Unsupported cell type. Double-click to inspect/edit the content.

```
#Q1) code to reverse a string :
def reverse_string(s):
    return s[::-1]

# Example usage
input_string = "Hello, Jupyter!"
reversed_string = reverse_string(input_string)

# Print the result
print("Original string:", input_string)
print("Reversed string:", reversed_string)
```

```
Original string: Hello, Jupyter!
Reversed string: !retypuJ ,olleH
```

```
#2) code to count the num
ber 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

# Example usage
input_string = "Hello, Jupyter!"
vowel_count = count_vowels(input_string)

# Print the result
print("Original string:", input_string)
print("Number of vowels:", vowel_count)
```

```
Original string: Hello, Jupyter!
Number of vowels: 4
```

```
#3 Code to check if a given string is a palindrome or not:
def is_palindrome(s):
    # Remove spaces and convert to lowercase for uniformity
    s = s.replace(" ", "").lower()
    return s == s[::-1]

# Example usage
input_string = "A man a plan a canal Panama"
palindrome_check = is_palindrome(input_string)

# Print the result
print("Original string:", input_string)
print("Is palindrome:", palindrome_check)
```

```
Original string: A man a plan a canal Panama
Is palindrome: True
```

```
#4  Code to check if two given strings are anagrams of each other:

def are_anagrams(str1, str2):
    # Remove spaces and convert to lowercase for uniformity
    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()
    return sorted(str1) == sorted(str2)

# Example usage
str1 = "Listen"
str2 = "Silent"
anagram_check = are_anagrams(str1, str2)

# Print the result
print("String 1:", str1)
print("String 2:", str2)
print("Are anagrams:", anagram_check)
```

```
String 1: Listen
String 2: Silent
Are anagrams: True
```

```
#5. Code to find all occurrences of a given substring within another string:
def find_substring_occurrences(s, sub):
    start = 0
    occurrences = []
    while True:
        start = s.find(sub, start)
        if start == -1:
            break
        occurrences.append(start)
        start += len(sub)  # Use start += 1 to find overlapping matches
    return occurrences

# Example usage
main_string = "Hello, hello, hello!"
substring = "hello"
occurrences = find_substring_occurrences(main_string.lower(), substring.lower())

# Print the result
print("Main string:", main_string)
print("Substring:", substring)
print("Occurrences at indices:", occurrences)
```

```
Main string: Hello, hello, hello!
Substring: hello
Occurrences at indices: [0, 7, 14]
```

```
#6. Code to perform basic string compression using the counts of repeated characters:
def compress_string(s):
    if not s:
        return ""

    compressed = []
    count = 1
    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
    compressed.append(s[-1] + str(count))

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

# Example usage
input_string = "aabcccccaaa"
compressed = compress_string(input_string)

# Print the result
print("Original string:", input_string)
print("Compressed string:", compressed)
```

```
Original string: aabcccccaaa
Compressed string: a2b1c5a3
```

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

# Example usage
input_string = "abcdefg"
unique_check = has_unique_characters(input_string)

# Print the result
print("Original string:", input_string)
print("Has all unique characters:", unique_check)
```

```
Original string: abcdefg
Has all unique characters: True
```

```
#8. Code to convert a given string to uppercase or lowercase:
def convert_case(s):
    return s.upper(), s.lower()

input_string = "Hello, World!"
uppercase_string, lowercase_string = convert_case(input_string)

# Print the result
print("Original string:", input_string)
print("Uppercase string:", uppercase_string)
print("Lowercase string:", lowercase_string)
```

```
Original string: Hello, World!
Uppercase string: HELLO, WORLD!
Lowercase string: hello, world!
```

```
#9. Code to count the number of words in a string:

def count_words(s):
    words = s.split()
    return len(words)

# Example usage
input_string = "Hello, world! Welcome to Jupyter."
word_count = count_words(input_string)

# Print the result
print("Original string:", input_string)
print("Number of words:", word_count)
```

```
Original string: Hello, world! Welcome to Jupyter.
Number of words: 5
```

```
#10. Code to concatenate two strings without using the + operator:
def concatenate_strings(str1, str2):
    return "{}{}".format(str1, str2)

# Example usage
str1 = "Hello"
str2 = "World"
concatenated_string = concatenate_strings(str1, str2)

# Print the result
print("String 1:", str1)
print("String 2:", str2)
print("Concatenated string:", concatenated_string)
```

```
String 1: Hello
String 2: World
Concatenated string: HelloWorld
```

```
#11. Code to remove all occurrences of a specific element from a list:

def remove_all_occurrences(lst, element):
    return [x for x in lst if x != element]

# Example usage
input_list = [1, 2, 3, 4, 2, 2, 5, 6]
element_to_remove = 2
updated_list = remove_all_occurrences(input_list, element_to_remove)

# Print the result
print("Original list:", input_list)
print("Element to remove:", element_to_remove)
print("Updated list:", updated_list)
```

```
Original list: [1, 2, 3, 4, 2, 2, 5, 6]
Element to remove: 2
Updated list: [1, 3, 4, 5, 6]
```

```
#12. Code to find the second largest number in a given list of integers:
def second_largest(nums):
    unique_nums = list(set(nums))  # Remove duplicates
    unique_nums.sort()
    return unique_nums[-2] if len(unique_nums) >= 2 else None

# Example usage
input_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
second_largest_number = second_largest(input_list)

# Print the result
print("Original list:", input_list)
print("Second largest number:", second_largest_number)
```

```
Original list: [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
Second largest number: 6
```

```
#13. 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 elem in lst:
        if elem in counts:
            counts[elem] += 1
        else:
            counts[elem] = 1
    return counts

# Example usage
input_list = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
element_counts = count_occurrences(input_list)

# Print the result
print("Original list:", input_list)
print("Element counts:", element_counts)
```

```
Original list: [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
Element counts: {1: 1, 2: 2, 3: 3, 4: 4}
```

```
#14. 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:
        lst[left], lst[right] = lst[right], lst[left]
        left += 1
        right -= 1
    return lst

# Example usage
input_list = [1, 2, 3, 4, 5]
reversed_list = reverse_list(input_list.copy())

# Print the result
print("Original list:", input_list)
print("Reversed list:", reversed_list)
```

```
Original list: [1, 2, 3, 4, 5]
Reversed list: [5, 4, 3, 2, 1]
```

```
#15. Code to find and remove duplicates from a list while preserving the original order of elements:
def remove_duplicates(lst):
    seen = set()
    result = []
    for item in lst:
        if item not in seen:
            seen.add(item)
            result.append(item)
    return result

# Example usage
input_list = [1, 2, 2, 3, 4, 4, 5]
deduplicated_list = remove_duplicates(input_list)

# Print the result
print("Original list:", input_list)
print("List after removing duplicates:", deduplicated_list)
```

```
Original list: [1, 2, 2, 3, 4, 4, 5]
List after removing duplicates: [1, 2, 3, 4, 5]
```

```
#16. 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

# Example usage
input_list = [1, 2, 3, 4, 5]
sorted_check = is_sorted(input_list)

# Print the result
print("Original list:", input_list)
print("Is sorted:", sorted_check)
```

```
Original list: [1, 2, 3, 4, 5]
Is sorted: True
```

```
#17. Code to merge two sorted lists into a single sorted list:
def merge_sorted_lists(list1, list2):
    merged_list = []
    i, j = 0, 0
    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
    merged_list.extend(list1[i:])
    merged_list.extend(list2[j:])
    return merged_list

# Example usage
list1 = [1, 3, 5, 7]
list2 = [2, 4, 6, 8]
merged_list = merge_sorted_lists(list1, list2)

# Print the result
print("List 1:", list1)
print("List 2:", list2)
print("Merged list:", merged_list)
```

```
List 1: [1, 3, 5, 7]
List 2: [2, 4, 6, 8]
Merged list: [1, 2, 3, 4, 5, 6, 7, 8]
```

```
#18. Code to find the intersection of two given lists:
def list_intersection(list1, list2):
    return list(set(list1) & set(list2))

# Example usage
list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
intersection = list_intersection(list1, list2)

# Print the result
print("List 1:", list1)
print("List 2:", list2)
print("Intersection:", intersection)
```

```
List 1: [1, 2, 3, 4]
List 2: [3, 4, 5, 6]
Intersection: [3, 4]
```

```
#19. Code to find the union of two lists without duplicates:
def list_union(list1, list2):
    return list(set(list1) | set(list2))

# Example usage
list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
union = list_union(list1, list2)

# Print the result
print("List 1:", list1)
print("List 2:", list2)
print("Union:", union)
```

```
List 1: [1, 2, 3, 4]
List 2: [3, 4, 5, 6]
Union: [1, 2, 3, 4, 5, 6]
```

```
#20. Code to shuffle a given list randomly without using any built-in shuffle functions:
import random

# Function to shuffle a list
def shuffle_list(lst):
    shuffled = lst[:]
    length = len(shuffled)
    for i in range(length):
        swap_idx = random.randint(0, length - 1)
        shuffled[i], shuffled[swap_idx] = shuffled[swap_idx], shuffled[i]
    return shuffled

# Example usage
input_list = [1, 2, 3, 4, 5]
shuffled_list = shuffle_list(input_list)

# Print the result
print("Original list:", input_list)
print("Shuffled list:", shuffled_list)
```

```
Original list: [1, 2, 3, 4, 5]
Shuffled list: [3, 1, 4, 2, 5]
```

```
#21. 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):
    return tuple(set(tuple1) & set(tuple2))

# Example usage
tuple1 = (1, 2, 3, 4)
tuple2 = (3, 4, 5, 6)
common_tuple = common_elements(tuple1, tuple2)

# Print the result
print("Tuple 1:", tuple1)
print("Tuple 2:", tuple2)
print("Common elements tuple:", common_tuple)
```

```
Tuple 1: (1, 2, 3, 4)
Tuple 2: (3, 4, 5, 6)
Common elements tuple: (3, 4)
```

```
#22. Code that prompts the user to enter two sets of integers separated by commas, then prints the intersection of these two sets:
def intersection_from_input():
    set1 = set(map(int, input("Enter the first set of integers, separated by commas: ").split(',')))
    set2 = set(map(int, input("Enter the second set of integers, separated by commas: ").split(',')))
    intersection = set1 & set2
    return intersection

# Example usage
intersection = intersection_from_input()

# Print the result
print("Intersection of the two sets:", intersection)
```

```
#23. Code to concatenate two tuples. The function should take two tuples as input and return a new tuple containing all elements from bo
def concatenate_tuples(tuple1, tuple2):
    return tuple1 + tuple2

# Example usage
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
concatenated_tuple = concatenate_tuples(tuple1, tuple2)

# Print the result
print("Tuple 1:", tuple1)
print("Tuple 2:", tuple2)
print("Concatenated tuple:", concatenated_tuple)
```

```
Tuple 1: (1, 2, 3)
Tuple 2: (4, 5, 6)
Concatenated tuple: (1, 2, 3, 4, 5, 6)
```

```
#24. Code to prompt the user to input two sets of strings and print the elements that are present in the first set but not in the second
def difference_of_sets():
    set1 = set(input("Enter the first set of strings, separated by commas: ").split(','))
    set2 = set(input("Enter the second set of strings, separated by commas: ").split(','))
    difference = set1 - set2
    return difference

# Example usage
difference = difference_of_sets()

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


#25. Code that takes a tuple and two integers as input, and returns a new tuple containing elements from the original tuple within the s
def extract_range_from_tuple(tpl, start, end):
    return tpl[start:end]

# Example usage
tuple_input = (1, 2, 3, 4, 5, 6, 7, 8)
start_index = 2
end_index = 5
extracted_tuple = extract_range_from_tuple(tuple_input, start_index, end_index)

# Print the result
print("Original tuple:", tuple_input)
print("Extracted tuple:", extracted_tuple)
```

```
Original tuple: (1, 2, 3, 4, 5, 6, 7, 8)
Extracted tuple: (3, 4, 5)
```

```
#26. Code to prompt the user to input two sets of characters and print the union of these two sets:
def union_of_char_sets():
    set1 = set(input("Enter the first set of characters, separated by commas: ").split(','))
    set2 = set(input("Enter the second set of characters, separated by commas: ").split(','))
    union = set1 | set2
    return union

# Example usage
union = union_of_char_sets()

# Print the result
print("Union of the two sets:", union)


#27. Code that takes a tuple of integers as input and returns the maximum and minimum values from the tuple using tuple unpacking:
def max_min_tuple(tpl):
    return max(tpl), min(tpl)

# Example usage
tuple_input = (5, 3, 9, 1, 6)
max_value, min_value = max_min_tuple(tuple_input)

# Print the result
print("Original tuple:", tuple_input)
print("Maximum value:", max_value)
print("Minimum value:", min_value)
```

```
Original tuple: (5, 3, 9, 1, 6)
Maximum value: 9
Minimum value: 1
```

```
#28. Code to define two sets of integers and print the union, intersection, and difference of these two sets:
def set_operations(set1, set2):
    union = set1 | set2
    intersection = set1 & set2
    difference = set1 - set2
    return union, intersection, difference

# Example usage
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
union, intersection, difference = set_operations(set1, set2)

# Print the result
print("Set 1:", set1)
print("Set 2:", set2)
print("Union:", union)
print("Intersection:", intersection)
print("Difference (Set1 - Set2):", difference)
```

```
Set 1: {1, 2, 3, 4}
Set 2: {3, 4, 5, 6}
Union: {1, 2, 3, 4, 5, 6}
Intersection: {3, 4}
Difference (Set1 - Set2): {1, 2}
```

```
#29. Code that takes a tuple and an element as input and returns the count of occurrences of the given element in the tuple:
def count_element_in_tuple(tpl, element):
    return tpl.count(element)

# Example usage
tuple_input = (1, 2, 3, 2, 4, 2, 5)
element = 2
count = count_element_in_tuple(tuple_input, element)

# Print the result
print("Original tuple:", tuple_input)
print("Element to count:", element)
print("Count of element:", count)
```

```
Original tuple: (1, 2, 3, 2, 4, 2, 5)
Element to count: 2
Count of element: 3
```

```
#30. Code to prompt the user to input two sets of strings and print the symmetric difference of these two sets:
def symmetric_difference_of_sets():
    set1 = set(input("Enter the first set of strings, separated by commas: ").split(','))
    set2 = set(input("Enter the second set of strings, separated by commas: ").split(','))
    sym_diff = set1 ^ set2
    return sym_diff

# Example usage
sym_diff = symmetric_difference_of_sets()

# Print the result
print("Symmetric difference of the two sets:", sym_diff)
```

```
#31. Code that takes a list of words as input and returns a dictionary where the keys are unique words and the values are the frequencie
def word_frequencies(words):
    frequency_dict = {}
    for word in words:
        if word in frequency_dict:
            frequency_dict[word] += 1
        else:
            frequency_dict[word] = 1
    return frequency_dict

# Example usage
words_list = ["apple", "banana", "apple", "orange", "banana", "apple"]
frequencies = word_frequencies(words_list)

# Print the result
print("Original list:", words_list)
print("Word frequencies:", frequencies)
```

```
Original list: ['apple', 'banana', 'apple', 'orange', 'banana', 'apple']
Word frequencies: {'apple': 3, 'banana': 2, 'orange': 1}
```

```
#32. Code that takes two dictionaries as input and merges them into a single dictionary. If there are common keys, the values should be
def merge_dictionaries(dict1, dict2):
    merged_dict = dict1.copy()
    for key, value in dict2.items():
        if key in merged_dict:
            merged_dict[key] += value
        else:
            merged_dict[key] = value
    return merged_dict

# Example usage
dict1 = {'a': 1, 'b': 2, 'c': 3}
dict2 = {'b': 3, 'c': 4, 'd': 5}
merged_dict = merge_dictionaries(dict1, dict2)

# Print the result
print("Dictionary 1:", dict1)
print("Dictionary 2:", dict2)
print("Merged dictionary:", merged_dict)
```

```
Dictionary 1: {'a': 1, 'b': 2, 'c': 3}
Dictionary 2: {'b': 3, 'c': 4, 'd': 5}
```

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

```
#33. 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 c
def get_nested_value(nested_dict, keys):
    current_level = nested_dict
    for key in keys:
        if key in current_level:
            current_level = current_level[key]
        else:
            return None
    return current_level

# Example usage
nested_dict = {'a': {'b': {'c': 42}}}
keys = ['a', 'b', 'c']
value = get_nested_value(nested_dict, keys)

# Print the result
print("Nested dictionary:", nested_dict)
print("Keys:", keys)
print("Value:", value)
```

```
Nested dictionary: {'a': {'b': {'c': 42}}}
Keys: ['a', 'b', 'c']
Value: 42
```

```
#34. Code that takes a dictionary as input and returns a sorted version of it based on the values. You can specify whether to sort in as
def sort_dict_by_values(d, ascending=True):
    return dict(sorted(d.items(), key=lambda item: item[1], reverse=not ascending))

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

# Print the result
print("Original dictionary:", input_dict)
print("Sorted dictionary (ascending):", sorted_dict_asc)
print("Sorted dictionary (descending):", sorted_dict_desc)
```

```
Original dictionary: {'a': 3, 'b': 1, 'c': 2}
Sorted dictionary (ascending): {'b': 1, 'c': 2, 'a': 3}
Sorted dictionary (descending): {'a': 3, 'c': 2, 'b': 1}
```

```
#35. Code that inverts a dictionary, swapping keys and values. Ensure that the inverted dictionary handles cases where multiple keys have
def invert_dict(d):
    inverted_dict = {}
    for key, value in d.items():
        if value in inverted_dict:
            inverted_dict[value].append(key)
        else:
            inverted_dict[value] = [key]
    return inverted_dict
```

-