**Question 1**

Given an array **arr[ ]** of size **N** having elements, the task is to find the next greater element for each element of the array in order of their appearance in the array.Next greater element of an element in the array is the nearest element on the right which is greater than the current element.If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

**Example 1:**

Input:

N = 4, arr[] = [1 3 2 4]

Output:

3 4 4 -1

Explanation:

In the array, the next larger element

to 1 is 3 , 3 is 4 , 2 is 4 and for 4 ?

since it doesn't exist, it is -1.

**Ans.**

**def** createStack():

    stack **=** []

**return** stack

**def** isEmpty(stack):

**return** len(stack) **==** 0

**def** push(stack, x):

    stack.append(x)

**def** pop(stack):

**if** isEmpty(stack):

        print("Error : stack underflow")

**else**:

**return** stack.pop()

**def** printNGE(arr):

    s **=** createStack()

    element **=** 0

    next **=** 0

    push(s, arr[0])

**for** i **in** range(1, len(arr), 1):

        next **=** arr[i]

**if** isEmpty(s) **==** False:

            element **=** pop(s)

**while** element < next:

**print**(str(element) **+** " -- " **+** str(next))

**if** isEmpty(s) **==** True:

**break**

                element **=** pop(s)

**if** element > next:

                push(s, element)

        push(s, next)

**while** isEmpty(s) **==** False:

        element **=** pop(s)

        next **=** **-**1

        print(str(element) **+** " -- " **+** str(next))

arr **=** [11, 13, 21, 3]

printNGE(arr)

**Question 2**

Given an array **a** of integers of length **n**, find the nearest smaller number for every element such that the smaller element is on left side.If no small element present on the left print -1.

**Example 1:**

Input: n = 3

a = {1, 6, 2}

Output: -1 1 1

Explaination: There is no number at the

left of 1. Smaller number than 6 and 2 is 1.

**Ans.** Solution from my interviewbit :- <https://www.interviewbit.com/profile/gopsa>

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def prevSmaller(self, A):

        S = list()

        res = []

        for i in range(len(A)):

            while (len(S) > 0 and S[-1] >= A[i]):

                S.pop()

            if (len(S) == 0):

                res.append(-1)

            else:

                res.append(S[-1])

            S.append(A[i])

        return res

**Question 3**

Implement a Stack using two queues **q1** and **q2**.

**Example 1:**

Input:

push(2)

push(3)

pop()

push(4)

pop()

Output:3 4

Explanation:

push(2) the stack will be {2}

push(3) the stack will be {2 3}

pop() poped element will be 3 the

  stack will be {2}

push(4) the stack will be {2 4}

pop()   poped element will be 4

**Ans.**

**from** \_collections **import** deque

**class** Stack:

**def** \_\_init\_\_(self):

        # Two inbuilt queues

        self.q1 **=** deque()

        self.q2 **=** deque()

**def** push(self, x):

        # Push x first in empty q2

        self.q2.append(x)

        # Push all the remaining

        # elements in q1 to q2.

**while** (self.q1):

            self.q2.append(self.q1.popleft())

        # swap the names of two queues

        self.q1, self.q2 **=** self.q2, self.q1

**def** pop(self):

        # if no elements are there in q1

**if** self.q1:

            self.q1.popleft()

**def** top(self):

**if** (self.q1):

**return** self.q1[0]

**return** None

**def** size(self):

**return** len(self.q1)

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    s **=** Stack()

    s.push(1)

    s.push(2)

    s.push(3)

**print**("current size: ", s.size())

    print(s.top())

    s.pop()

**print**(s.top())

    s.pop()

    print(s.top())

**print**("current size: ", s.size())

**Question 4**

You are given a stack **St**. You have to reverse the stack using recursion.

**Example 1:**

Input:St = {3,2,1,7,6}

Output:{6,7,1,2,3}

**Ans.**

**def** insertAtBottom(stack, item):

**if** isEmpty(stack):

        push(stack, item)

**else**:

        temp **=** pop(stack)

        insertAtBottom(stack, item)

        push(stack, temp)

# Below is the function that

# reverses the given stack

# using insertAtBottom()

**def** reverse(stack):

**if** **not** isEmpty(stack):

        temp **=** pop(stack)

        reverse(stack)

        insertAtBottom(stack, temp)

# Below is a complete running

# program for testing above

# functions.

# Function to create a stack.

# It initializes size of stack

# as 0

**def** createStack():

    stack **=** []

**return** stack

# Function to check if

# the stack is empty

**def** isEmpty(stack):

**return** len(stack) **==** 0

# Function to push an

# item to stack

**def** push(stack, item):

    stack.append(item)

# Function to pop an

# item from stack

**def** pop(stack):

    # If stack is empty

    # then error

**if**(isEmpty(stack)):

        print("Stack Underflow ")

        exit(1)

**return** stack.pop()

# Function to print the stack

**def** prints(stack):

**for** i **in** range(len(stack)**-**1, **-**1, **-**1):

        print(stack[i], end**=**' ')

    print()

# Driver Code

stack **=** createStack()

push(stack, str(4))

push(stack, str(3))

push(stack, str(2))

push(stack, str(1))

print("Original Stack ")

prints(stack)

reverse(stack)

print("Reversed Stack ")

prints(stack)

**Question 5**

You are given a string **S**, the task is to reverse the string using stack.

**Example 1:**

Input: S="GeeksforGeeks"

Output: skeeGrofskeeG

**Ans.**

**def** createStack():

    stack **=** []

**return** stack

# Function to determine the size of the stack

**def** size(stack):

**return** len(stack)

# Stack is empty if the size is 0

**def** isEmpty(stack):

**if** size(stack) **==** 0:

**return** true

# Function to add an item to stack .

# It increases size by 1

**def** push(stack, item):

    stack.append(item)

# Function to remove an item from stack.

# It decreases size by 1

**def** pop(stack):

**if** isEmpty(stack):

**return**

**return** stack.pop()

# A stack based function to reverse a string

**def** reverse(string):

    n **=** len(string)

    # Create a empty stack

    stack **=** createStack()

    # Push all characters of string to stack

**for** i **in** range(0, n, 1):

        push(stack, string[i])

    # Making the string empty since all

    # characters are saved in stack

    string **=** ""

    # Pop all characters of string and

    # put them back to string

**for** i **in** range(0, n, 1):

        string **+=** pop(stack)

**return** string

# Driver program to test above functions

string **=** "GeeksQuiz"

string **=** reverse(string)

print("Reversed string is " **+** string)

**Question 6**

Given string **S** representing a postfix expression, the task is to evaluate the expression and find the final value. Operators will only include the basic arithmetic operators like \***, /, + and -**.

**Example 1:**

Input: S = "231\*+9-"

Output: -4

Explanation:

After solving the given expression,

we have -4 as result.

**Ans.**

**class** Evaluate:

    # Constructor to initialize the class variables

**def** \_\_init\_\_(self, capacity):

        self.top **=** **-**1

        self.capacity **=** capacity

        # This array is used a stack

        self.array **=** []

    # Check if the stack is empty

**def** isEmpty(self):

**return** True **if** self.top **==** **-**1 **else** False

    # Return the value of the top of the stack

**def** peek(self):

**return** self.array[**-**1]

    # Pop the element from the stack

**def** pop(self):

**if** **not** self.isEmpty():

            self.top **-=** 1

**return** self.array.pop()

**else**:

**return** "$"

    # Push the element to the stack

**def** push(self, op):

        self.top **+=** 1

        self.array.append(op)

    # The main function that converts given infix expression

    # to postfix expression

**def** evaluatePostfix(self, exp):

        # Iterate over the expression for conversion

**for** i **in** exp:

            # If the scanned character is an operand

            # (number here) push it to the stack

**if** i.isdigit():

                self.push(i)

            # If the scanned character is an operator,

            # pop two elements from stack and apply it.

**else**:

                val1 **=** self.pop()

                val2 **=** self.pop()

                self.push(str(eval(val2 **+** i **+** val1)))

**return** int(self.pop())

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    exp **=** "231\*+9-"

    obj **=** Evaluate(len(exp))

    # Function call

**print**("postfix evaluation: %d" **%** (obj.evaluatePostfix(exp)))

**Question 7**

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

**Ans.** Solution from my leetcode :- <https://leetcode.com/gopsa2001/>

class MinStack:

    def \_\_init\_\_(self):

        self.stack = []

        self.minStack = []

    def push(self, val: int) -> None:

        self.stack.append(val)

        val = min(val, self.minStack[-1] if self.minStack else val)

        self.minStack.append(val)

    def pop(self) -> None:

        self.stack.pop()

        self.minStack.pop()

    def top(self) -> int:

        return self.stack[-1]

    def getMin(self) -> int:

        return self.minStack[-1]

**Question 8**

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

**Example 1:**

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

**Ans.** Solution from my leetcode :- <https://leetcode.com/gopsa2001/>

class Solution:

    def trap(self, height: List[int]) -> int:

        stack = []

        ans = 0

        n = len(height)

        for i in range(n):

            while stack and height[stack[-1]] < height[i]:

                j = stack.pop()

                if len(stack) == 0:

                    break

                h = min(height[i], height[stack[-1]]) - height[j]

                w = i - stack[-1] - 1

                ans += h \* w

            stack.append(i)

        return ans