

Name – Gathram Kesava Murthy

Registered mail id- kesavagathram.murthy@outlook.com

Course Name – Master Generative AI: Data Science Course

Assignment Name – Python Advanced Data Structure Assignment

Submission Date – 25th October 2024

Drive Link –

<https://colab.research.google.com/drive/1X9Yc4n0VEQFNFYCCEIKdA0GqseVTPIBU?usp=sharing>

Github Link –

<https://github.com/GKesavamurthy1241/Python-Advanced-Data-Structure-Assignment>

Name – Gathram Kesava Murthy

Registered mail id- kesavagathram.murthy@outlook.com

Course Name – Master Generative AI: Data Science Course

Assignment Name – Python Advanced Data Structure Assignment

Submission Date – 25th October 2024

Drive Link -

<https://colab.research.google.com/drive/1X9Yc4n0VEQFNFYCCEIKdA0GqseVTPIBU?usp=sharing>

Github Link -

<https://github.com/GKesavamurthy1241/Python-Advanced-Data-Structure-Assignment>

✓ 1. Write a code to reverse a string

```
def reverse_string(s):  
    return s[::-1]  
  
original_string = input()  
reversed_string = reverse_string(original_string)  
print(reversed_string)
```

↔ PW Skills
sllikS WP

✓ 2. Write a code to count the number 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

input_string = input()
vowel_count = count_vowels(input_string)
print(f"Number of vowels in '{input_string}': {vowel_count}")
```

⇒ random
Number of vowels in 'random': 2

✓ 3. Write a code to check if a given string is a palindrome or not

```
def is_palindrome(s):
    cleaned_string = ''.join(s.split()).lower()
    return cleaned_string == cleaned_string[::-1]

input_string = input()
if is_palindrome(input_string):
    print(f'"{input_string}" is a palindrome.')
else:
    print(f'"{input_string}" is not a palindrome.')
```

⇒ Madam
"Madam" is a palindrome.

✓ 4. Write a code to check if two given strings are anagrams of each other

```
def are_anagrams(s1, s2):
    s1 = ''.join(s1.split()).lower()
    s2 = ''.join(s2.split()).lower()
    return sorted(s1) == sorted(s2)

string1 = input()
```

```

string2 = input()
if are_anagrams(string1, string2):
    print(f'"{string1}" and "{string2}" are anagrams.')
else:
    print(f'"{string1}" and "{string2}" are not anagrams.')

```



```

Mam
Mam
"Mam" and "Mam" are anagrams.

```

5. Write a code to find all occurrences of a given substring within another string

```

def find_substring_occurrences(s, substring):
    indices = []
    start = 0
    while start < len(s):
        pos = s.find(substring, start)
        if pos != -1:
            indices.append(pos)
            start = pos + 1
        else:
            break
    return indices
text = input()
substring = input()
occurrences = find_substring_occurrences(text, substring)
print(f"Occurrences of '{substring}' in '{text}': {occurrences}")

```



```

PW Skills
PW
Occurrences of 'PW' in 'PW Skills': [0]

```

6. Write a 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))
    return ''.join(compressed)

```

```

input_string = input()
compressed_string = compress_string(input_string)
print(f"Compressed string: {compressed_string}")

```

```

⇒ aaaaabbbbbbbcccccdefghgh
   Compressed string: a5b6c4d1e1f1g1h1g1h1

```

✓ 7. Write a code to determine if a string has all unique characters

```

def has_unique_characters(s):
    seen_characters = set()
    for char in s:
        if char in seen_characters:
            return False
        seen_characters.add(char)
    return True

input_string = input()
if has_unique_characters(input_string):
    print(f'"{input_string}" has all unique characters.')
else:
    print(f'"{input_string}" does not have all unique characters.')

```

```

⇒ abcdefga
   "abcdefga" does not have all unique characters.

```

✓ 8. 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()
    else:
        return s.lower()

input_string = input()

```

```
uppercase_string = convert_case(input_string, to_upper=True)
print(f"Uppercase: {uppercase_string}")
lowercase_string = convert_case(input_string, to_upper=False)
print(f"Lowercase: {lowercase_string}")
```

```
⇒ Hello World
   Uppercase: HELLO WORLD
   Lowercase: hello world
```

✓ 9. Write a code to count the number of words in a string

```
def count_words(s):
    words = s.split()
    return len(words)
input_string = input()
word_count = count_words(input_string)
print(f"Number of words in the string: {word_count}")
```

```
⇒ He is looking outside the home through window
   Number of words in the string: 8
```

✓ 10. Write a code to concatenate two strings without using the + operator

```
def concatenate_strings(s1, s2):
    return ''.join([s1, s2])
string1 = input()
string2 = input()
result = concatenate_strings(string1, string2)
print(f"Concatenated string: {result}")
```

```
⇒ Kesava
   Murthy
   Concatenated string: KesavaMurthy
```

✓ 11. Write a code to remove all occurrences of a specific element from a list

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

```
input_list = eval(input())
element_to_remove = int(input())
result = remove_occurrences(input_list, element_to_remove)
print(f"List after removing {element_to_remove}: {result}")
```

```
⇒ [1,2,3,4,5,2]
2
List after removing 2: [1, 3, 4, 5]
```

✓ 12. Implement a code to find the second largest number in a given list of integers

```
def find_second_largest(lst):
    unique_numbers = set(lst)
    if len(unique_numbers) < 2:
        return None
    sorted_numbers = sorted(unique_numbers, reverse=True)
    return sorted_numbers[1]
input_list = eval(input())
second_largest = find_second_largest(input_list)
if second_largest is not None:
    print(f"The second largest number is: {second_largest}")
else:
    print("There are not enough unique numbers to determine the second largest.")
```

```
⇒ [1,2,3,4,5,6,7,8,1,2]
The second largest number is: 7
```

✓ 13. 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):
    count_dict = {}
    for element in lst:
        if element in count_dict:
            count_dict[element] += 1
        else:
            count_dict[element] = 1
    return count_dict
input_list = eval(input())
```

```
occurrences = count_occurrences(input_list)
print(f"Occurrences of each element: {occurrences}")
```

```
→ [1,2,3,4,5,6,1,2,3,4]
Occurrences of each element: {1: 2, 2: 2, 3: 2, 4: 2, 5: 1, 6: 1}
```

✓ 14. Write a code to reverse a list in-place without using any built-in reverse functions

```
def reverse_list_in_place(lst):
    left = 0
    right = len(lst) - 1
    while left < right:
        lst[left], lst[right] = lst[right], lst[left]
        left += 1
        right -= 1
input_list = eval(input("Enter a list of elements : "))
reverse_list_in_place(input_list)
print(f"Reversed list: {input_list}")
```

```
→ Enter a list of elements : [1,2,3,4,5,6,7]
Reversed list: [7, 6, 5, 4, 3, 2, 1]
```

✓ 15. Implement a code to find and remove duplicates from a list while preserving the original order of elements

```
def remove_duplicates_preserve_order(lst):
    seen = set()
    result = []
    for element in lst:
        if element not in seen:
            seen.add(element)
            result.append(element)
    return result
input_list = eval(input("Enter a list of elements : "))
unique_list = remove_duplicates_preserve_order(input_list)
print(f"List after removing duplicates: {unique_list}")
```

```
→ Enter a list of elements : [1,2,3,4,5,6,7,1,2,3,4]
List after removing duplicates: [1, 2, 3, 4, 5, 6, 7]
```


✓ 16. 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) <= 1:
        return True
    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
input_list = eval(input("Enter a list of elements :"))
if is_sorted(input_list):
    print("The list is sorted.")
else:
    print("The list is not sorted.")
```

➞ Enter a list of elements :[1,2,3,4,5,6,7,8,9]
The list is sorted.

✓ 17. 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
    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
    while i < len(list1):
        merged_list.append(list1[i])
        i += 1
    while j < len(list2):
        merged_list.append(list2[j])
        j += 1
    return merged_list
list1 = eval(input("Enter the first sorted list : "))
list2 = eval(input("Enter the second sorted list : "))
merged = merge_sorted_lists(list1, list2)
print(f"Merged sorted list: {merged}")
```

```
➞ Enter the first sorted list : [1,2,3,4]
Enter the second sorted list : [5,6,7]
Merged sorted list: [1, 2, 3, 4, 5, 6, 7]
```

18. Implement a code to find the intersection of two given lists

```
def intersection_of_lists(list1, list2):
    set1 = set(list1)
    set2 = set(list2)
    return list(set1.intersection(set2))
list1 = eval(input("Enter the first list : "))
list2 = eval(input("Enter the second list : "))
intersection = intersection_of_lists(list1, list2)
print(f"Intersection of the two lists: {intersection}")
```

```
➞ Enter the first list : [1,2,3,4,5,6]
Enter the second list : [4,5,6,7,8,9,1]
Intersection of the two lists: [1, 4, 5, 6]
```

19. Create a code to find the union of two lists without duplicates

```
def union_of_lists(list1, list2):
    union_set = set(list1)
    union_set.update(list2)
    return list(union_set)
list1 = eval(input("Enter the first list : "))
list2 = eval(input("Enter the second list : "))
union = union_of_lists(list1, list2)
print(f"Union of the two lists without duplicates: {union}")
```

```
➞ Enter the first list : [1,2,3,4,5]
Enter the second list : [6,7,8,9,0]
Union of the two lists without duplicates: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

20. Write a code to shuffle a given list randomly without using any built-in shuffle functions

```
import random
def shuffle_list(lst):
    shuffled_list = lst[:]
    n = len(shuffled_list)
    for i in range(n - 1, 0, -1):
        j = random.randint(0, i)
        shuffled_list[i], shuffled_list[j] = shuffled_list[j], shuffled_list[i]
    return shuffled_list
input_list = eval(input("Enter a list of elements : "))
shuffled = shuffle_list(input_list)
print(f"Shuffled list: {shuffled}")
```

↩ Enter a list of elements : [1,2,3,4,5,6,7,8,9,0]
Shuffled list: [5, 9, 1, 6, 7, 3, 2, 0, 8, 4]

21. 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_ordered(tuple1, tuple2):
    set2 = set(tuple2)
    return tuple(element for element in tuple1 if element in set2)
tuple1 = eval(input("Enter the first tuple : "))
tuple2 = eval(input("Enter the second tuple : "))
common_ordered_tuple = common_elements_ordered(tuple1, tuple2)
print(f"Common elements in both tuples (ordered): {common_ordered_tuple}")
```

↩ Enter the first tuple : [1,2,3,4,5]
Enter the second tuple : [1,3,4,5,6]
Common elements in both tuples (ordered): (1, 3, 4, 5)

22. 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):
    user_input = input(prompt)
    return set(int(x.strip()) for x in user_input.split(','))
set1 = get_set_from_input("Enter the first set of integers : ")
set2 = get_set_from_input("Enter the second set of integers : ")
```

```
intersection = set1.intersection(set2)
print(f"Intersection of the two sets: {intersection}")
```

```
➞ Enter the first set of integers : 1,2,3,4
Enter the second set of integers : 4,5,6,7
Intersection of the two sets: {4}
```

23. 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):
    return tuple1 + tuple2
tuple1 = eval(input("Enter the first tuple : "))
tuple2 = eval(input("Enter the second tuple: "))
result_tuple = concatenate_tuples(tuple1, tuple2)
print(f"Concatenated tuple: {result_tuple}")
```

```
➞ Enter the first tuple : 1,2,3,4
Enter the second tuple: 5,6,7,8
Concatenated tuple: (1, 2, 3, 4, 5, 6, 7, 8)
```

24. 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):
    user_input = input(prompt)
    return set(x.strip() for x in user_input.split(','))
set1 = get_set_from_input("Enter the first set of strings : ")
set2 = get_set_from_input("Enter the second set of strings : ")
difference = set1.difference(set2)
print(f"Elements present in the first set but not in the second set: {difference}")
```

```
➞ Enter the first set of strings : a,b,c
Enter the second set of strings : b,c,d
Elements present in the first set but not in the second set: {'a'}
```

- ✓ 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 specified range of indices

```
def slice_tuple(original_tuple, start_index, end_index):
    return original_tuple[start_index:end_index]
original_tuple = eval(input("Enter a tuple : "))
start_index = int(input("Enter the start index: "))
end_index = int(input("Enter the end index: "))
sliced_tuple = slice_tuple(original_tuple, start_index, end_index)
print(f"Sliced tuple: {sliced_tuple}")
```

```
➞ Enter a tuple : 1,2,3,4,5
Enter the start index: 3
Enter the end index: 4
Sliced tuple: (4,)
```

- ✓ 26. 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):
    user_input = input(prompt)
    return set(x.strip() for x in user_input.split(','))
set1 = get_set_from_input("Enter the first set of characters : ")
set2 = get_set_from_input("Enter the second set of characters : ")
union_set = set1.union(set2)
print(f"Union of the two sets: {union_set}")
```

```
➞ Enter the first set of characters : a,b,c
Enter the second set of characters : c,d,e
Union of the two sets: {'b', 'e', 'd', 'a', 'c'}
```

- 27. 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 max_min_from_tuple(int_tuple):
    max_value = max(int_tuple)
    min_value = min(int_tuple)
    return max_value, min_value
input_tuple = eval(input("Enter a tuple of integers : "))
max_value, min_value = max_min_from_tuple(input_tuple)
print(f"Maximum value: {max_value}")
print(f"Minimum value: {min_value}")
```

```
➞ Enter a tuple of integers : 1,2,3,4,5
Maximum value: 5
Minimum value: 1
```

28. Create a code that defines two sets of integers. Then,

- ✓ print the union, intersection, and difference of these two sets

```
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
union_set = set1.union(set2)
intersection_set = set1.intersection(set2)
difference_set = set1.difference(set2)
print(f"Set 1: {set1}")
print(f"Set 2: {set2}")
print(f"Union of the two sets: {union_set}")
print(f"Intersection of the two sets: {intersection_set}")
print(f"Difference (Set 1 - Set 2): {difference_set}")
```

```
➞ Set 1: {1, 2, 3, 4, 5}
Set 2: {4, 5, 6, 7, 8}
Union of the two sets: {1, 2, 3, 4, 5, 6, 7, 8}
Intersection of the two sets: {4, 5}
Difference (Set 1 - Set 2): {1, 2, 3}
```

29. 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):
    return input_tuple.count(element)
```

```
input_tuple = eval(input("Enter a tuple : "))
element = eval(input("Enter the element to count : "))
occurrences = count_occurrences(input_tuple, element)
print(f"The element {element} occurs {occurrences} time(s) in the tuple.")
```

```
⇒ Enter a tuple : 1,2,3,4,5,6,1
Enter the element to count : 1
The element 1 occurs 2 time(s) in the tuple.
```

30. 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):
    user_input = input(prompt)
    return set(x.strip() for x in user_input.split(','))
set1 = get_set_from_input("Enter the first set of strings : ")
set2 = get_set_from_input("Enter the second set of strings : ")
symmetric_difference = set1.symmetric_difference(set2)
print(f"Symmetric difference of the two sets: {symmetric_difference}")
```

```
⇒ Enter the first set of strings : a,b,c
Enter the second set of strings : c,b,a
Symmetric difference of the two sets: set()
```

31. 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

```
def word_frequencies(word_list):
    frequency_dict = {}
    for word in word_list:
        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()
```

```
frequencies = word_frequencies(input_words)
print("Word frequencies:", frequencies)
```

➞ Enter a list of words separated by spaces: a a a b b b c c
 Word frequencies: {'a': 3, 'b': 3, 'c': 2}

32. 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):
    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
dict1 = eval(input("Enter the first dictionary : "))
dict2 = eval(input("Enter the second dictionary : "))
result_dict = merge_dictionaries(dict1, dict2)
print("Merged dictionary:", result_dict)
```

➞ Enter the first dictionary : {'a':1,'b':2}
 Enter the second dictionary : {'c':3,'d':4}
 Merged dictionary: {'a': 1, 'b': 2, 'c': 3, 'd': 4}

33. 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 access_nested_dict(nested_dict, keys):
    current_value = nested_dict
    for key in keys:
        if key in current_value:
            current_value = current_value[key]
```



```

        else:
            return None
        return current_value
nested_dictionary = {
    'a': {
        'b': {
            'c': 1
        },
        'd': 2
    },
    'e': 3
}
keys_input = input("Enter the list of keys separated by commas : ")
keys_list = [key.strip() for key in keys_input.split(',')]
result = access_nested_dict(nested_dictionary, keys_list)
if result is not None:
    print(f"The value corresponding to the keys {keys_list} is: {result}")
else:
    print(f"One or more keys {keys_list} do not exist in the dictionary.")

```


 Enter the list of keys separated by commas : a,b,c
 The value corresponding to the keys ['a', 'b', 'c'] is: 1

34. 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_dictionary_by_values(input_dict, descending=False):
    sorted_dict = dict(sorted(input_dict.items(), key=lambda item: item[1], reverse=descending))
    return sorted_dict
user_input = eval(input("Enter a dictionary : "))
order_input = input("Sort in descending order? (yes/no): ").strip().lower()
descending_order = order_input == 'yes'
sorted_result = sort_dictionary_by_values(user_input, descending=descending_order)
print("Sorted dictionary:", sorted_result)

```

 Enter a dictionary : {'a': 3, 'b': 1, 'c': 2}
 Sort in descending order? (yes/no): no
 Sorted dictionary: {'b': 1, 'c': 2, 'a': 3}

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 invert_dictionary(input_dict):  
    inverted_dict = {}  
    for key, value in input_dict.items():  
        if value in inverted_dict:  
            inverted_dict[value].append(key)  
        else:  
            inverted_dict[value] = [key]  
    return inverted_dict  
user_input = eval(input("Enter a dictionary : "))  
inverted_result = invert_dictionary(user_input)  
print("Inverted dictionary:", inverted_result)
```

```
➞ Enter a dictionary : {'a': 1, 'b': 2, 'c': 1}  
Inverted dictionary: {1: ['a', 'c'], 2: ['b']}
```