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

Program:

def reverse\_string(string):

reversed\_string = ""

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

reversed\_string += string[i]

return reversed\_string

input\_string = "Hello, World!"

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

print(reversed\_string)

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

Program:

def is\_palindrome(string):

reversed\_string = reverse\_string(string)

return string == reversed\_string

input\_string = "madam"

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):

largest = lst[0]

for num in lst:

if num > largest:

largest = num

return largest

numbers = [10, 7, 23, 4, 15, 9]

largest\_number = find\_largest\_element(numbers)

print("The largest number is:", largest\_number)

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

def count\_occurrences(lst):

occurrence\_count = {}

for item in lst:

if item in occurrence\_count:

occurrence\_count[item] += 1

else:

occurrence\_count[item] = 1

return occurrence\_count

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

occurrence\_count = count\_occurrences(numbers)

print("Occurrence count:", occurrence\_count)

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

def find\_second\_largest(lst):

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, 7, 23, 4, 15, 9]

second\_largest\_number = find\_second\_largest(numbers)

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

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

def remove\_duplicates(lst):

unique\_list = []

for item in lst:

if item not in unique\_list:

unique\_list.append(item)

return unique\_list

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

unique\_numbers = remove\_duplicates(numbers)

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

1. 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)

number = 5

factorial\_result = factorial(number)

print("Factorial of", number, "is", factorial\_result)

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

def is\_prime(number):

if number <= 1:

return False

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

if number % i == 0:

return False

return True

number = 17

if is\_prime(number):

print(number, "is a prime number.")

else:

print(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[:]

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

numbers = [5, 2, 9, 1, 7, 3]

sorted\_numbers = sort\_list(numbers)

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

1. 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

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

sum\_of\_numbers = find\_sum(numbers)

print("Sum of numbers:", sum\_of\_numbers)

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

def find\_common\_elements(list1, list2):

common\_elements = []

for item in list1:

if item in list2 and item not in common\_elements:

common\_elements.append(item)

return common\_elements

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.replace(" ", "").lower()

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

return sorted(str1) == sorted(str2)

string1 = "listen"

string2 = "silent"

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.

def permutations(string):

if len(string) == 1:

return [string]

perms = []

for i in range(len(string)):

char = string[i]

remaining = string[:i] + string[i+1:]

for perm in permutations(remaining):

perms.append(char + perm)

return perms

string = "abc"

string\_perms = permutations(string)

print("Permutations of", string, ":", string\_perms)

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

def fibonacci(n):

sequence = [0, 1]

if n <= 1:

return sequence[:n+1]

while len(sequence) < n+1:

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

sequence.append(next\_num)

return sequence

num\_terms = 10

fibonacci\_sequence = fibonacci(num\_terms)

print("Fibonacci sequence:", fibonacci\_sequence)

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

def find\_median(lst):

sorted\_list = sorted(lst)

n = len(sorted\_list)

if n % 2 == 0:

mid1 = n // 2

mid2 = mid1 - 1

median = (sorted\_list[mid1] +Certainly! Here are the solutions to the remaining problems:

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

```python

def find\_median(lst):

sorted\_list = sorted(lst)

n = len(sorted\_list)

if n % 2 == 0:

mid1 = n // 2

mid2 = mid1 - 1

median = (sorted\_list[mid1] + sorted\_list[mid2]) / 2

else:

mid = n // 2

median = sorted\_list[mid]

return median

numbers = [7, 2, 10, 5, 4, 8]

median = find\_median(numbers)

print("The median is:", median)

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

def find\_intersection(list1, list2):

return list(set(list1) & set(list2))

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

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

intersection = find\_intersection(list1, list2)

print("Intersection:", intersection)

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

def find\_intersection(list1, list2):

return list(set(list1) & set(list2))

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 = max\_sum = lst[0]

for num in lst[1:]:

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]

max\_sum = find\_max\_subarray\_sum(numbers)

print("Maximum subarray sum:", max\_sum)

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

def remove\_vowels(string):

vowels = ['a', 'e', 'i', 'o', 'u']

return ''.join(char for char in string if char.lower() not in vowels)

input\_string = "Hello, World!"

result = remove\_vowels(input\_string)

print("String without vowels:", result)

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

def reverse\_words(sentence):

words = sentence.split()

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

return reversed\_words

input\_sentence = "Hello, World! Welcome to OpenAI."

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

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

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

def is\_anagram(str1, str2):

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

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

return sorted(str1) == sorted(str2)

string1 = "listen"

string2 = "silent"

if is\_anagram(string1, string2):

print("The strings are anagrams.")

else:

print("The strings are not anagrams.")

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

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

char\_count = {}

for char in string:

if char in char\_count:

char\_count[char] += 1

else:

char\_count[char] = 1

for char in string:

if char\_count[char] == 1:

return char

return None

input\_string = "abracadabra"

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

if non\_repeating\_char:

print("First non-repeating character:", non\_repeating\_char)

else:

print("No non-repeating character found.")

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 = number // divisor

else:

divisor += 1

return prime\_factors

number = 84

prime\_factors = find\_prime\_factors(number)

print("Prime factors of", number, "are:", 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

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

number = 16

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

print(number, "is a power of two.")

else:

print(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

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

merged\_list.extend(list1[i:])

merged\_list.extend(list2[j:])

return merged\_list

list1 = [1, 3, 5, 7]

list2 = [2, 4, 6, 8]

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

print("Merged and sorted list:", merged\_list)

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

from collections import Counter

def find\_mode(lst):

counts = Counter(lst)

max\_count = max(counts.values())

mode = [num for num, count in counts.items() if count == max\_count]

return mode

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

mode = find\_mode(numbers)

print("Mode(s):", mode)

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

def find\_gcd(a, b):

while b:

a, b = b, a % b

return a

num1 = 36

num2 = 48

gcd = find\_gcd(num1, num2)

print("GCD of", num1, "and", num2, "is:", gcd)

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

def square\_root(number):

if number < 0:

return None

if number == 0:

return 0

guess = number

while True:

new\_guess = (guess + number / guess) / Apologies, I ran into the response character limit. Here are the remaining solutions:

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

```python

def square\_root(number):

if number < 0:

return None

if number == 0:

return 0

guess = number

while True:

new\_guess = (guess + number / guess) / 2

if abs(new\_guess - guess) < 0.0001:

return new\_guess

guess = new\_guess

number = 25

sqrt = square\_root(number)

print("Square root of", number, "is:", sqrt)

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

def is\_valid\_palindrome(string):

alphanumeric\_string = ''.join(char.lower() for char in string if char.isalnum())

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

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

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

print("The string is a valid palindrome.")

else:

print("The string is not a valid palindrome.")

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

def find\_minimum(lst):

left = 0

right = len(lst) - 1

while left < right:

mid = (left + right) // 2

if lst[mid] > lst[right]:

left = mid + 1

else:

right = mid

return lst[left]

numbers = [5, 6, 7, 8, 9, 1, 2, 3, 4]

minimum = find\_minimum(numbers)

print("Minimum element:", minimum)

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

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 even numbers:", sum\_of\_evens)

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

def power(base, exponent):

if exponent == 0:

return 1

if exponent < 0:

return 1 / power(base, -exponent)

return base \* power(base, exponent-1)

base = 2

exponent = 3

result = power(base, exponent)

print(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(lst):

unique\_list = []

for item in lst:

if item not in unique\_list:

unique\_list.append(item)

return unique\_list

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

unique\_numbers = remove\_duplicates(numbers)

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

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

def longest\_common\_prefix(strs):

if not strs:

return ""

shortest = min(strs, key=len)

for i, char in enumerate(shortest):

if any(string[i] != char for string in strs):

return shortest[:i]

return shortest

strings = ["flower", "flow", "flight"]

common\_prefix = longest\_common\_prefix(strings)

print("Longest common prefix:", 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

root = int(number \*\* 0.5)

return root \* root == number

number = 25

if is\_perfect\_square(number):

print(number, "is a perfect square.")

else:

print(number, "is not a perfect square.")

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

def find\_product(lst):

product = 1

for num in lst:

product \*= num

return product

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

product = find\_product(numbers)

print("Product of numbers:", product)

1. 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

input\_sentence = "Hello, World! Welcome to OpenAI."

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

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

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

def sum\_of\_digits(number):

digit\_sum = 0

while number > 0:

digit = number % 10

digit\_sum += digit

number = number // 10

return digit\_sum

number = 12345

digit\_sum = sum\_of\_digits(number)

print("Sum of digits:", digit\_sum)

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

def find\_smallest\_missing\_positive(lst):

n = len(lst)

for i in range(n):

while 1 <= lst[i] <= n and lst[lst[i] - 1] != lst[i]:

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]

smallest\_missing = find\_smallest\_missing\_positive(numbers)

print("Smallest missing positive integer:", smallest\_missing)

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

def find\_smallest\_missing\_positive(lst):

n = len(lst)

for i in range(n):

while 1 <= lst[i] <= n and lst[lst[i] - 1] != lst[i]:

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]

smallest\_missing = find\_smallest\_missing\_positive(numbers)

print("Smallest missing positive integer:", smallest\_missing)

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

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

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

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

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

n = len(lst)

for i in range(n):

while 1 <= lst[i] <= n and lst[lst[i] - 1] != lst[i]:

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 integer:", first\_missing\_positive)

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

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

n = len(lst)

for i in range(n):

while 1 <= lst[i] <= n and lst[lst[i] - 1] != lst[i]:

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 integer:", first\_missing\_positive)

46. Implement a function to find the first missing positive

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

n = len(lst)

for i in range(n):

while 1 <= lst[i] <= n and lst[lst[i] - 1] != lst[i]:

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 integer:", first\_missing\_positive)