Different Implementations of Stack

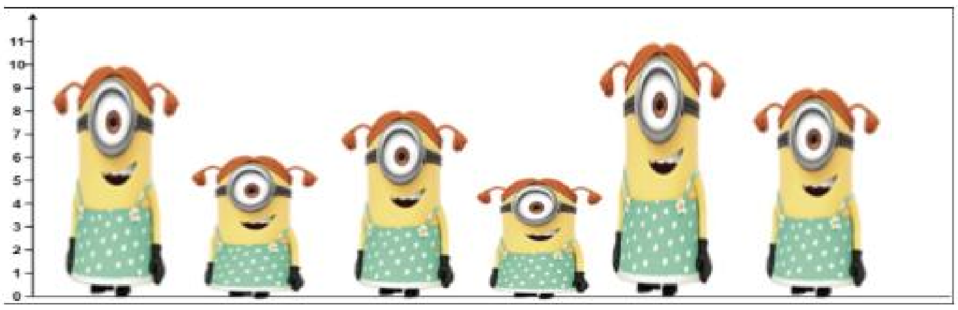
Lesson Plan

Q. Number of Visible People in a Queue (leetcode 1944)
There are n people standing in a queue, and they numbered from 0 to n - 1 in left to right order. You are given an array heights of distinct integers where heights[i] represents the height of the ith person.

A person can see another person to their right in the queue if everybody in between is shorter than both of them. More formally, the ith person can see the jth person if i < j and min(heights[i], heights[j]) > max(heights[i+1], heights[i+2], ..., heights[j-1]).

Return an array answer of length n where answer[i] is the number of people the ith person can see to their right in the queue.

Input: heights = [10,6,8,5,11,9]
Output: [3,1,2,1,1,0]
Explanation:



Person 0 can see person 1, 2, and 4.
Person 1 can see person 2.
Person 2 can see person 3 and 4.
Person 3 can see person 4.
Person 4 can see person 5.
Person 5 can see no one since nobody is to the right of them.

Code

```
from typing import List

class Solution:
    def canSeePersonsCount(self, heights: List[int]) ->
List[int]:
        stk = []
        ans = []

        for i in range(len(heights) - 1, -1, -1):
            cnt = 0
 while stk and stk[-1] <= heights[i]:
                cnt += 1
                stk.pop()
            ans.append(cnt + (1 if stk else 0))
            stk.append(heights[i])

        return ans[::-1]

# Example usage:
if __name__ == "__main__":
    heights = [10, 6, 8, 5, 11, 9]
    sol = Solution()
    result = sol.canSeePersonsCount(heights)
    print(result)  # Output: [3, 1, 2, 1, 1, 0]
```

Ques: Sliding Window Maximum (LEETCODE 239)
You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

Return the max sliding window.

Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]

Explanation:

```
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

Example 2:

Input: nums = [1], k = 1
Output: [1]

Code

```
from typing import List

class Solution:
  def maxSlidingWindow(self, nums: List[int], k: int) →
List[int]:
        from collections import deque

        dq = deque()
        result = []

        for i in range(len(nums)):
            # Remove elements out of the current window
            if dq and dq[0] < i - k + 1:
                dq.popleft()

            # Remove elements smaller than the current
element
            while dq and nums[dq[-1]] ⩽ nums[i]:
                dq.pop()
```

```
   dq.append(i)

            # Start adding results after the first window
            if i ≥ k - 1:
                result.append(nums[dq[0]])

        return result

# Example usage:
if __name__ == "__main__":
    nums = [1, 3, -1, -3, 5, 3, 6, 7]
    k = 3
    sol = Solution()
    result = sol.maxSlidingWindow(nums, k)
    print(result)  # Output: [3, 3, 5, 5, 6, 7]
```

Q. Min Stack (leetcode 155)
Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

MinStack() initializes the stack object.
void push(int val) pushes the element val onto the stack.
void pop() removes the element on the top of the stack.
int top() gets the top element of the stack.
int getMin() retrieves the minimum element in the stack.
You must implement a solution with O(1) time complexity for each function.

Example 1:

Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top();   // return 0
minStack.getMin(); // return -2

Code

```
class MinStack:
    def __init__(self):
        self.stack = []
        self.min_stack = []

    def push(self, val: int) -> None:
        self.stack.append(val)
        if not self.min_stack or val <= self.min_stack[-1]:
            self.min_stack.append(val)

    def pop(self) -> None:
        if self.stack:
            val = self.stack.pop()
            if val == self.min_stack[-1]:
                self.min_stack.pop()
 def top(self) -> int:
        if self.stack:
            return self.stack[-1]
        return None

    def getMin(self) -> int:
        if self.min_stack:
  return self.min_stack[-1]
        return None

# Example usage:
if __name__ == "__main__":
    minStack = MinStack()
    minStack.push(-2)
    minStack.push(0)
    minStack.push(-3)
    print(minStack.getMin())  # Output: -3
    minStack.pop()
    print(minStack.top())     # Output: 0
    print(minStack.getMin())  # Output: -2
```