AGENDA

Queue & Deque:

- Queue Introduction (FIFO)
- Deque Introduction
- Implement Queue using 2 stacks [only discuss]
- First non-repeating character in a stream of characters
- First negative in every window of size k
- Sliding window maximum

First non-repeating character in a stream of characters

Given an input stream **s** consisting only of lowercase alphabets. While reading characters from the stream, you have to tell which character has appeared only once in the stream upto that point. If there are many characters that have appeared only once, you have to tell which one of them was the first one to appear. If there is no such character then append '#' to the answer.

NOTE:
1. You need to find the answer for every i (0 <= i < n)
2. In order to find the solution for every you need to consider the string from starting position till the ith position.

Examples:

Input: s = "aabc"
Output: "a#bb"
Explanation: For every ith character we will consider the string from index 0 till index i first non repeating character is as follow- "a" - first non-repeating character is 'a' "aa" - no non-repeating character so '#' "aab" - first non-repeating character is 'b' "aabc" - there are two non repeating characters 'b' and 'c', first non-repeating character is 'b' because 'b' comes before 'c' in the stream.

Input: s = "zz"
Output: "z#"
Explanation: For every character first non repeating character is as follow- "z" - first non-repeating character is 'z' "zz" - no non-repeating character so '#'

Input: s = "bb"
Output: "b#"
Explanation: For every character first non repeating character is as follow- "b" - first non-repeating character is 'b' "bb" - no non-repeating character so '#'

First negative in every window of size k

Given an array **arr[]** and a positive integer **k**, find the first negative integer for each and every window(contiguous subarray) of size **k.**

Note: If a window does not contain a negative integer, then return 0 for that window.

Examples:

```
Input: arr[] = [-8, 2, 3, -6, 10] , k = 2
Output: [-8, 0, -6, -6]
Explanation:
Window [-8, 2] First negative integer is -8.
Window [2, 3] No negative integers, output is 0.
Window [3, -6] First negative integer is -6.
Window [-6, 10] First negative integer is -6.
```

```
Input: arr[] = [12, -1, -7, 8, -15, 30, 16, 28] , k = 3
Output: [-1, -1, -7, -15, -15, 0]
Explanation:
Window [12, -1, -7] First negative integer is -1.
Window [-1, -7, 8] First negative integer is -1.
Window [-7, 8, -15] First negative integer is -7.
Window [8, -15, 30] First negative integer is -15.
Window [-15, 30, 16] First negative integer is -15.
Window [30, 16, 28] No negative integers, output is 0.
```

```
Input: arr[] = [12, 1, 3, 5] , k = 3
Output: [0, 0]
Explanation:
Window [12, 1, 3] No negative integers, output is 0.
Window [1, 3, 5] No negative integers, output is 0.
```

Sliding window maximum

Given an array **arr[]** of integers and an integer **k**, your task is to find the maximum value for each contiguous subarray of size k. The output should be an array of maximum values corresponding to each contiguous subarray.

Examples:

```
Input: arr[] = [1, 2, 3, 1, 4, 5, 2, 3, 6], k = 3
Output: [3, 3, 4, 5, 5, 5, 6]
Explanation:
1st contiguous subarray = [1 2 3] max = 3
2nd contiguous subarray = [2 3 1] max = 3
3rd contiguous subarray = [3 1 4] max = 4
4th contiguous subarray = [1 4 5] max = 5
5th contiguous subarray = [4 5 2] max = 5
6th contiguous subarray = [5 2 3] max = 5
7th contiguous subarray = [2 3 6] max = 6
```

```
Input: arr[] = [8, 5, 10, 7, 9, 4, 15, 12, 90, 13], k = 4
Output: [10, 10, 10, 15, 15, 90, 90]
Explanation:
1st contiguous subarray = [8 5 10 7], max = 10
2nd contiguous subarray = [5 10 7 9], max = 10
3rd contiguous subarray = [10 7 9 4], max = 10
4th contiguous subarray = [7 9 4 15], max = 15
5th contiguous subarray = [9 4 15 12], max = 15
6th contiguous subarray = [4 15 12 90], max = 90
7th contiguous subarray = [15 12 90 13], max = 90
```

```
Input: arr[] = [5, 1, 3, 4, 2, 6], k = 1
Output: [5, 1, 3, 4, 2, 6]
Explanation:
When k = 1, each element in the array is its own subarray, so the output is simply the same array
```