Given two strings s and t, determine if they are isomorphic*.*

Two strings s and t are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

**Example 1:**

**Input:** s = "egg", t = "add"

**Output:** true

**Answer-** def is\_isomorphic(s, t):

if len(s) != len(t):

return False

s\_to\_t = {}

t\_to\_s = {}

for i in range(len(s)):

char\_s = s[i]

char\_t = t[i]

if char\_s in s\_to\_t:

if s\_to\_t[char\_s] != char\_t:

return False

else:

s\_to\_t[char\_s] = char\_t

if char\_t in t\_to\_s:

if t\_to\_s[char\_t] != char\_s:

Return False

else:

t\_to\_s[char\_t] = char\_s

return True

Given a string num which represents an integer, return true if num is a **strobogrammatic number**.

A **strobogrammatic number** is a number that looks the same when rotated 180 degrees (looked at upside down).

**Example 1:**

**Input:** num = "69"

**Output:true**

**Answer- def isStrobogrammatic(num):**

**strobogrammatic\_pairs = {'0': '0', '1': '1', '6': '9', '8': '8', '9': '6'}**

**left, right = 0, len(num) - 1**

**while left <= right:**

**if num[left] not in strobogrammatic\_pairs or num[right] != strobogrammatic\_pairs[num[left]]:**

**return False**

**left += 1**

**right -= 1**

**return True**

Given two non-negative integers, num1 and num2 represented as string, return the sum of num1 and num2 as a string.

You must solve the problem without using any built-in library for handling large integers (such as BigInteger). You must also not convert the inputs to integers directly.

**Example 1:**

**Input:** num1 = "11", num2 = "123"

**Output:** "134"

**Answer-** def addStrings(num1, num2):

result = ""

i = len(num1) - 1

j = len(num2) - 1

carry = 0

while i >= 0 or j >= 0:

digit1 = int(num1[i]) if i >= 0 else 0

digit2 = int(num2[j]) if j >= 0 else 0

temp = digit1 + digit2 + carry

carry = temp // 10

temp %= 10

result = str(temp) + result

i -= 1

j -= 1

if carry > 0:

result = str(carry) + result

return result

Given a string s, reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

**Example 1:**

**Input:** s = "Let's take LeetCode contest"

**Output:** "s'teL ekat edoCteeL tsetnoc"

**Answer-** def reverse\_words(s):

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

reversed\_words = [word[::-1] for word in words] # Reverse each word

reversed\_sentence = ' '.join(reversed\_words) # Join the reversed words with whitespace

return reversed\_sentence

# Example usage

s = "Let's take LeetCode contest"

reversed\_sentence = reverse\_words(s)

print(reversed\_sentence)

Given a string s and an integer k, reverse the first k characters for every 2k characters counting from the start of the string.

If there are fewer than k characters left, reverse all of them. If there are less than 2k but greater than or equal to k characters, then reverse the first k characters and leave the other as original.

**Example 1:**

**Input:** s = "abcdefg", k = 2

**Output:** Given a string s and an integer k, reverse the first k characters for every 2k characters counting from the start of the string.

If there are fewer than k characters left, reverse all of them. If there are less than 2k but greater than or equal to k characters, then reverse the first k characters and leave the other as original.

**Example 1:**

**Input:** s = "abcdefg", k = 2

**Output:**

"bacdfeg

**Answer-** def reverse\_string(s, k):

chars = list(s) # Convert string to a list of characters

n = len(chars)

for i in range(0, n, 2 \* k):

# Reverse the first k characters

chars[i:i+k] = reversed(chars[i:i+k])

return ''join(chars) # Convert the list back to a string

# Example usage

s = "abcdefg"

k = 2

result = reverse\_string(s, k)

print(result) # Output: "bacdfeg"

Given two strings s and goal, return true if and only if s can become goal after some number of **shifts** on s.

A **shift** on s consists of moving the leftmost character of s to the rightmost position.

* For example, if s = "abcde", then it will be "bcdea" after one shift.

**Example 1:**

**Input:** s = "abcde", goal = "cdeab"

**Output:**

true

**Answer-** def can\_shift(s, goal):

if len(s) != len(goal):

return False

# Concatenate s with itself

s += s

# Check if goal is a substring of the concatenated string

if goal in s:

return True

else:

return False

s = "abcde"

goal = "cdeab"

print(can\_shift(s, goal))

True

Given two strings s and t, return true if they are equal when both are typed into empty text editors. '#' means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

**Example 1:**

**Input:** s = "ab#c", t = "ad#c"

**Output:** true

**Explanation:**

Both s and t become "ac".

**Answer-** def processString(s):

stack = []

for char in s:

if char != '#':

stack.append(char)

elif stack:

stack.pop()

return ''.join(stack)

def backspaceCompare(s, t):

processedS = processString(s)

processedT = processString(t)

return processedS == processed

s = "ab#c"

t = "ad#c"

print(backspaceCompare(s, t)) # Output: True