IF INPUT IS 1 LINE: (file = open)

f = open("swap3.txt", "r")

f\_out = open("swap.out3.txt", "w")

n = int(f.readline())

a1 = int(f.readline())

b1 = int(f.readline())

c1 = int(f.readline())

print(n)

print(a1)

print(b1)

print(c1)

f\_out.write(str(n) + str(a1) + str(b1) + str(c1))

IF INPUT IS 2 LINE: (file = open)

f = open("swap.txt", "r")

f\_out = open("swap.out1.txt", "w")

n, k = map(int, f.readline().split())

a1, a2 = map(int, f.readline().split())

b1, b2 = map(int, f.readline().split())

print(n, k)

print(a1, a2)

print(b1, b2)

f\_out.write(str(n) + str(k)+"\n")

f\_out.write(str(a1) + str(a2)+ "\n")

f\_out.write(str(b1) + str(b2))

IF INPUT IS 3 LINE: (file = open)

f = open("swap2.txt", "r")

f\_out = open("swap.out2.txt", "w")

n, k , p = map(int, f.readline().split())

a1, a2, a3 = map(int, f.readline().split())

b1, b2, b3 = map(int, f.readline().split())

print(n, k, p)

print(a1, a2, a3)

print(b1, b2, b3)

f\_out.write(str(n) + str(k) + str(p)+"\n")

f\_out.write(str(a1) + str(a2) + str(a3)+ "\n")

f\_out.write(str(b1) + str(b2) + str(b3))

IF INPUT IS 1 LINE: (sys)

import sys

sys.stdin = open("swap3.txt", "r")

sys.stdout = open("swap.out3.txt", "w")

n = int(input())

a1 = int(input())

b1 = int(input())

c1 = int(input())

print(n)

print(a1)

print(b1)

print(c1)

IF INPUT IS 2 LINE: (sys)

import sys

sys.stdin = open("swap.txt", "r")

sys.stdout = open("swap.out1.txt", "w")

n, k = map(int, input().split())

a1, a2 = map(int, input().split())

b1, b2 = map(int, input().split())

print(n, k)

print(a1, a2)

print(b1, b2)

IF INPUT IS 3 LINE: (sys)

import sys

sys.stdin = open("swap2.txt", "r")

sys.stdout = open("swap.out2.txt", "w")

n, k , p = map(int, input().split())

a1, a2, a3 = map(int, input().split())

b1, b2, b3 = map(int, input().split())

print(n, k, p)

print(a1, a2, a3)

print(b1, b2, b3)

HOW TO FIND THE DATE OF X MONTHS BEFORE/AFTER TODAY

import datetime

print("6 months after today")

print((datetime.date.today()+datetime.timedelta(6\*365/12)).isoformat())

print("6 months before today")

print((datetime.date.today()-datetime.timedelta(3\*365/12)).isoformat())

print("4 months after today")

print((datetime.date.today()+datetime.timedelta(4\*365/12)).isoformat())

HOW TO PRINT A CALENDAR

import calendar

cal = calendar.TextCalendar(calendar.SUNDAY)

print(cal.formatyear(1998,2,1,1,3))

year:1998, (2) = column width, 1 = lines per week, 1 = space between  columns

3 = how many months per column

HOW TO PRINT A CALENDAR(USER ENTER YEAR/MONTH)

import calendar

print("print a calendar for any year and month:")

month = int(input("month(mm):"))

year = int(input("year(yyyy):"))

print('\n')

HOW TO PRINT WHEN IS TODAY, TOMORROW AND YESTERDAY

import datetime

today = datetime.date.today()

yesterday = today - datetime.timedelta(days = 1)

tomorrow = today + datetime.timedelta(days = 1)

print('Yesterday : ',yesterday)

print('Today : ',today)

print('Tomorrow : ',tomorrow)

HOW TO TELL THE CURRENT DATE/TIME IN PYTHON

import time

import datetime

print("current date and time", datetime.datetime.now())

print("current year", datetime.date.today().strftime("%Y"))

print("month of the year", datetime.date.today().strftime("%B"))

print("week of the year", datetime.date.today().strftime("W"))

print("week of the year", datetime.date.today().strftime("w"))

print("day of the year", datetime.date.today().strftime("%j"))

print("day of the month", datetime.date.today().strftime("%d"))

print("day of the week", datetime.date.today().strftime("%a"))

HOW TO TELL THE DATE MORE THAN 1 DAYS FROM TODAY

from datetime import date, timedelta

newdatethrus = date.today()-timedelta(2)

print("current date:", date.today())

print("2 days ago:", newdatethrus)

HOW TO USE SIMULTANEOUS EQUATIONS IN PYTHON

from sympy import symbols, Eq, solve

x,y,z=symbols("x y z")

equation1 = Eq((3\*x + 2\*y - z),4)

equation2 = Eq((-2\*x + -2\*y + 3\*z),30)

equation3 = Eq((5\*x - 2\*y - z),30)

print(" equation1: ", equation1)

print(" equation2: ", equation2)

print(" equation3: ", equation3)

solution = solve((equation1,equation2,equation3),(x,y,z))

print(" this solution is: ", solution)

HOW TO USE DICTIONARIES IN PYTHON

make\_a\_dictionary\_1 = {"a":1, "b":2, "c":3}

make\_a\_dictionary\_2 = {"ben":10, "tom":4, "mike":8, "james":7}

key1="a"

key2="b"

key3="c"

key4="d"

print(key1 in make\_a\_dictionary\_1)

print(key2 in make\_a\_dictionary\_1)

print(key3 in make\_a\_dictionary\_1)

print(key4 in make\_a\_dictionary\_1)

key5="ben"

key6="tom"

key7="mike"

key8="timothy"

print(key5 in make\_a\_dictionary\_2)

print(key6 in make\_a\_dictionary\_2)

print(key7 in make\_a\_dictionary\_2)

print(key8 in make\_a\_dictionary\_2)

HOW TO TELL IF AN ENTERED INTEGER IS IN EXISTING DICTIONARY

my\_dictionary={"january":5, "march":10,"may":15, "july":20,"september":25, "december":30}

new\_key = "april"

new\_value = 8

if new\_key not in my\_dictionary:

    my\_dictionary[new\_key]=new\_value

else:

    print("This month and date already exist in this dictionary")

print(my\_dictionary)

HOW TO COMBINE DICTIONARIES IN PYTHON

dictionary1 = {"a":1, "b":2 , "c":3}

dictionary2 = {"d":1, "e":2 , "f":3}

dictionary3 = {"g":1, "h":2 , "i":3}

dictionary4 = {"j":1, "k":2 , "l":3}

dictionary5 = {"c":4, "d":5 , "e":7}

dictionary\_one\_two = dictionary1|dictionary2

print(dictionary\_one\_two)

dictionary\_three\_four = dictionary3|dictionary4

print(dictionary\_three\_four)

dictionary\_one\_two\_three\_four = dictionary1|dictionary2|dictionary3|dictionary4

print(dictionary\_one\_two\_three\_four)

dictionary\_one\_five = dictionary1|dictionary5

print(dictionary\_one\_five)

HOW TO NEST DICTIONARIES (NESTED LISTS)

my\_list = [["a",1],["b",1],["c",3],["d",3],["f",3],["f",4],["g",4]]

new\_dict={}

for nested\_list in my\_list:

key=nested\_list[0]

vaue = nested\_list[1]

    new\_dict[key] = value

print(new\_dict)

HOW TO TELL THE DIFFERENCE IN DICTIONARIES IN PYTHON

dictionary1 = {"a":1, "b":2 , "c":3}

dictionary2 = {"d":1, "e":4 , "f":7}

dictionary3 = {"g":1, "h":9 , "i":3}

value = len(set(dictionary1.values()))

value2 = len(set(dictionary2.values()))

value3 = len(set(dictionary3.values()))

if value == 0:

print("empty")

elif value == 1:

print("same values")

else:

    print("different values")

if value2 == 0:

print("empty")

elif value2 == 1:

print("same values")

else:

    print("different values")

if value3 == 0:

print("empty")

elif value3 == 1:

print("same values")

else:

print("different values")

HOW TO FIND MAX NUMBER IN A DICTIONARY

dictionary1 = {"a":3, "b":3 , "c":3}

dictionary2 = {"a":0, "b":1 , "c":2}

dictionary3 = {}

if dictionary1:

    max\_value = max(set(dictionary1.values()))

    print(max\_value)

else:

    print(None)

if dictionary2:

    max\_value = max(set(dictionary2.values()))

    print(max\_value)

else:

    print(None)

if dictionary3:

    max\_value = max(set(dictionary3.values()))

    print(max\_value)

else:

print(None)

HOW TO FIND MAX/MIN SUM OF NUMBERS IN A DICTIONARY

my\_list = {

    "a":[1,2,3,4,5],

    "b":[-5,10,20,-10,5],

    "c":[2,4,6,8,10,12],

    "d":[5,10,15,20,15],

    "e":[100,20,-50,-75,-30]

}

max\_sum = None

for list\_value in my\_list.values():

    list\_sum = sum(list\_value)

    if max\_sum is None:

        max\_sum = list\_sum

    elif max\_sum < list\_sum:

        max\_sum = list\_sum

print(max\_sum)

min\_sum = None

for list\_value in my\_list.values():

    list\_sum = sum(list\_value)

    if min\_sum is None:

        min\_sum = list\_sum

    elif min\_sum > list\_sum:

        min\_sum = list\_sum

print(min\_sum)

HOW TO FIND FRQUENCIES IN A DICTIONARY

my\_dict={

    "a":4,

    "b":4,

    "c":5,

    "d":7,

    "e":8,

    "f":1,

    "g":9,

    "h":10

}

freq\_dict = {}

for value in my\_dict.values():

if value in freq\_dict:

        freq\_dict[value]+=1

else:

        freq\_dict[value]=1

print(freq\_dict)

HOW TO MAKE A GRAPH IN PYTHON

import matplotlib.pyplot as plt

x = [1,2,3,4,5,6,7,8,9]

y = [2,4,6,8,10,12,14,16,18]

plt.plot(x,y)

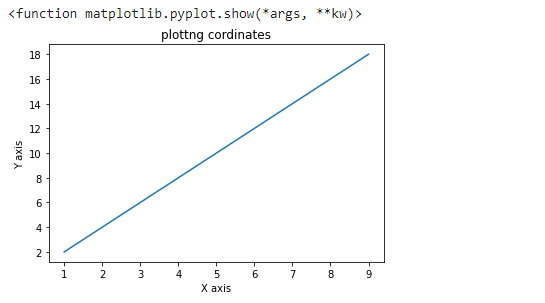
plt.xlabel("X axis")

plt.ylabel("Y axis")

plt.title("plottng cordinates")

plt.show

OUTPUT:



"r" - Read - Default value. Opens a file for reading, error if the file does not exist

“a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

HOW TO READ AND OPEN FILES IN PYTHON

file = open('bx.txt', 'r')

readfile = file.readline()

while readfile:

    print(readfile)

    readfile=file.readline()

import sys

sys.stdin = open('bx.txt','r')

while sys.stdout:

    print(read.close)

    bx.txt = int(read.readline())

fileToRead = open('some\_data.txt','r')

fileToWrite = open('newfile1.txt','w')

temp = fileToRead.read()

fileToWrite.write(temp.upper())

fileToRead.close()

fileToWrite.close()

fileToRead = open('some\_data.txt','r')

fileToWrite = open('newfile2.txt','w')

temp = fileToRead.read()

fileToWrite.write(temp.upper())

fileToRead.close()

fileToWrite.close()

HOW TO READ AND PRINT THE INFORMATION IN FILES OUT IN PYTHON

with open("names.txt","r") as o:

    data = o.read()

print(data)

print("break")

with open("names1.txt","w") as read:

    data1=read.readline()

print(data)

print("break")

with open("names2.txt","w") as read:

    data1=read.readline()

print(data)

print("break")

with open("names3.txt","r") as read:

    data1=read.readline(20)

print(data)

f=open("names.txt","r")

print(f.readline())

print(f.readline())

print(f.readline())

print(f.readline())

f=open("names.txt","r")

for x in f:

    print(x)

    print("name",x)

f.close()

HOW TO READ AND OPEN FILES IN PYTHON

file = open('some\_data.txt', 'r')

readfile = file.readline()

while readfile:

    print(readfile)

    readfile=file.readline

fileToRead = open('bx.txt','r')

fileToWrite = open('newfile2.txt','w')

temp = fileToRead.read()

fileToWrite.write(temp.upper())

fileToRead.close()

fileToWrite.close()

HOW TO COUNT THE NUMBER OF LINES IN A FILE

file = open('SHORTSTORY.txt','r')

read = file.readline()

count = 0

while read:

    count += 1

    read =  file.readline()

print(count)

HOW TO FIND IF NUMBERS IN ANOTHER FILE IS EQUAL IN PYTHON

with open("data.txt","r") as o:

    for o in range:

        if(a==b) and (a==c):

            print("equal")

        elif(d==e == f):

            print("equal")

        else:

            print("not equal")

HOW TO REVERSE INTEGERS (4 METHODS)

METHOD 1

a = "python"

for char in reversed(a):

    print(char, end="")

    print(" ")

METHOD 2

b = "benjamin"

for i in reversed(range(len(b))):

    print(b[i],end="")

    print(" ")

METHOD 3

c = "hello"

for i in range(len(c)-1):

    print(c[i],end="")

    print(" ")

METHOD 4

d = "happy"

for char in reversed(b):

    print(char,end="")

    print(" ")

Python code to find how many lines have letter starting with t

def line\_count():

    f = open("word.txt","r")

    count = 0

    for line in f:

        if line[0] not in "t":

                count +=1

        f.close()

        print("no lines start with letter t :)")

line\_count()

#program to count words in a file

file = open("number.txt", "rt")

data = file.read()

words = data.split()

print(len(words))

#part 2

def count\_words():

    f = open("word4.txt","r")

    print(f.reaad())

    count = 0

    data = f.read()

    words = dta.strip().split()

    for words in words:

        count+= 1

    print("total word in my file is: " , count)

    f.close()

count\_words()

How to print the Nth line of a file

def file\_read(text,nlines):

    from itertools import islice #islice is to cut things

    with open(text) as f:

        or line in islice(f,nlines):

            print(line)

file\_read("word2.txt", 4) #1,2,3 and 4

how to count number of lowercase letter in file

with open("word5.txt") as file:

    count = 0

    text = file.read()

    for letter in text:

        if letter.islower():

            count += 1

    print(count)

#how to count number of uppercase letter in file

with open("word5.txt") as file:

    count = 0

    text = file.read()

    for letter in text:

        if letter.isupper():

            count += 1

    print(count)

display words ending with letter 'e'

def read():

    f = open("word7.txt","r")

    data = f.read()

    words = data.split()

    for i in words:

        if i[-1] == 'e':

            print(i)

            count = i.split()

            number\_of\_words = len(count)

            print(number\_of\_words)

read()

HOW TO SOLVE THE FOLLOWING STATEMENT (ALGEBRA) USING PYTHON:

x +   y +  z = 4

x + 2y + 3z = 10

x -y - 4z = 20

from sympy import symbols, Eq, solve

x, y, z = symbols("x y z")

equation\_1 = Eq((x + y + z), 4)

equation\_2 = Eq((x + 2\*y + 3\*z), 10)

equation\_3 = Eq((x - y - 4\*z), 20)

print("Equation 1:", equation\_1)

print("Equation 2:", equation\_2)

print("Equation 3:", equation\_3)

solution = solve((equation\_1, equation\_2, equation\_3), (x, y, z))

print("Solution:", solution)

Write a Python program that checks if a number is prime or not. If the number is prime, it should print "Prime". If the number is not prime, it should print "Not prime".

n=int(input("enter number here"))

if n==0 or n==1:

    print("not a prime number")

else:

    for i in range(2,n):

        if n%i ==0:

            print("not a prime number")

        else:

            print("prime")

Write a Python program that solves for the factorial of an entered integer

n=int(input("enter some numbers : "))

factorial=1

for i in range(2,n+1):

factorial\*=i

print(factorial)

write a python program to print triangle given the number entered by user

EXAMPLE INPUT: 4

EXAMPLE OUTPUT:

\*

\* \*

\* \* \*

\* \* \* \*

EXAMPLE INPUT: 3

EXAMPLE OUTPUT:

\*

\* \*

\* \* \*

n=int(input("enter number:"))

for i in range(1,n+1):

print("\*" \* i)

The first cow eat, one bread, the second cow eats 2 bread, and so on,

what is the toatl amount of bread that will be eaten at the 20th cow?

totalbreadeaten = 0

for cow in range(1,13):

    totalbreadeaten += cow

print(totalbreadeaten)

# totalbreadeaten = 0

totalbread = 100

for cow in range(1,13):

    totalbreadeaten += cow

totalbreadleft = totalbread-totalbreadeaten

print(totalbreadleft)

FOR LOOPS IN PYTHON (ASK USER FOR INPUT)

n = int(input("Enter the value of n: "))

for i in range (1, 13):

    print(f"{n} x {i} = {n \* i}")

b = int(input("Enter the value of b: "))

for i in range (1, 13):

    print(f"{n} + {i} = {n + i}")

c = int(input("Enter the value of c: "))

for i in range (1, 13):

    print(f"{n} - {i} = {n - i}")

d = int(input("Enter the value of d: "))

for i in range (1, 13):

    print(f"{n} / {i} = {n / i}")

HOW TO WRITE A PROGRAM THAT PRINT NUMBER BETWEEN START AND END NUMBER ENTERED BY THE USER

EXAMPLE INPUT:

4, 9

EXAMPLE OUTPUT:

4

5

6

7

8

9

EXAMPLE INPUT: 1, 10

EXAMPLE OUTPUT:

1

2

3

4

5

6

7

8

9

10

start\_value = int(input("enter number"))

end\_value = int(input("enter number"))

for i in range(start\_value,end\_value + 1):

    if i%2 == 0:

        print(i)

HOW TO MAKE A CALCULATOR WITH PYTHON

addition = 1

subtraction = 2

multiplication = 3

division = 4

interger\_division = 5

modulo = 6

number1 = int(input('Enter your first number: '))

number2 = int(input('Enter your second number: '))

operation = int(input("enter the number for the operation you would like"))

if operation == addition:

    result = number1+number2

    print(f"the result of {number1} + {number2} is: {result}")

elif operation == subtraction:

    result = number1-number2

    print(f"the result of {number1} - {number2} is: {result}")

elif operation == multiplication:

    result = number1\*number2

    print(f"the result of {number1} \* {number2} is: {result}")

elif operation == division:

    result = number1 / number2

    print(f"the result of {number1} / {number2} is: {result}")

elif operation == interger\_division:

    result = number1 // number2

    print(f"the result of {number1} // {number2} is: {result}")

elif operation == modulo:

    result = number1 % number2

    print(f"the result of {number1} % {number2} is: {result}")

else:

    print("invalid input")

📌 Description:

Write a Python program that simulates the "Rock, Paper, Scissors" game.

The game should ask the user to enter an option (either "Rock", "Paper",  or "Scissors").

The player should play against the computer, which will select a random  option

The computer's selection will be compared against the player's selection  to determine who wins.

A descriptive message should be displayed indicating if the player won,  lost, or if the game ended in a tie.

import random

choice = input("hi welcome to rock paper sisor please choose rock sissor or paper!: ")

list1 = ["rock", "paper", "scissors"]

computer = random.choice(list1)

if choice == computer:

    print("it’s a tie! try again")

elif choice == "rock":

    if computer == "scissors":

        print("you win! your opponent chose sissors")

    else:

        print("u lose! your opponent chose paper")

elif choice == "paper":

    if computer == "rock":

        print("you win! your opponent chose rock")

    else:

        print("u lose! your opponent chose sissor")

elif choice == "scissors":

    if computer == "paper":

        print("you win! your opponent chose paper")

    else:

        print("you lose! your opponent chose sissor")

HOW TO WRITE A PYTHON CODE THAT FINDS THE LARGEST NUMBER AONG 3 NUMBERS

num1 = 1

num2 = 3

num3 = 4

if (num1 >= num2) and (num1 >= num3):

   largest = num1

elif (num2 >= num1) and (num2 >= num3):

   largest = num2

else:

   largest = num3

print(largest)

num4 = 1

num5 = 4

num6 = 15

if (num4 >= num5) and (num5 >= num6):

   largest = num1

elif (num5 >= num4) and (num4 >= num6):

   largest = num5

else:

   largest = num6

print(largest)

HOW TO WRITE A PYTHON CODE THAT CHECK IS LETTER ENTERED IS A VOWEL OR CONSONENT

vowel\_list = ["a", "e", "i", "o", "u"]

letter = input("Input a letter of the alphabet: ")

if letter in vowel\_list:

    print("this is a vowel")

elif letter not in vowel\_list:

    print("this word is a consonant")

else:

    print("only enter alphabets")

#PYTHON CODE THAT DETERMINE WHICH YEAR IS A LEAP YEAR

method 1

year = 1912

if year % 4 != 0:

    print( f"{year} is a common year")

elif year % 100 != 0:

    print( f"{year} is a leap year")

elif year % 400 != 0:

    print( f"{year} is a common year")

else:

    print( f"{year} is a leap year")

method 2

year = int(input("enter year"))

if year % 400 ==0:

    print(f"{year} is leap year")

elif year % 100 == 0:

    print(f"{year} is not a leap year")

elif year %4 == 0:

    print(f"{year} is leap year")

else:

    print(f"{year} is not a leap year")

method 3

year = int(input("enter year"))

if year % 4 == 0:

        if year % 100 == 0:

            if year % 400 == 0:

                print(f"{year} is leap year")

            else:

                print(f"{year} is not a leap year")

        else:

        print(f"{year}is leap year")

else:

print(f"{year} is not a leap year")

Write a Python program that prints the number of days in a given month.

The value of the variable month is the name of the month with the first letter capitalized.

Do not consider leap years for the number of days in February.

You can add a customized message. For example: "<month> has: <num\_days> days."

METHOD 1

from calendar import monthrange

month=int(input("enter a year"))

year=int(input("enter a month"))

day = monthrange(month,year)[1]

print(day)

METHOD 2

print("find number of day in month choose ur month:")

month = input()

month\_31\_days = ("January, March, May, July, August, October, and December")

Write a Python program that determines if three numbers (a, b, and c) are in increasing order, decreasing order, or none.

If a > b > c, print "Decreasing Order".

If a < b < c, print "Increasing Order".

Else, print "None".

if a>b>c:

    print("abc: decreasing order")

elif a<b<c:

    print("abc: increasing order")

else:

    print("NIL")

if d>e>f:

    print("def: decreasing oder")

elif d<e<f:

    print("def: increasing order")

else:

    print("NIL")

Write a Python program that prints the corresponding season based on the  value of the variable season\_num.

The possible values of season\_num are: 1 for Spring, 2 for Summer, 3 for  Fall, 4 for Winter.

If the value of season\_num is neither one of these values, print "Please  enter a valid number".

spring = 1

summer = 2

fall = 3

winter = 4

seasonnum = int(input("enter a season number 1,2,3 or 4"))

if seasonnum == spring:

    print("it is spring now")

elif seasonnum == summer:

    print("it is summer now")

elif seasonnum == fall:

    print("it is fall now")

elif seasonnum == winter:

    print("it is swinter now")

else:

    print("underfined")

You are given a list of n integers, and your task is to calculate the  number of distinct values in the list.

Input

The first input line has an integer n: the number of values.

The second line has n integers x1,x2,…,xn.

Output

Print one integers: the number of distinct values.

Constraints

1≤n≤2⋅105

1≤xi≤109

Example

Input:

5

2 3 2 2 3

Output:

2

n = int(input())

number = sorted(map(int, input().split()))

count = 1

for i in range(1, n):

    if number[i] != number[i - 1]:

        count += 1

print(count)

Farmer John's 3 prize cows, Bessie, Elsie, and Mildred, are always wandering off to the far reaches of the farm! He needs your help herding them  back together.

The main field in the farm is long and skinny -- we can think of it as a number line, on which a cow can occupy any integer location. The 3 cows are currently situated at different integer locations, and Farmer John wants to move them so they occupy three consecutive locations (e.g., positions 6, 7, and 8).

Unfortunately, the cows are rather sleepy, and Farmer John has trouble getting their attention to make them move. At any point in time, he can only make a cow move if she is an "endpoint" (either the minimum or maximum position among all the cows). When he moves a cow, he can instruct her to move to any unoccupied integer location as long as in this new location she is no longer an endpoint. Observe that over time, these types of moves tend to push the cows closer and closer together.

Please determine the minimum and maximum number of moves possible before the cows become grouped in three consecutive locations.

INPUT FORMAT (file herding.in):

The input file contains one line with three space-separated integers, giving the locations of Bessie, Elsie, and Mildred. Each location is an integer in the range 1…109.

OUTPUT FORMAT (file herding.out):

The first line of output should contain the minimum number of moves Farmer John needs to make to group the cows together. The second line of output should contain the maximum number of such moves he could conceivably make before the cows become grouped together.

SAMPLE INPUT:

4 7 9

SAMPLE OUTPUT:

1

2

import sys

sys.stdin = open('herding.in', 'r')

sys.stdout = open('herding.out', 'w')

cowb,cowe,cowm = map(int, input().split())

if cowm == cowb + 2:

  print("moves is 0")

elif cowe == cowb + 2 or cowm == cowe + 2:

  print("moves is 1")

else:

  print("moves is 2")

print(max(cowe - cowb, cowm- cowe) - 1)

Farmer John owns 7 dairy cows: Bessie, Elsie, Daisy, Gertie, Annabelle, Maggie, and Henrietta. He milks them every day and keeps detailed records on the amount of milk provided by each cow during each milking session. Not surprisingly, Farmer John highly prizes cows that provide large amounts of milk.

Cows, being lazy creatures, don't necessarily want to be responsible for producing too much milk. If it were up to them, they would each be perfectly content to be the lowest-producing cow in the entire herd. However, they keep hearing Farmer John mentioning the phrase "farm to table" with his human friends, and while they don't quite understand what this means, they have a suspicion that it actually may not be the best idea to be the cow producing the least amount of milk. Instead, they figure it's safer to be in the position of producing the second-smallest amount of milk in the herd. Please help the cows figure out which of them currently occupies this desirable position.

INPUT FORMAT (file notlast.in):

The input file for this task starts with a line containing the integer N (1≤N≤100), giving the number of entries in Farmer John's milking log.

Each of the N following lines contains the name of a cow (one of the seven above) followed by a positive integer (at most 100), indicating the amount of milk produced by the cow during one of its milking sessions.

Any cow that does not appear in the log at all is assumed to have produced no milk.

OUTPUT FORMAT (file notlast.out):

On a single line of output, please print the name of the cow that produces the second-smallest amount of milk. More precisely, if M is the minimum total amount of milk produced by any cow, please output the name of the cow whose total production is minimal among all cows that produce more than M units of milk. If several cows tie for this designation, or if no cow has this designation (i.e., if all cows have production equal to M), please output the word "Tie". Don't forget to add a newline character at the end of your line of output. Note that M=0 if one of the seven cows is completely absent from the milking log, since this cow would have produced no milk.

SAMPLE INPUT:

10

Bessie 1

Maggie 13

Elsie 3

Elsie 4

Henrietta 4

Gertie 12

Daisy 7

Annabelle 10

Bessie 6

Henrietta 5

SAMPLE OUTPUT:

Henrietta

In this example, Bessie, Elsie, and Daisy all tie for the minimum by each producing 7 units of milk. The next-largest production, 9 units, is due to Henrietta.

def main():

    f = open("notlast.in.txt","r")

    f1 = open("notlast.out.txt","w")

    n = int(f.readline())

    cow\_log = {"bessie":0,"maggie":0,"elsie":0,"henriett":0,"gertie":0,"daisy":0,"annabelle":0}

    for i in range(n):

        row = f.readline().strip().split()

        cow\_log[row[1]] += int[row(1)]

    cow\_log = {k:a for k, a in sorted(cow\_log.items(), key = lambda item:item[1])}

    result = ""

    best\_producer = max(cow\_log.values())

    for key,value in cow\_Log.itmes():

        if cow\_log[key] > best\_producer:

            result = key

            break

        best\_producer = cow\_log[key]

    values = list(cow\_log.values())

    if len(set(values)) == 1:

        result = "draw"

    elif values.count(cow\_milk[result]) > 1:

        result = "draw"

    print(result,file = f1)

    print(str(cow\_log))

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Bessie the cow has designed what she thinks will be the next big hit videogame: "Angry Cows". The premise, which she believes is completely original, is that the player shoots a cow with a slingshot into a one-dimensional scene consisting of a set of hay bales located at various points on a number line; the cow lands on a hay bale with sufficient force to cause the bale to explode, which in turn might set of a chain reaction that causes additional nearby hay bales to explode. The goal is to use a single cow to start a chain reaction that detonates as many hay bales as possible.

There are N hay bales located at distinct integer positions x1,x2,…,xN on the number line. If a cow is launched onto a hay bale at position x, this hay bale explodes with a "blast radius" of 1, meaning that any other hay bales within 1 unit of distance are also engulfed by the explosion. These neighboring bales then themselves explode (all simultaneously), each with a blast radius of 2, so these explosions may engulf additional yet-unexploded bales up to 2 units of distance away. In the next time step, these bales also explode (all simultaneously) with blast radius 3. In general, at time t a set of hay bales will explode, each with blast radius t. Bales engulfed by these explosions will themselves explode at time t+1 with blast radius t+1, and so on.

Please determine the maximum number of hay bales that can explode if a single cow is launched onto the best possible hay bale to start a chain reaction.

INPUT FORMAT (file angry.in):

The first line of input contains N (1≤N≤100). The remaining N lines all contain integers x1…xN (each in the range 0…1,000,000,000).

OUTPUT FORMAT (file angry.out):

Please output the maximum number of hay bales that a single cow can cause to explode.

SAMPLE INPUT:

6

8

5

6

13

3

4

SAMPLE OUTPUT:

5

In this example, launching a cow onto the hay bale at position 5 will cause the bales at positions 4 and 6 to explode, each with blast radius 2. These explosions in turn cause the bales at positions 3 and 8 to explode, each with blast radius 3. However, these final explosions are not strong enough to reach the bale at position 13.

#method 1 -> import system

import sys

sys.stdin = open("cowqueue.in.txt", "r")

sys.stdout = open("cowqueue.out.txr", "w")

n = int(input())

cows = []

time = 0

for i in range(n):

    cows.append(list(map(int, input().split())))

cows.sort()

for cow in cows:

  if time > cow[0]:

      time += cow[1]

  else:

      time = cow[0] + cow[1]

print(time)

#method 2-> file in, file out

fin = open("cowqueue.in.txt", "r")

fout = open("cowqueue2.out.txt", "w")

n = int(fin.readline())

cows = list()

time = 0

for i in range(n):

    x,y = map(int,fin.readline().split())

    cows.append((x,y))

    print(cows)

cows.sort()

for cow in cows:

    if time > cow[0]:

        time += cow[1]

    else:

        time = cow[0] + cow[1]

print(time)

fout.write(str(time))

fout.close()

fin.close()

#method 3 -> with open

with open('cowqueue.in.txt', 'r') as fin:

    time = []

    for i in range(int(fin.readline())):

            time.append(list(map(int, fin.readline().split())))

    time = sorted(time)

    currtime = 0

with open('cowqueue.out3.txt', 'w') as fout:

    for i in  time:

        currtime = max(i[0],currtime) + i[1]

    fout.write(str(currtime))

Farmer John has purchased a subscription to Good Hooveskeeping magazine for his cows, so they have plenty of material to read while waiting around in the barn during milking sessions. Unfortunately, the latest issue contains a rather inappropriate article on how to cook the perfect steak, which FJ would rather his cows not see (clearly, the magazine is in need of better editorial oversight).

FJ has taken all of the text from the magazine to create the string S of length at most 10^6 characters. From this, he would like to remove occurrences of a substring T to censor the inappropriate content. To do this, Farmer John finds the \_first\_ occurrence of T in S and deletes it. He then repeats the process again, deleting the first occurrence of T again, continuing until there are no more occurrences of T in S. Note that the deletion of one occurrence might create a new occurrence of T that didn't exist before. Please help FJ determine the final contents of S after censoring is complete.

INPUT FORMAT: (file censor.in)

The first line will contain S. The second line will contain T. The length of T will be at most that of S, and all characters of S and T will be lower-case alphabet characters (in the range a..z).

OUTPUT FORMAT: (file censor.out)

The string S after all deletions are complete. It is guaranteed that S will not become empty during the deletion process.

SAMPLE INPUT:

whatthemomooofun

moo

SAMPLE OUTPUT:

whatthefun

METHOD 1 -> FIN FOUT

fin = open("censor.in.txt", "r")

fout = open("censor.out.txt", "w")

S = fin.readline().strip()

T = fin.readline().strip()

ans = " "

for i in S:

    ans += i

    if len(ans) >= len(T) and ans[len(ans)-len(T):] == T:

        ans = ans[:len(ans) - len(T)]

fout.write(str(ans))

fout.close()

fin.close()

METHOD 2 -> IMPORT SYS

import sys

sys.stdin = open('censor.in.txt', 'r')

sys.stdout = open('censor.out', 'w')

s = input().strip()

t = input().strip()

ans = " "

for char in s:

  ans += char

  if ans[-len(t):] == t:

    ans=ans[:-len(t)]

print(ans)

METHOD 3 -> WITH OPEN

with open('censor.in.txt', 'r') as fin:

    data = fin.readlines()

    S = data[0].strip()

    T = data[1].strip()

    ans = " "

    for i in S:

        ans += i

        if ans[-len(T):] == T:

            ans = ans[:-len(T)]

print(ans)

with open('censor.out3.txt', 'w') as f\_out:

  f\_out.write(str(ans))

farmer john is a mad scientist. he believes that aliens have taken his two cows from his farm. farmer  john wants to find out what day the cows were  taken from his far.he think besis was taken (x(x0 days ago and carrie will be taken y(y)days from t day

x(x)

4x + y = 22

x - y = 3

from sympy import symbols, Eq, solve

import datetime

x,y = symbols("x y")

equation1 = Eq((4\*x + y ),22)

equation2 = Eq((x-y),3)

solution = solve((equation1,equation2),(x,y))

print(solution)

x = 5

y = 2

today = datetime.date.today()

bessie = today - datetime.timedelta(days = 25)

print("bessie was taken on,",bessie)

carrie = today + datetime.timedelta(days = 4)

print("carrie was taken on",carrie)

KNOWLEDGE ON LISTS

#KNOWLEDGE 1

cowa = [6,52,74,62,92]

cowb = [15,30,45,50,89]

cowa.extend(cowb) #extend always add the {cowa} into beginning of {cowb}

print(cowa)

cowb.append(cowa)

print(cowb)

#KNOWLEDGE 2

list1 = {6,52,74,62,92}

list2 = {15,30,45,50,89}

list1.extend(list2)

print(list2)

# () -> 'tuple' object has no attribute 'extend'

# {} -> 'set' object has no attribute 'extend'

#KNOWLEDGE 3

#copy list a and add list 2 to list a

#list b is at the back now

#make new var, extend data

cowa = [6,52,74,62,92]

cowb = [15,30,45,50,89]

bc1 = cowa.copy()

bc1.extend(cowb)

print(bc1)

#can switch data

cowa.extend(cowb)

print(cowa)

#KNOWLEDGE 4

cowa = [6,52,74,62,92]

cowb = [15,30,45,50,89]

bc1 = cowa.copy()

bc1.extend(cowb)

print(bc1)

#prints the same thing as what is in cow a

for item in cowa:

    bc1.append(item)

print(bc1)

print(cowb)

#KNOWLEDGE 5

#only print out digitsfrom list with numbers

cowa = [6,52,74,62,92]

cowb = [15,30,45,50,89]

cowc = [6,52,74,62,92]

cowd = [15,30,45,50,89]

cowe = ["bessie",34,"james",36,"john",50]

a1 = cowc[0]

b1 = cowc[1]

c1 = cowc[2]

d1 = cowc[3]

e1 = cowc[4]

print(a1)

print(c1)

print(e1)

answer=[ans for ans in cowe if isinstance(ans, (int,float))]

print(answer)

list1 = ["bessie",34,"james",36,"john",50]

#N = names

#A = ages

#name/age: eg: Bessie, 34

print("=============================")

print("A")

digit=[ans for ans in list1 if isinstance(ans, (int,float))]

print(digit)

print("=============================")

b = list1[0:2]

j = list1[2:4]

a = list1[4:6]

print(b)

print(j)

print(a)

print("=============================")

print("N")

list1.remove(34)

list1.remove(36)

list1.remove(50)

print(list1)

print("=============================")

#KNOWLEDGE 6

a = [1,2,3,4,5,6,7,8,9,10]

b = [2,4,6,8,10]

c = [9,7,5,3,1]

#sort only from small to big

a.sort()

print(a)

b.sort()

print(b)

c.sort()

print(c)

#smallest number

print("find smallest number")

a.sort()

print(a[0])

b.sort()

print(b[0])

c.sort()

print(c[0])

#biggest number

print("find largest number")

a.sort()

print(a[-1])

b.sort()

print(b[-1])

c.sort()

print(c[-1])

#KNOWLEDGE 7

#biggest

a = [1,2,3,4,5,6,7,8,9,10]

b = [2,4,6,8,10]

c = [9,7,5,3,1]

biggest = a[0]

for i in a:

    if i > biggest:

        biggest = i

print(biggest)

#smallest

smallest = a[0]

for i in a:

    if i < smallest:

        smallest = i

print(smallest)

big = b[0]

for i in b:

    if i>big:

        big = i

print(big)

#KNOWLEDGE 8

list\_f = [1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6]

list\_g = [9,1,1,5,9,4,2,7,2,9,5,3,2]

list\_h = [2,4,1,5,9,4,2,7,3,9,6,3,2,7,5,8,9,3]

myset\_f = set(list\_f)

myset\_g = set(list\_g)

print(myset\_f)

print(myset\_g)

for itemin list\_h:

    itemExist = False

    for x in uniquelist:

        if x == item:

            itemExist = True

            break

    if not itemExist:

        uniquelist.append(item)

print(uniquelist)

def function1():

    print("this is function1")

def function2():

    print("this is function 2")

list1 = [function1,function2]

Farmer John, in his old age, has unfortunately become increasingly grumpy and paranoid. Forgetting the extent to which bovine diversity helped his farm truly flourish over the years, he has recently decided to build a huge fence around the farm, discouraging cows from neighboring farms from visiting, and completely prohibiting entry from a handful of neighboring farms. The cows are quite upset by this state of affairs, not only since they can no longer visit with their friends, but since it has caused them to cancel participation in the International Milking Olympiad, an event to which they look forward all year.

Neighboring cows that still have the ability to enter Farmer John's property find the process has become more arduous, as they can enter only through a single gate where each cow is subject to intense questioning, often causing the cows to queue up in a long line.

For each of the NN cows visiting the farm, you are told the time she arrives at the gate and the duration of time required for her to answer her entry questions. Only one cow can be undergoing questioning at any given time, so if many cows arrive near the same time, they will likely need to wait in line to be processed one by one. For example, if a cow arrives at time 5 and answers questions for 7 units of time, another cow arriving at time 8 would need to wait until time 12 to start answering questions herself.

Please determine the earliest possible time by which all cows are able to enter the farm.

**INPUT FORMAT (file cowqueue.in):**

The first line of input contains NN, a positive integer at most 100. Each of the next NN lines describes one cow, giving the time it arrives and the time it requires for questioning; each of these numbers are positive integers at most 1,000,000.

**OUTPUT FORMAT (file cowqueue.out):**

Please determine the minimum possible time at which all the cows could have completed processing.

**SAMPLE INPUT:**

3

2 1

8 3

5 7

**SAMPLE OUTPUT:**

15

Here, first cow arrives at time 2 and is quickly processed. The gate remains briefly idle until the third cow arrives at time 5, and begins processing. The second cow then arrives at time 8 and waits until time 5+7=12 to start answering questions, finishing at time 12+3 = 15.

#METHOD 1

import sys

sys.stdin = open("cowqueue.in.txt", "r")

sys.stdout = open("cowqueue.out2.txt", "w")

n = int(input())

cows = []

for i in range(n):

    # cows.append(list(map(int, input().split())))

    cows.append([int(x)for x in input().split()])

cows.sort()

start = 0

for cow in cows:

    if start>cow[0]:

        start+=cow[1]

    else:

        start = cow[0]+cow[1]

print(start)

#METHOD 2

import sys

sys.stdin = open("cowqueue.in.txt", "r")

sys.stdout = open("cowqueue.out2.txt", "w")

n = int(input())

for i in range(n):

    cow.append([cows.append([int(x)for x in input().split()])

cows.sort()

start = cows[0][0] + cows[0][1]

for cow in cows[1:]:

    if cow[0]>start:

       start = cow[0] + cow[1]

    else:

        start+=cow[1]

print(start)

#METHOD 3

import sys

sys.stdin = open("cowqueue.in.txt", "r")

sys.stdout = open("cowqueue.out2.txt", "w")

n = int(input())

arrive = []

question = []

for i in range(N):

    first\_number,second\_number = map(int,input().split())

    arrive.append(first\_number)

    question.append(second\_number)

arrives.sory()

start = arrives[0]+question[0]

for cow in range(1,N):

    if start>arrives[i]:

        start = start+question[i]

    if start<arrives[i]:

        start = arrives[i]+question[i]

print(start)

print(N)

print(arrive)

print(question)

#METHOD 4

fin = open("cowqueue.in.txt", "r")

fout = open("cowqueueanswer.out.txt", "w")

n = int(fin.readline())

cows = []

for i in range(n):

    cows.append([int(x)for x in fin.readline().split()])

cows.sort()

start = cows[0][0] + cows[0][1]

for cow in cows[1:]:

    if cow[0]>start:

       start = cow[0] + cow[1]

    else:

        start+=cow[1]

fout.write(str(start))

fout.close()

CONDITIONAL PRACTICE

#logic for methord oone and two

def divide\_or\_substract(number1, number2):

    divide = number1 / number2

    if divide <= 5:

        return divide

    else:

       return number1 - number2

#methord 1

result\_1 = divide\_or\_substract(20, 5)

print("the answer is", result\_1)

result\_2 = divide\_or\_substract(40, 2)

print("the answer is", result\_2)

#methord 2

number1=int(input("first number: "))

number2=int(input("second number: "))

result\_3 = divide\_or\_substract(number1, number2)

print("the answer is", result\_3)

#methord 3 (use the data from a file)

f = open("n1.txt")

data= f.readline().strip().split("\n")

n1, n2 = map(int, data[0].split(' '))

print(n1)

print(n2)

result3 = divide\_or\_substract(n1, n2)

print("the answer is", result3)

fout=open("data.out4.txt",'w')

fout.write(str(result3))

question 1:

we have bessie and her friend bc

input

N = the number if times that they both got marbles from from john

the next line a and b represent how many marbles they got from from john each time.

output:

one line: who has the most marbles

question 2:

we have bessie and her friend bc

bc always got 3 times as many marbles as beesie

input

N = the number if times that farmer john have bessie marbles

the next line a represent how many marbles that bessie got from from john each time.

output:

one what is the diffrence between bessie and bc.(marbles)

'''

import sys

sys.stdin = open("filein\_1.txt", "r")

sys.stdout = open("filein1\_out.txt", "w")

n = input()

a1, b1 = map(int, input().split())

a2, b2 = map(int, input().split())

a3, b3 = map(int, input().split())

a4, b4 = map(int, input().split())

sum\_bessie = a1+a2+a3+a4

sum\_bc = b1+b2+b3+b4

if sum\_bc > sum\_bessie:

    print(sum\_bc)

elif sum\_bessie > sum\_bc:

    print(sum\_bessie)

else:

    print("0")

'''

import sys

sys.stdin = open("filein\_2.txt", "r")

sys.stdout = open("answer1.txt", "w")

n = int(input())

bessie = []

for i in range(n):

    first = map(int,input().split())

    bessie.extend(first)

print(bessie)

print(sum(bessie))

bc= []

for m in bessie:

    bc.append(m\*3)

print(bc)

print(sum(bc))

bessie\_sum = sum(bessie)

bc\_sum = sum(bc)

difference = bc\_sum-bessie\_sum

print("bc-bessie: ", difference)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**C - Made Up**

Time Limit: 2 sec / Memory Limit: 1024 MB

Score : *300300* points

### Problem Statement

Given are three sequences of length *NN* each: *A = (A\_1, A\_2, \dots, A\_N)A=(A1​,A2​,…,AN​)*, *B = (B\_1, B\_2, \dots, B\_N)B=(B1​,B2​,…,BN​)*, and *C = (C\_1, C\_2, \dots, C\_N)C=(C1​,C2​,…,CN​)*, consisting of integers between *11* and *NN* (inclusive).

How many pairs *(i, j)(i,j)* of integers between *11* and *NN* (inclusive) satisfy *A\_i = B\_{C\_j}Ai​=BCj​​*?

### Constraints

* *1 \leq N \leq 10^51≤N≤105*
* *1 \leq A\_i, B\_i, C\_i \leq N1≤Ai​,Bi​,Ci​≤N*
* All values in input are integers.

### Input

Input is given from Standard Input in the following format:

*NN*

*A\_1A1​* *A\_2A2​* *\ldots…* *A\_NAN​*

*B\_1B1​* *B\_2B2​* *\ldots…* *B\_NBN​*

*C\_1C1​* *C\_2C2​* *\ldots…* *C\_NCN​*

### Output

Print the number of pairs *(i, j)(i,j)* such that *A\_i = B\_{C\_j}Ai​=BCj​​*.

### Sample Input 1

3

1 2 2

3 1 2

2 3 2

### Sample Output 1

4

Four pairs satisfy the condition: *(1, 1), (1, 3), (2, 2), (3, 2)(1,1),(1,3),(2,2),(3,2)*.

### Sample Input 2

4

1 1 1 1

1 1 1 1

1 2 3 4

### Sample Output 2

16

All the pairs satisfy the condition.

### Sample Input 3

3

2 3 3

1 3 3

1 1 1

### Sample Output 3

0

f = open("input3.txt", "r")

def read\_data():

    N = int(f.readline())

    a = list(map(int, f.readline().split()))

    b = list(map(int, f.readline().split()))

    c = [int(x) for x in f.readline().split()]

    pairs = dict()

    for i in a:

        if i not in pairs:

            pairs[i]=1

        else:

            pairs[i]+=1

    valid\_pairs = 0

    for i in range(N):

        valid\_pairs += pairs.get(b[c[i]-1], 0)

    print(valid\_pairs, file = open("output3.txt","w"))

if \_\_name\_\_ == "\_\_main\_\_":

    read\_data()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Sum of Two Values

* **Time limit:** 1.00 s

* **Memory limit:** 512 MB

You are given an array of nn integers, and your task is to find two values (at distinct positions) whose sum is xx.  
  
**Input**  
  
The first input line has two integers nn and xx: the array size and the target sum.  
  
The second line has nn integers a1,a2,…,ana1,a2,…,an: the array values.  
  
**Output**  
  
Print two integers: the positions of the values. If there are several solutions, you may print any of them. If there are no solutions, print IMPOSSIBLE.  
  
**Constraints**

* 1≤n≤2⋅1051≤n≤2⋅105
* 1≤x,ai≤1091≤x,ai≤109

**Example**  
  
Input:  
4 8  
2 7 5 1  
  
Output:  
2 4

import sys

sys.stdin = open("number.in.txt", "r")

sys.stdout = open("number.out1.txt", "w")

n, x = map(int, input().split())

a = [int(i) for i in input().split()]

numbers = []

for i in range(n):

    numbers.append([a[i], i + 1])

numbers.sort()

ans = []

read\_left = 0

read\_right = n - 1

while read\_left < read\_right:

    left\_to\_right\_sum = numbers[read\_left][0] + numbers[read\_right][0]

    if left\_to\_right\_sum == x:

        ans.append(str(numbers[read\_right][1]))

        ans.append(str(numbers[read\_left][1]))

        break

    elif left\_to\_right\_sum > target:

        read\_right -= 1

    else:

        read\_left += 1

print(' '.join(ans) if ans else 'IMPOSSIBLE')

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
USACO 2017 January Contest, Bronze

## Problem 1. Don't Be Last!

Farmer John owns 7 dairy cows: Bessie, Elsie, Daisy, Gertie, Annabelle, Maggie, and Henrietta. He milks them every day and keeps detailed records on the amount of milk provided by each cow during each milking session. Not surprisingly, Farmer John highly prizes cows that provide large amounts of milk.

Cows, being lazy creatures, don't necessarily want to be responsible for producing too much milk. If it were up to them, they would each be perfectly content to be the lowest-producing cow in the entire herd. However, they keep hearing Farmer John mentioning the phrase "farm to table" with his human friends, and while they don't quite understand what this means, they have a suspicion that it actually may not be the best idea to be the cow producing the least amount of milk. Instead, they figure it's safer to be in the position of producing the second-smallest amount of milk in the herd. Please help the cows figure out which of them currently occupies this desirable position.

**INPUT FORMAT (file notlast.in):**

The input file for this task starts with a line containing the integer NN (1≤N≤1001≤N≤100), giving the number of entries in Farmer John's milking log.

Each of the NN following lines contains the name of a cow (one of the seven above) followed by a positive integer (at most 100), indicating the amount of milk produced by the cow during one of its milking sessions.

Any cow that does not appear in the log at all is assumed to have produced no milk.

**OUTPUT FORMAT (file notlast.out):**

On a single line of output, please print the name of the cow that produces the second-smallest amount of milk. More precisely, if MM is the minimum total amount of milk produced by any cow, please output the name of the cow whose total production is minimal among all cows that produce more than MM units of milk. If several cows tie for this designation, or if no cow has this designation (i.e., if all cows have production equal to MM), please output the word "Tie". Don't forget to add a newline character at the end of your line of output. Note that M=0M=0 if one of the seven cows is completely absent from the milking log, since this cow would have produced no milk.

**SAMPLE INPUT:**

10

Bessie 1

Maggie 13

Elsie 3

Elsie 4

Henrietta 4

Gertie 12

Daisy 7

Annabelle 10

Bessie 6

Henrietta 5

**SAMPLE OUTPUT:**

Henrietta

In this example, Bessie, Elsie, and Daisy all tie for the minimum by each producing 7 units of milk. The next-largest production, 9 units, is due to Henrietta.

fin = open("notlast.in.txt", "r")

fout = open("notlast.out1.txt", "w")

N = int(fin.readline())

data = [fin.readline().split() for cow in range(N)]

cows = {'Bessie':0, 'Maggie':0, 'Elsie':0, 'Henrietta':0, 'Gertie':0, 'Annabelle':0, 'Daisy':0}

for c in data:

    cows[c[0]] += int(c[1])

if len(set(cows.values())) == 1:

    print("Tie")

else:

    sl1 = sorted(set(cows.values()))[3]

    sl2 = sorted(set(cows.values()))[3]

    count = 0

    for key,value in cows.items():

        if value == sl1:

            answer = key

            test\_answer = key,value

            count += 1

            if count>1:

                answer = "Tie"

            if sl1 == sl2:

                print("tie")

fout.write(str(answer))

fout.write(str(test\_answer))

fout.close()

C. White Sheet

time limit per test

1 second

memory limit per test

256 megabytes

input

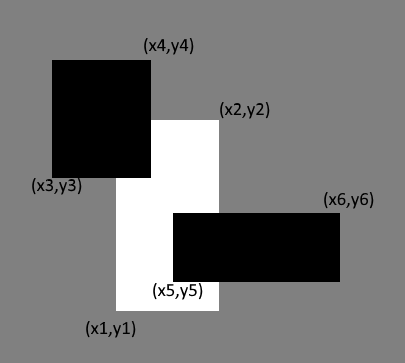
standard input

output

standard output

There is a white sheet of paper lying on a rectangle table. The sheet is a rectangle with its sides parallel to the sides of the table. If you will take a look from above and assume that the bottom left corner of the table has coordinates (0,0)(0,0), and coordinate axes are left and bottom sides of the table, then the bottom left corner of the white sheet has coordinates (x1,y1)(�1,�1), and the top right — (x2,y2)(�2,�2).

After that two black sheets of paper are placed on the table. Sides of both black sheets are also parallel to the sides of the table. Coordinates of the bottom left corner of the first black sheet are (x3,y3)(�3,�3), and the top right — (x4,y4)(�4,�4). Coordinates of the bottom left corner of the second black sheet are (x5,y5)(�5,�5), and the top right — (x6,y6)(�6,�6).

Example of three rectangles.

Determine if some part of the white sheet can be seen from the above after the two black sheets are placed. The part of the white sheet can be seen if there is at least one point lying **not strictly inside** the white sheet and **strictly outside** of both black sheets.

**Input**

The first line of the input contains four integers x1,y1,x2,y2�1,�1,�2,�2 (0≤x1<x2≤106,0≤y1<y2≤106)(0≤�1<�2≤106,0≤�1<�2≤106) — coordinates of the bottom left and the top right corners of the white sheet.

The second line of the input contains four integers x3,y3,x4,y4�3,�3,�4,�4 (0≤x3<x4≤106,0≤y3<y4≤106)(0≤�3<�4≤106,0≤�3<�4≤106) — coordinates of the bottom left and the top right corners of the first black sheet.

The third line of the input contains four integers x5,y5,x6,y6�5,�5,�6,�6 (0≤x5<x6≤106,0≤y5<y6≤106)(0≤�5<�6≤106,0≤�5<�6≤106) — coordinates of the bottom left and the top right corners of the second black sheet.

**The sides of each sheet of paper are parallel (perpendicular) to the coordinate axes.**

**Output**

If some part of the white sheet can be seen from the above after the two black sheets are placed, print "YES" (without quotes). Otherwise print "NO".

**Examples**

**input**

**Copy**

2 2 4 4

1 1 3 5

3 1 5 5

**output**

**Copy**

NO

**input**

**Copy**

3 3 7 5

0 0 4 6

0 0 7 4

**output**

**Copy**

YES

**input**

**Copy**

5 2 10 5

3 1 7 6

8 1 11 7

**output**

**Copy**

YES

**input**

**Copy**

0 0 1000000 1000000

0 0 499999 1000000

500000 0 1000000 1000000

**output**

**Copy**

YES

**Note**

In the first example the white sheet is fully covered by black sheets.

In the second example the part of the white sheet can be seen after two black sheets are placed. For example, the point (6.5,4.5)(6.5,4.5) lies not strictly inside the white sheet and lies strictly outside of both black sheets.

import sys

sys.stdin = open("in1.txt","r")

sys.stdout = open("in1out.txt","w")

white = list(map(int,input().split()))

black1 = list(map(int,input().split()))

black2 = list(map(int,input().split()))

'''

def intersect(rectangle1,rectangle2):

    x\_intersect = max(0,min(rectangle1[2],rectangle2[2]) - max(rectangle1[0],rectangle2[0]))

    y\_intersect = max(0,min(rectangle1[3],rectangle2[3]) - max(rectangle1[1],rectangle2[1]))

    return x\_intersect \* y\_intersect

area = (white[2]-white[0])\*(white[3]-white[1])

intersection1 = intersect(white,black1)

intersection2 = intersect(white,black2)

black\_overlap = max(black1[0],black2[0]),max(black1[1],black2[1]),min(black1[2]black2[2]),min(black1[3],black2[3])

intersection3 = intersect(white,black\_overlap)

white\_area -= intersection1 + intersection2 - intersection3

if white\_area <= 0:

    print('no")

else:

    print("yes")

'''

def area(black\_x:int,black\_y:int,white\_x:int,:white\_y;int)->int:

    length = white\_y - black\_y

    width = white\_x - black\_x

    return length\*width

def intersect(sheet1,sheet2) -> bool:

    black\_sheetx1,black\_sheety1,white\_sheetx1,white\_sheety1 = sheet1[0],sheet1[1],sheet1[2],sheet1[3]

    black\_sheetx2,black\_sheety1,white\_sheetx1,white\_sheety1 = sheet1[0],sheet1[1],sheet1[2],sheet1[3]

    if (black\_sheetx1 >= white\_sheetx2 or white\_sheetx1 <= black\_sheetx2)

        or black\_sheety1 >= white\_sheet2 or white\_sheety1 <= black\_sheety2):

      return False

    else:

        return True

def intersection\_rear(sheet2,sheet2) ->int:

    black\_sheetx1,black\_sheety1,white\_sheetx1,white\_sheety1 = sheet1[0],sheet1[1],sheet1[2],sheet1[3]

    black\_sheetx2,black\_sheety1,white\_sheetx1,white\_sheety1 = sheet1[0],sheet1[1],sheet1[2],sheet1[3]

    return (

        (min(white\_sheetx1,white\_sheetx2) - max(black\_sheetx1,black\_sheetx2)))

        \* (min(white\_sheetx1,white\_sheetx2) - max(black\_sheetx1,black\_sheetx2)))

area = area(white[0],white[1],white[2],white[3])

if intersect(black1,white) and intersect(back2.white):

    black1\_area = intersection\_area(white,black1)

    black2\_area = intersection\_area(white,black2)

    if black1\_area + black2\_area > area:

        print("no")

    else:

        print("yes")

## USACO 2022 US Open Contest, Bronze

## Problem 1. Photoshoot

Return to Problem List  
Contest has ended.



**Log in to allow submissions in analysis mode**

Top of Form



Bottom of Form

Farmer John, desperate to win the award for best cow photographer at the county fair, is trying to take the perfect photograph of his N� cows (2≤N≤2⋅1052≤�≤2⋅105, N� even).

Farmer John owns cows of two potential breeds: Guernseys and Holsteins. To make his photo as aesthetic as possible, he wants to line up his cows so that as many Guernseys are in even-numbered positions in the line as possible (the first position in the line is an odd position, the next is an even position, and so on). Due to his lack of strong communication with his cows, the only way he can achieve his goal is by asking even length "prefixes" of his cows to reverse themselves (a prefix consists of the range of cows from the first cow up to the j�th cow for some position j�).

Please count the minimum number of reversals required for Farmer John to achieve his goal.

#### INPUT FORMAT (input arrives from the terminal / stdin):

The first line of input contains the value of N�.

The second line contains a string of length N,�, specifying the initial ordering of the cows from left to right. Each 'H' represents a Holstein, while each 'G' represents a Guernsey.

#### OUTPUT FORMAT (print output to the terminal / stdout):

Output the minimum number of reversals needed on a single line.

#### SAMPLE INPUT:

14

GGGHGHHGHHHGHG

#### SAMPLE OUTPUT:

1

In this example, it suffices to reverse the prefix consisting of the first six cows.

GGGHGHHGHHHGHG (Before)

-> HGHGGGHGHHHGHG (After)

Before the reversal, four Guernseys were at even positions. After the reversal, six Guernseys are at even positions. It is impossible for there to be more than six Guernseys at even positions.

#### SCORING:

* Test cases 2-6 satisfy N≤1000�≤1000.
* Test cases 7-11 satisfy no additional constraints.

Problem credits: Aryansh Shrivastava

Contest has ended. No further submissions allowed.

'''

n = int(input())

cows = input().strip()

change = []

answer = 0

for i in range(0, len(cows), 2):

    if cows[i] + cows[i + 1] == 'HG':

        change.append('check')

    if cows[i] + cows[i + 1] == 'GH':

        change.append('switch')

for i in range(len(change) -1):

    if change[i] != change[i + 1]:

        answer += 1

if change[-1] == 'switch':

    answer += 1

print(answer)

'''

n = int(input())

cows = input()

split\_cows = [cows[i:i+2]for i in range(0,n,2) if cows[i:i+2] in ('GH','HG')]

answer = 1

for i in range(len(split\_cows) -1):

    answer += split\_cows[i] != split\_cows[i+1]

if split\_cows[-1] == 'HG':

    print(answer - 1)

else:

    print(answer)

import sys

sys.stdin = open('data1.txt')

sys.stdout = open('data2.out.txt', 'w')

def overlap(white, black2):

    x1,y1,x2,y2 = white

    x3,y3,x4,y4 = black2

    if x1>=x4 or x2<=x3 or y1>=y4 or y2<=y3:

        return (0,0,0,0)

    else:

        return max(x1,x3),max(y1,y3),min(x2,x4),min(y2,y4)

def area(t):

    return (t[3]-t[1])\*(t[2]-t[0])

white = tuple(map(int,input().split()))

black1 = tuple(map(int,input().split()))

black2 = tuple(map(int,input().split()))

total = area(white) - area(overlap(white,black1)) - area(overlap(white,black2))

+ area(overlap(white,overlap(black1,black2)))

if total==0:

    print("NO")

else:

    print("YES")

import sys

sys.stdin = open("jurymark2.txt", "r")

sys.stdout = open("jurymarkout3.txt", "w")

def solve():

k,n = map(int,input().split())

A = list(map(int,input().split()))

B = list(map(int,input().split()))

first = set()

for i in range(1,k):

A[i] = A[i]+A[i-1]

for i in range(n):

possible = set()

for j in range(k):

if i == 0:

possible.add(B[i]-A[j])

continue

if B[i]-A[j] in first:

possible.add(B[i]-A[j])

first = possible

print(len(possible))

if \_\_name\_\_ == "\_\_main\_\_":

solve()

import sys

sys.stdin = open('square.in.txt')

sys.stdout = open('square.out2.txt', 'w')

r1 = list(map(int, input().strip().split()))

r2 = list(map(int, input().strip().split()))

new\_rectangle=(max(abs(min(r1[0], r2[0]) - max(r1[2], r2[2])),

                   abs(min(r1[1], r2[1]) - max(r1[3], r2[3]))) \*\* 2)

print(new\_rectangle)

import sys

sys.stdin = open('square.in.txt')

sys.stdout = open('square.out1.txt', 'w')

x1, y1, x2, y2 = map(int, input().split())

x3, y3, x4, y4 = map(int, input().split())

new\_rectangle = max(max(x1,x2,x3,x4) - min(x1,x2,x3,x4),

                    max(y1,y2,y3,y4) - min(y1,y2,y3,y4))

print(new\_rectangle \*\* 2)

import sys

sys.stdin = open("billboard.in", "r")

sys.stdout = open("billboard.out", "w")

max\_range = 2000

def find():

        visible = [[Fasle for in range(max\_range)] for \_ in range(max\_range)]

        for i in range(3):

                x1, y1, y2 = map(int, input().split())

                x1 += max\_range  // 2

                x2 += max\_range  // 2

                y1 += max\_range  // 2

                y2 += max\_range  // 2

                for x in range(x1, x2):

                        for y in range(y1, y2):

                                visible[x][y] = i < 2

        answer = 0

        for x in range(max\_range):

                for y in range(max\_range):

                        answer += visible[x][y]

        print(answer)

main()