Q.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

Ans: To determine if two strings, `s` and `t`, are isomorphic, we can compare the character mappings between the two strings. For two characters to be considered isomorphic, they should have the same mapping in both strings.

Here's the step-by-step approach to solve this problem:

1. Initialize two dictionaries, `s\_map` and `t\_map`, to store the character mappings of `s` and `t`, respectively.

2. Iterate through each character `s\_char` in `s` and its corresponding character `t\_char` in `t`.

3. Check if `s\_char` is already in `s\_map` and if its mapping is not equal to `t\_char`. If this condition is true, return `False` because the characters are not isomorphic.

4. Check if `t\_char` is already in `t\_map` and if its mapping is not equal to `s\_char`. If this condition is true, return `False` because the characters are not isomorphic.

5. If `s\_char` is not in `s\_map` and `t\_char` is not in `t\_map`, add their mappings to their respective dictionaries.

6. Return `True` if the loop completes without returning `False`.

Here's the implementation in Python:

```python

def isIsomorphic(s, t):

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

return False # Strings with different lengths cannot be isomorphic

s\_map = {}

t\_map = {}

for s\_char, t\_char in zip(s, t):

if s\_char in s\_map and s\_map[s\_char] != t\_char:

return False

if t\_char in t\_map and t\_map[t\_char] != s\_char:

return False

s\_map[s\_char] = t\_char

t\_map[t\_char] = s\_char

return True

```

Let's test it with the given example:

```python

s = "egg"

t = "add"

print(isIsomorphic(s, t))

```

Output:

```

True

```

The function returns `True`, indicating that the strings "egg" and "add" are isomorphic.

Q. 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

Ans: To determine if a string `num` is a strobogrammatic number, we can iterate through the string from both ends and check if the corresponding characters have a valid mapping. The valid mappings for a strobogrammatic number are:

- "0" and "0"

- "1" and "1"

- "6" and "9"

- "8" and "8"

- "9" and "6"

Here's the step-by-step approach to solve this problem:

1. Initialize two pointers, `left` and `right`, to the first and last indices of the string `num`, respectively.

2. While `left` is less than or equal to `right`:

- Check if `num[left]` and `num[right]` form a valid mapping:

- If they don't, return `False` because the string is not a strobogrammatic number.

- If they do, move `left` one step to the right and `right` one step to the left.

3. Return `True` if the loop completes without returning `False`.

Here's the implementation in Python:

```python

def isStrobogrammatic(num):

mappings = {'0': '0', '1': '1', '6': '9', '8': '8', '9': '6'}

left = 0

right = len(num) - 1

while left <= right:

if num[left] not in mappings or num[right] != mappings[num[left]]:

return False

left += 1

right -= 1

return True

```

Let's test it with the given example:

```python

num = "69"

print(isStrobogrammatic(num))

```

Output:

```

True

```

The function returns `True`, indicating that the string "69" is a strobogrammatic number.

Q. 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"

Ans: To add two non-negative integers represented as strings, `num1` and `num2`, we can perform digit-by-digit addition similar to how we add numbers manually. We start from the rightmost digits and move towards the left, adding the corresponding digits and keeping track of any carry.

Here's the step-by-step approach to solve this problem:

1. Initialize an empty string called `result` to store the sum.

2. Initialize two pointers, `i` and `j`, to the rightmost indices of `num1` and `num2`, respectively.

3. Initialize a variable `carry` to 0 to keep track of the carry during addition.

4. While `i` and `j` are greater than or equal to 0:

- Convert the characters at indices `i` and `j` of `num1` and `num2` to integers.

- Add the integers along with the carry and convert the sum to a string.

- Append the rightmost digit of the sum to the `result` string.

- Update the carry by dividing the sum by 10.

- Decrement `i` and `j` by 1.

5. If `i` is greater than or equal to 0:

- Perform the above steps for the remaining digits of `num1` and append the result to the `result` string.

6. If `j` is greater than or equal to 0:

- Perform the above steps for the remaining digits of `num2` and append the result to the `result` string.

7. If the carry is non-zero, append it to the `result` string.

8. Reverse the `result` string to get the correct order of digits.

9. Return the `result` string.

Here's the implementation in Python:

```python

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

sum = digit1 + digit2 + carry

result += str(sum % 10)

carry = sum // 10

i -= 1

j -= 1

if carry > 0:

result += str(carry)

return result[::-1]

```

Let's test it with the given example:

```python

num1 = "11"

num2 = "123"

print(addStrings(num1, num2))

```

Output:

```

"134"

```

The function returns the string "134", which represents the sum of the two non-negative integers `num1` and `num2`.

Q. 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"

Ans: To reverse the order of characters in each word within a sentence while preserving whitespace and the initial word order, we can follow these steps:

1. Split the string `s` into individual words. We can do this by using the `split()` method.

2. Iterate through each word in the list of words.

3. Reverse the characters of each word using string slicing or the `[::-1]` notation.

4. Join the reversed words back together using a space as the delimiter.

5. Return the resulting string.

Here's the implementation in Python:

```python

def reverseWords(s):

words = s.split()

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

return ' '.join(reversed\_words)

```

Let's test it with the given example:

```python

s = "Let's take LeetCode contest"

print(reverseWords(s))

```

Output:

```

"s'teL ekat edoCteeL tsetnoc"

```

The function returns the string `"s'teL ekat edoCteeL tsetnoc"`, which represents the reversed order of characters in each word while preserving whitespace and the initial word order.

Q. 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"

Ans: To reverse the first `k` characters for every `2k` characters counting from the start of the string `s`, we can follow these steps:

1. Convert the string `s` into a list of characters to make it mutable.

2. Iterate through the characters of `s` with a step size of `2k`.

3. For each iteration, reverse the first `k` characters of the sublist starting at the current index.

- If there are fewer than `k` characters remaining, reverse all of them.

4. Convert the modified list of characters back into a string and return it.

Here's the implementation in Python:

```python

def reverseStr(s, k):

chars = list(s)

n = len(chars)

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

left = i

right = min(i + k - 1, n - 1)

while left < right:

chars[left], chars[right] = chars[right], chars[left]

left += 1

right -= 1

return ''.join(chars)

```

Let's test it with the given example:

```python

s = "abcdefg"

k = 2

print(reverseStr(s, k))

```

Output:

```

"bacdfeg"

```

The function returns the string `"bacdfeg"`, which represents the reversed first `k` characters for every `2k` characters counting from the start of the string `s`.

Q. 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

Ans: To determine if one string `s` can become another string `goal` after some number of shifts on `s`, we can check if `goal` is a substring of the concatenated string `s + s`.

Here's the step-by-step approach to solve this problem:

1. Check if the lengths of `s` and `goal` are different. If they are different, return `False` because it is not possible to obtain `goal` from `s` through shifts.

2. Concatenate `s` with itself to create a new string `concat`.

3. Check if `goal` is a substring of `concat`. If it is, return `True`; otherwise, return `False`.

Here's the implementation in Python:

```python

def rotateString(s, goal):

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

return False

concat = s + s

if goal in concat:

return True

return False

```

Let's test it with the given example:

```python

s = "abcde"

goal = "cdeab"

print(rotateString(s, goal))

```

Output:

```

True

```

The function returns `True`, indicating that the string "abcde" can become "cdeab" after some number of shifts.

Q. 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".

Ans: To determine if two strings `s` and `t` are equal when both are typed into empty text editors, considering that '#' represents a backspace character, we can simulate the typing process and compare the resulting strings.

Here's the step-by-step approach to solve this problem:

1. Create helper functions to process each string and return the final result after considering the backspace characters.

2. In each helper function, initialize an empty stack to keep track of the characters.

3. Iterate through each character `c` in the string:

- If `c` is not a backspace character '#', push it onto the stack.

- If `c` is a backspace character '#', check if the stack is empty. If it is not, pop the topmost character from the stack.

4. After processing both strings, compare the resulting strings.

- If they are equal, return `True`; otherwise, return `False`.

Here's the implementation in Python:

```python

def processString(s):

stack = []

for c in s:

if c != '#':

stack.append(c)

elif stack:

stack.pop()

return ''.join(stack)

def backspaceCompare(s, t):

return processString(s) == processString(t)

```

Let's test it with the given example:

```python

s = "ab#c"

t = "ad#c"

print(backspaceCompare(s, t))

```

Output:

```

True

```

The function returns `True`, indicating that the strings "ab#c" and "ad#c" are equal when both are typed into empty text editors considering the backspace characters.