Introduction to Stack Data Structure

Lesson Plan

PW SKILLS

Q.Check for Balanced Brackets in an expression (well-formedness).

Input: exp = “[()]{}{[()()]()}”
Output: Balanced

Explanation: all the brackets are well-formed

Input: exp = “[(])”
Output: Not Balanced

Explanation: 1 and 4 brackets are not balanced because
there is a closing ‘]’ before the closing ‘(‘

Code

```
class BalancedBrackets:
    @staticmethod
    def areBracketsBalanced(expr):
        temp = []
        for char in expr:
            if not temp:
                temp.append(char)
            elif (temp[-1] == '(' and char == ')') or \
                 (temp[-1] == '{' and char == '}') or \
                 (temp[-1] == '[' and char == ']'):
                temp.pop()
            else:
                temp.append(char)
        return not temp

if __name__ == "__main__":
    expression = "{[()]}";
    if BalancedBrackets.areBracketsBalanced(expression):
        print("Brackets are balanced.")
    else:
        print("Brackets are not balanced.")
```

Q. Given a string S, The task is to remove all the consecutive duplicate characters of the string and return the resultant string.

Input: S= “aaaaabbbbbb”
Output: ab

Code

```
class DeleteConsecutiveStrings:
    @staticmethod
    def deleteConsecutiveStrings(s):
        i, j = 0, 0
        newElements = []
        while j < len(s):
            if s[i] == s[j]:
                j += 1
            else:
                newElements.append(s[i])
                i = j
            j += 1
        newElements.append(s[j - 1])
        return ''.join(newElements)

if __name__ == "__main__":
    input_str = "aabbcdeeff"
    result =
DeleteConsecutiveStrings.deleteConsecutiveStrings(input_str
)
    print(result)
```

Ques: Given an array, print the Next Greater Element (NGE) for every element.
Input: arr[] = [4 , 5 , 2 , 25]

Output:
```
4    ->  5
5    ->  25
2    ->  25
25   ->  -1
```

Explanation: except 25 every element has an element greater than them present on the right side

Q.Given an array of distinct elements, find the previous greater element for every element. If the previous greater element does not exist, print -1.

Input : arr[] = {10, 4, 2, 20, 40, 12, 30}
Output : -1, 10, 4, -1, -1, 40, 40

Code

```
class PreviousGreaterElement:
    @staticmethod
    def prevGreater(arr):
        n = len(arr)
        s = []
        result = []
        for i in range(n):
            while s and s[-1] <= arr[i]:
                s.pop()
            if not s:
                result.append(-1)
            else:
                result.append(s[-1])
            s.append(arr[i])
        print(', '.join(map(str, result)))

if __name__ == "__main__":
    arr = [10, 4, 2, 20, 40, 12, 30]
    PreviousGreaterElement.prevGreater(arr)
```

Q.The stock span problem is a financial problem where we have a series of N daily price quotes for a stock and we need to calculate the span of the stock's price for all N days. The span Si of the stock's price on a given day i is defined as the maximum number of consecutive days just before the given day, for which the price of the stock on the current day is less than or equal to its price on the given day.

Input: **N = 7, price[] = [100 80 60 70 60 75 85]**
Output: **1 1 1 2 1 4 6**

Explanation: **Traversing the given input span for 100 will be 1, 80 is smaller than 100 so the span is 1, 60 is smaller than 80 so the span is 1, 70 is greater than 60 so the span is 2 and so on. Hence the output will be 1 1 1 2 1 4 6.**

Code

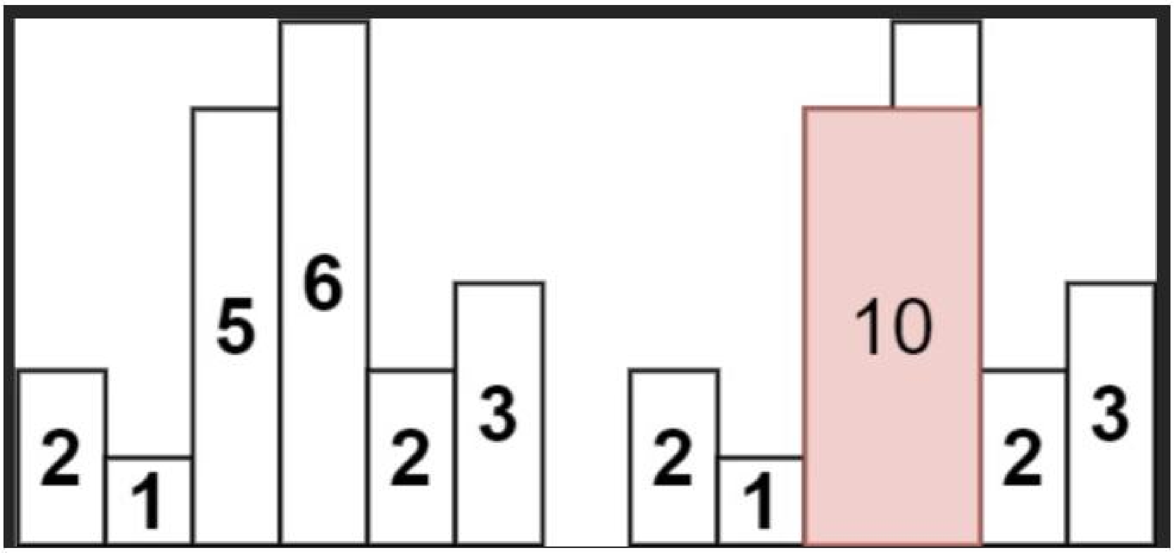
```
class StockSpan:
    @staticmethod
    def calculateSpan(price):
        n = len(price)
        S = [0] * n
        st = []
        st.append(0)
        S[0] = 1
        for i in range(1, n):
            while st and price[st[-1]] <= price[i]:
                st.pop()
            S[i] = i + 1 if not st else i - st[-1]
            st.append(i)
        return S

if __name__ == "__main__":
    price = [10, 4, 5, 90, 120, 80]
    span = StockSpan.calculateSpan(price)
    for num in span:
        print(num, end=" ")
    print()
```

Ques. Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram.(leetcode 84)

Input: heights = [2,1,5,6,2,3]
Output: 10

Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10 units.



Code

```
from typing import List

class Solution:
    def largestRectangleArea(self, heights: List[int]) ->
int:
        n = len(heights)
        if n == 0:
            return 0

        # Initialize left and right arrays
        left = [0] * n
        right = [0] * n

        # Stack to store indices
        stack = []

        # Fill left array
        for i in range(n):
            while stack and heights[stack[-1]] >=
heights[i]:
                stack.pop()
            left[i] = stack[-1] + 1 if stack else 0
            stack.append(i)

        # Clear stack for reuse
        stack = []

        # Fill right array
        for i in range(n - 1, -1, -1):
```

```
     while stack and heights[stack[-1]] >= heights[i]:
                stack.pop()
            right[i] = stack[-1] - 1 if stack else n - 1
            stack.append(i)

        # Calculate max area
        max_area = 0
        for i in range(n):
            max_area = max(max_area, (right[i] - left[i] +
1) * heights[i])

        return max_area

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