### Programming Guide for “Stacks and Queues”

#### 1. Validate Brackets

**Overview:** This function checks whether the brackets in a given string are balanced.

**Pseudo Code:**

FUNCTION validate\_brackets(string)  
 INITIALIZE an empty stack  
 FOR each character in string  
 IF character is an opening bracket  
 PUSH it onto the stack  
 ELSE IF character is a closing bracket  
 IF stack is empty OR top of stack doesn't match the closing bracket  
 RETURN False  
 ELSE  
 POP the top element from the stack  
 RETURN True IF stack is empty, ELSE False  
END FUNCTION

**Implementation Guide:** - Use a stack to keep track of opening brackets. - For each character in the string, if it’s an opening bracket, push it onto the stack. - If it’s a closing bracket, check if it matches the top element of the stack. - If the stack is empty or the brackets don’t match, return False. - If the entire string is processed and the stack is empty, the brackets are balanced.

#### 2. Next Greater Element

**Overview:** For each element in a list, find the next greater element within the list.

**Pseudo Code:**

FUNCTION next\_greater\_element(nums)  
 INITIALIZE result list filled with -1 for each element in nums  
 INITIALIZE an empty stack  
 FOR each index, i, in nums  
 WHILE stack is not empty AND nums[i] is greater than nums at top of stack  
 SET result at index of top of stack to nums[i]  
 POP the top element from stack  
 PUSH i onto stack  
 RETURN result  
END FUNCTION

**Implementation Guide:** - Maintain a stack to store indices of elements. - Iterate through the list, comparing the current element with the element at the index on top of the stack. - Update the result for elements in the stack if the current element is greater. - Push the current index onto the stack.

#### 3. Reverse Stack

**Overview:** Reverse a stack using only push and pop operations.

**Pseudo Code:**

FUNCTION reverse\_stack(stack)  
 INITIALIZE an auxiliary stack  
 WHILE the original stack is not empty  
 POP the top element from the original stack  
 PUSH the popped element onto the auxiliary stack  
 RETURN the auxiliary stack  
END FUNCTION

**Implementation Guide:** - Create an auxiliary stack. - Pop each element from the original stack and push it onto the auxiliary stack. - Once all elements are transferred, the auxiliary stack will hold the elements in reversed order. - Return the auxiliary stack.

This guide should assist students in understanding the approach and logic behind each function. The pseudo code provides a structural overview, while the implementation guide offers step-by-step instructions for translating this logic into Python code.