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

def reverse\_string(string):

reversed\_string = ""

for i in range(len(string) - 1, -1, -1):

reversed\_string += string[i]

return reversed\_string

# Example usage

input\_string = input("Enter a string: ")

reversed\_string = reverse\_string(input\_string)

print("Reversed string:", reversed\_string)

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

def is\_palindrome(string):

# Remove spaces and convert to lowercase

string = string.replace(" ", "").lower()

# Check if the string is equal to its reverse

reversed\_string = reverse\_string(string)

return string == reversed\_string

def reverse\_string(string):

reversed\_string = ""

for i in range(len(string) - 1, -1, -1):

reversed\_string += string[i]

return reversed\_string

# Example usage

input\_string = input("Enter a string: ")

if is\_palindrome(input\_string):

print("The string is a palindrome.")

else:

print("The string is not a palindrome."

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

def find\_largest\_element(lst):

if not lst:

return None

largest\_element = lst[0]

for num in lst:

if num > largest\_element:

largest\_element = num

return largest\_element

# Example usage

input\_list = [12, 45, 67, 23, 9, 100]

largest = find\_largest\_element(input\_list)

print("The largest element is:", largest)

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

def count\_occurrences(lst):

occurrences = {}

for element in lst:

if element in occurrences:

occurrences[element] += 1

else:

occurrences[element] = 1

return occurrences

# Example usage

input\_list = [1, 2, 3, 2, 1, 3, 4, 5, 4, 4, 4]

occurrences = count\_occurrences(input\_list)

print("Element occurrences:", occurrences)

1. 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:

second\_largest = lst[i]

return second\_largest

# Example usage

input\_list = [12, 45, 67, 23, 9, 100]

second\_largest = find\_second\_largest(input\_list)

if second\_largest is not None:

print("The second largest number is:", second\_largest)

else:

print("There is no second largest number.")

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

def remove\_duplicates(lst):

return list(set(lst))

# Example usage

input\_list = [1, 2, 3, 2, 1, 4, 5, 4, 4]

unique\_list = remove\_duplicates(input\_list)

print("List with duplicates removed:", unique\_list)

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

def factorial(n):

if n < 0:

return None

elif n == 0:

return 1

else:

result = 1

for i in range(1, n + 1):

result \*= i

return result

# Example usage

input\_number = int(input("Enter a number: "))

factorial\_result = factorial(input\_number)

if factorial\_result is not None:

print("Factorial of", input\_number, "is:", factorial\_result)

else:

print("Factorial cannot be calculated for a negative number.

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

def is\_prime(number):

if number < 2:

return False

for i in range(2, int(number \*\* 0.5) + 1):

if number % i == 0:

return False

return True

# Example usage

input\_number = int(input("Enter a number: "))

if is\_prime(input\_number):

print(input\_number, "is a prime number.")

else:

print(input\_number, "is not a prime number.")

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

def sort\_list(lst):

sorted\_list = lst.copy()

sorted\_list.sort()

return sorted\_list

# Example usage

input\_list = [5, 2, 8, 1, 9, 3]

sorted\_list = sort\_list(input\_list)

print("Sorted list:", sorted\_list)

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

def find\_sum(lst):

total\_sum = 0

for num in lst:

total\_sum += num

return total\_sum

# Example usage

input\_list = [1, 2, 3, 4, 5]

sum\_of\_numbers = find\_sum(input\_list)

print("Sum of numbers:", sum\_of\_number

1. 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 and element not in common\_elements:

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:", common\_elements)

1. 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(" ", "")

if len(str1) != len(str2):

return False

char\_count = {}

for char in str1:

if char in char\_count:

char\_count[char] += 1

else:

char\_count[char] = 1

for char in str2:

if char in char\_count:

char\_count[char] -= 1

if char\_count[char] == 0:

del char\_count[char]

else:

return False

return len(char\_count) == 0

# Example usage

string1 = input("Enter the first string: ")

string2 = input("Enter the second string: ")

if is\_anagram(string1, string2):

print("The strings are anagrams.")

else:

print("The strings are not anagrams.")

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

import itertools

def generate\_permutations(string):

# Convert the string into a list of characters

characters = list(string)

# Generate all permutations using itertools.permutations()

permutations = list(itertools.permutations(characters))

# Convert each permutation back to a string

permutations = [''.join(permutation) for permutation in permutations]

return permutation

# Test the function

input\_string = input("Enter a string: ")

permutations = generate\_permutations(input\_string)

print("All permutations:")

for permutation in permutations:

print(permutation)

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

def fibonacci\_sequence(num\_terms):

# Check if the number of terms is 0 or negative

if num\_terms <= 0:

return []

sequence = [0, 1] # Initialize the sequence with the first two terms

# Calculate the Fibonacci sequence

while len(sequence) < num\_terms:

next\_term = sequence[-1] + sequence[-2]

sequence.append(next\_term)

return sequence

# Test the function

num\_terms = int(input("Enter the number of terms: "))

fib\_sequence = fibonacci\_sequence(num\_terms)

print("Fibonacci sequence:")

print(fib\_sequence)

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

def find\_median(numbers):

sorted\_numbers = sorted(numbers)

length = len(sorted\_numbers)

if length % 2 == 0:

# If the length is even, calculate the average of the middle two elements

mid = length // 2

median = (sorted\_numbers[mid - 1] + sorted\_numbers[mid]) / 2

else:

# If the length is odd, the median is the middle element

mid = length // 2

median = sorted\_numbers[mid]

return median

# Test the function

num\_list = [2, 4, 1, 5, 3]

median = find\_median(num\_list)

print("Median:", median)

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

def is\_sorted(lst):

# Iterate over the list from the second element

for i in range(1, len(lst)):

# If the current element is less than the previous element, the list is not sorted

if lst[i] < lst[i-1]:

return False

# If the loop completes without finding any out-of-order elements, the list is sorted

return True

# Test the function

num\_list = [1, 2, 3, 4, 5]

print(is\_sorted(num\_list)) # True

num\_list = [1, 3, 2, 4, 5]

print(is\_sorted(num\_list)) # False

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

def find\_intersection(list1, list2):

# Convert the lists to sets to eliminate duplicate elements

set1 = set(list1)

set2 = set(list2)

# Find the common elements using set intersection

intersection = set1.intersection(set2)

return list(intersection)

# Test the function

list1 = [1, 2, 3, 4, 5]

list2 = [4, 5, 6, 7, 8]

intersection = find\_intersection(list1, list2)

print("Intersection:", intersection)

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

def find\_max\_subarray\_sum(lst):

current\_sum = 0

max\_sum = float('-inf') # Initialize max\_sum as negative infinity

for num in lst:

current\_sum = max(num, current\_sum + num)

max\_sum = max(max\_sum, current\_sum)

return max\_sum

# Test the function

num\_list = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

max\_sum = find\_max\_subarray\_sum(num\_list)

print("Maximum Subarray Sum:", max\_sum)

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

def remove\_vowels(string):

vowels = 'aeiouAEIOU'

result = ''

for char in string:

if char not in vowels:

result += char

return result

# Test the function

input\_string = input("Enter a string: ")

new\_string = remove\_vowels(input\_string)

print("String after removing vowels:", new\_string)

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

def reverse\_sentence(sentence):

words = sentence.split() # Split the sentence into words

reversed\_words = words[::-1] # Reverse the order of words

reversed\_sentence = ' '.join(reversed\_words) # Join the reversed words into a sentence

return reversed\_sentence

# Test the function

input\_sentence = input("Enter a sentence: ")

reversed\_sentence = reverse\_sentence(input\_sentence)

print("Reversed sentence:", reversed\_sentence)

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

def are\_anagrams(str1, str2):

# Remove spaces and convert both strings to lowercase

str1 = str1.replace(" ", "").lower()

str2 = str2.replace(" ", "").lower()

# Check if the lengths of the strings are equal

if len(str1) != len(str2):

return False

# Convert strings to lists and sort them

str1\_list = sorted(list(str1))

str2\_list = sorted(list(str2))

# Check if the sorted lists are equal

if str1\_list == str2\_list:

return True

else:

return False

# Example usage

string1 = "listen"

string2 = "silent"

if are\_anagrams(string1, string2):

print(f"{string1} and {string2} are anagrams.")

else:

print(f"{string1} and {string2} are not anagrams.")

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

def find\_first\_non\_repeating\_char(string):

# Create a dictionary to store character frequencies

char\_freq = {}

# Traverse the string and count the frequency of each character

for char in string:

if char in char\_freq:

char\_freq[char] += 1

else:

char\_freq[char] = 1

# Find the first non-repeating character

for char in string:

if char\_freq[char] == 1:

return char

# If no non-repeating character is found, return None

return None

# Example usage

input\_string = "hello"

result = find\_first\_non\_repeating\_char(input\_string)

if result:

print(f"The first non-repeating character in '{input\_string}' is '{result}'.")

else:

print(f"There are no non-repeating characters in '{input\_string}'.")

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

def find\_prime\_factors(number):

prime\_factors = []

divisor = 2

while divisor <= number:

if number % divisor == 0:

prime\_factors.append(divisor)

number /= divisor

else:

divisor += 1

return prime\_factors

# Example usage

input\_number = 56

result = find\_prime\_factors(input\_number)

if len(result) > 0:

print(f"The prime factors of {input\_number} are: {result}")

else:

print(f"{input\_number} has no prime factors.")

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

def is\_power\_of\_two(number):

if number <= 0:

return False

# A number is a power of two if and only if it has a single set bit (binary representation)

# Using bitwise operations, we can check if the number is a power of two

return (number & (number - 1)) == 0

# Example usage

input\_number = 16

if is\_power\_of\_two(input\_number):

print(f"{input\_number} is a power of two.")

else:

print(f"{input\_number} is not a power of two."

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

def merge\_sorted\_lists(list1, list2):

merged\_list = []

i = j = 0

# Merge the two lists while maintaining the sorted order

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

# Add the remaining elements from list1 (if any)

while i < len(list1):

merged\_list.append(list1[i])

i += 1

# Add the remaining elements from list2 (if any)

while j < len(list2):

merged\_list.append(list2[j])

j += 1

return merged\_list

# Example usage

list1 = [1, 3, 5, 7, 9]

list2 = [2, 4, 6, 8, 10]

merged = merge\_sorted\_lists(list1, list2)

print("Merged List:", merged)

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

from collections import Counter

def find\_mode(numbers):

# Count the occurrences of each number in the list

count = Counter(numbers)

# Find the mode(s) with the highest count

mode = max(count.values())

# Create a list to store the mode(s)

mode\_list = [num for num, freq in count.items() if freq == mode]

return mode\_list

# Example usage

number\_list = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]

result = find\_mode(number\_list)

if len(result) == 1:

print(f"The mode of the list is: {result[0]}")

else:

print(f"The modes of the list are: {', '.join(str(num) for num in result)}")

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

def find\_gcd(a, b):

# Ensure a is greater than or equal to b

if a < b:

a, b = b, a

while b != 0:

# Calculate the remainder

remainder = a % b

# Swap a and b, and update the remainder

a = b

b = remainder

return a

# Example usage

number1 = 36

number2 = 48

gcd = find\_gcd(number1, number2)

print(f"The GCD of {number1} and {number2} is: {gcd}")

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

def calculate\_square\_root(number):

if number < 0:

return None

# Initial guess for the square root

guess = number /

# Keep improving the guess until it converges

while True:

new\_guess = 0.5 \* (guess + number / guess)

if abs(guess - new\_guess) < 1e-9: # Adjust the desired precision as needed

return new\_guess

guess = new\_guess

# Example usage

input\_number = 25

result = calculate\_square\_root(input\_number)

if result is not None:

print(f"The square root of {input\_number} is: {result}")

else:

print("Invalid input: Cannot calculate square root of a negative number.")

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

import re

def is\_valid\_palindrome(string):

# Remove non-alphanumeric characters from the string and convert it to lowercase

cleaned\_string = re.sub(r'\W+', '', string.lower())

# Check if the cleaned string is a palindrome

return cleaned\_string == cleaned\_string[::-1]

# Example usage

input\_string = "A man, a plan, a canal: Panama!"

if is\_valid\_palindrome(input\_string):

print(f"The string '{input\_string}' is a valid palindrome.")

else:

print(f"The string '{input\_string}' is not a valid palindrome.")

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

def find\_minimum\_element(nums):

left = 0

right = len(nums) - 1

# If the list is not rotated, the first element is the minimum

if nums[left] <= nums[right]:

return nums[left]

# Perform binary search to find the minimum element

while left < right:

mid = (left + right) // 2

# Check if mid element is greater than its next element

if nums[mid] > nums[mid + 1]:

return nums[mid + 1]

# Check if mid element is smaller than its previous element

if nums[mid] < nums[mid - 1]:

return nums[mid]

# If mid element is greater than the leftmost element,

# the minimum element is on the right side

if nums[mid] >= nums[left]:

left = mid + 1

# If mid element is smaller than the rightmost element,

# the minimum element is on the left side

else:

right = mid - 1

# If the loop ends, left and right pointers coincide

return nums[left]

# Example usage

nums = [4, 5, 6, 7, 0, 1, 2]

minimum = find\_minimum\_element(nums)

print(f"The minimum element in the list is: {minimum}")

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

def sum\_even\_numbers(numbers):

sum\_even = 0

for num in numbers:

if num % 2 == 0:

sum\_even += num

return sum\_even

# Example usage

number\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

sum\_even = sum\_even\_numbers(number\_list)

print(f"The sum of even numbers in the list is: {sum\_even}")

1. 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 / power(base, -exponent)

# Example usage

base = 2

exponent = 3

result = power(base, exponent)

print(f"{base} raised to the power of {exponent} is: {result}"

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

def remove\_duplicates\_preserve\_order(lst):

unique\_list = []

seen = set()

for item in lst:

if item not in seen:

unique\_list.append(item)

seen.add(item)

return unique\_list

# Example usage

input\_list = [1, 2, 3, 2, 4, 1, 5, 6, 5, 4]

result = remove\_duplicates\_preserve\_order(input\_list)

print("List with duplicates removed:", result)

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

def find\_longest\_common\_prefix(strings):

if not strings:

return ""

# Find the minimum length string in the list

min\_len = min(len(string) for string in strings)

# Initialize the longest common prefix

prefix = ""

# Iterate over the characters in the minimum length string

for i in range(min\_len):

# Get the current character

current\_char = strings[0][i]

# Check if the current character is common to all strings

if all(string[i] == current\_char for string in strings):

prefix += current\_char

else:

break

return prefix

# Example usage

string\_list = ["flower", "flow", "flight"]

common\_prefix = find\_longest\_common\_prefix(string\_list)

print(f"The longest common prefix is: {common\_prefix}")

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

def is\_perfect\_square(number):

if number < 0:

return False

# Find the square root of the number

square\_root = int(number \*\* 0.5)

# Check if the square of the square root is equal to the number

return square\_root \* square\_root == number

# Example usage

input\_number = 16

if is\_perfect\_square(input\_number):

print(f"{input\_number} is a perfect square.")

else:

print(f"{input\_number} is not a perfect square.")

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

def calculate\_product(numbers):

product = 1

for num in numbers:

product \*= num

return product

# Example usage

number\_list = [2, 3, 4, 5]

product = calculate\_product(number\_list)

print(f"The product of all elements in the list is: {product}")

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

def reverse\_sentence(sentence):

# Split the sentence into words

words = sentence.split()

# Reverse the order of words

reversed\_words = words[::-1]

# Join the reversed words back into a sentence

reversed\_sentence = ' '.join(reversed\_words)

return reversed\_sentence

# Example usage

input\_sentence = "Hello, how are you?"

reversed\_sentence = reverse\_sentence(input\_sentence)

print("Reversed sentence:", reversed\_sentence)

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

def find\_missing\_number(numbers):

# Calculate the expected sum of consecutive numbers from 1 to n

n = len(numbers) + 1

expected\_sum = n \* (n + 1) // 2

# Calculate the actual sum of numbers in the list

actual\_sum = sum(numbers)

# The missing number is the difference between the expected sum and the actual sum

missing\_number = expected\_sum - actual\_sum

return missing\_number

# Example usage

number\_list = [1, 2, 3, 5, 6, 7, 8, 9]

missing\_number = find\_missing\_number(number\_list)

print(f"The missing number in the list is: {missing\_number}")

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

def sum\_of\_digits(number):

# Convert the number to a string to iterate over its digits

number\_str = str(number)

# Initialize the sum of digits

digit\_sum = 0

# Iterate over each digit and add it to the sum

for digit in number\_str:

digit\_sum += int(digit)

return digit\_sum

# Example usage

input\_number = 12345

digit\_sum = sum\_of\_digits(input\_number)

print(f"The sum of digits in {input\_number} is: {digit\_sum}")

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

def is\_valid\_palindrome(string):

# Remove spaces and convert the string to lowercase

cleaned\_string = ''.join(string.split()).lower()

# Check if the cleaned string is equal to its reverse

return cleaned\_string == cleaned\_string[::-1]

# Example usage

input\_string = "A man, a plan, a canal: Panama"

if is\_valid\_palindrome(input\_string):

print(f"The string '{input\_string}' is a valid palindrome.")

else:

print(f"The string '{input\_string}' is not a valid palindrome.")

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

def find\_smallest\_missing\_positive\_integer(nums):

# Step 1: Move all non-positive numbers to the left side of the list

i = 0

n = len(nums)

while i < n:

if 1 <= nums[i] <= n and nums[i] != nums[nums[i] - 1]:

# Swap the number at i with the number at its correct position

nums[nums[i] - 1], nums[i] = nums[i], nums[nums[i] - 1]

else:

i += 1

# Step 2: Find the first index where the number is not equal to its index + 1

for i in range(n):

if nums[i] != i + 1:

return i + 1

# If all numbers from 1 to n are present, the smallest missing positive integer is n + 1

return n + 1

# Example usage

number\_list = [3, 4, -1, 1]

smallest\_missing = find\_smallest\_missing\_positive\_integer(number\_list)

print(f"The smallest missing positive integer in the list is: {smallest\_missing}")

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

def find\_longest\_palindrome\_substring(string):

if len(string) < 2:

return string

start = 0

max\_length = 1

for i in range(len(string)):

# Check for odd-length palindromes

left = i - 1

right = i + 1

while left >= 0 and right < len(string) and string[left] == string[right]:

left -= 1

right += 1

length = right - left - 1

if length > max\_length:

start = left + 1

max\_length = length

# Check for even-length palindromes

left = i

right = i + 1

while left >= 0 and right < len(string) and string[left] == string[right]:

left -= 1

right += 1

length = right - left - 1

if length > max\_length:

start = left + 1

max\_length = length

return string[start:start + max\_length]

# Example usage

input\_string = "babad"

longest\_palindrome = find\_longest\_palindrome\_substring(input\_string)

print(f"The longest palindrome substring is: {longest\_palindrome}")

1. 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

# Example usage

number\_list = [1, 2, 3, 2, 4, 1, 5, 6, 5, 4]

element = 2

occurrences = count\_occurrences(number\_list, element)

print(f"The number of occurrences of {element} in the list is: {occurrences}")

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

def is\_perfect\_number(number):

if number <= 0:

return False

divisors = [1] # Start with 1 as a divisor

sum\_of\_divisors = 1 # Initialize sum of divisors as 1

# Find all divisors of the number (excluding the number itself)

for i in range(2, int(number\*\*0.5) + 1):

if number % i == 0:

divisors.append(i)

if i != number // i:

divisors.append(number // i)

# Calculate the sum of the divisors

sum\_of\_divisors = sum(divisors)

# Check if the sum of divisors equals the number

return sum\_of\_divisors == number

# Example usage

input\_number = 28

if is\_perfect\_number(input\_number):

print(f"{input\_number} is a perfect number.")

else:

print(f"{input\_number} is not a perfect number.")

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

def remove\_duplicates(string):

unique\_chars = set()

result = []

for char in string:

if char not in unique\_chars:

result.append(char)

unique\_chars.add(char)

return ''.join(result)

# Example usage

input\_string = "Hello, world!"

result\_string = remove\_duplicates(input\_string)

print("String with duplicates removed:", result\_string)

1. Implement a function to find the first missing positive.

def find\_first\_missing\_positive(nums):

n = len(nums)

# Step 1: Move all non-positive numbers to the left side of the list

i = 0

while i < n:

if 1 <= nums[i] <= n and nums[i] != nums[nums[i] - 1]:

# Swap the number at i with the number at its correct position

nums[nums[i] - 1], nums[i] = nums[i], nums[nums[i] - 1]

else:

i += 1

# Step 2: Find the first index where the number is not equal to its index + 1

for i in range(n):

if nums[i] != i + 1:

return i + 1

# If all numbers from 1 to n are present, the first missing positive is n + 1

return n + 1

# Example usage

number\_list = [3, 4, -1, 1]

first\_missing\_positive = find\_first\_missing\_positive(number\_list)

print(f"The first missing positive integer is: {first\_missing\_positive}")