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

```
def reverse_string(input_string):
    reversed_string = ""
    for i in range(len(input_string) - 1, -1, -1):
        reversed_string += input_string[i]
    return reversed_string

# Example usage
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(input_string):
    input_string = input_string.lower()
    reversed_string = input_string[::-1]
    return input_string == reversed_string

# Example usage
string = "radar"
is_pal = is_palindrome(string)
print(is_pal)
```

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

```
def find_largest_element(lst):
    largest_element = lst[0]
    for element in lst:
        if element > largest_element:
            largest_element = element
    return largest_element

# Example usage
numbers = [10, 5, 20, 30, 15]
largest = find_largest_element(numbers)
print(largest)
```

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

```
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 usage
numbers = [1, 2, 1, 3, 2, 1, 3, 4, 4, 2]
occurrences = count_occurrences(numbers)
print(occurrences)
```

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

```
def find_second_largest(lst):
    if len(lst) < 2:
        return None
    largest = max(lst[0], lst[1])
    second_largest = min(lst[0], lst[1])
    for i in range(2, len(lst)):
        if lst[i] > largest:
            second_largest = largest
            largest = lst[i]
        elif lst[i] > second_largest and lst[i] != largest:
            second_largest = lst[i]
    return second_largest

# Example usage
numbers = [10, 5, 20, 30, 15]
second_largest = find_second_largest(numbers)
print(second_largest)
```

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

```
def remove_duplicates(lst):
    return list(set(lst))
```

```
# Example usage
numbers = [1, 2, 1, 3, 2, 1, 3, 4, 4, 2]
unique_numbers = remove_duplicates(numbers)
print(unique_numbers)
```

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

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

# Example usage
num = 5
fact = factorial(num)
print(fact)
```

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

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

# Example usage
number = 13
is_prime_number = is_prime(number)
print(is_prime_number)
```

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

```
def sort_list(lst):
    sorted_list = lst[:]
    for i in range(len(sorted_list)):
        for j in range(i+1, len(sorted_list)):
            if sorted_list[i] > sorted_list[j]:
                sorted_list[i], sorted_list[j] = sorted_list[j], sorted_list[i]
```

```
    return sorted_list

# Example usage
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.

```
def find_sum(lst):
    total = 0
    for num in lst:
        total += num
    return total

# Example usage
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.

```
def find_common_elements(list1, list2):
    common_elements = []
    for element in list1:
        if element in list2:
            common_elements.append(element)
    return common_elements

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

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

```
def is_anagram(str1, str2):
    str1 = str1.lower().replace(" ", "")
    str2 = str2.lower().replace(" ", "")
    return sorted(str1) == sorted(str2)
```

```
# Example usage
string1 = "listen"
string2 = "silent"
if is_anagram(string1, string2):
    print("The strings are anagrams.")
else:
    print("The strings are not anagrams.")
```

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

```
import itertools

def generate_permutations(string):
    permutations = [''.join(perm) for perm in itertools.permutations(string)]
    return permutations

# Example usage
string = "abc"
permutations = generate_permutations(string)
print(permutations)
```

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

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

# Example usage
num_terms = 10
fibonacci = fibonacci_sequence(num_terms)
print(fibonacci)
```

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

```
def find_median(numbers):
    numbers.sort()
```

```
    length = len(numbers)
    if length % 2 == 0:
        middle = length // 2
        median = (numbers[middle - 1] + numbers[middle]) / 2
    else:
        middle = length // 2
        median = numbers[middle]
    return median

# Example usage
number_list = [4, 2, 9, 6, 7, 1, 5]
median = find_median(number_list)
print(median)
```

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

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

# Example usage
sorted_list = [1, 2, 3, 4, 5]
if is_sorted(sorted_list):
    print("The list is sorted.")
else:
    print("The list is not sorted.")
```

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

```
def find_intersection(list1, list2):
    intersection = list(set(list1) & set(list2))
    return intersection

# Example usage
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.

```
def find_max_subarray_sum(numbers):
```

```
    max_sum = current_sum = numbers[0]
    for number in numbers[1:]:
        current_sum = max(number, current_sum + number)
        max_sum = max(max_sum, current_sum)
    return max_sum

# Example usage
number_list = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
max_sum = find_max_subarray_sum(number_list)
print(max_sum)
```

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

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

# Example usage
input_string = "Hello, World!"
result = remove_vowels(input_string)
print(result)
```

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

```
def reverse_words(sentence):
    words = sentence.split()
    reversed_sentence = ' '.join(reversed(words))
    return reversed_sentence

# Example usage
input_sentence = "Hello, world!"
reversed_sentence = reverse_words(input_sentence)
print(reversed_sentence)
```

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

```
def are_anagrams(str1, str2):
    # Convert strings to lowercase and remove spaces
    str1 = str1.lower().replace(" ", "")
    str2 = str2.lower().replace(" ", "")
```

```
    # Check if the sorted characters are the same
    return sorted(str1) == sorted(str2)

# Example usage
print(are_anagrams("listen", "silent"))  # True
print(are_anagrams("hello", "world"))    # False
```

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

```
from collections import Counter

def first_non_repeating_char(string):
    # Count the occurrences of each character
    char_counts = Counter(string)

    # Find the first character with count 1
    for char in string:
        if char_counts[char] == 1:
            return char

    # If no non-repeating character found
    return None

# Example usage
print(first_non_repeating_char("abracadabra"))  # 'c'
print(first_non_repeating_char("hello"))        # 'h'
```

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

```
def prime_factors(n):
    factors = []
    i = 2

    while i * i <= n:
        if n % i:
            i += 1
        else:
            n //= i
            factors.append(i)

    if n > 1:
        factors.append(n)
```

```
    return factors

# Example usage
print(prime_factors(56))    # [2, 2, 2, 7]
print(prime_factors(120))   # [2, 2, 2, 3, 5]
```

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

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

# Example usage
print(is_power_of_two(16))   # True
print(is_power_of_two(10))   # False
```

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

```
def merge_sorted_lists(list1, list2):
    merged = []
    i = j = 0

    while i < len(list1) and j < len(list2):
        if list1[i] <= list2[j]:
            merged.append(list1[i])
            i += 1
        else:
            merged.append(list2[j])
            j += 1

    # Add remaining elements from list1, if any
    merged.extend(list1[i:])

    # Add remaining elements from list2, if any
    merged.extend(list2[j:])

    return merged

# Example usage
list1 = [1, 3, 5, 7]
```

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

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

```
from collections import Counter

def find_mode(numbers):
    counts = Counter(numbers)
    max_count = max(counts.values())
    mode = [num for num, count in counts.items() if count == max_count]
    return mode

# Example usage
numbers = [1, 2, 3, 2, 2, 4, 5, 4, 2]
print(find_mode(numbers))   # [2]
```

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

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# Example usage
print(gcd(24, 36))   # 12
print(gcd(49, 14))   # 7
```

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

```
def sqrt(n):
    if n < 0:
        raise ValueError("Square root is not defined for negative numbers.")
    elif n == 0:
        return 0

    x = n
    y = (x + n / x) / 2

    while abs(y - x) >= 0.0001:
        x = y
```

```
        y = (x + n / x) / 2

    return y

# Example usage
print(sqrt(16))   # 4.0
print(sqrt(2))    # 1.4142156862745097
```

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

```
import re

def is_valid_palindrome(s):
    s = re.sub(r'[^a-zA-Z0-9]', '', s.lower())
    return s == s[::-1]

# Example usage
print(is_valid_palindrome("A man, a plan, a canal: Panama"))  # True
print(is_valid_palindrome("race a car"))                      # False
```

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

```
def find_minimum(nums):
    left = 0
    right = len(nums) - 1

    while left < right:
        mid = left + (right - left) // 2

        if nums[mid] > nums[right]:
            left = mid + 1
        else:
            right = mid

    return nums[left]

# Example usage
nums = [4, 5, 6, 7, 0, 1, 2]
print(find_minimum(nums))   # 0
```

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

```
def sum_of_even_numbers(numbers):
    return sum(num for num in numbers if num % 2 == 0)

# Example usage
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(sum_of_even_numbers(numbers))   # 30
```

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

```
def power(base, exponent):
    if exponent == 0:
        return 1
    elif exponent > 0:
        return base * power(base, exponent - 1)
    else:
        return 1 / (base * power(base, -exponent - 1))

# Example usage
print(power(2, 3))     # 8
print(power(5, -2))    # 0.04
```

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

```
def remove_duplicates(numbers):
    seen = set()
    return [num for num in numbers if not (num in seen or seen.add(num))]

# Example usage
numbers = [1, 2, 3, 2, 1, 4, 5, 6, 4]
print(remove_duplicates(numbers))   # [1, 2, 3, 4, 5, 6]
```

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

```
def longest_common_prefix(strings):
    if not strings:
        return ""

    prefix = strings[0]
```

```
    for string in strings[1:]:
        while not string.startswith(prefix):
            prefix = prefix[:-1]

    return prefix

# Example usage
strings = ["flower", "flow", "flight"]
print(longest_common_prefix(strings))   # "fl"
```

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

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

# Example usage
print(is_perfect_square(16))   # True
print(is_perfect_square(14))   # False
```

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

```
from functools import reduce
import operator

def product_of_elements(numbers):
    return reduce(operator.mul, numbers, 1)

# Example usage
numbers = [1, 2, 3, 4, 5]
print(product_of_elements(numbers))   # 120
```

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

```
def reverse_words(sentence):
    words = sentence.split()
    reversed_words = " ".join(reversed(words))
    return reversed_words
```

```
# Example usage
sentence = "Hello, world! I am Python."
print(reverse_words(sentence))   # "Python. am I world! Hello,"
```

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

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

# Example usage
numbers = [1, 2, 3, 5, 6, 7, 8, 9, 10]
print(find_missing_number(numbers))   # 4
```

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

```
def sum_of_digits(n):
    total = 0
    while n > 0:
        total += n % 10
        n //= 10
    return total

# Example usage
print(sum_of_digits(12345))   # 15
print(sum_of_digits(9876))    # 30
```

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

```
def is_valid_palindrome(string):
    # Remove non-alphanumeric characters and convert to lowercase
    string = ''.join(ch.lower() for ch in string if ch.isalnum())
    # Check if the reversed string is equal to the original string
    return string == string[::-1]

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

```
print(is_valid_palindrome("racecar"))  # True
print(is_valid_palindrome("Hello, World!"))  # False
```

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

```
def find_smallest_missing_positive(nums):
    n = len(nums)
    for i in range(n):
        while 1 <= nums[i] <= n and nums[i] != nums[nums[i] - 1]:
            nums[nums[i] - 1], nums[i] = nums[i], nums[nums[i] - 1]

    for i in range(n):
        if nums[i] != i + 1:
            return i + 1

    return n + 1

print(find_smallest_missing_positive([1, 2, 0]))  # 3
print(find_smallest_missing_positive([3, 4, -1, 1]))  # 2
print(find_smallest_missing_positive([7, 8, 9, 11, 12]))  # 1
```

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

```
def longest_palindrome_substring(string):
    n = len(string)
    if n < 2:
        return string

    start = 0
    max_len = 1

    for i in range(n):
        # Check odd-length palindromes centered at i
        left = right = i
        while left >= 0 and right < n and string[left] == string[right]:
            if right - left + 1 > max_len:
                start = left
                max_len = right - left + 1
```

```
            left -= 1
            right += 1

        # Check even-length palindromes centered at i, i+1
        left = i
        right = i + 1
        while left >= 0 and right < n and string[left] == string[right]:
            if right - left + 1 > max_len:
                start = left
                max_len = right - left + 1
            left -= 1
            right += 1

    return string[start:start + max_len]

print(longest_palindrome_substring("babad"))  # "bab"
print(longest_palindrome_substring("cbbd"))  # "bb"
print(longest_palindrome_substring("a"))  # "a"
```

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

```
def count_occurrences(lst, element):
    count = 0
    for item in lst:
        if item == element:
            count += 1
    return count

numbers = [2, 5, 3, 2, 4, 2, 7, 2]
print(count_occurrences(numbers, 2))  # 4
print(count_occurrences(numbers, 7))  # 1
print(count_occurrences(numbers, 9))  # 0
```

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

```
def is_perfect_number(num):
    if num <= 0:
        return False

    divisor_sum = sum(i for i in range(1, num) if num % i == 0)
```

```
    return divisor_sum == num

print(is_perfect_number(6))  # True
print(is_perfect_number(28))  # True
print(is_perfect_number(12))  # False
```

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

```
def remove_duplicates(string):
    unique_chars = []
    for char in string:
        if char not in unique_chars:
            unique_chars.append(char)
    return ''.join(unique_chars)

print(remove_duplicates("Hello, World!"))  # "Helo, Wrd!"
print(remove_duplicates("abbcccddddeeeee"))  # "abcde"
```

46. Implement a function to find the first missing positive

```
def find_first_missing_positive(nums):
    n = len(nums)

    # Move each positive number to its corresponding index
    for i in range(n):
        while 1 <= nums[i] <= n and nums[i] != nums[nums[i] - 1]:
            nums[nums[i] - 1], nums[i] = nums[i], nums[nums[i] - 1]

    # Find the first index where the value doesn't match the index
    for i in range(n):
        if nums[i] != i + 1:
            return i + 1

    return n + 1


print(find_first_missing_positive([1, 2, 0]))  # 3
print(find_first_missing_positive([3, 4, -1, 1]))  # 2
print(find_first_missing_positive([7, 8, 9, 11, 12]))  # 1
```