MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY

«KHARKIV POLYTECHNIC INSTITUTE»

Department of Software Engineering and Management Information Technologies

Report from lab № 6

discipline «Fundamentals of python»

Executed by: Student of group 1.КН.201.8e1

CHUKWU IRELE AMORIN

Checked by: Senior lecturer

Kharkiv

2019

***Laboratory work 7***

***Lists***

*1. “Even indices”*

Print all elements of the list with even indices (that is, A [0], A [2], A [4], ...).

*2. “Even elements”*

Print all the even elements of the list. In doing so, use a for loop that iterates over the list items, not their indexes!

*3. "More than the previous"*

A list of numbers is given. Print all list items that are larger than the previous item.

*4. “Neighbors of the same sign”*

A list of numbers is given. If it has two adjacent elements of the same sign, print these numbers. If there are no adjacent elements of the same sign, do not print anything. If there are several such pairs of neighbors, print the first pair.

*5. “More than your neighbors”*

A list of numbers is given. Determine how many elements in this list are more than two of their neighbors, and print the number of such elements. The extreme elements of the list are never taken into account, since they do not have enough neighbors.

*6. "The greatest element"*

A list of numbers is given. Print the value of the largest element in the list, and then the index of this element in the list. If there are several largest elements, print the index of the first of them.

*7, "The line"*

Peter moved to another school. In a physical education lesson, he needed to determine his place in the ranks. Help him do this.

The program receives a non-increasing sequence of natural numbers, which means the growth of each person in the system. After that, enter the number X - the growth of Peter. All numbers in the input are natural and do not exceed 200.

Print the number under which Peter should stand in order. If there are people in the ranks with the same height, the same as Petya, then he must stand after them.

*8. "The number of different elements"*

Given a list ordered by non-decreasing elements in it. Determine how many different elements are in it.

*9. “Rearrange neighboring”*

Rearrange the adjacent list items (A [0] c A [1], A [2] c A [3], etc.). If there are an odd number of elements, then the last element remains in its place.

*10. “Rearrange min and max”*

In the list, all elements are different. Swap the minimum and maximum element of this list.

*11. "Delete item"*

Given a list of numbers and the index of the element in the list k. Remove the element with index k from the list by moving to the left all the elements to the right of the element with index k.

The program receives a list of inputs, then the number k. The program shifts all elements, and after that deletes the last element of the list using the pop () method without parameters.

The program should shift directly in the list, and not do this when displaying items. Also, you cannot use an additional list. Also, you should not use the pop (k) method with a parameter.

*12. "Insert item"*

Given a list of integers, the number k and the value of C. It is necessary to insert into the list at the position with index k an element equal to C, shifting all elements with an index of at least k to the right.

Since this increases the number of elements in the list, after reading the list, you will need to add a new element to its end using the append method.

The insert must be carried out already in a read list, without doing this at the output and without creating an additional list.

*13. "The number of matching pairs"*

A list of numbers is given. Count how many pairs of elements there are equal to each other. It is believed that any two elements equal to each other form one pair, which must be calculated.

*14. “Unique elements”*

The list is given. Print those elements that appear in the list only once. Elements must be displayed in the order in which they appear in the list.

*15 "Bowling alley"*

N pins were put in one row, numbering them from left to right with numbers from 1 to N. Then K balls were thrown along this row, and the ith ball knocked down all the pins with numbers from li to ri inclusive. Determine which pins are left standing in place.

The program receives at the input the number of pins N and the number of throws K. Next comes K pairs of numbers li, ri, with 1≤ li≤ ri≤ N.

The program should output a sequence of N characters, where the j-th character is “I” if the j-th pin remains standing, or “.” If the j-th pin was knocked down.

*16, The Queens*

It is known that 8 queens can be placed on an 8 × 8 board so that they do not beat each other. You have been given the arrangement of 8 queens on the board, determine if there is a pair of beating each other among them.

The program receives eight pairs of numbers as an input, each number from 1 to 8 - coordinates of 8 queens. If the queens do not beat each other, print the word NO, otherwise print YES.

Solution:

#1

def evenindecies():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    for \_ in range(0,len(seq)-1):

        if (\_ % 2 == 0):

            print(seq[\_])

#2

def evenelements():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    c = [i for i in seq if i % 2 == 0]

    for \_ in c:

        print(\_)

#3

def more():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    prev,curr = seq[0]

    for \_ in seq :

        curr = \_

        if curr > prev :

            print(curr)

        prev = curr

#4

def neigh():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    prev = int(seq[0])

    count =1

    for \_ in seq :

        if count >1:

            curr = int(\_)

            if (prev >= 0 and curr >= 0 ) or (prev < 0 and curr < 0) :

                print(prev, end=" ")

                print(curr, end=" ")

                break

            prev = curr

        count += 1

#5

def bigneigh():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    prev = int(seq[0])

    count =0

    num=0

    for \_ in seq :

        if count >0 and count < len(seq)-1:

            curr = int(\_)

            nex = int(seq[count+1])

            if curr > prev and curr > nex:

                num += 1

            prev = curr

        count += 1

    print(num)

#6

def maxlist():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    length =  len(seq)

    i = 0

    maxm =int(seq[i])

    ind = 0

    while  i < length:

        if maxm < int(seq[i]):

            maxm = int(seq[i])

            ind = i

        i += 1

    print("largest of member of the sequence is : {0} with index of :{1}".format(maxm,ind))

#7

def line():

    seq = []

    value = 0

    peet = int(input("input peter's value :  "))

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    fixed = 0

    position = 0

    for \_ in seq :

        if int(\_) > peet:

            position +=1

            continue

        elif int(\_) < peet:

            fixed = 1

            break

        position +=1

    print("peter should stand in {0} position ".format(position))

#8

def diff():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    seq = list(dict.fromkeys(seq))

    print("number of distict elements : " ,len(seq))

#9

def rearr():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    i = 0

    while i <len(seq) and (i+1) != len(seq) :

        if i % 2 == 0:

            curr = seq[i]

            prev = seq[i+1]

            seq[i] = prev

            seq[i+1] = curr

        i +=1

    for \_ in seq:

        print(\_)

#10

def  rearrMM():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    length =  len(seq)

    i = 0

    maxm =int(seq[i])

    while  i < length:

        if maxm < int(seq[i]):

            maxm = int(seq[i])

            maxin = i

        i += 1

    i = 0

    minm =int(seq[i])

    while  i < length:

        if minm > int(seq[i]):

            minm = int(seq[i])

            minin = i

        i += 1

    temp = seq[minin]

    seq[minin] = seq[maxin]

    seq[maxin] = temp

    for \_ in seq:

        print(\_)

#11

def delete():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    ind  = int(input("enter the value of index : "))

    for \_ in range(ind, len(seq)):

        if (\_+1) != len(seq):

            temp = seq[\_]

            seq[\_] = seq[\_+1]

            seq[\_+1] = temp

    seq.pop()

    for \_ in seq:

        print(\_)

#12

def inset():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    ind  = int(input("enter the value of index : "))

    val = input("enter the value to insert : ")

    for \_ in range(ind, len(seq)):

        temp = seq[\_]

        seq[\_] =  val

        val = temp

    seq.append(val)

    for \_ in seq:

        print(\_)

    #13

def pair():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

        ans = 0

    n = len(seq)

    for i in range(0 , n):

        for j in range(i + 1, n):

            if (seq[i] == seq[j]):

                ans += 1

    print("number of pairs : ", ans )

#14

def uniq():

    seq = []

    value = 0

    while value != ('y' or 'Y'):

        value = input("enter a value  , enter y to stop : ")

        if value != ('y' or 'Y'):

            seq.append(value)

    unique\_list = []

    for x in seq:

        if x not in unique\_list:

            unique\_list.append(x)

    for x in unique\_list:

        print(x)

#15

def bowling():

    seq = []

    N = int(input("enter the value of N"))

    for \_ in range(0,N):

        seq.append(\_)

    k = int(input("enter the number of throws : "))

    for \_ in range(1,k+1):

        print("enter pair the values of {0} throw".format(\_))

#16

global N

N = 8

def printSolution(board):

    for i in range(N):

        for j in range(N):

            print (board[i][j])

        print

# A utility function to check if a queen can

# be placed on board[row][col]. Note that this

# function is called when "col" queens are

# already placed in columns from 0 to col -1.

# So we need to check only left side for

# attacking queens

def isSafe(board, row, col):

    # Check this row on left side

    for i in range(col):

        if board[row][i] == 1:

            return False

    # Check upper diagonal on left side

    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

        if board[i][j] == 1:

            return False

    # Check lower diagonal on left side

    for i, j in zip(range(row, N, 1), range(col, -1, -1)):

        if board[i][j] == 1:

            return False

    return True

def solveNQUtil(board, col):

    # base case: If all queens are placed

    # then return true

    if col >= N:

        return True

    # Consider this column and try placing

    # this queen in all rows one by one

    for i in range(N):

        if isSafe(board, i, col):

            # Place this queen in board[i][col]

            board[i][col] = 1

            # recur to place rest of the queens

            if solveNQUtil(board, col + 1) == True:

                return True

            # If placing queen in board[i][col

            # doesn't lead to a solution, then

            # queen from board[i][col]

            board[i][col] = 0

    # if the queen can not be placed in any row in

    # this colum col then return false

    return False

# This function solves the N Queen problem using

# Backtracking. It mainly uses solveNQUtil() to

# solve the problem. It returns false if queens

# cannot be placed, otherwise return true and

# placement of queens in the form of 1s.

# note that there may be more than one

# solutions, this function prints one of the

# feasible solutions.

def solveNQ():

    board = [ [0, 0, 0, 0, 0, 0, 0, 0],

    [0, 0, 0, 0, 0, 0, 0, 0],

    [0, 0, 0, 0, 0, 0, 0, 0],

    [0, 0, 0, 0, 0, 0, 0, 0],

    [0, 0, 0, 0, 0, 0, 0, 0],

    [0, 0, 0, 0, 0, 0, 0, 0],

    [0, 0, 0, 0, 0, 0, 0, 0],

    [0, 0, 0, 0, 0, 0, 0, 0]

    ]

    sol=False

    for \_ in range (0,8):

        row=int(input("enter a queen row : "))

        col = int(input("enter a quen col : "))

        sol =isSafe(board,row,col)

        board[row][col] = 1

    if sol == True:

        print("No")

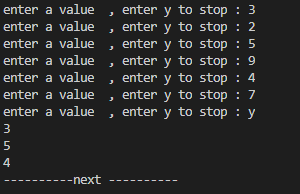
    else:

        print("yes")

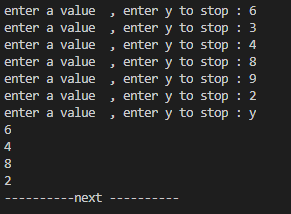
if \_\_name\_\_ == "\_\_main\_\_":

    solveNQ()

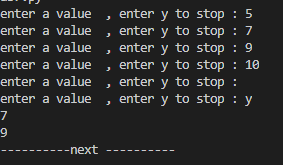
No1:



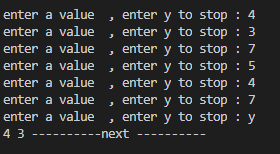
No2:



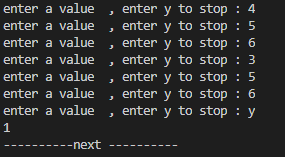
No3:



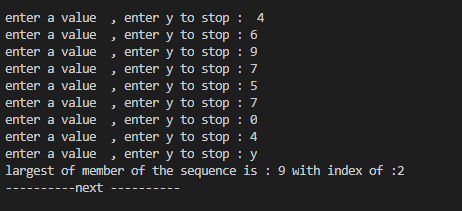
No4:



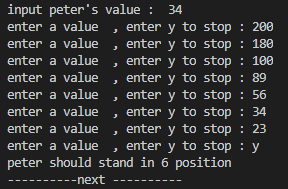
No5:



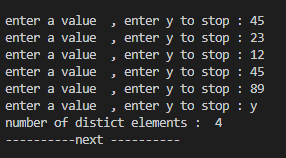
No6:



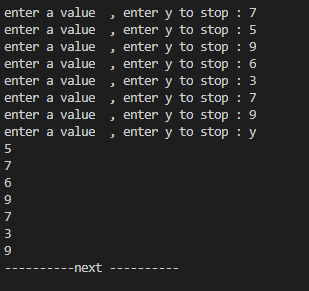
No7:



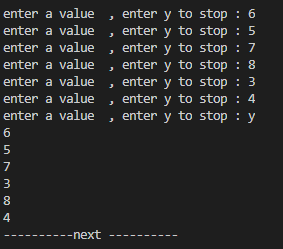
No8:



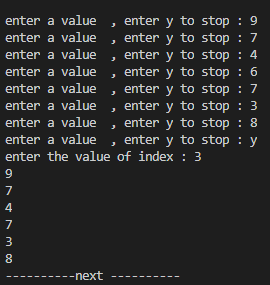
No9:



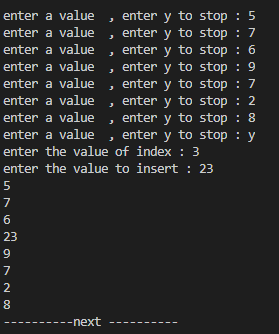
No10:



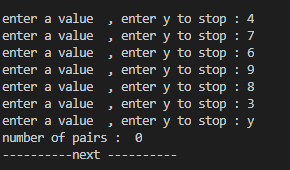
No11:



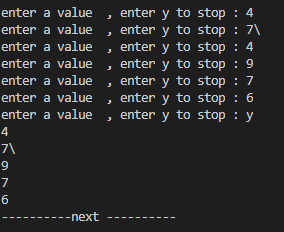
No12:



No13:



No14:



No16:

