Question 1: LeetCode Questino Q155. Easy. "Min Stack"

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

Constraints:

-231 <= val <= 231 - 1
Methods pop, top and getMin operations will always be called on non-empty stacks.
At most 3 * 104 calls will be made to push, pop, top, and getMin.

```
By using two stacks, stack and min_stack, we can keep track of all elements and
their respective minimums at each point.

The min_stack is updated in sync with the stack:
- (1) when a new element is pushed and it is less than or equal to the current
minimum, it is added to min_stack.
- (2) when an element is popped and it is the current minimum, it is also removed
from min_stack.

Push and Pop operations are O(1) because adding/removing elements from the end of
a list is constant time. Top and getMin operations are also O(1) because they
simply return the last element of a list, which is a constant-time operation.
```

```
Psuedo code:

function __init__:
    initialize stack as an empty list
    initialize min_stack as an empty list

function push(x):
    append x to stack
    if min_stack is empty or x is less than or equal to the last element in
min_stack:
        append x to min_stack

function pop:
    if the last element in stack is equal to the last element in min_stack:
        remove the last element from min_stack
    remove the last element from stack

function top:
    return the last element in stack

function getMin:
    return the last element in min_stack
```

```python
# Define the MinStack
class MinStack:
    def __init__(self):
        self.stack = []
        self.min_stack = []

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

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

    def top(self) -> int:
        return self.stack[-1]

    def getMin(self) -> int:
        return self.min_stack[-1]

""" test"""
def test_min_stack(operations, values):
    min_stack = MinStack()
    results = []

    for op, val in zip(operations, values):
        if op == "MinStack":
            results.append(None)
        elif op == "push":
            min_stack.push(val[0])
            results.append(None)
        elif op == "pop":
            min_stack.pop()
            results.append(None)
        elif op == "top":
            results.append(min_stack.top())
        elif op == "getMin":
            results.append(min_stack.getMin())

    return results

operations = ["MinStack","push","push","push","getMin","pop","top","getMin"]
values = [[],[-2],[0],[-3],[],[],[],[]]
output = test_min_stack(operations, values)
print(output)
```

```
[None, None, None, None, -3, None, 0, -2]
```

Question 2: LeetCode Questino #1472. Medium. "Design Browser History"
You have a browser of one tab where you start on the homepage and you can visit another url, get back in the history number of steps or move forward in the history number of steps.

Implement the BrowserHistory class:

BrowserHistory(string homepage) Initializes the object with the homepage of the browser.
void visit(string url) Visits url from the current page. It clears up all the forward history.
string back(int steps) Move steps back in history. If you can only return x steps in the history and steps > x, you will return only x steps. Return the current url after moving back in history at most steps.
string forward(int steps) Move steps forward in history. If you can only forward x steps in the history and steps > x, you will forward only x steps. Return the current url after forwarding in history at most steps.

Example:

Input:
["BrowserHistory","visit","visit","visit","back","back","forward","visit","forward","back","back"]
[["leetcode.com"],["google.com"],["facebook.com"],["youtube.com"],[1],[1],[1],["linkedin.com"],[2],[2],[7]]
Output:
[null,null,null,null,"facebook.com","google.com","facebook.com",null,"linkedin.com","google.com","leetcode.com"]

Explanation:
BrowserHistory browserHistory = new BrowserHistory("leetcode.com");
browserHistory.visit("google.com");   // You are in "leetcode.com". Visit "google.com"
browserHistory.visit("facebook.com");   // You are in "google.com". Visit "facebook.com"
browserHistory.visit("youtube.com");   // You are in "facebook.com". Visit "youtube.com"
browserHistory.back(1);   // You are in "youtube.com", move back to "facebook.com" return "facebook.com"
browserHistory.back(1);   // You are in "facebook.com", move back to "google.com" return "google.com"
browserHistory.forward(1);   // You are in "google.com", move forward to "facebook.com" return "facebook.com"
browserHistory.visit("linkedin.com");   // You are in "facebook.com". Visit "linkedin.com"
browserHistory.forward(2);   // You are in "linkedin.com", you cannot move forward any

steps.
browserHistory.back(2);   // You are in "linkedin.com", move back two steps to "facebook.com" then to "google.com". return "google.com"
browserHistory.back(7);   // You are in "google.com", you can move back only one step to "leetcode.com". return "leetcode.com"

Constraints:

1 <= homepage.length <= 20
1 <= url.length <= 20
1 <= steps <= 100
homepage and url consist of '.' or lower case English letters.
At most 5000 calls will be made to visit, back, and forward.

```
Initialize the history list a with the homepage.
Set the current index i to 0.

When a new URL is visited, all forward history from the current position i is
removed.
The new URL is appended to the history list a.
Update the current index i to point to the new URL.

Move i back by the specified number of steps, ensuring it does not go below 0.
Return the URL at the new current index i.

Move i forward by the specified number of steps, ensuring it does not exceed the
last index of a.
Return the URL at the new current index i.
```

```
function __init__(homepage):
    initialize array a with homepage
    initialize index i to 0

function visit(url):
    delete elements in a from index i+1 to end
    append url to a
    increment i by 1

function back(steps):
    set i to max(0, i - steps)
    return a[i]

function forward(steps):
    set i to min(i + steps, length of a - 1)
    return a[i]
```

```python
class BrowserHistory:
    def __init__(self, homepage: str):
        self.a = [homepage]
        self.i = 0

    def visit(self, url: str) -> None:
        del self.a[self.i + 1:]
        self.a.append(url)
```

```python
            self.i += 1

    def back(self, steps: int) -> str:
        self.i = max(0, self.i - steps)
        return self.a[self.i]

    def forward(self, steps: int) -> str:
        self.i = min(self.i + steps, len(self.a) - 1)
        return self.a[self.i]

""" test """
def test_browser_history(operations, values):
    browser_history = None
    results = []

    for op, val in zip(operations, values):
        if op == "BrowserHistory":
            browser_history = BrowserHistory(val[0])
            results.append(None)
        elif op == "visit":
            browser_history.visit(val[0])
            results.append(None)
        elif op == "back":
            results.append(browser_history.back(val[0]))
        elif op == "forward":
            results.append(browser_history.forward(val[0]))

    return results

""" input """
operations = 
["BrowserHistory","visit","visit","visit","back","back","forward","visit","forwar
d","back","back"]
values = [["leetcode.com"],["google.com"],["facebook.com"],["youtube.com"],[1],
[1],[1],["linkedin.com"],[2],[2],[7]]

output = test_browser_history(operations, values)
print(output)
```

```
[None, None, None, None, 'facebook.com', 'google.com', 'facebook.com', None,
'linkedin.com', 'google.com', 'leetcode.com']
```