1. Given an array, print the Next Greater Element (NGE) for every element. The Next greater Element for an element x is the first greater element on the right side of x in array. Elements for which no greater element exist, consider next greater element as -1.

Examples:

a) For any array, rightmost element always has next greater element as -1.

b) For an array which is sorted in decreasing order, all elements have next greater element as -1.

c) For the input array [4, 5, 2, 25}, the next greater elements for each element are as follows.

Element       NGE

   4      -->   5

   5      -->   25

   2      -->   25

   25     -->   -1

d) For the input array [13, 7, 6, 12}, the next greater elements for each element are as follows.

  Element        NGE

   13      -->    -1

   7       -->     12

   6       -->     12

   12     -->     -1

def printNGE(arr):

    for i in range(0, len(arr), 1):

        next = -1

        for j in range(i+1, len(arr), 1):

            if arr[i] < arr[j]:

                next = arr[j]

                break

        print(str(arr[i]) + " -- " + str(next))

O(n^2).

Use two loops: The outer loop picks all the elements one by one. The inner loop looks for the first greater element for the element picked by outer loop. If a greater element is found then that element is printed as next, otherwise -1 is printed.

Implement a Queue using 2 stacks**s1** and**s2** . DONE

enQueue(q, x) 1) While stack1 is not empty, push everything from stack1 to stack2. 2) Push x to stack1 (assuming size of stacks is unlimited). 3) Push everything back to stack1. Here time complexity will be O(n) deQueue(q) 1) If stack1 is empty then error 2) Pop an item from stack1 and return it Here time complexity will be O(1)

# implement stacks using plain lists with push and pop functions

Stack1 = [] Stack2 = []

# implement enqueue method by using only stacks # and the append and pop functions

def Enqueue(element):

Stack1.append(element)

# implement dequeue method by pushing all elements # from stack 1 into stack 2, which reverses the order # and then popping from stack 2

def Dequeue():

if len(Stack2) == 0:

if len(Stack1) == 0:

return 'Cannot dequeue because queue is empty'

while len(Stack1) > 0:

p = Stack1.pop()

Stack2.append(p)

return Stack2.pop()

Enqueue('a')

Enqueue('b')

Enqueue('c')

print Dequeue()

1. Implement a Stack using 2 queue**q1** and**q2** . DONE

import queue class MyStack:

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

""" Initialize your data structure here. """

self.q1 = queue.Queue()

def push(self, x):

""" Push element x onto stack. :type x: int :rtype: void """

q2 = queue.Queue()

q2.put(x)

while not self.q1.empty():

q2.put(self.q1.get())

self.q1 = q2

def pop(self):

""" Removes the element on top of the stack and returns that element. :rtype: int """

return self.q1.get()

def top(self):

""" Get the top element. :rtype: int """

topElement = self.pop()

self. push(topElement)

return topElement

def empty(self):

""" Returns whether the stack is empty. :rtype: bool """

return self.q1.empty()

1. Implement a Stack in which you can get min element in O(1) time and O(1) space. DONE

From collection import deque

Q = deque()

q.append ('eat')

q.append ('sleep')

q.append ('code')

Q

Deque (['eat', 'sleep', 'code'])

q.pop()

'code'

q.pop()

'sleep'

q.pop()

'eat'

q.pop()

1. The task is to design and implement methods of an **LRU cache**. The class has two methods get and set which are defined as follows.  
   get(x)   : Gets the value of the key x if the key exists in the cache otherwise returns -1  
   set(x,y) : inserts the value if the key x is not already present. If the cache reaches its capacity it should invalidate the least recently used item before inserting the new item.  
   In the constructor of the class the size of the cache should be initialized.

*From <*[*https://www.kunxi.org/blog/2014/05/lru-cache-in-python*](https://www.kunxi.org/blog/2014/05/lru-cache-in-python)*>*

1. Given an input stream of n characters consisting only of small case alphabets the task is to find the first non repeating character each time a character is inserted to the stream. If no non repeating element is found print -1. DONE  
     
   Example

Flow in stream : a, a, b, c

a goes to stream : 1st non repeating element a (a)

a goes to stream : no non repeating element -1 (a, a)

b goes to stream : 1st non repeating element is b (a, a, b)

c goes to stream : 1st non repeating element is b (a, a, b, c)

NO\_OF\_CHARS **=** 256

# Returns an array of size 256 containg count

# of characters in the passed char array

def getCharCountArray(string):

    count = [0] \* NO\_OF\_CHARS

    for i in string:

        count[ord(i)]+=1

    return count

# The function returns index of first non-repeating

# character in a string. If all characters are repeating

# then returns -1

def firstNonRepeating(string):

    count = getCharCountArray(string)

    index = -1

    k = 0

    for i in string:

        if count[ord(i)] == 1:

            index = k

            break

        k += 1

    return index