1. Write a Python program to reverse a string without using any built-in string reversal functions.

```
   def reverse_string(s):
   reversed_str = ''
   for i in range(len(s) - 1, -1, -1):
       reversed_str += s[i]
   return reversed_str

string = "Hello, World!"
reversed_string = reverse_string(string)
print(reversed_string)
```

2. Implement a function to check if a given string is a palindrome.

```
   def is_palindrome(s):
   reversed_str = reverse_string(s)
   return s == reversed_str

string = "madam"
print(is_palindrome(string))  # True

string = "hello"
print(is_palindrome(string))  # False
```

3. Write a program to find the largest element in a given list.

```
 Ans - def find_largest_element(lst):
   if len(lst) == 0:
       return None
   largest = lst[0]
   for num in lst:
       if num > largest:
           largest = num
   return largest

numbers = [10, 5, 20, 15, 30]
largest_number = find_largest_element(numbers)
print(largest_number)
```

4. Implement a function to count the occurrence of each element in a list.

```
Ans - def count_occurrences(lst):
   occurrence_count = {}
   for element in lst:
        occurrence_count[element] = occurrence_count.get(element, 0) + 1
    return occurrence_count

numbers = [1, 2, 3, 2, 1, 3, 4, 5, 2, 1]
occurrence_count = count_occurrences(numbers)
print(occurrence_count)
```

5. Write a Python program to find the second largest number in a list.

Ans - def find_second_largest(lst):

```
    if len(lst) < 2:
        return None
    largest = lst[0]
    second_largest = float('-inf')
    for num in lst:
        if num > largest:
            second_largest = largest
            largest = num
        elif num > second_largest and num < largest:
            second_largest = num
    return second_largest

numbers = [10, 5, 20, 15, 30]
second_largest_number = find_second_largest(numbers)
print(second_largest_number)
```

6. Implement a function to remove duplicate elements from a list.

Ans - def remove_duplicates(lst):

```
    return list(set(lst))

numbers = [1, 2, 3, 2, 1, 3, 4, 5, 2, 1]
unique_numbers = remove_duplicates(numbers)
print(unique_numbers)
```

7. Write a program to calculate the factorial of a given number.

Ans - def factorial(n):

```
    if n == 0 or n == 1:
        return 1
    fact = 1
    for i in range(2, n + 1):
        fact *= i
    return fact
```

```
number = 5
factorial_result = factorial(number)
print(factorial_result)
```

8. Implement a function to check if a given number is prime.

Ans -
```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

number = 17
print(is_prime(number))  # True

number = 20
print(is_prime(number))  # False
```

9. Write a Python program to sort a list of integers in ascending order.

Ans-
```
def sort_list(lst):
    return sorted(lst)

numbers = [5, 2, 8, 1, 9]
sorted_numbers = sort_list(numbers)
print(sorted_numbers)
```

10. Implement a function to find the sum of all numbers in a list.

Ans -
```
def find_sum(lst):
    return sum(lst)

numbers = [1, 2, 3, 4, 5]
sum_of_numbers = find_sum(numbers)
print(sum_of_numbers)
```

11. Write a program to find the common elements between two lists.

Ans -
```
def find_common_elements(lst1, lst2):
    return list(set(lst1) & set(lst2))
```

```
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
common_elements = find_common_elements(list1, list2)
print(common_elements)
```

12. Implement a function to check if a given string is an anagram of another string.

Ans -
```
def is_anagram(str1, str2):
    return sorted(str1) == sorted(str2)

string1 = "listen"
string2 = "silent"
print(is_anagram(string1, string2))  # True

string1 = "hello"
string2 = "world"
print(is_anagram(string1, string2))  # False
```

13. Write a Python program to generate all permutations of a given string.

Ans-
```
def generate_permutations(s):
    if len(s) == 0:
        return []
    if len(s) == 1:
        return [s]
    permutations = []
    for i in range(len(s)):
        char = s[i]
        remaining_chars = s[:i] + s[i + 1:]
        for permutation in generate_permutations(remaining_chars):
            permutations.append(char + permutation)
    return permutations

string = "abc"
permutations = generate_permutations(string)
print(permutations)
```

14. Implement a function to calculate the Fibonacci sequence up to a given number of terms.

Ans -
```
def fibonacci_sequence(n):
    sequence = [0, 1]
    if n <= 1:
        return sequence[:n + 1]
    while len(sequence) <= n:
        sequence.append(sequence[-1] + sequence[-2])
    return sequence

terms = 10
fibonacci = fibonacci_sequence(terms)
print(fibonacci)
```

15. Write a program to find the median of a list of numbers.

```
Ans - def find_median(lst):
    sorted_lst = sorted(lst)
    n = len(sorted_lst)
    if n % 2 == 0:
        mid = n // 2
        return (sorted_lst[mid - 1] + sorted_lst[mid]) / 2
    else:
        return sorted_lst[n // 2]

numbers = [1, 3, 2, 5, 4]
median = find_median(numbers)
print(median)
```

16. Implement a function to check if a given list is sorted in non-decreasing order.

```
Ans- def is_sorted(lst):
    return all(lst[i] <= lst[i + 1] for i in range(len(lst) - 1))

numbers1 = [1, 2, 3, 4, 5]
print(is_sorted(numbers1))  # True

numbers2 = [5, 2, 8, 1, 9]
print(is_sorted(numbers2))  # False
```

17. Write a Python program to find the intersection of two lists.

```
Ans- def find_intersection(lst1, lst2):
    return list(set(lst1) & set(lst2))

list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
intersection = find_intersection(list1, list2)
print(intersection)
```

18. Implement a function to find the maximum subarray sum in a given list.

Ans -
```
def find_maximum_subarray_sum(lst):
    max_sum = float('-inf')
    current_sum = 0
    for num in lst:
        current_sum = max(num, current_sum + num)
        max_sum = max(max_sum, current_sum)
    return max_sum

numbers = [-2, -3, 4, -1, -2, 1, 5, -3]
maximum_subarray_sum = find_maximum_subarray_sum(numbers)
print(maximum_subarray_sum)
```

19. Write a program to remove all vowels from a given string.

Ans -
```
def remove_vowels(s):
    vowels = "aeiouAEIOU"
    return ''.join(char for char in s if char not in vowels)

string = "Hello, World!"
string_without_vowels = remove_vowels(string)
print(string_without_vowels)
```

20. Implement a function to reverse the order of words in a given sentence.

Ans -
```
def reverse_words(sentence):
    words = sentence.split()
    reversed_words = ' '.join(reversed(words))
    return reversed_words

sentence = "Hello, World!"
reversed_sentence = reverse_words(sentence)
print(reversed_sentence)
```

21. Write a Python program to check if two strings are anagrams of each other.

Ans -
```
def are_anagrams(str1, str2):
    return sorted(str1) == sorted(str2)

string1 = "listen"
string2 = "silent"
print(are_anagrams(string1, string2))  # True

string1 = "hello"
```

```
string2 = "world"
print(are_anagrams(string1, string2))  # False
```

22. Implement a function to find the first non-repeating character in a string.

Ans-
```
def find_first_non_repeating_char(s):
    char_count = {}
    for char in s:
        char_count[char] = char_count.get(char, 0) + 1
    for char in s:
        if char_count[char] == 1:
            return char
    return None

string = "hello"
first_non_repeating_char = find_first_non_repeating_char(string)
print(first_non_repeating_char)
```

23. Write a program to find the prime factors of a given number.

Ans -
```
def find_prime_factors(n):
    factors = []
    divisor = 2
    while n > 1:
        if n % divisor == 0:
            factors.append(divisor)
            n /= divisor
        else:
            divisor += 1
    return factors

number = 48
prime_factors = find_prime_factors(number)
print(prime_factors)
```

24. Implement a function to check if a given number is a power of two.

Ans -
```
def is_power_of_two(n):
    if n <= 0:
        return False
    return (n & (n - 1)) == 0

number1 = 16
print(is_power_of_two(number1))  # True
```

```
number2 = 20
print(is_power_of_two(number2))  # False
```

25. Write a Python program to merge two sorted lists into a single sorted list.

Ans -
```
def merge_sorted_lists(lst1, lst2):
    merged_list = []
    i, j = 0, 0
    while i < len(lst1) and j < len(lst2):
        if lst1[i] < lst2[j]:
            merged_list.append(lst1[i])
            i += 1
        else:
            merged_list.append(lst2[j])
            j += 1
    merged_list.extend(lst1[i:])
    merged_list.extend(lst2[j:])
    return merged_list

list1 = [1, 3, 5, 7]
list2 = [2, 4, 6, 8]
merged_list = merge_sorted_lists(list1, list2)
print(merged_list)
```

26. Implement a function to find the mode of a list of numbers.

Ans -
```
def find_mode(lst):
    frequency_count = {}
    for num in lst:
        frequency_count[num] = frequency_count.get(num, 0) + 1
    max_frequency = max(frequency_count.values())
    modes = [num for num, frequency in frequency_count.items() if frequency == max_frequency]
    return modes

numbers = [1, 2, 3, 2, 1, 3, 4, 5, 2, 1]
modes = find_mode(numbers)
print(modes)
```

27. Write a program to find the greatest common divisor (GCD) of two numbers.

Ans -
```
def find_gcd(a, b):
    while b:
        a, b = b, a % b
    return a

number1 = 48
number2 = 36
gcd = find_gcd(number1, number2)
print(gcd)
```

28. Implement a function to calculate the square root of a given number

```
Ans- .def calculate_square_root(n):
    if n < 0:
        raise ValueError("Square root is not defined for negative numbers")
    if n == 0:
        return 0
    guess = n
    while True:
        new_guess = 0.5 * (guess + n / guess)
        if abs(guess - new_guess) < 1e-9:
            return new_guess
        guess = new_guess

number = 25
square_root = calculate_square_root(number)
print(square_root)
```

29. Write a Python program to check if a given string is a valid palindrome ignoring non-alphanumeric characters.

```
Ans - def is_valid_palindrome(s):
    alphanumeric_str = ''.join(char.lower() for char in s if char.isalnum())
    return alphanumeric_str == alphanumeric_str[::-1]

string1 = "A man, a plan, a canal: Panama"
print(is_valid_palindrome(string1))  # True
```

30. Implement a function to find the minimum element in a rotated sorted list.

```
Ans - def find_min_in_rotated_list(lst):
    left, right = 0, len(lst) - 1
    while left < right:
        mid = left + (right - left) // 2
        if lst[mid] > lst[right]:
            left = mid + 1
        else:
            right = mid
    return lst[left]

numbers = [4, 5, 6, 7, 0, 1, 2]
min_element = find_min_in_rotated_list(numbers)
print(min_element)
```

31. Write a program to find the sum of all even numbers in a list.

```
Ans - def find_sum_of_evens(lst):
    return sum(num for num in lst if num % 2 == 0)

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sum_of_evens = find_sum_of_evens(numbers)
print(sum_of_evens)
```

32. Implement a function to calculate the power of a number using recursion.

```
Ans - def calculate_power(base, exponent):
    if exponent == 0:
        return 1
    if exponent < 0:
        return 1 / calculate_power(base, -exponent)
    if exponent % 2 == 0:
        half = calculate_power(base, exponent // 2)
        return half * half
    return base * calculate_power(base, exponent - 1)

base = 2
exponent = 3
power = calculate_power(base, exponent)
print(power)
```

33. Write a Python program to remove duplicates from a list while preserving the order.

```
Ans - def remove_duplicates_preserve_order(lst):
    seen = set()
    return [x for x in lst if not (x in seen or seen.add(x))]

numbers = [1, 2, 3, 2, 1, 3, 4, 5, 2, 1]
unique_numbers = remove_duplicates_preserve_order(numbers)
print(unique_numbers)
```

34. Implement a function to find the longest common prefix among a list of strings.

```
Ans - def find_longest_common_prefix(strings):
    if not strings:
        return ''
    shortest_string = min(strings, key=len)
    for i, char in enumerate(shortest_string):
        if any(string[i] != char for string in strings):
            return shortest_string[:i]
    return shortest_string

strings = ["flower", "flow", "flight"]
longest_common_prefix = find_longest_common_prefix(strings)
print(longest_common_prefix)
```

35. Write a program to check if a given number is a perfect square.

```
Ans - def is_perfect_square(n):
    if n < 0:
        return False
    root = int(n**0.5)
    return root * root == n

number1 = 25
print(is_perfect_square(number1))  # True
```

36. Implement a function to calculate the product of all elements in a list.

```
Ans - def calculate_product(lst):
    product = 1
    for num in lst:
        product *= num
    return product

numbers = [1, 2, 3, 4, 5]
product = calculate_product(numbers)
print(product)
```

37. Write a Python program to reverse the order of words in a sentence while preserving the word order.

```
Ans - def reverse_sentence(sentence):
    words = sentence.split()
    reversed_words = ' '.join(reversed(words))
    return reversed_words

sentence = "Hello, World!"
reversed_sentence = reverse_sentence(sentence)
print(reversed_sentence)
```

38. Implement a function to find the missing number in a given list of consecutive numbers.

```
Ans - def find_missing_number(lst):
    n = len(lst) + 1
    total_sum = (n * (n + 1)) // 2
    actual_sum = sum(lst)
    return total_sum - actual_sum

numbers = [1, 2, 3, 5]
missing_number = find_missing_number(numbers)
print(missing_number)
```

39. Write a program to find the sum of digits of a given number.

```
Ans - def find_sum_of_digits(n):
    return sum(int(digit) for digit in str(n) if digit.isdigit())

number = 12345
sum_of_digits = find_sum_of_digits(number)
print(sum_of_digits)
```

40. Implement a function to check if a given string is a valid palindrome considering case sensitivity.

```
Ans - def is_valid_palindrome_case_sensitive(s):
    return s == s[::-1]

string1 = "Madam"
print(is_valid_palindrome_case_sensitive(string1))  # False

string2 = "racecar"
print(is_valid_palindrome_case_sensitive(string2))  # True
```

41. Write a Python program to find the smallest missing positive integer in a list.

```
Ans - def find_smallest_missing_positive(lst):
    smallest_missing = 1
    set_lst = set(lst)
    while smallest_missing in set_lst:
        smallest_missing += 1
    return smallest_missing

numbers = [1, 2, 0]
smallest_missing = find_smallest_missing_positive(numbers)
print(smallest_missing)
```

42. Implement a function to find the longest palindrome substring in a given string.

```
Ans - def find_longest_palindrome_substring(s):
    def expand_around_center(left, right):
        while left >= 0 and right < len(s) and s[left] == s[right]:
            left -= 1
            right += 1
        return s[left + 1: right]

    longest_palindrome = ''
    for i in range(len(s)):
        odd_palindrome = expand_around_center(i, i)
        even_palindrome = expand_around_center(i, i + 1)
        longest_palindrome = max(longest_palindrome, odd_palindrome, even_palindrome,
key=len)
    return longest_palindrome

string = "babad"
longest_palindrome = find_longest_palindrome_substring(string)
print(longest_palindrome)
```

43. Write a program to find the number of occurrences of a given element in a list.

```
Ans -  def count_occurrences(lst, element):
    return lst.count(element)

numbers = [1, 2, 3, 2, 1, 3, 4, 5, 2, 1]
element = 2
occurrence_count = count_occurrences(numbers, element)
print(occurrence_count)
```

44. Implement a function to check if a given number is a perfect number.

```
Ans - def is_perfect_number(n):
    if n < 2:
        return False
    divisors = [1]
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            divisors.extend([i, n // i])
    return sum(divisors) == n

number1 = 6
print(is_perfect_number(number1))  # True

number2 = 12
print(is_perfect_number(number2))  # False
```

45. Write a Python program to remove all duplicates from a string.

```
Ans - def remove_duplicates_from_string(s):
    return ''.join(set(s))

string = "Hello, World!"
string_without_duplicates = remove_duplicates_from_string(string)
print(string_without_duplicates)
```

46. Implement a function to find the first missing positive

```
Ans - def find_first_missing_positive(lst):
    n = len(lst)
    for i in range(n):
        while 1 <= lst[i] <= n and lst[i] != lst[lst[i] - 1]:
            lst[lst[i] - 1], lst[i] = lst[i], lst[lst[i] - 1]
    for i in range(n):
        if lst[i] != i + 1:
            return i + 1
    return n + 1

numbers = [3, 4, -1, 1]
first_missing_positive = find_first_missing_positive(numbers)
print(first_missing_positive)
```