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

Ans: Certainly! Here's a Python program that reverses a string without using any built-in string reversal functions:

```python

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\_to\_reverse = input("Enter a string to reverse: ")

reversed\_string = reverse\_string(string\_to\_reverse)

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

```

In this program, we define a function called `reverse\_string` that takes an input string as a parameter. We initialize an empty string called `reversed\_string`. Then, using a for loop, we iterate over the characters of the input string in reverse order, starting from the last character (`len(input\_string) - 1`) and going down to the first character (`0`) with a step of `-1`. Inside the loop, we append each character to the `reversed\_string` variable. Finally, we return the reversed string.

You can test this program by entering a string, and it will output the reversed version of that string.

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

Ans: Sure! Here's a Python function that checks if a given string is a palindrome:

```python

def is\_palindrome(input\_string):

# Remove any non-alphanumeric characters and convert to lowercase

cleaned\_string = ''.join(c.lower() for c in input\_string if c.isalnum())

# Check if the cleaned string is equal to its reverse

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

# Example usage

string\_to\_check = input("Enter a string to check if it's a palindrome: ")

if is\_palindrome(string\_to\_check):

print("The string is a palindrome.")

else:

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

```

In this program, we define a function called `is\_palindrome` that takes an input string as a parameter. First, we clean the string by removing any non-alphanumeric characters using a generator expression and the `isalnum()` method. We also convert the cleaned string to lowercase to make the comparison case-insensitive.

Then, we check if the cleaned string is equal to its reverse using the slicing syntax `[::-1]`, which creates a reversed version of the string. If the cleaned string and its reverse are equal, it means the original string is a palindrome, and the function returns `True`. Otherwise, it returns `False`.

You can test this function by entering a string, and it will output whether the string is a palindrome or not.

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

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

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

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

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

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

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

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

Ans 3 to 10 no’s:

Sure! Here are the solutions to the programming tasks you provided:

3. Program to find the largest element in a given list:

```python

def find\_largest\_element(lst):

largest = lst[0]

for num in lst:

if num > largest:

largest = num

return largest

# Example usage

numbers = [5, 2, 8, 10, 3]

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

print("Largest number:", largest\_num)

```

4. Function to count the occurrence of each element in a list:

```python

from collections import Counter

def count\_occurrences(lst):

count = Counter(lst)

return count

# Example usage

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

occurrences = count\_occurrences(numbers)

print("Number of occurrences:", occurrences)

```

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

```python

def find\_second\_largest(lst):

largest = float('-inf')

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

# Example usage

numbers = [5, 2, 8, 10, 3]

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

print("Second largest number:", second\_largest\_num)

```

6. Function to remove duplicate elements from a list:

```python

def remove\_duplicates(lst):

return list(set(lst))

# Example usage

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

unique\_numbers = remove\_duplicates(numbers)

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

```

7. Program to calculate the factorial of a given number:

```python

def calculate\_factorial(n):

if n == 0:

return 1

else:

return n \* calculate\_factorial(n - 1)

# Example usage

number = 5

factorial = calculate\_factorial(number)

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

```

8. Function to check if a given number is prime:

```python

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

# Example usage

number = 17

if is\_prime(number):

print(number, "is prime")

else:

print(number, "is not prime")

```

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

```python

def sort\_list(lst):

return sorted(lst)

# Example usage

numbers = [5, 2, 8, 10, 3]

sorted\_numbers = sort\_list(numbers)

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

```

10. Function to find the sum of all numbers in a list:

```python

def find\_sum(lst):

return sum(lst)

# Example usage

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

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

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

```

Feel free to run these programs and modify them as per your needs.

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

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

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

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

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

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

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

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

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

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

Ans 11 to 20 no”s:

11. Program to find the common elements between two lists:

```python

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

common\_elements = set(list1) & set(list2)

return list(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)

```

12. Function to check if a given string is an anagram of another string:

```python

from collections import Counter

def is\_anagram(str1, str2):

count\_str1 = Counter(str1)

count\_str2 = Counter(str2)

return count\_str1 == count\_str2

# Example usage

string1 = "listen"

string2 = "silent"

if is\_anagram(string1, string2):

print(string1, "and", string2, "are anagrams")

else:

print(string1, "and", string2, "are not anagrams")

```

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

```python

import itertools

def generate\_permutations(string):

permutations = list(itertools.permutations(string))

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

# Example usage

string = "abc"

permutations = generate\_permutations(string)

print("Permutations:", permutations)

```

14. Function to calculate the Fibonacci sequence up to a given number of terms:

```python

def fibonacci\_sequence(n):

sequence = [0, 1]

if n <= 2:

return sequence[:n]

while len(sequence) < n:

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

sequence.append(next\_number)

return sequence

# Example usage

terms = 8

fibonacci = fibonacci\_sequence(terms)

print("Fibonacci sequence:", fibonacci)

```

15. Program to find the median of a list of numbers:

```python

def find\_median(numbers):

numbers.sort()

n = len(numbers)

if n % 2 == 0:

median = (numbers[n//2 - 1] + numbers[n//2]) / 2

else:

median = numbers[n//2]

return median

# Example usage

numbers = [5, 2, 8, 10, 3]

median = find\_median(numbers)

print("Median:", median)

```

16. Function to check if a given list is sorted in non-decreasing order:

```python

def is\_sorted(lst):

return all(lst[i] <= lst[i+1] for i in range(len(lst)-1))

# Example usage

numbers1 = [1, 2, 3, 4, 5]

numbers2 = [5, 2, 8, 10, 3]

if is\_sorted(numbers1):

print("List 1 is sorted")

else:

print("List 1 is not sorted")

if is\_sorted(numbers2):

print("List 2 is sorted")

else:

print("List 2 is not sorted")

```

17. Python program to find the intersection of two lists:

```python

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:", intersection)

```

18. Function to find the maximum subarray sum in a given list:

```python

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

max\_sum = current\_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

# Example usage

numbers = [1, -2, 3, 4, -5, 2, 7, -3]

maximum\_sum = find\_maximum\_subarray\_sum(numbers)

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

```

19. Program to remove all vowels from a given string:

```python

def remove\_vowels(string):

vowels = "aeiouAEIOU"

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

# Example usage

string = "Hello, World!"

new\_string = remove\_vowels(string)

print("String without vowels:", new\_string)

```

20. Function to reverse the order of words in a given sentence:

```python

def reverse\_words(sentence):

words = sentence.split()

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

return reversed\_words

# Example usage

sentence = "Hello, World!"

reversed\_sentence = reverse\_words(sentence)

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

```

Feel free to run these programs and modify them as per your needs.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

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

46. Implement a function to find the first missing positive

Ans: 21 to 46 no’s:

Sure! Here are the solutions to the programming tasks you've provided:

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

```python

def are\_anagrams(str1, str2):

return sorted(str1) == sorted(str2)

# Example usage

string1 = "listen"

string2 = "silent"

if are\_anagrams(string1, string2):

print("The strings are anagrams.")

else:

print("The strings are not anagrams.")

```

22. Function to find the first non-repeating character in a string:

```python

def first\_non\_repeating\_char(string):

char\_count = {}

for char in string:

char\_count[char] = char\_count.get(char, 0) + 1

for char in string:

if char\_count[char] == 1:

return char

return None

# Example usage

input\_string = "abracadabra"

result = first\_non\_repeating\_char(input\_string)

if result:

print("The first non-repeating character is:", result)

else:

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

```

23. Program to find the prime factors of a given number:

```python

def find\_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

number = 84

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

print("Prime factors of", number, "are:", prime\_factors)

```

24. Function to check if a given number is a power of two:

```python

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

return n != 0 and (n & (n - 1)) == 0

# Example usage

number = 16

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

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

else:

print(number, "is not a power of two.")

```

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

```python

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

merged\_list = []

i, j = 0, 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

# Example usage

sorted\_list1 = [1, 3, 5, 7]

sorted\_list2 = [2, 4, 6, 8]

merged\_list = merge\_sorted\_lists(sorted\_list1, sorted\_list2)

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

```

26. Function to find the mode of a list of numbers:

```python

from collections import Counter

def find\_mode(numbers):

counter = Counter(numbers)

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

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

return modes

# Example usage

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

mode = find\_mode(number\_list)

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

```

27. Program to find the greatest common divisor (GCD) of two numbers:

```python

def gcd(a, b):

while b:

a, b = b, a % b

return a

# Example usage

num1 = 24

num2 = 36

result = gcd(num1, num2)

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

```

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

```python

def sqrt(n):

if n < 0:

raise ValueError("Square root is not defined for negative numbers.")

if n == 0 or n == 1:

return n

x = n

y = (x + 1) // 2

while y < x:

x = y

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

return x

# Example usage

number = 25

result = sqrt(number)

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

```

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

```python

import re

def is\_valid\_palindrome(string):

# Remove non-alphanumeric characters and convert to lowercase

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

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

# Example usage

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.")

```

30. Function to find the minimum element in a rotated sorted list:

```python

def find\_minimum\_rotated(nums):

left, right = 0, len(nums) - 1

while left < right:

mid = (left + right) // 2

if nums[mid] > nums[right]:

left = mid + 1

else:

right = mid

return nums[left]

# Example usage

rotated\_list = [4, 5, 6, 7, 0, 1, 2]

minimum = find\_minimum\_rotated(rotated\_list)

print("Minimum element in the rotated list:", minimum)

```

31. Program to find the sum of all even numbers in a list:

```python

def sum\_of\_evens(numbers):

return sum(num for num in numbers if num % 2 == 0)

# Example usage

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

sum\_even = sum\_of\_evens(number\_list)

print("Sum of even numbers:", sum\_even)

```

32. Function to calculate the power of a number using recursion:

```python

def power(base, exponent):

if exponent == 0:

return 1

elif exponent < 0:

return 1 / power(base, -exponent)

elif exponent % 2 == 0:

return power(base \* base, exponent // 2)

else:

return base \* power(base \* base, (exponent - 1) // 2)

# Example usage

base = 2

exponent = 5

result = power(base, exponent)

print(base, "raised to the power of", exponent, "is:", result)

```

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

```python

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

seen = set()

return [x for x in lst if not (x in seen or seen.add(x))]

# Example usage

original\_list = [1, 2, 3, 2, 4

, 3, 1]

deduplicated\_list = remove\_duplicates\_preserve\_order(original\_list)

print("Deduplicated list:", deduplicated\_list)

```

34. Function to find the longest common prefix among a list of strings:

```python

def longest\_common\_prefix(strings):

if not strings:

return ""

min\_length = min(len(s) for s in strings)

prefix = ""

for i in range(min\_length):

if all(s[i] == strings[0][i] for s in strings):

prefix += strings[0][i]

else:

break

return prefix

# Example usage

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

common\_prefix = longest\_common\_prefix(string\_list)

print("Longest common prefix:", common\_prefix)

```

35. Program to check if a given number is a perfect square:

```python

def is\_perfect\_square(n):

if n < 0:

return False

if n == 0 or n == 1:

return True

left, right = 1, n // 2

while left <= right:

mid = (left + right) // 2

square = mid \* mid

if square == n:

return True

elif square < n:

left = mid + 1

else:

right = mid - 1

return False

# Example usage

number = 25

if is\_perfect\_square(number):

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

else:

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

```

36. Function to calculate the product of all elements in a list:

```python

def product\_of\_elements(numbers):

product = 1

for num in numbers:

product \*= num

return product

# Example usage

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

product = product\_of\_elements(number\_list)

print("Product of elements:", product)

```

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

```python

def reverse\_words(sentence):

words = sentence.split()

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

return reversed\_sentence

# Example usage

input\_sentence = "Hello, world! This is a sentence."

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

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

```

38. Function to find the missing number in a given list of consecutive numbers:

```python

def find\_missing\_number(numbers):

n = len(numbers) + 1

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

actual\_sum = sum(numbers)

missing\_number = expected\_sum - actual\_sum

return missing\_number

# Example usage

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

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

print("Missing number:", missing\_number)

```

39. Program to find the sum of digits of a given number:

```python

def sum\_of\_digits(number):

return sum(int(digit) for digit in str(number) if digit.isdigit())

# Example usage

number = 12345

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

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

```

40. Function to check if a given string is a valid palindrome considering case sensitivity:

```python

def is\_valid\_palindrome\_case\_sensitive(string):

return string == string[::-1]

# Example usage

input\_string = "level"

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

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

else:

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

```

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

```python

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

n = len(nums)

for i in range(n):

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

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

# Example usage

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

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

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

```

42. Function to find the longest palindrome substring in a given string:

```python

def longest\_palindrome\_substring(string):

if not string:

return ""

def expand\_around\_center(left, right):

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

left -= 1

right += 1

return string[left + 1:right]

longest = ""

for i in range(len(string)):

odd\_palindrome = expand\_around\_center(i, i)

even\_palindrome = expand\_around\_center(i, i + 1)

if len(odd\_palindrome) > len(longest):

longest = odd\_palindrome

if len(even\_palindrome) > len(longest):

longest = even\_palindrome

return longest

# Example usage

input\_string = "babad"

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

print("Longest palindrome substring:", longest\_palindrome)

```

43. Program to find the number of occurrences of a given element in a list:

```python

def count\_occurrences(numbers, target):

count = 0

for num in numbers:

if num == target:

count += 1

return count

# Example usage

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

target\_number = 2

occurrences = count\_occurrences(number\_list, target\_number)

print("Number of occurrences of", target\_number, ":", occurrences)

```

44. Function to check if a given number is a perfect number:

```python

def is\_perfect\_number(number):

if number <= 0:

return False

divisors = [1]

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)

return sum(divisors) == number

# Example usage

number = 28

if is\_perfect\_number(number):

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

else:

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

```

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

```python

def remove\_duplicates(string):

return ''.join(set(string))

# Example usage

input\_string = "abracadabra"

result = remove\_duplicates(input\_string)

print("String after removing duplicates:", result)

```

46. Function to find the first missing positive integer in a given list:

```python

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

n = len(nums)

for i in range(n):

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

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

# Example usage

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

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

print("First missing positive integer:", first\_missing)

```

These are the solutions to the programming tasks you provided. Hope you find them helpful!