoint from set2 and set3?", result)
result = is_disjoint(set1, set3, set4)
print("Are set1 disjoint from set3 and set4?", result)

```

38. Create a script that returns the symmetric difference between a list of sets.

```
def symmetric_difference_sets(sets):
    result = set()
    for s in sets:
        result ^= s
    return result

# Example usage
sets_list = [{1, 2, 3}, {3, 4, 5}, {5, 6, 7}]
result = symmetric_difference_sets(sets_list)
print("Symmetric difference of sets:", result)

```

39. Develop a method to remove a random element from a set without using the pop method.

```
1 import random
2
```

40. Write a script to find all subsets of a set that meet a certain condition (e.g., subsets whose elements sum to a prime number).