Q1. Write a code to reverse String.

Input :-

```
a = "Himanshu Panchal"
a[::-1]
```

Output :-

```
'lahcnaP uhsnamiH'
```

Q2. Write a code to count the number of vowels in String.

Input :-

```
def count_vowels(s):

vowels = 'aeiouAEIOU'

count = 0


for char in s:

    if char in vowels:
        count += 1

return count


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

Output :-

```
enter any string raman

Number of vowels in 'raman': 2
```

Q3. Write a code to check if a given string is a palindrome or not.

```
 Input :-   def is_palindrome(s):
   # Compare the string with its reverse
   return s == s[::-1]

input_string = input("enter a string")
input_string = input_string.upper()

if is_palindrome(input_string):
   print(f"'{input_string}' is a palindrome.")

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

```
 Output :-  enter a string naman

'NAMAN' is a palindrome.
```

Q4. Write a code to check if two given strings are anagrams of each other.

```
 Input :-  def anagram(str1, str2):

   # Check if both strings have the same length
   if len(str1) != len(str2):
       return False

   # Sort the characters of both strings and compare
   return sorted(str1) == sorted(str2)
```

```
string1 = input("Enter the first string: ")
string2 = input("Enter the second string: ")

if anagram(string1, string2):
    print(f"'{string1}' and '{string2}' are anagrams.")
else:
    print(f"'{string1}' and '{string2}' are not anagrams.")
```

```
 Output :- Enter the first string:  rat
Enter the second string:  art

'RAT' and 'ART' are anagrams.
```

Q5. Write a code to find all occurrences of a given substring within another string.

```
   Input :-  def find_substring(main_string, sub_string):
    return [i for i in range(len(main_string)) if
main_string.startswith(sub_string, i)]

main_str = "Hello World, Hello Python, Hello Again"
sub_str = "Hello"

indices = find_substring(main_str, sub_str)

print(f"The substring '{sub_str}' is found at indices: {indices}")
```

```
Output :-  The substring 'Hello' is found at indices: [0, 13, 27]
```

Q6. Write a code to perform basic string compression using the counts repeated characters.

```
 Input :-   def compress_string(s):
  """
  Compress a string by counting repeated characters.

  Args:
      s (str): The input string.

  Returns:
      str: The compressed string.
  """
  if not s:
      return s

  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)

# Example usage:
print(compress_string("AAABBBCCC"))
print(compress_string("ABC"))
```

Output :- A3B3C3

```
A1B1C1
```

Q7. Write a code to determine if a string has all unique characters.

Input :-

```
def has_unique_characters(s):
    # Create an empty set to keep track of seen characters
    seen_chars = set()

    # Iterate through each character in the string
    for char in s:
        if char in seen_chars:
            return False  # Duplicate character found
        seen_chars.add(char)

    return True


input_string = input("Enter a string to check for uniqueness: ")
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.")
```

Output :-

```
Enter a string to check for uniqueness:  rahul

'rahul' has all unique characters.
```

Q8. Write a code to convert a string to uppercase or lowercase.

Input :-

```
# Convert to uppercase
```

```
string = "hello world"
uppercase_string = string.upper()
print(uppercase_string)

# Convert to lowercase
string = "HELLO WORLD"
lowercase_string = string.lower()
print(lowercase_string)
```

```
Output :- HELLO WORLD
hello world
```

Q9. Write a code to count the number of words in a string.

```
Input :- def count_words(string):
    words = string.split()
    return len(words)

string = "Himanshu Panchal"
word_count = count_words(string)
print("Number of words:", word_count)
```

```
Output :-  2
```

Q10. Write a code to concatenate two strings without using the + operator.

```
Input:- def  strings(s1, s2):

    return ".join([s1, s2])
```

```
string1 = "Hello, "
string2 = "world!"
print("String:",  strings(string1, string2))
```

Output :- String: Hello, world!

Q11. Write a code to remove all occurrences of a specific element from a list.

Input :-
```
def remove_element(lst, element):
    return [i for i in lst if i!= element]

my_list = [1, 2, 3, 4, 2, 5, 2, 6]
element_to_remove = 2
new_list = remove_element(my_list, element_to_remove)
print(new_list)
```

Output :- [1, 3, 4, 5, 6]

Q12. Implement a code to find the second largest number in a given list of integers.

Input :-
```
def find_second_largest(lst):

    if len(lst) < 2:
        return None  # Not enough elements to find the second largest

    # Initialize the largest and second largest to None
    largest = second_largest = None

    for num in lst:
        if largest is None or num > largest:
            # Update second largest before updating largest
            second_largest = largest
            largest = num
        elif num != largest and (second_largest is None or num >
second_largest):
            second_largest = num

    return second_largest

numbers = [10, 20, 4, 45, 99, 99, 20]
print("Second largest number:", find_second_largest(numbers))
```

Output :- Second largest number: 45

**Q13. Create a code to count the occurrences of each element in a list and return a dictionary with elements as keys and their count
as values.**

```
 Input :- 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

example_list = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
occurrences = count_occurrences(example_list)
print("Occurrences:", occurrences)
```

Output :-

```
Occurrences: {1: 1, 2: 2, 3: 3, 4: 4}
```

Q14. write a code to reverse list in-place without using any built-in reverse functions.

Input :-

```
def reverse_list_in_place(lst):
    left = 0
    right = len(lst) - 1

    while left < right:
        # Swap the elements at the left and right indices
        lst[left], lst[right] = lst[right], lst[left]
        # Move the pointers towards the center
        left += 1
        right -= 1

example_list = [1, 2, 3, 4, 5]
print("Original List:", example_list)
reverse_list_in_place(example_list)
print("Reversed List:", example_list)
```

Output :-

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

Q15 Implement a code t o find and remove duplicates from a list while preserving the original order of elements.

Input :-

```
def remove_duplicates(lst):    seen = set()
    unique_list = []
    for element in lst:
        if element not in seen:
            seen.add(element)
            unique_list.append(element)

    return unique_list


example_list = [1, 2, 2, 3, 4, 4, 5, 1, 6]
print("Original List:", example_list)
unique_list = remove_duplicates(example_list)
print("List after removing duplicates:", unique_list)
```

Output :-

```
 Original List: [1, 2, 2, 3, 4, 4, 5, 1, 6]
List after removing duplicates: [1, 2, 3, 4, 5, 6]
```

Q16. Create a code to check if a given is sorted (either is ascending or descending order) or not.

Input :-

```
def is_sorted(lst):

    if not lst:
        return 'not sorted'  # An empty list is not sorted in either order

    # Check ascending order
    ascending = all(lst[i] <= lst[i + 1] for i in range(len(lst) - 1))
    if ascending:
        return 'ascending'

    # Check descending order
    descending = all(lst[i] >= lst[i + 1] for i in range(len(lst) - 1))
    if descending:
        return 'descending'

    return 'not sorted'


example_list1 = [1, 2, 2, 3, 4, 5]  # Ascending
example_list2 = [5, 4, 4, 3, 2, 1]  # Descending
example_list3 = [1, 3, 2, 4, 5]     # Not sorted

print("List 1 is:", is_sorted(example_list1))
print("List 2 is:", is_sorted(example_list2))
print("List 3 is:", is_sorted(example_list3))
```

Output :-

```
 List 1 is: ascending
List 2 is: descending
List 3 is: not sorted
```

Q17. Write a code to merge two sorted lists into a single sorted list.

Input :-

```
def merge_sorted_lists(list1, list2):
    result = []
    i, j = 0, 0
    while i < len(list1) and j < len(list2):
        if list1[i] < list2[j]:
            result.append(list1[i])
            i += 1
        else:
            result.append(list2[j])
            j += 1
    result.extend(list1[i:])
    result.extend(list2[j:])
    return result

list1 = [1, 3, 5, 7]
list2 = [2, 4, 6, 8]
print("List 1:", list1)
print("List 2:", list2)
print("Merged list:", merge_sorted_lists(list1, list2))
```

Output :-

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

Q18. Implement a code to find the intersection of two given lists.

Input :-

```
def intersection(list1, list2):
    return [element for element in list1 if element in list2]

list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
print("List 1:", list1)
print("List 2:", list2)
print("Intersection:", intersection(list1, list2))
```

Output :-

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

Q19. Create a code to find the union of two lists without duplicates.

Input :-

```
def union_of_lists(list1, list2):

    set1 = set(list1)
    set2 = set(list2)


    union = set1 | set2


    return list(union)

list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
union_list = union_of_lists(list1, list2)
print("Union of lists:", union_list)
```

Output :- Union of lists: [1, 2, 3, 4, 5, 6, 7, 8]

Q20. Write a code to shuffle a given list randomly without using any built-in shuffle functions.

Input :- import random

```
def shuffle_list(lst):

    n = len(lst)
    for i in range(n - 1, 0, -1):
        # Generate a random index from 0 to i
        j = random.randint(0, i)
        # Swap elements at index i and j
        lst[i], lst[j] = lst[j], lst[i]

example_list = [1, 2, 3, 4, 5]
```

```
print("Original List:", example_list)
shuffle_list(example_list)
print("Shuffled List:", example_list)
```

Output :-

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

Q21. Write a code that takes two tuples as input and returns a new tuple containing elements that are common to both input tuples.

Input :-

```
def common_elements(tuple1, tuple2):

    # Convert tuples to sets to find common elements
    set1 = set(tuple1)
    set2 = set(tuple2)

    # Find the intersection of both sets
    common_set = set1 & set2

    # Convert the result back to a sorted tuple and return
    return tuple(sorted(common_set))

tuple1 = (1, 2, 3, 4, 5)
tuple2 = (4, 5, 6, 7, 8)
common_tuple = common_elements(tuple1, tuple2)
print("Common Elements Tuple:", common_tuple)
```

Output :- Common Elements Tuple: (4, 5)

Q22. Create a code that prompts the user to enter two sets of integers separated by commas. Then print the Intersection of these two sets.

Input :-
```
def get_set_from_user(prompt):
    user_input = input(prompt)
    return set(map(int, user_input.split(',')))

def intersection(set1, set2):
    return set1 & set2

set1 = get_set_from_user("Enter the first set of integers separated by commas: ")
set2 = get_set_from_user("Enter the second set of integers separated by commas: ")

print("Intersection:", intersection(set1, set2))
```

Output :-
```
 Enter the first set of integers separated by commas:   1,12,4
Enter the second set of integers separated by commas:  4,12,13

Intersection: {4, 12}
```

Q23. 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.

Input :-
```
def concatenate_tuples(tuple1, tuple2):

    # Concatenate the two tuples using the + operator
    result = tuple1 + tuple2
    return result

tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
concatenated_tuple = concatenate_tuples(tuple1, tuple2)
print("Concatenated Tuple:", concatenated_tuple)
```

Output :- Concatenated Tuple: (1, 2, 3, 4, 5, 6)

Q24. 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.

Input :-

```
def get_set_from_user(prompt):
    user_input = input(prompt)
    return set(user_input.split(','))

def difference(set1, set2):
    return set1 - set2

set1 = get_set_from_user("Enter the first set of strings separated by commas: ")
set2 = get_set_from_user("Enter the second set of strings separated by commas: ")

print("Elements in the first set but not in the second set:", difference(set1, set2))
```

Output :-

```
Enter the first set of strings separated by commas: ram,shyam,rohan

Enter the second set of strings separated by commas: shyam,rahul,abhay
```

```
Elements in the first set but not in the second set: {'ram', 'rohan'}
```

Q25. Create a code that takes a tuple and two strings as input. The function should return a new tuple containing elements from the original tuple within the specified range of indices.

Input :-

```
def slice_tuple(t, start, end):
    start_idx = int(start)
    end_idx = int(end)
    return t[start_idx:end_idx]

t = (1, 2, 3, 4, 5, 6, 7, 8, 9)
start = "2"
end = "5"
result = slice_tuple(t, start, end)
print(result)
```

Output :-

```
(3, 4, 5)
```

Q26. Write a code that prompts the users to input two sets of characters. Then, print the union of these two sets.

Input :-

```
def get_set_from_user(prompt):
    user_input = input(prompt)
    return set(user_input)

def union(set1, set2):
    return set1.union(set2)

set1 = get_set_from_user("Enter the first set of characters: ")
set2 = get_set_from_user("Enter the second set of characters: ")

print("Union of the two sets:", union(set1, set2))
```

Output :-

```
Enter the first set of characters:  Himanshu
Enter the second set of characters:  Panchal

Union of the two sets: {'i', 'n', 'u', 'a', 'P', 'c', 'l', 'm', 'h', 'H', 's'}
```

Q27. 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.

Input :-

```
def find_max_min(tup):

    if not tup:
        raise ValueError("The input tuple is empty. Cannot determine max and min values.")

    # Compute the maximum and minimum values
    max_value = max(tup)
    min_value = min(tup)

    # Return the results using tuple unpacking
    return (max_value, min_value)

input_tuple = (10, 20, 30, 40, 50, 60)
max_val, min_val = find_max_min(input_tuple)
print("Maximum Value:", max_val)
print("Minimum Value:", min_val)
```

Output :-

```
 Maximum Value: 60
Minimum Value: 10
```

Q28. Create a code that defines two sets of integers. Then, print the union, intersection, and difference of these two sets.

Input :-

```
 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("Set 1:", set1)
print("Set 2:", set2)
print("Union:", union_set)
print("Intersection:", intersection_set)
print("Difference:", difference_set)
```

Output :-

```
 Set 1: {1, 2, 3, 4, 5}
Set 2: {4, 5, 6, 7, 8}
Union: {1, 2, 3, 4, 5, 6, 7, 8}
Intersection: {4, 5}
Difference: {1, 2, 3}
```

Q29. 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.

Input :-

```
def count_occurrences(t, elem):
    return t.count(elem)

t = (1, 2, 3, 2, 4, 2, 5)
elem = 2
count = count_occurrences(t, elem)
print(f"The element {elem} occurs {count} times in the tuple.")
```

Output :-

```
The element 2 occurs 3 times in the tuple.
```

Q30. Develop a code that prompts the user to input two sets of strings. Then, print the symmetric difference of these two sets.

Input :-

```
 def get_set_from_user(prompt):
    user_input = input(prompt)
    return set(user_input.split())

def print_symmetric_difference(set1, set2):
    symmetric_difference = set1.symmetric_difference(set2)
    print("Symmetric difference:", symmetric_difference)

def main():
    set1 = get_set_from_user("Enter the first set of strings (separated by spaces): ")
    set2 = get_set_from_user("Enter the second set of strings (separated by spaces): ")
    print_symmetric_difference(set1, set2)

if __name__ == "__main__":
    main()
```

Output :-

```
 Enter the first set of strings (separated by spaces):  hp re de
Enter the second set of strings (separated by spaces):  de

Symmetric difference: {'hp', 're'}
```

Q31. 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.

Input :-

```
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


words = ["apple", "banana", "apple", "orange", "banana", "banana"]
frequencies = word_frequencies(words)
print(frequencies)  # Output: {'apple': 2, 'banana': 3, 'orange': 1}
```

Output :- {'apple': 2, 'banana': 3, 'orange': 1}

Q32. 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.

Input :-

```
def merge_dicts(dict1, dict2):
    merged_dict = {**dict1}
    for key, value in dict2.items():
        if key in merged_dict:
            merged_dict[key] += value
        else:
            merged_dict[key] = value
    return merged_dict


dict1 = {"a": 1, "b": 2, "c": 3}
dict2 = {"b": 4, "c": 5, "d": 6}
merged_dict = merge_dicts(dict1, dict2)
print(merged_dict)
```

Output :- `{'a': 1, 'b': 6, 'c': 8, 'd': 6}`

Q33. Write a code to access a value in 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.

Input :-

```
 def get_value(nested_dict, keys):
    current_dict = nested_dict
    for key in keys:
        if key not in current_dict:
            return None
        current_dict = current_dict[key]
    return current_dict
```

```
nested_dict = {"a": {"b": {"c": 1}}}
keys = ["a", "b", "c"]
value = get_value(nested_dict, keys)
print(value)  # Output: 1

keys = ["a", "b", "d"]
value = get_value(nested_dict, keys)
print(value)  # Output: None
```

Output :-

```
 1
None
```

Q34. 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.

Input :-

```
def sort_dict_by_value(dict, ascending=True):

    return dict(sorted(dict.items(), key=lambda item: item[1],
reverse=not ascending))


dict = {"a": 3, "b": 1, "c": 2, "d": 4}
sorted_dict_ascending = sort_dict_by_value(dict)
```

```
print(sorted_dict_ascending) sorted_dict_descending =
sort_dict_by_value(dict, ascending=False)

print(sorted_dict_descending)
```

Output :-

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

Q35. 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 invented dictionary.

Input :-

```
def invert_dict(dict):

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

dict = {"a": 1, "b": 2, "c": 1, "d": 3, "e": 2}
inverted_dict = invert_dict(dict)
print(inverted_dict)
```

Output :- **{1: ['a', 'c'], 2: ['b', 'e'], 3: ['d']}**

Link :-

https://docs.google.com/document/d/1l7ITJohLynj7ViK4VGzGF26ScabGnyg4kZyBtkas8bw/edit