💡 \*\*Question 1\*\*

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

**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 **and** s\_to\_t[char\_s] **!=** char\_t:

**return** **False**

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

**return** **False**

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

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

**return** **True**

s **=** "egg"

t **=** "add"

is\_isomorphic(s,t)

#Output:- True

💡 \*\*Question 2\*\*

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

**def** is\_strobogrammatic(num):

valid\_nums **=** {'0', '1', '6', '8', '9'}

strob\_map **=** {'0': '0', '1': '1', '6': '9', '8': '8', '9': '6'}

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

**while** left **<=** right:

**if** num[left] **not** **in** valid\_nums **or** num[right] **not** **in** valid\_nums:

**return** **False**

**if** num[left] **!=** strob\_map[num[right]]:

**return** **False**

left **+=** 1

right **-=** 1

**return** **True**

num **=** "69"

is\_strobogrammatic(num)

#Output:- True

💡 \*\*Question 3\*\*

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"

**def** add\_strings(num1, num2):

i **=** len(num1) **-** 1

j **=** len(num2) **-** 1

carry **=** 0

result **=** []

**while** i **>=** 0 **or** j **>=** 0 **or** carry **>** 0:

**if** i **>=** 0:

carry **+=** int(num1[i])

i **-=** 1

**if** j **>=** 0:

carry **+=** int(num2[j])

j **-=** 1

result**.**append(str(carry **%** 10))

carry **//=** 10

**return** ''**.**join(result[::**-**1])

num1 **=** "11"

num2 **=** "123"

add\_strings(num1,num2)

#Output:- '134'

💡 \*\*Question 4\*\*

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"

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

words **=** s**.**split()

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

**return** ' '**.**join(reversed\_words)

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

reverse\_words(s)

#Output:- "s'teL ekat edoCteeL tsetnoc"

💡 \*\*Question 5\*\*

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"

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

s **=** list(s)

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

s[i:i**+**k] **=** reversed(s[i:i**+**k])

**return** ''**.**join(s)

s **=** "abcdefg"

k **=** 2

result **=** reverse\_string(s, k)

print(result)

#Output:- bacdfeg

💡 \*\*Question 6\*\*

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

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

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

**return** **False**

**for** \_ **in** range(len(s)):

**if** s **==** goal:

**return** **True**

s **=** s[1:] **+** s[0]

**return** **False**

s **=** "abcde"

goal **=** "cdeab"

result **=** can\_shift(s, goal)

print(result)

#Output:- True

💡 \*\*Question 7\*\*

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".

**def** compare\_strings(s, t):

**def** process\_string(string):

stack **=** []

**for** char **in** string:

**if** char **==** '#':

**if** stack:

stack**.**pop()

**else**:

stack**.**append(char)

**return** ''**.**join(stack)

**return** process\_string(s) **==** process\_string(t)

s **=** "ab#c"

t **=** "ad#c"

result **=** compare\_strings(s, t)

print(result)

#Output:- True

💡 \*\*Question 8\*\*

You are given an array coordinates, coordinates[i] = [x, y], where [x, y] represents the coordinate of a point. Check if these points make a straight line in the XY plane.

**Input:** coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]

**Output:** true

**def** check\_straight\_line(coordinates):

**if** len(coordinates) **<=** 2:

**return** **True**

x0, y0 **=** coordinates[0]

x1, y1 **=** coordinates[1]

**for** i **in** range(2, len(coordinates)):

x, y **=** coordinates[i]

**if** (y1 **-** y0) **\*** (x **-** x0) **!=** (y **-** y0) **\*** (x1 **-** x0):

**return** **False**

**return** **True**

coordinates **=** [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]

result **=** check\_straight\_line(coordinates)

print(result)

#Output:- True