<aside> 💡 **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

</aside>

```
class Solution {
  public boolean isIsomorphic(String s, String t) {
    if(s.length() != t.length()) return false;
    HashMap<Character, Character> map = new HashMap<>();
    for(int i =  0 ; i< s.length(); i++){
      char a= s.charAt(i);
      char b= t.charAt(i);
      if(map.containsKey(a)){
        if(map.get(a) !=b)
          return false;

      }else if(!map.containsKey(a) && !map.containsValue(b)){
      map.put(a,b);
    }else{
      return false;
    }

  }
  return true;
  }

}
```

<aside> 💡 **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

</aside>

```
public boolean isStrobogrammatic(String num) {
    if(num == null || num.length() == 0) return false;
    int left = 0, right = num.length() - 1;
    while(left <= right){
        char c = num.charAt(left);
        if(!map.containsKey(c) || map.get(c) != num.charAt(right)) return false;
        left++;
        right--;
    }
    return true;
}

private Map<Character, Character> map = new HashMap<Character, Character>(){
  {
      put('0', '0');
      put('1', '1');
      put('6', '9');
      put('8', '8');
      put('9', '6');
  }
};
```

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"

```
public class Solution {
    public static String addStrings(String num1, String num2) {
        int i = num1.length() - 1;
        int j = num2.length() - 1;
        int carry = 0;
```

```
        StringBuilder res = new StringBuilder();

        while (i >= 0 || j >= 0 || carry > 0) {
            int x = i >= 0 ? num1.charAt(i) - '0' : 0;
            int y = j >= 0 ? num2.charAt(j) - '0' : 0;

            int sum = x + y + carry;
            carry = sum / 10;
            res.append(sum % 10);

            i--;
            j--;
        }

        if (carry > 0) {
            res.append(carry);
        }

        return res.reverse().toString();
    }
```

💡 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"

</aside>

```
class Solution {
  public String reverseWords(String s) {


    String[] word= s.split(" ");


    for(int i = 0; i<word.length; i++){

      word[i]= reverseWord(word[i]);
    }
     return String.join(" ", word);

  }
  public static String reverseWord(String word){
    StringBuilder str= new StringBuilder(word);
```

```
    return str.reverse().toString();
     }
}
```

<aside> 💡 **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"

</aside>

```
class Solution {
  public String reverseStr(String s, int k) {
    int n = s.length();
    char[] ch= s.toCharArray();
    for(int i= 0; i<n; i += 2*k){
     int start =i;
     int end= Math.min(i+k-1, n-1);
     while(start<end){
       char temp = ch[start];
       ch[start]= ch[end];
       ch[end]= temp;
       start++;
       end--;
     }
    }
    return new String(ch);

  }
}
```

<aside> 💡 **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

</aside>

```
class Solution {
  public boolean rotateString(String s, String goal) {
    if(s.length() != goal.length()) return false;
    String t= s+s; // temporary string which include all the
rotated string
    return t.contains(goal);

  }
}
```

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".

```
class Solution {
  public boolean backspaceCompare(String s, String t) {
     int i = s.length() - 1;
    int j = t.length() - 1;

    int skipS = 0;
    int skipT = 0;

    while (i >= 0 || j >= 0) {
      // Find the position of the next non-backspace character in s
      while (i >= 0) {
        if (s.charAt(i) == '#') {
          skipS++;
        } else if (skipS > 0) {
          skipS--;
```

```
        } else {
          break;
        }
        i--;
      }

      // Find the position of the next non-backspace character in t
      while (j >= 0) {
        if (t.charAt(j) == '#') {
          skipT++;
        } else if (skipT > 0) {
          skipT--;
        } else {
          break;
        }
        j--;
      }

      // Compare the current characters at positions i and j
      if (i >= 0 && j >= 0 && s.charAt(i) != t.charAt(j)) {
        return false;
      }

      // If one string has reached its end but the other hasn't,
      // they are not equal after backspacing
      if ((i >= 0) != (j >= 0)) {
        return false;
      }

      i--;
      j--;
    }

    return true;
  }

}
```

<aside> 💡 **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.

</aside>

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

Output: true

```
if (coordinates.length <= 2) {
            return true;
        }

        int[] point1 = coordinates[0];
        int[] point2 = coordinates[1];

        for (int i = 2; i < coordinates.length; i++) {
            int[] currPoint = coordinates[i];

            // Calculate the slopes
            int slopeX = point2[0] - point1[0];
            int slopeY = point2[1] - point1[1];
            int currSlopeX = currPoint[0] - point1[0];
            int currSlopeY = currPoint[1] - point1[1];

            // Check if slopes are equal
            if (slopeX * currSlopeY != slopeY * currSlopeX) {
                return false;
            }
        }

        return true;
    }
```