* Print Hello, World! to stdout.

Ans:- print("Hello, World!")

* Given an integer, , perform the following conditional actions:
* If  is odd, print Weird
* If  is even and in the inclusive range of  to , print Not Weird
* If  is even and in the inclusive range of  to , print Weird
* If  is even and greater than , print Not Weird

Ans:- n=int(input())

print("Weird") if n%2!=0 else print("Not Weird") if n>=2 and n<=5 else print("Weird") if n>=6 and n<=20 else print("Not Weird")

* The provided code stub reads two integers from STDIN,  a and b. Add code to print three lines where:

1. The first line contains the sum of the two numbers.
2. The second line contains the difference of the two numbers (first - second).
3. The third line contains the product of the two numbers.

Ans:- a = int(input())

b = int(input())

print(a+b)

print(a-b)

print(a\*b)

* The first line contains the first integer,a .  
  The second line contains the second integer,b

The result of the integer division a//b.

The result of the float division is a/b.

Ans:- a = int(input())

b = int(input())

print(a//b)

print(a/b)

* The provided code stub reads and integer, n, from STDIN. For all non-negative integers i<n , print i2 .

Ans:- n = int(input())

print(\*(i\*\*2 for i in range(n)), sep="\n")

* The included code stub will read an integer,n , from STDIN.Without using any string methods, try to print the following 123…n Print the list of integers from 1 through n as a string, without spaces.

Ans:- n = int(input())

print(\*(range(1, n + 1)), sep="")

* You are given a string and your task is to swap cases. In other words, convert all lowercase letters to uppercase letters and vice versa.

Ans:- swap\_case=lambda s: s.swapcase()

* You are given a string. Split the string on a " " (space) delimiter and join using a - hyphen.

Ans:- split\_and\_join=lambda line: "-".join(line.split(" "))

* Given a year, determine whether it is a leap year. If it is a leap year, return the Boolean True, otherwise return False.Note that the code stub provided reads from STDIN and passes arguments to the is\_leap function.

Ans:- is\_leap=lambda year: (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

* A close-up of a number

  Description automatically generated

*int arr:* an array of integers

**Returns**:*float:* the resulting float value rounded to 3 places after the decimal

Ans:- average=lambda array: f"{sum(set(array)) / len(set(array)):.3f}"

* The first line contains n. The second line contains an array A[]  of n integers each separated by a space. Print the runner-up score. Given list is [2,3,6,6,5]. The maximum score is 6, second maximum is 5. Hence, we print 5 as the runner-up score.

Ans:-     n = int(input())

    arr = map(int, input().split())

     print(sorted(set(arr),reverse=True)[1])

* You are given three integers  x,y and z representing the dimensions of a cuboid along with an integer n. Print a list of all possible coordinates given by (i,j,k) on a 3D grid where the sum of i+j+k is not equal to n. Four integers  x,y,z and n, each on a separate line.

Input:- 1

1

1

2

Output:- [[0, 0, 0], [0, 0, 1], [0, 1, 0], [1, 0, 0], [1, 1, 1]]

Ans:-  x = int(input())

     y = int(input())

     z = int(input())

     n = int(input())

     print([[a,b,c] for a in range(x+1) for b in range(y+1) for c in range(z+1) if a+b+c != n])

* You are given the firstname and lastname of a person on two different lines. Your task is to read them and print the following:

Hello firstname lastname! You just delved into python.

Ans:- print\_full\_name=lambda first,last: print("Hello ",first," ",last,"! You just delved into python.",sep="")

* Complete the *mutate\_string* function.

*mutate\_string* has the following parameters:

*string string:* the string to change

*int position:* the index to insert the character at

*string character:* the character to insert

**Returns** *string:* the altered string

Ans:- mutate\_string=lambda string, position, character: "".join([character if x==position else y for x,y in enumerate(string)])

* The provided code stub will read in a dictionary containing key/value pairs of name:[marks] for a list of students. Print the average of the marks array for the student name provided, showing 2 places after the decimal.

Input: The first line contains the integer n, the number of students' records. The next n lines contain the names and marks obtained by a student, each value separated by a space. The final line contains **query\_name**, the name of a student to query.

**Output Format**:Print one line: The average of the marks obtained by the particular student correct to 2 decimal places.

Ans:-  n = int(input())

     student\_marks = {}

     for i in range(n):

        name, \*line = input().split()

        scores = list(map(float, line))

        student\_marks[name] = scores

     query\_name = input()

     print(f"{sum(student\_marks[query\_name])/len(student\_marks[query\_name]):.2f}")

* User enters a string and a substring. You have to print the number of times that the substring occurs in the given string. String traversal will take place from left to right, not from right to left. The first line of input contains the original string. The next line contains the substring. Output the integer number indicating the total number of occurrences of the substring in the original string.

Input:-

ABCDCDC

CDC

Output:-

2

Ans:- count\_substring=lambda string, sub\_string: len(set(string.find(sub\_string,i) for i in range(len(string)) if string.find(sub\_string,i)!=-1))

* A single line of input containing the full name, S. Print the capitalized string, S.

**Sample Input:** chris alan

**Sample Output:** Chris Alan

Ans:- solve=lambda s: "".join([y.upper() if (x == " " and y != " ") else y for x, y in zip(" " + s, s)])

* You are given a string s and width w.  
  Your task is to wrap the string s into a paragraph of width w.

**Sample Input:** ABCDEFGHIJKLIMNOQRSTUVWXYZ

4

**Sample Output:**

ABCD

EFGH

IJKL

IMNO

QRST

UVWX

YZ

Ans:- wrap=lambda string, max\_width: "\n".join([string[i:i+max\_width] for i in range(0,len(string),max\_width)])

* **Input Format**: The first line contains a, the second line contains b, and the third line contains m.

**Output:** Print two lines.  
On the first line, print the result of pow(a,b). On the second line, print the result of pow(a,b,m),i.e., a^b mod m.

Ans:- a=int(input())

b=int(input())

m=int(input())

print(a\*\*b)

print((a\*\*b)%m)

* **Input Format**: The first line contains the first integer,a , and the second line contains the second integer, b.

The first line is the integer division a//b.  
The second line is the result of the modulo operator: a%b.  
The third line prints the divmod of a and b.

**Sample Input**

177

10

**Sample Output**

17

7

(17, 7)

Ans:- a=int(input())

b=int(input())

print(a//b)

print(a%b)

print((a//b,a%b))

* You are given a positive integer N. Print a numerical triangle of height N-1 like the one below:

1

22

333

4444

55555

......

Ans:- for i in range(1,int(input())):

    print(((10\*\*i-1)//9)\*i)

* Read four numbers,a ,b ,c and d, and print the result of ab+cd.

Ans:- a=int(input())

b=int(input())

c=int(input())

d=int(input())

print(a\*\*b+c\*\*d)

* [**Polar coordinates**](https://en.wikipedia.org/wiki/Polar_coordinate_system) are an alternative way of representing Cartesian coordinates or [Complex Numbers](https://en.wikipedia.org/wiki/Complex_number). A complex number  z=x+yj is completely determined by its real part x and imaginary part y.  
  Here, j is the [imaginary unit](https://en.wikipedia.org/wiki/Imaginary_unit). A polar coordinate (r,w) is completely determined by modulus r and phase angle w.

A blue arrow pointing up

Description automatically generated

If we convert complex number z to its polar coordinate, we find:  
r: Distance from z to origin, i.e., math.sqrt(x^2+y^2)   
w: Counter clockwise angle measured from the positive x-axis to the line segment that joins z to the origin.

**Input Format**: A single line containing the complex number z. Note: complex() function can be used in python to convert the input as a complex number.

Output two lines:  
The first line should contain the value of r.  
The second line should contain the value of w.

Ans:- import math

z=complex(input())

print((z.real\*\*2+z.imag\*\*2)\*\*0.5)

print(math.atan2(z.imag,z.real))

* Let's learn some new Python concepts! You have to generate a list of the first N fibonacci numbers, 0 being the first number. Then, apply the map function and a lambda expression to cube each fibonacci number and print the list.

One line of input: an integer N.

Output: A list on a single line containing the cubes of the first N fibonacci numbers.

**Sample Input**

5

**Sample Output**

[0, 1, 1, 8, 27]

Ans: cube = lambda x: x\*\*3

fibonacci=lambda n, seq=[0, 1]:[] if n==0 else [seq[0]] if n==1 else seq if len(seq) >= n else fibonacci(n, seq + [seq[-1] + seq[-2]])

     n = int(input())

     print(list(map(cube, fibonacci(n))))

* Given a list of rational numbers,find their product.

**Input Format**: First line contains n, the number of rational numbers.  
The ith of next n lines contain two integers each, the numerator(Ni) and denominator(Di) of the ith rational number in the list.

**Output Format:** Print only one line containing the numerator and denominator of the product of the numbers in the list in its simplest form, i.e. numerator and denominator have no common divisor other than 1.

Ans:- from fractions import Fraction

from functools import reduce

product=lambda fracs:(reduce(lambda a,b:a\*b,fracs).numerator, reduce(lambda a,b:a\*b,fracs).denominator)

     fracs = []

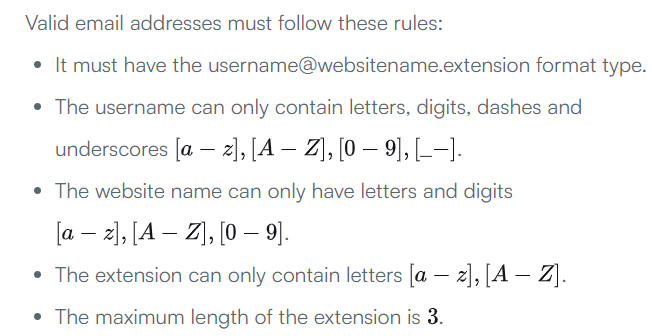
     for i in range(int(input())):

        fracs.append(Fraction(\*map(int, input().split())))

     result = product(fracs)

     print(\*result)

* You are given an integer N followed by N email addresses. Your task is to print a list containing only valid email addresses in lexicographical order.



**Concept**

A filter takes a function returning True or False and applies it to a sequence, returning a list of only those members of the sequence where the function returned True. A Lambda function can be used with filters.

**Input Format**

The first line of input is the integer N, the number of email addresses.  
 N lines follow, each containing a string.

**Sample Input**

3

lara@hackerrank.com

brian-23@hackerrank.com

britts\_54@hackerrank.com

**Sample Output**

['brian-23@hackerrank.com', 'britts\_54@hackerrank.com'

Ans:- def fun(email):

     import re

     pattern = r'^[a-zA-Z0-9\_-]+@[a-zA-Z0-9]+\.[a-zA-Z]{1,3}$'

     return re.match(pattern, email) is not None

def filter\_mail(emails):

     return list(filter(fun, emails))

n = int(input())

    emails = []

    for i in range(n):

        emails.append(input())

filtered\_emails = filter\_mail(emails)

filtered\_emails.sort()

print(filtered\_emails)

* A valid email address meets the following criteria:

It's composed of a username, domain name, and extension assembled in this format: username@domain.extension

The username starts with an English alphabetical character, and any subsequent characters consist of one or more of the following: [alphanumeric characters](https://en.wikipedia.org/wiki/Alphanumeric), -,., and \_.

The domain and extension contain only [English alphabetical characters](https://en.wikipedia.org/wiki/English_alphabet).

The extension is 1,2, or 3 characters in length.

Given  pairs of names and email addresses as input, print each name and email address pair having a valid email address on a new line.

**Input Format**:The first line contains a single integer, n, denoting the number of email address.  
Each line i of the n subsequent lines contains a name and an email address as two space-separated values following this format:

name [user@email.com](mailto:user@email.com)

**Output Format:**Print the space-separated name and email address pairs containing valid email addresses only. Each pair must be printed on a new line in the following format:

name [user@email.com](mailto:user@email.com)

You must print each valid email address in the same order as it was received as input.

**Sample Input**

2

DEXTER <dexter@hotmail.com>

VIRUS <virus!@variable.:p>

**Sample Output**

DEXTER [dexter@hotmail.com](mailto:dexter@hotmail.com)

Ans:- n=int(input())

name=[]

email=[]

for i in range(n):

     temp\_name, temp\_email = input().split()

     name.append(temp\_name)

     email.append(temp\_email)

def fun(email1):

     import re

     pattern = r'^<[a-zA-Z][a-zA-Z0-9\_.-]+@[a-zA-Z]+\.[a-zA-Z]{1,3}>$'

     return re.match(pattern, email1) is not None

for i in range(n):

     if fun(email[i]):

         print(name[i],email[i])

* You are given a string N.Your task is to verify that N is a floating point number.

In this task, a valid float number must satisfy all of the following requirements:

 Number can start with +, - or . symbol.  
For example:  
✔+4.50  
✔-1.0  
✔.5  
✔-.7  
✔+.4  
✖ -+4.5

 Number must contain at least 1 decimal value.  
For example:  
✖ 12.  
✔12.0

 Number must have exactly one . symbol.  
 Number must not give any exceptions when converted using float(N).

**Input Format**:The first line contains an integer T, the number of test cases.  
The next T line(s) contains a string S.

**Output Format**: Output True or False for each test case.

**Sample Input 0**

4

4.0O0

-1.00

+4.54

SomeRandomStuff

**Sample Output 0**

False

True

True

False

Ans:- T=int(input())

s=[]

for i in range(T):

     s.append(input())

def is\_valid\_float(s):

     import re

     pattern = r"^[+-]?(\d+\.\d+|\.\d+)$"

     return re.match(pattern, s) is not None

for i in s:

     print(is\_valid\_float(i))

### [Exceptions](https://docs.python.org/2/tutorial/errors.html#exceptions):Errors detected during execution are called exceptions.

**Examples**:

[**ZeroDivisionError**](https://docs.python.org/2/library/exceptions.html#exceptions.ZeroDivisionError)  
This error is raised when the second argument of a division or modulo operation is zero.

[**ValueError**](https://docs.python.org/2/library/exceptions.html#exceptions.ValueError):This error is raised when a built-in operation or function receives an argument that has the right type but an inappropriate value.

### [Handling Exceptions](https://docs.python.org/2/tutorial/errors.html#handling-exceptions): The statements try and except can be used to handle selected exceptions. A try statement may have more than one except clause to specify handlers for different exceptions.

You are given two values a and b.  
Perform integer division and print a/b.

**Input Format**: The first line contains T, the number of test cases.  
The next T lines each contain the space separated values of a and b.

**Output Format**

Print the value of a/b.  
In the case of ZeroDivisionError or ValueError, print the error code.

**Sample Input**

3

1 0

2 $

3 1

**Sample Output**

Error Code: integer division or modulo by zero

Error Code: invalid literal for int() with base 10: '$'

3

Ans:- T=int(input())

a, b=[], []

for i in range(T):

     temp\_a, temp\_b=input().split()

     a.append(temp\_a)

     b.append(temp\_b)

for i in range(T):

     try:

         print(int(a[i])//int(b[i]))

     except ZeroDivisionError:

print("Error Code: integer division or modulo by zero")

     except ValueError as e:

         print("Error Code:", e)

* [**shape**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.shape.html#numpy-ndarray-shape)

The *shape* tool gives a tuple of array dimensions and can be used to change the dimensions of an array.

**(a). Using *shape* to get array dimensions**

import numpy

my\_\_1D\_array = numpy.array([1, 2, 3, 4, 5])

print my\_1D\_array.shape #(5,) -> 1 row and 5 columns

my\_\_2D\_array = numpy.array([[1, 2],[3, 4],[6,5]])

print my\_2D\_array.shape #(3, 2) -> 3 rows and 2 columns

[**reshape**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html#numpy.reshape)

The *reshape* tool gives a new shape to an array without changing its data. It creates a new array and does not modify the original array itself.

import numpy

my\_array = numpy.array([1,2,3,4,5,6])

print numpy.reshape(my\_array,(3,2))

#Output

[[1 2]

[3 4]

**Task:**

You are given a space separated list of nine integers. Your task is to convert this list into a X *NumPy* array.

**Input Format**

A single line of input containing  space separated integers.

**Output Format**

Print the X *NumPy* array.

**Sample Input**

1 2 3 4 5 6 7 8 9

**Sample Output**

[[1 2 3]

[4 5 6]

[7 8 9]]

Ans:- import numpy as np

l=np.array(list(map(int,input().split())))

print(np.reshape(l,(3,3)))

* **Input Format**

A single line of input containing space separated numbers.

**Output Format**

Print the reverse NumPy array with type float.

**Sample Input**

1 2 3 4 -8 -10

**Sample Output**

[-10. -8. 4. 3. 2. 1.]

Ans:- import numpy

arrays=lambda arr: numpy.array(arr[::-1],float)

arr = input().strip().split()

result = arrays(arr)

print(result)

* [**Transpose**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.transpose.html#numpy-transpose)

We can generate the transposition of an array using the tool numpy.transpose.  
It will not affect the original array, but it will create a new array.

**Task**

You are given a NXM integer array matrix with space separated elements (N = rows and M = columns).  
Your task is to print the *transpose* and *flatten* results.

**Input Format**:The first line contains the space separated values of N and M.  
The next N lines contains the space separated elements of M columns.

**Output Format**:First, print the *transpose* array and then print the *flatten*.

**Sample Input**

2 2

1 2

3 4

**Sample Output**

[[1 3]

[2 4]]

[1 2 3 4]

Ans:- import numpy as np

N, M=list(map(int, input().split()))

l=[]

for i in range(N):

     l.append(input().split())

print(np.transpose(np.array(l)))

print(np.array(l).flatten())

* [**sum**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.sum.html):The *sum* tool returns the sum of array elements over a given axis.

import numpy

my\_array = numpy.array([ [1, 2], [3, 4] ])

print numpy.sum(my\_array, axis = 0) #Output : [4 6]

print numpy.sum(my\_array, axis = 1) #Output : [3 7]

print numpy.sum(my\_array, axis = None) #Output : 10

print numpy.sum(my\_array) #Output : 10

By default, the axis value is None. Therefore, it performs a sum over all the dimensions of the input array.

[**prod**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.prod.html):The *prod* tool returns the product of array elements over a given axis.

import numpy

my\_array = numpy.array([ [1, 2], [3, 4] ])

print numpy.prod(my\_array, axis = 0) #Output : [3 8]

print numpy.prod(my\_array, axis = 1) #Output : [ 2 12]

print numpy.prod(my\_array, axis = None) #Output : 24

print numpy.prod(my\_array) #Output : 24

By default, the axis value is None. Therefore, it performs the product over all the dimensions of the input array.

**Task**:You are given a 2-D array with dimensions N XM.  
Your task is to perform the sum tool over axis 0 and then find the product of that result.

**Input Format**:The first line of input contains space separated values of N and M.  
The next N lines contains M space separated integers.

**Output Format**:Compute the sum along axis 0. Then, print the product of that sum.

**Sample Input**

2 2

1 2

3 4

**Sample Output**

24

**Explanation**

The sum along axis  0= [4 6 ]  
The product of this sum = 24

Ans:- import numpy as np

N, M=list(map(int, input().split()))

l=[]

for i in range(N):

     l.append(input().split())

print(np.prod(np.sum(np.array(l),axis=0)))

* [**Concatenate**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html): Two or more arrays can be concatenated together using the *concatenate* function with a tuple of the arrays to be joined:

import numpy

array\_1 = numpy.array([1,2,3])

array\_2 = numpy.array([4,5,6])

array\_3 = numpy.array([7,8,9])

print numpy.concatenate((array\_1, array\_2, array\_3))

#Output

[1 2 3 4 5 6 7 8 9]

If an array has more than one dimension, it is possible to specify the axis along which multiple arrays are concatenated. By default, it is along the first dimension.

import numpy

array\_1 = numpy.array([[1,2,3],[0,0,0]])

array\_2 = numpy.array([[0,0,0],[7,8,9]])

print numpy.concatenate((array\_1, array\_2), axis = 1)

#Output

[[1 2 3 0 0 0]

[0 0 0 7 8 9]]

**Task**

You are given two integer arrays of size NXP and MXP (N & M are rows, and P is the column). Your task is to *concatenate* the arrays along axis 0.

**Input Format**:The first line contains space separated integers N,M  and P.  
The next N lines contains the space separated elements of the P columns.  
After that, the next M lines contains the space separated elements of the P columns.

**Output Format**: Print the concatenated array of size (N+M)XP.

**Sample Input**

4 3 2

1 2

1 2

1 2

1 2

3 4

3 4

3 4

**Sample Output**

[[1 2]

[1 2]

[1 2]

[1 2]

[3 4]

[3 4]

[3 4]]

Ans:- import numpy as np

N, M, P=list(map(int, input().split()))

l1,l2=[],[]

for i in range(N):

     l1.append(input().split())

for i in range(M):

l2.append(input().split())

print(np.concatenate((np.array(l1,int),np.array(l2,int)),axis=0))

* Basic mathematical functions operate element-wise on arrays. They are available both as operator overloads and as functions in the *NumPy* module.

import numpy

a = numpy.array([1,2,3,4], float)

b = numpy.array([5,6,7,8], float)

print a + b #[ 6. 8. 10. 12.]

print numpy.add(a, b) #[ 6. 8. 10. 12.]

print a - b #[-4. -4. -4. -4.]

print numpy.subtract(a, b) #[-4. -4. -4. -4.]

print a \* b #[ 5. 12. 21. 32.]

print numpy.multiply(a, b) #[ 5. 12. 21. 32.]

print a / b #[ 0.2 0.33333333 0.42857143 0.5 ]

print numpy.divide(a, b) #[ 0.2 0.33333333 0.42857143 0.5 ]

print a % b #[ 1. 2. 3. 4.]

print numpy.mod(a, b) #[ 1. 2. 3. 4.]

print a\*\*b #[ 1.00000000e+00 6.40000000e+01 2.18700000e+03 6.55360000e+04]

print numpy.power(a, b) #[ 1.00000000e+00 6.40000000e+01 2.18700000e+03 6.55360000e+04]

**Task**:You are given two integer arrays A  and B of dimensions NXM.  
Your task is to perform the following operations:

1. Add ( A+B )
2. Subtract (A -B )
3. Multiply ( A\*B )
4. Integer Division ( A//B )
5. Mod ( A%B )
6. Power ( A\*\*B )

**Note**:There is a method numpy.floor\_divide() that works like numpy.divide() except it performs a floor division.

**Input Format**:The first line contains two space separated integers,  N and M.  
The next N lines contains M space separated integers of array A.  
The following N lines contains M space separated integers of array B.

**Output Format**:Print the result of each operation in the given order under **Task**.

**Sample Input**

1 4

1 2 3 4

5 6 7 8

**Sample Output**

[[ 6 8 10 12]]

[[-4 -4 -4 -4]]

[[ 5 12 21 32]]

[[0 0 0 0]]

[[1 2 3 4]]

[[ 1 64 2187 65536]]

Ans:- import numpy as np

N,M=list(map(int,input().split()))

A,B=[],[]

for i in range(N):

     A.append(input().split())

for i in range(N):

     B.append(input().split())

A, B=np.array(A,int), np.array(B,int)

print(A+B,A-B,A\*B,A//B,A%B,A\*\*B,sep="\n")

* [**min**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.min.html): The tool *min* returns the minimum value along a given axis.

import numpy

my\_array = numpy.array([[2, 5],

[3, 7],

[1, 3],

[4, 0]])

print numpy.min(my\_array, axis = 0) #Output : [1 0]

print numpy.min(my\_array, axis = 1) #Output : [2 3 1 0]

print numpy.min(my\_array) #Output : 0

By default, the axis value is None. Therefore, it finds the minimum over all the dimensions of the input array.

[**max**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.max.html): The tool *max* returns the maximum value along a given axis.

import numpy

my\_array = numpy.array([[2, 5],

[3, 7],

[1, 3],

[4, 0]])

print numpy.max(my\_array, axis = 0) #Output : [4 7]

print numpy.max(my\_array) #Output : 7

By default, the axis value is None. Therefore, it finds the maximum over all the dimensions of the input array.

**Task:** You are given a 2-D array with dimensions NXM.  
Your task is to perform the *min* function over axis  1 and then find the *max* of that.

**Input Format:** The first line of input contains the space separated values of N and M.  
The next N lines contains M space separated integers.

**Output Format:** Compute the *min* along axis 1 and then print the *max* of that result.

**Sample Input**

4 2

2 5

3 7

1 3

4 0

**Sample Output**

3

**Explanation**

The *min* along axis  1=[2,3,1,0]   
The *max* of [2,3,1,0]  = 3

Ans:- import numpy as np

N,M=list(map(int,input().split()))

A=[]

for i in range(N):

    A.append(input().split())

print(np.max(np.min(np.array(A, int),axis=1)))

* [**floor**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.floor.html#numpy-floor): The tool *floor* returns the floor of the input element-wise.

import numpy

my\_array = numpy.array([1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9])

print numpy.floor(my\_array) #[ 1. 2. 3. 4. 5. 6. 7. 8. 9.]

[**ceil**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.ceil.html#numpy-ceil): The tool *ceil* returns the ceiling of the input element-wise.  
The ceiling of  is the smallest integer  where .

import numpy

my\_array = numpy.array([1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9])

print numpy.ceil(my\_array) #[ 2. 3. 4. 5. 6. 7. 8. 9. 10.]

[**rint**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.rint.html): The *rint* tool rounds to the nearest integer of input element-wise.

import numpy

my\_array = numpy.array([1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9])

print numpy.rint(my\_array) #[ 1. 2. 3. 4. 6. 7. 8. 9. 10.]

**Task:** You are given a 1-D array, A. Your task is to print the floor, ceil  and rint of all the elements of A.

**Note:** In order to get the correct output format, add the line numpy.set\_printoptions(legacy='1.13')below the numpy import.

**Input Format**: A single line of input containing the space separated elements of array A.

**Output Format:** On the first line, print the floor of A.  
On the second line, print the ceil of A.  
On the third line, print the rint of A.

**Sample Input**

1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9

**Sample Output**

[ 1. 2. 3. 4. 5. 6. 7. 8. 9.]

[ 2. 3. 4. 5. 6. 7. 8. 9. 10.]

[ 1. 2. 3. 4. 6. 7. 8. 9. 10.]

Ans:- import numpy as np

np.set\_printoptions(legacy='1.13')

A=np.array(input().split(),float)

print(np.floor(A),np.ceil(A),np.rint(A),sep="\n")

* [**mean**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html): The *mean* tool computes the arithmetic mean along the specified axis.

import numpy

my\_array = numpy.array([ [1, 2], [3, 4] ])

print numpy.mean(my\_array, axis = 0) #Output : [ 2. 3.]

print numpy.mean(my\_array) #Output : 2.5

By default, the axis is None. Therefore, it computes the mean of the flattened array.

[**var**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.var.html#numpy-var): The *var* tool computes the arithmetic variance along the specified axis.

import numpy

my\_array = numpy.array([ [1, 2], [3, 4] ])

print numpy.var(my\_array, axis = 0) #Output : [ 1. 1.]

print numpy.var(my\_array) #Output : 1.25

By default, the axis is None. Therefore, it computes the variance of the flattened array.

[**std**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.std.html#numpy.std): The *std* tool computes the arithmetic standard deviation along the specified axis.

import numpy

my\_array = numpy.array([ [1, 2], [3, 4] ])

print numpy.std(my\_array, axis = 0) #Output : [ 1. 1.]

print numpy.std(my\_array) #Output : 1.11803398875

By default, the axis is None. Therefore, it computes the standard deviation of the flattened array.

**Task:** You are given a 2-D array of size NXM.  
Your task is to find:

1. The mean along axis 1
2. The var along axis 0
3. The std along axis None

**Input Format:**The first line contains the space separated values of N and M.  
The next N lines contains M space separated integers.

**Output Format:** First, print the *mean*.  
Second, print the *var*.  
Third, print the *std*.

**Sample Input**

2 2

1 2

3 4

**Sample Output**

[ 1.5 3.5]

[ 1. 1.]

1.11803398875

Ans:- import numpy as np

N, M=list(map(int,input().split()))

A=[]

for i in range(N):

     A.append(input().split())

A=np.array(A, float) print(np.mean(A, axis=1),np.var(A, axis=0),np.around(np.std(A),decimals=11),sep="\n")

* Given an integer,n , print the following values for each integer i from 1 to n:

1. Decimal
2. Octal
3. Hexadecimal (capitalized)
4. Binary

**Function Description**: Complete the *print\_formatted* function in the editor below.

*print\_formatted* has the following parameters:

* *int number:* the maximum value to print

**Prints**: The four values must be printed on a single line *in the order specified above* for each i from 1 to number. Each value should be space-padded to match the width of the *binary* value of number and the values should be separated by a single space.

**Input Format:** A single integer denoting n.

**Sample Input**

17

**Sample Output**

1 1 1 1

2 2 2 10

3 3 3 11

4 4 4 100

5 5 5 101

6 6 6 110

7 7 7 111

8 10 8 1000

9 11 9 1001

10 12 A 1010

11 13 B 1011

12 14 C 1100

13 15 D 1101

14 16 E 1110

15 17 F 1111

16 20 10 10000

17 21 11 10001

Ans:- def print\_formatted(number):

    width = len(bin(number)[2:])

     for i in range(1, number + 1):

        print(f"{i:{width}d} {i:{width}o} {i:{width}X} {i:{width}b}")

n=int(input())

     print\_formatted(n)

* In Python, a string of text can be aligned *left, right* and *center*.

**.ljust(width)**

This method returns a left aligned string of length *width*.

>>> width = 20

>>> print 'HackerRank'.ljust(width,'-')

HackerRank----------

**.center(width)**

This method returns a centered string of length *width*.

>>> width = 20

>>> print 'HackerRank'.center(width,'-')

-----HackerRank-----

**.rjust(width)**

This method returns a right aligned string of length *width*.

>>> width = 20

>>> print 'HackerRank'.rjust(width,'-')

----------HackerRank

**Task**

You are given a partial code that is used for generating the *HackerRank Logo* of variable *thickness*.

**Input Format**

A single line containing the *thickness* value for the logo.

**Constraints**

The *thickness* must be an *odd* number.

**Output Format**

Output the desired logo.

**Sample Input**

5

**Sample Output**

H

HHH

HHHHH

HHHHHHH

HHHHHHHHH

HHHHH HHHHH

HHHHH HHHHH

HHHHH HHHHH

HHHHH HHHHH

HHHHH HHHHH

HHHHH HHHHH

HHHHHHHHHHHHHHHHHHHHHHHHH

HHHHHHHHHHHHHHHHHHHHHHHHH

HHHHHHHHHHHHHHHHHHHHHHHHH

HHHHH HHHHH

HHHHH HHHHH

HHHHH HHHHH

HHHHH HHHHH

HHHHH HHHHH

HHHHH HHHHH

HHHHHHHHH

HHHHHHH

HHHHH

HHH

H

Ans:-

thickness =int(input()) #This must be an odd number

c = 'H'

#Top Cone

for i in range(thickness):

    print((c\*i).rjust(thickness-1)+c+(c\*i).ljust(thickness-1))

#Top Pillars

for i in range(thickness+1):

    print((c\*thickness).center(thickness\*2)+(c\*thickness).center(thickness\*6))

#Middle Belt

for i in range((thickness+1)//2):

    print((c\*thickness\*5).center(thickness\*6))

#Bottom Pillars

for i in range(thickness+1):

    print((c\*thickness).center(thickness\*2)+(c\*thickness).center(thickness\*6))

#Bottom Cone

for i in range(thickness):

    print(((c\*(thickness-i-1)).rjust(thickness)+c+(c\*(thickness-i-1)).ljust(thickness)).rjust(thickness\*6))

* If we want to add a single element to an existing set, we can use the .add() operation.

**Example:** >>> s = set('HackerRank')

>>> s.add('H')

>>> print s

**Task**: Apply your knowledge of the .add() operation to help your friend Rupal.  
Rupal has a huge collection of country stamps. She decided to count the total number of distinct country stamps in her collection. She asked for your help. You pick the stamps one by one from a stack of N country stamps.  
Find the total number of distinct country stamps.

**Input Format**: The first line contains an integer N, the total number of country stamps.  
The next N lines contains the name of the country where the stamp is from.

**Output Format**: Output the total number of distinct country stamps on a single line.

**Sample Input**

7

UK

China

USA

France

New Zealand

UK

France

**Sample Output**

5

Ans:- N=int(input())

print(len({input() for i in range(N)}))

* **.union()**

The *.union()* operator returns the union of a set and the set of elements in an iterable.  
Sometimes, the *|* operator is used in place of *.union()* operator, but it operates only on the set of elements in *set*.  
Set is immutable to the *.union()* operation (or *|* operation).

**Example**: >>> s = set("Hacker")

>>> print s.union("Rank")

set(['a', 'R', 'c', 'r', 'e', 'H', 'k', 'n'])

>>> print s.union(set(['R', 'a', 'n', 'k']))

set(['a', 'R', 'c', 'r', 'e', 'H', 'k', 'n'])

>>> print s.union(['R', 'a', 'n', 'k'])

set(['a', 'R', 'c', 'r', 'e', 'H', 'k', 'n'])

>>> print s.union(enumerate(['R', 'a', 'n', 'k']))

set(['a', 'c', 'r', 'e', (1, 'a'), (2, 'n'), 'H', 'k', (3, 'k'), (0, 'R')])

>>> print s.union({"Rank":1})

set(['a', 'c', 'r', 'e', 'H', 'k', 'Rank'])

>>> s | set("Rank")

set(['a', 'R', 'c', 'r', 'e', 'H', 'k', 'n'])

**Task**: The students of District College have subscriptions to *English* and *French* newspapers. Some students have subscribed only to *English*, some have subscribed to only *French* and some have subscribed to both newspapers.

You are given two sets of student roll numbers. One set has subscribed to the *English* newspaper, and the other set is subscribed to the *French* newspaper. The same student could be in both sets. Your task is to find the total number of students who have subscribed to *at least one* newspaper.

**Input Format**: The first line contains an integer, n, the number of students who have subscribed to the *English* newspaper.  
The second line contains n space separated roll numbers of those students.  
The third line contains b, the number of students who have subscribed to the *French* newspaper.  
The fourth line contains b space separated roll numbers of those students.

**Output Format**: Output the total number of students who have *at least one* subscription.

**Sample Input**

9

1 2 3 4 5 6 7 8 9

9

10 1 2 3 11 21 55 6 8

**Sample Output**: 13

Ans:- n=int(input())

s1=set(input().split())

b=int(input())

s2=set(input().split())

print(len(s1| s2))

* **.intersection()**

The .intersection() operator returns the intersection of a set and the set of elements in an iterable.  
Sometimes, the & operator is used in place of the .intersection() operator, but it only operates on the set of elements in set.  
The set is immutable to the .intersection() operation (or & operation).

>>> s = set("Hacker")

>>> print s.intersection("Rank")

set(['a', 'k'])

>>> print s.intersection(set(['R', 'a', 'n', 'k']))

set(['a', 'k'])

>>> print s.intersection(['R', 'a', 'n', 'k'])

set(['a', 'k'])

>>> print s.intersection(enumerate(['R', 'a', 'n', 'k']))

set([])

>>> print s.intersection({"Rank":1})

set([])

>>> s & set("Rank")

set(['a', 'k'])

**Continuation to the above question.**

**Output Format**: Output the total number of students who have subscriptions to **both** English and French newspapers.

Ans:- n=int(input())

s1=set(input().split())

b=int(input())

s2=set(input().split())

print(len(s1& s2))

* **.difference()**: The tool .difference() returns a set with all the elements from the set that are not in an iterable.  
  Sometimes the - operator is used in place of the .difference() tool, but it only operates on the set of elements in set.  
  Set is immutable to the .difference() operation (or the - operation).

>>> s = set("Hacker")

>>> print s.difference("Rank")

set(['c', 'r', 'e', 'H'])

>>> print s.difference(set(['R', 'a', 'n', 'k']))

set(['c', 'r', 'e', 'H'])

>>> print s.difference(['R', 'a', 'n', 'k'])

set(['c', 'r', 'e', 'H'])

>>> print s.difference(enumerate(['R', 'a', 'n', 'k']))

set(['a', 'c', 'r', 'e', 'H', 'k'])

>>> print s.difference({"Rank":1})

set(['a', 'c', 'e', 'H', 'k', 'r'])

>>> s - set("Rank")

set(['H', 'c', 'r', 'e'])

**Continuation to the above question.**

**Output Format**: Output the total number of students who are subscribed to the English newspaper only.

Ans:- n=int(input())

s1=set(input().split())

b=int(input())

s2=set(input().split())

print(len(s1-s2))

* **.symmetric\_difference()**: The .symmetric\_difference() operator returns a set with all the elements that are in the set and the iterable but not both.  
  Sometimes, a ^ operator is used in place of the .symmetric\_difference() tool, but it only operates on the set of elements in set.  
  The set is immutable to the .symmetric\_difference() operation (or ^ operation).

>>> s = set("Hacker")

>>> print s.symmetric\_difference("Rank")

set(['c', 'e', 'H', 'n', 'R', 'r'])

>>> print s.symmetric\_difference(set(['R', 'a', 'n', 'k']))

set(['c', 'e', 'H', 'n', 'R', 'r'])

>>> print s.symmetric\_difference(['R', 'a', 'n', 'k'])

set(['c', 'e', 'H', 'n', 'R', 'r'])

>>> print s.symmetric\_difference(enumerate(['R', 'a', 'n', 'k']))

set(['a', 'c', 'e', 'H', (0, 'R'), 'r', (2, 'n'), 'k', (1, 'a'), (3, 'k')])

>>> print s.symmetric\_difference({"Rank":1})

set(['a', 'c', 'e', 'H', 'k', 'Rank', 'r'])

>>> s ^ set("Rank")

set(['c', 'e', 'H', 'n', 'R', 'r'])

**Continuation to the above question.**

**Output Format**: Output total number of students who have subscriptions to the English or the French newspaper but not both.

Ans:- n=int(input())

s1=set(input().split())

b=int(input())

s2=set(input().split())

print(len(s1^ s2))

* You are given two sets, A and B.  
  Your job is to find whether set  A is a subset of set B.  
  If set A is subset of set B, print **True**.  
  If set A is not a subset of set B, print **False**.

**Input Format:** The first line will contain the number of test cases,T .  
The first line of each test case contains the number of elements in set A.  
The second line of each test case contains the space separated elements of set A.  
The third line of each test case contains the number of elements in set B.  
The fourth line of each test case contains the space separated elements of set B.

**Output Format**: Output **True** or **False** for each test case on separate lines.

**Sample Input**

3

5

1 2 3 5 6

9

9 8 5 6 3 2 1 4 7

1

2

5

3 6 5 4 1

7

1 2 3 5 6 8 9

3

9 8 2

**Sample Output**: True

False

False

Ans:- T=int(input())

for i in range(T):

     nA=int(input())

     A=set(input().split())

     nB=int(input())

     B=set(input().split())

     print(A.issubset(B))

* You are given a set A and n other sets.  
  Your job is to find whether set A is a strict superset of each of the N sets.

Print True, if A is a *strict superset* of each of the N sets. Otherwise, print False.

A strict superset has at least one element that does not exist in its subset.

**Input Format**: The first line contains the space separated elements of set A.  
The second line contains integer n, the number of other sets.  
The next n lines contains the space separated elements of the other sets.

**Output Format**: Print True if set A is a *strict superset* of all other N sets. Otherwise, print False.

**Sample Input:** 1 2 3 4 5 6 7 8 9 10 11 12 23 45 84 78

2

1 2 3 4 5

100 11 12

**Sample Output:** False

Ans:- A=set(input().split())

n=int(input())

B = set().union(\*[set(input().split()) for \_ in range(n)])

print(A.issuperset(B))

### [any()](https://docs.python.org/2/library/functions.html#any): This expression returns True if **any** element of the iterable is true. If the iterable is empty, it will return False.

### [all()](https://docs.python.org/2/library/functions.html#all): This expression returns True if **all** of the elements of the iterable are true. If the iterable is empty, it will return True.

**Task**: You are given a space separated list of integers. If all the integers are positive, then you need to check if any integer is a [palindromic integer](https://en.wikipedia.org/wiki/Palindromic_number).

**Input Format**: The first line contains an integer N.N  is the total number of integers in the list.  
The second line contains the space separated list of  N integers.

**Output Format:** Print True if all the conditions of the problem statement are satisfied. Otherwise, print False.

**Sample Input:** 5

12 9 61 5 14

**Sample Output**: True

**Explanation**: **Condition 1**: All the integers in the list are positive.  
**Condition 2**: 5 is a palindromic integer.

Ans:- N=int(input())

l=list(map(int, input().split()))

print(all(i > 0 for i in l) and any(str(i) == str(i)[::-1] for i in l))

* You are given a string S.  
  S contains alphanumeric characters only.  
  A close up of a letter

  Description automatically generated Your task is to sort the string S in the following manner:
* All sorted *lowercase letters* are ahead of *uppercase letters*.
* All sorted *uppercase letters* are ahead of digits.
* All sorted *odd digits* are ahead of sorted *even digits*.

**Input Format:** A single line of input contains the string S.

**Output Format**: Output the sorted string S.

**Sample Input**: Sorting1234

**Sample Output**: ginortS1324

Ans:- s=input()

lower\_case = sorted([i for i in s if i.islower()])

upper\_case = sorted([i for i in s if i.isupper()])

odd\_digits = sorted([i for i in s if i.isdigit() and int(i)%2!= 0])

even\_digits = sorted([i for i in s if i.isdigit() and int(i)%2==0])

print(''.join(lower\_case+upper\_case+odd\_digits+even\_digits))

* You are given a string s consisting only of digits 0-9, commas ,, and dots .

Your task is to complete the regex\_pattern defined below, which will be used to re.split() all of the , and . symbols in s.

It’s guaranteed that every comma and every dot in s is preceeded and followed by a digit.

**Sample Input:** 100,000,000.000

**Sample Output:** 100

000

000

000

Ans:- regex\_pattern = "[,.]"

import re

print("\n".join(re.split(regex\_pattern, input())))

* Let's dive into the interesting topic of regular expressions! You are given some input, and you are required to check whether they are valid mobile numbers.

A valid mobile number is a ten digit number starting with a 7,8 or 9.

<https://developers.google.com/edu/python/regular-expressions>

**Input Format**

The first line contains an integer N, the number of inputs.  
 N lines follow, each containing some string.

**Output Format**: For every string listed, print "YES" if it is a valid mobile number and "NO" if it is not on separate lines. Do not print the quotes.

**Sample Input:** 2

9587456281

1252478965

**Sample Output:** YES

NO

Ans:- import re

l=[input() for i in range(int(input()))]

for i in l: print("YES") if re.match(r"^[789]\d{9}$",i) else print("NO")

* You are given a string, and you have to validate whether it's a valid Roman numeral. If it is valid, print True. Otherwise, print False. Try to create a regular expression for a valid Roman numeral.

**Input Format**: A single line of input containing a string of Roman characters.

**Output Format:** Output a single line containing True or False according to the instructions above.

**Sample Input:** CDXXI

**Sample Output:** True

Ans:- regex\_pattern = r'^M{0,3}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$'

import re

print(str(bool(re.match(regex\_pattern, input()))))

* Consider a list (list = []). You can perform the following commands:

1. insert i e: Insert integer e at position i.
2. print: Print the list.
3. remove e: Delete the first occurrence of integer e.
4. append e: Insert integer e at the end of the list.
5. sort: Sort the list.
6. pop: Pop the last element from the list.
7. reverse: Reverse the list.

**Input Format:** The first line contains an integer,n , denoting the number of commands.  
Each line i of the n subsequent lines contains one of the commands described above.

**Output Format**: For each command of type print, print the list on a new line.

**Sample Input:** 12

insert 0 5

insert 1 10

insert 0 6

print

remove 6

append 9

append 1

sort

print

pop

reverse

print

**Sample Output:** [6, 5, 10]

[1, 5, 9, 10]

[9, 5, 1]

Ans:-l=[]

n=int(input())

for \_ in range(n):

     command = input().split()

     if command[0] == "insert":

        # Insert integer e at position i

         i, e = map(int, command[1:])

        l.insert(i, e)

     elif command[0] == "print":

         # Print the list

         print(l)

     elif command[0] == "remove":

         # Delete the first occurrence of integer e

         e = int(command[1])

         l.remove(e)

     elif command[0] == "append":

         # Insert integer e at the end of the list

         e = int(command[1])

         l.append(e)

     elif command[0] == "sort":

         # Sort the list

         l.sort()

     elif command[0] == "pop":

         # Pop the last element from the list

         l.pop()

     elif command[0] == "reverse":

         # Reverse the list

         l.reverse()

* Given the names and grades for each student in a class of N students, store them in a nested list and print the name(s) of any student(s) having the second lowest grade.

**Note:** If there are multiple students with the second lowest grade, order their names alphabetically and print each name on a new line.

**Input Format:** The first line contains an integer, N, the number of students.  
The 2N  subsequent lines describe each student over 2 lines.  
- The first line contains a student's name.  
- The second line contains their grade.

**Output Format:** Print the name(s) of any student(s) having the second lowest grade in. If there are multiple students, order their names alphabetically and print each one on a new line.

**Sample Input:** 5

Harry

37.21

Berry

37.21

Tina

37.2

Akriti

41

Harsh

39

**Sample Output:** Berry

Harry

Ans:- n = int(input())

s=[[input(), float(input())] for \_ in range(n)]

# Sort the s list based on grades in ascending order

s=sorted(s, key=lambda x: x[1])

# Find the second lowest grade among the students

slg=[s[i][1] for i in range(n) if s[i][1] != s[0][1]][0]

# Print the names of students with the slg

print("\n".join(sorted([s[i][0] for i in range(n) if s[i][1] == slg])))

* If the inputs are given on one line separated by a character (the delimiter), use split() to get the separate values in the form of a list. The delimiter is space (ascii 32) by default. To specify that comma is the delimiter, use string.split(',').

>> a = raw\_input()

5 4 3 2

>> lis = a.split()

>> print (lis)

['5', '4', '3', '2']

If the list values are all integer types, use the map() method to convert all the strings to integers.

>> newlis = list(map(int, lis))

>> print (newlis)

[5, 4, 3, 2]

Sets are an unordered collection of unique values. A single set contains values of any immutable data type.  
**CREATING SETS:** >> myset = {1, 2} # Directly assigning values to a set

>> myset = set() # Initializing a set

>> myset = set(['a', 'b']) # Creating a set from a list

>> myset

{'a', 'b'}

**MODIFYING SETS**: Using the add() function:

>> myset.add('c')

>> myset

{'a', 'c', 'b'}

>> myset.add('a') # As 'a' already exists in the set, nothing happens

>> myset.add((5, 4))

>> myset

{'a', 'c', 'b', (5, 4)}  
Using the update() function:

>> myset.update([1, 2, 3, 4]) # update() only works for iterable objects

>> myset

{'a', 1, 'c', 'b', 4, 2, (5, 4), 3}

>> myset.update({1, 7, 8})

>> myset

{'a', 1, 'c', 'b', 4, 7, 8, 2, (5, 4), 3}

>> myset.update({1, 6}, [5, 13])

>> myset

{'a', 1, 'c', 'b', 4, 5, 6, 7, 8, 2, (5, 4), 13, 3}  
**REMOVING ITEMS**: Both the discard() and remove() functions take a single value as an argument and removes that value from the set. If that value is not present, discard() does nothing, but remove() will raise a KeyError exception.

>> myset.discard(10)

>> myset

{'a', 1, 'c', 'b', 4, 5, 7, 8, 2, 12, (5, 4), 13, 11, 3}

>> myset.remove(13)

>> myset

{'a', 1, 'c', 'b', 4, 5, 7, 8, 2, 12, (5, 4), 11, 3}

**COMMON SET OPERATIONS** Using union(), intersection() and difference() functions.

>> a = {2, 4, 5, 9}

>> b = {2, 4, 11, 12}

>> a.union(b) # Values which exist in a or b

{2, 4, 5, 9, 11, 12}

>> a.intersection(b) # Values which exist in a and b

{2, 4}

>> a.difference(b) # Values which exist in a but not in b

{9, 5}  
The union() and intersection() functions are symmetric methods:

>> a.union(b) == b.union(a)

True

>> a.intersection(b) == b.intersection(a)

True

>> a.difference(b) == b.difference(a)

False

These [other built-in data structures in Python](http://www.thelearningpoint.net/computer-science/learning-python-programming-and-data-structures/learning-python-programming-and-data-structures--tutorial-4--built-in-data-structures-strings-lists-tuples-dictionaries-mutability) are also useful.

**Input Format:** The first line of input contains an integer,M .  
The second line contains M space-separated integers.  
The third line contains an integer, N.  
The fourth line contains N space-separated integers.

**Output Format**: Output the symmetric difference integers in ascending order, one per line.

**Sample Input:**

STDIN Function

----- --------

4 set a size M = 4

2 4 5 9 a = {2, 4, 5, 9}

4 set b size N = 4

2 4 11 12 b = {2, 4, 11, 12}

**Sample Output:** 5

9

11

12

Ans:- M=int(input())

s1=set(map(int,input().split()))

N=int(input())

s2=set(map(int,input().split()))

print(\*sorted(s1^s2),sep="\n")

* [zip([iterable, ...])](https://docs.python.org/2/library/functions.html#zip)

This function returns a list of tuples. The ith tuple contains the ith element from each of the argument sequences or iterables.

If the argument sequences are of unequal lengths, then the returned list is truncated to the length of the shortest argument sequence.

**Sample Code:** >>> print zip([1,2,3,4,5,6],'Hacker')

[(1, 'H'), (2, 'a'), (3, 'c'), (4, 'k'), (5, 'e'), (6, 'r')]

>>> print zip([1,2,3,4,5,6],[0,9,8,7,6,5,4,3,2,1])

[(1, 0), (2, 9), (3, 8), (4, 7), (5, 6), (6, 5)]

>>> A = [1,2,3]

>>> B = [6,5,4]

>>> C = [7,8,9]

>>> X = [A] + [B] + [C]

>>> print zip(\*X)

[(1, 6, 7), (2, 5, 8), (3, 4, 9)]

**Task**: The National University conducts an examination of N students in X subjects.  
Your task is to compute the average scores of each student.

The format for the general mark sheet is:

Student ID → \_\_\_1\_\_\_\_\_2\_\_\_\_\_3\_\_\_\_\_4\_\_\_\_\_5\_\_

Subject 1 | 89 90 78 93 80

Subject 2 | 90 91 85 88 86

Subject 3 | 91 92 83 89 90.5

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Average 90 91 82 90 85.5

**Input Format**: The first line contains N and X separated by a space.  
The next X lines contains the space separated marks obtained by students in a particular subject.

**Output Format:** Print the averages of all students on separate lines.

The averages must be correct up to 1 decimal place.

**Sample Input:** 5 3

89 90 78 93 80

90 91 85 88 86

91 92 83 89 90.5

**Sample Output:** 90.0

91.0

82.0

90.0

85.5

Ans:- N, X = map(int, input().split())

marks=[list(map(float, input().split())) for \_ in range(X)]

for i in range(N):

     print(f"{sum([marks[j][i] for j in range(X)])/X:.1f}")

* **Task**: You are given a [polynomial](https://en.wikipedia.org/wiki/Polynomial) P of a single indeterminate (or variable),x .  
  You are also given the values of x and k. Your task is to verify if P(x)=k.

**Input Format:** The first line contains the space separated values of x and k.  
The second line contains the polynomial P.

**Output Format**: Print True if P(x)=k. Otherwise, print False.

**Sample Input:** 1 4

x\*\*3 + x\*\*2 + x + 1

**Sample Output:** True

Ans:- x,k =map(int, input().split())

P=input()

print(eval(P)==k)

* The eval() expression is a very powerful built-in function of Python. It helps in evaluating an expression. The expression can be a Python statement.Here, eval() can also be used to work with Python keywords or defined functions and variables. These would normally be stored as strings.

>>> type(eval("len"))

<type 'builtin\_function\_or\_method'>

Without eval()

>>> type("len")

<type 'str'>

**Task**: You are given an expression in a line. Read that line as a string variable, such as var, and print the result using eval(var).

**Sample Input:** print(2 + 3)

**Sample Output:** 5

Ans:- eval(input())

* Mr. Vincent works in a door mat manufacturing company. One day, he designed a new door mat with the following specifications:
* Mat size must be NXM . (N is an odd natural number, and M is 3 times N.)
* The design should have 'WELCOME' written in the center.
* The design pattern should only use |, . and - characters.

**Input Format:** A single line containing the space separated values of  N and M.

**Output Format**: Output the design pattern.

**Sample Input:** 9 27

**Sample Output:**

------------.|.------------

---------.|..|..|.---------

------.|..|..|..|..|.------

---.|..|..|..|..|..|..|.---

----------WELCOME----------

---.|..|..|..|..|..|..|.---

------.|..|..|..|..|.------

---------.|..|..|.---------

------------.|.------------

Ans:- N, M = map(int, input().split())

for i in range(1, N, 2): print((i \* ".|.").center(M, "-"))

print("WELCOME".center(M, "-"))

for i in range(N - 2, -1, -2): print((i \* ".|.").center(M, "-"))

* [**dot**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html): The dot tool returns the dot product of two arrays.

import numpy

A = numpy.array([ 1, 2 ])

B = numpy.array([ 3, 4 ])

print numpy.dot(A, B) #Output : 11

[**cross**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.cross.html): The cross tool returns the cross product of two arrays.

import numpy

A = numpy.array([ 1, 2 ])

B = numpy.array([ 3, 4 ])

print numpy.cross(A, B) #Output : -2

**Task:** You are given two arrays A and B. Both have dimensions of NXN.  
Your task is to compute their [matrix product](https://en.wikipedia.org/wiki/Matrix_multiplication#Matrix_product_.28two_matrices.29).

**Input Format:** The first line contains the integer N.  
The next N lines contains N space separated integers of array A.  
The following N lines contains N space separated integers of array B.

**Output Format:** Print the matrix multiplication of A and B.

**Sample Input:** 2

1 2

3 4

1 2

3 4

**Sample Output:** [[ 7 10]

[15 22]]

Ans:- import numpy as np

n = int(input())

A=np.array([list(map(int, input().split())) for \_ in range(n)])

B=np.array([list(map(int, input().split())) for \_ in range(n)])

print(np.dot(A, B))

* [**inner**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.inner.html): The inner tool returns the [inner product](https://en.wikipedia.org/wiki/Inner_product_space) of two arrays.

import numpy

A = numpy.array([0, 1])

B = numpy.array([3, 4])

print numpy.inner(A, B) #Output : 4

[**outer**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.outer.html): The outer tool returns the [outer product](https://en.wikipedia.org/wiki/Outer_product) of two arrays.

import numpy

A = numpy.array([0, 1])

B = numpy.array([3, 4])

print numpy.outer(A, B) #Output : [[0 0]

# [3 4]]

**Task:** You are given two arrays: A and B.  
Your task is to compute their inner and outer product.

**Input Format:** The first line contains the space separated elements of array A.  
The second line contains the space separated elements of array B.

**Output Format:** First, print the inner product.  
Second, print the outer product.

**Sample Input:** 0 1

2 3

**Sample Output:** 3

[[0 0]

[2 3]]

Ans:- import numpy as np

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

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

print(np.inner(A, B),np.outer(A, B),sep="\n")

* [zeros](http://docs.scipy.org/doc/numpy/reference/generated/numpy.zeros.html#numpy-zeros)**:** The zeros tool returns a new array with a given shape and type filled with 0's.

import numpy

print numpy.zeros((1,2)) #Default type is float

#Output : [[ 0. 0.]]

print numpy.zeros((1,2), dtype = numpy.int) #Type changes to int

#Output : [[0 0]]

[ones](http://docs.scipy.org/doc/numpy/reference/generated/numpy.ones.html#numpy-ones)**:** The ones tool returns a new array with a given shape and type filled with 1's.

import numpy

print numpy.ones((1,2)) #Default type is float

#Output : [[ 1. 1.]]

print numpy.ones((1,2), dtype = numpy.int) #Type changes to int

#Output : [[1 1]]

**Task:** You are given the shape of the array in the form of space-separated integers, each integer representing the size of different dimensions, your task is to print an array of the given shape and integer type using the tools numpy.zeros and numpy.ones.

**Input Format:** A single line containing the space-separated integers.

**Output Format:** First, print the array using the numpy.zeros tool and then print the array with the numpy.ones tool.

**Sample Input:** 3 3 3

**Sample Output:**

[[[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]]]

[[[1 1 1]

[1 1 1]

[1 1 1]]

[[1 1 1]

[1 1 1]

[1 1 1]]

[[1 1 1]

[1 1 1]

[1 1 1]]]

**Explanation:** Print the array built using numpy.zeros and numpy.ones tools and you get the result as shown.

Ans:- import numpy as np

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

print(np.zeros(n,dtype=int),np.ones(n,dtype=int),sep="\n")

* **.remove(x):** This operation removes element  from the set.  
  If element  does not exist, it raises a KeyError.  
  The *.remove(x)* operation returns None.

**Example**

>>> s = set([1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> s.remove(5)

>>> print s

set([1, 2, 3, 4, 6, 7, 8, 9])

**.discard(x):** This operation also removes element x from the set.  
If element x does not exist, it **does not** raise a KeyError.  
The *.discard(x)* operation returns None.

**Example**

>>> s = set([1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> s.discard(5)

>>> print s

set([1, 2, 3, 4, 6, 7, 8, 9])

**.pop():** This operation removes and return an arbitrary element from the set.  
If there are no elements to remove, it raises a KeyError.

**Example**

>>> s = set([1])

>>> print s.pop()

1

>>> print s

set([])

**Task:** You have a non-empty set s, and you have to execute N commands given in N lines.

The commands will be *pop, remove* and *discard*.

**Input Format:** The first line contains integer n, the number of elements in the set s.  
The second line contains n space separated elements of set s. All of the elements are non-negative integers, less than or equal to 9.  
The third line contains integer N, the number of commands.  
The next N lines contains either *pop, remove* and/or *discard* commands followed by their associated value.

**Output Format**: Print the sum of the elements of set s on a single line.

**Sample Input:** 9

1 2 3 4 5 6 7 8 9

10

pop

remove 9

discard 9

discard 8

remove 7

pop

discard 6

remove 5

pop

discard 5

**Sample Output:** 4

Ans: n = int(input())

s = set(map(int, input().split()))

N = int(input())

for \_ in range(N):

      command = input().split()

     if command[0]=='pop':s.pop()

     elif command[0]=='remove':s.remove(int(command[1]))

     elif command[0]=='discard':s.discard(int(command[1]))

print(sum(s))

* **Input Format:** The first line contains an integer,n , denoting the number of elements in the tuple.  
  The second line contains n space-separated integers describing the elements in tuple t.

**Output Format:** Print the result of hash(t).

**Sample Input:** 2

1 2

**Sample Output:** 3713081631934410656

Ans: n = int(input())

print(hash(tuple(map(int, input().split()))))

* **We can use the following operations to create mutations to a set:**

**.update()** or |= :Update the set by adding elements from an iterable/another set.

>>> H = set("Hacker")

>>> R = set("Rank")

>>> H.update(R)

>>> print H

set(['a', 'c', 'e', 'H', 'k', 'n', 'r', 'R'])

**.intersection\_update()** or &= :Update the set by keeping only the elements found in it and an iterable/another set.

>>> H = set("Hacker")

>>> R = set("Rank")

>>> H.intersection\_update(R)

>>> print H

set(['a', 'k'])

**.difference\_update()** or -= :Update the set by removing elements found in an iterable/another set.

>>> H = set("Hacker")

>>> R = set("Rank")

>>> H.difference\_update(R)

>>> print H

set(['c', 'e', 'H', 'r'])

**.symmetric\_difference\_update()** or ^=  
Update the set by only keeping the elements found in either set, but not in both.

>>> H = set("Hacker")

>>> R = set("Rank")

>>> H.symmetric\_difference\_update(R)

>>> print H

set(['c', 'e', 'H', 'n', 'r', 'R'])

**TASK:** You are given a set A and N number of other sets. These N number of sets have to perform some specific mutation operations on set A.

Your task is to execute those operations and print the sum of elements from set A.

**Input Format:** The first line contains the number of elements in set A.  
The second line contains the space separated list of elements in set A.  
The third line contains integer N, the number of other sets.  
The next 2\*N lines are divided into N parts containing two lines each.  
The first line of each part contains the space separated entries of the *operation name* and the *length of the other set*.  
The second line of each part contains space separated list of elements in the other set.

**Output Format:** Output the sum of elements in set A.

**Sample Input:** 16

1 2 3 4 5 6 7 8 9 10 11 12 13 14 24 52

4

intersection\_update 10

2 3 5 6 8 9 1 4 7 11

update 2

55 66

symmetric\_difference\_update 5

22 7 35 62 58

difference\_update 7

11 22 35 55 58 62 66

**Sample Output**: 38

Ans: n =int(input())

A =set(map(int,input().split()))

N =int(input())

for \_ in range(N):

     operation,\_=input().split()

     other\_set=set(map(int, input().split()))

     if operation=='intersection\_update':A&=other\_set

     elif operation=='update':A|=other\_set

     elif operation=='symmetric\_difference\_update':A^=other\_set

     elif operation=='difference\_update':A-=other\_set

print(sum(A))

* [**poly**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.poly.html): The poly tool returns the coefficients of a polynomial with the given sequence of roots.

print numpy.poly([-1, 1, 1, 10]) #Output : [ 1 -11 9 11 -10]

[**roots**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.roots.html): The roots tool returns the roots of a polynomial with the given coefficients.

print numpy.roots([1, 0, -1]) #Output : [-1. 1.]

[**polyint**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyint.html): The polyint tool returns an antiderivative (indefinite integral) of a polynomial.

print numpy.polyint([1, 1, 1]) #Output : [ 0.33333333 0.5 1. 0. ]

[**polyder**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyder.html#numpy.polyder): The polyder tool returns the derivative of the specified order of a polynomial.

print numpy.polyder([1, 1, 1, 1]) #Output : [3 2 1]

[**polyval**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyval.html#numpy.polyval): The polyval tool evaluates the polynomial at specific value.

print numpy.polyval([1, -2, 0, 2], 4) #Output : 34

[**polyfit**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html): The polyfit tool fits a polynomial of a specified order to a set of data using a least-squares approach.

print numpy.polyfit([0,1,-1, 2, -2], [0,1,1, 4, 4], 2)

#Output : [ 1.00000000e+00 0.00000000e+00 -3.97205465e-16]

The functions [polyadd](http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyadd.html" \l "numpy.polyadd" \t "_blank), [polysub](http://docs.scipy.org/doc/numpy/reference/generated/numpy.polysub.html" \l "numpy.polysub), [polymul](http://docs.scipy.org/doc/numpy/reference/generated/numpy.polymul.html), and [polydiv](http://docs.scipy.org/doc/numpy/reference/generated/numpy.polydiv.html" \l "numpy.polydiv) also handle proper addition, subtraction, multiplication, and division of polynomial coefficients, respectively.

**Task:** You are given the coefficients of a polynomial P.  
Your task is to find the value of P at point x.

**Input Format**: The first line contains the space separated value of the coefficients in P.  
The second line contains the value of x.

**Output Format**: Print the desired value.

**Sample Input:** 1.1 2 3

0

**Sample Output:** 3.0

Ans: import numpy as np

P=list(map(float,input().split()))

x=float(input())

print(np.polyval(P,x))

* [**identity**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.identity.html#numpy.identity): The *identity* tool returns an identity array. An identity array is a square matrix with all the main diagonal elements as 1 and the rest as 0. The default type of elements is float.

import numpy

print numpy.identity(3) #3 is for dimension 3 X 3

#Output: [[ 1. 0. 0.]

[ 0. 1. 0.]

[ 0. 0. 1.]]

[**eye**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.eye.html#numpy-eye): The *eye* tool returns a 2-D array with 1's as the diagonal and 0's elsewhere. The diagonal can be main, upper or lower depending on the optional parameter k. A positive k is for the upper diagonal, a negative k is for the lower, and a 0 k (default) is for the main diagonal.

import numpy

print numpy.eye(8, 7, k = 1) # 8 X 7 Dimensional array with first upper diagonal 1.

#Output: [[ 0. 1. 0. 0. 0. 0. 0.]

[ 0. 0. 1. 0. 0. 0. 0.]

[ 0. 0. 0. 1. 0. 0. 0.]

[ 0. 0. 0. 0. 1. 0. 0.]

[ 0. 0. 0. 0. 0. 1. 0.]

[ 0. 0. 0. 0. 0. 0. 1.]

[ 0. 0. 0. 0. 0. 0. 0.]

[ 0. 0. 0. 0. 0. 0. 0.]]

print numpy.eye(8, 7, k = -2) # 8 X 7 Dimensional array with second lower diagonal 1.

**Task:** Your task is to print an array of size NXM with its main diagonal elements as 1's and 0's everywhere else.

**Note**: In order to get alignment correct, please insert the line

np.set\_printoptions(legacy="1.13")

below the numpy import.

**Input Format**: A single line containing the space separated values of  N and M.  
N denotes the rows.  
M denotes the columns.

**Output Format:** Print the desired N X M array.

**Sample Input**:3 3

**Sample Output**: [[ 1. 0. 0.]

[ 0. 1. 0.]

[ 0. 0. 1.]]

Ans: import numpy as np

np.set\_printoptions(legacy="1.13")

a=np.array(input().split(),dtype=int)

print(np.eye(a[0],a[1]))

* A valid postal code P have to fullfil both below requirements:

1. P must be a number in the range from 100000 to 999999 inclusive.
2. P must not contain more than one alternating repetitive digit pair.

Alternating repetitive digits are digits which repeat immediately after the next digit. In other words, an alternating repetitive digit pair is formed by two equal digits that have just a single digit between them.

For example: 121426 # Here, 1 is an alternating repetitive digit.

**Input Format**: Locked stub code in the editor reads a single string denoting P from stdin and uses provided expression and your regular expressions to validate if P is a valid postal code.

**Output Format:** You are not responsible for printing anything to stdout. Locked stub code in the editor does that.

**Sample Input:** 110000

**Sample Output:** False

Ans: regex\_integer\_in\_range = r"^[1-9][0-9]{5}$"

regex\_alternating\_repetitive\_digit\_pair = r"(?=(\d)\d\1)"

import re

P=input()

print(bool(re.match(regex\_integer\_in\_range, P))

and len(re.findall(regex\_alternating\_repetitive\_digit\_pair, P)) < 2)

* Neo has a complex matrix script. The matrix script is a N X M grid of strings. It consists of alphanumeric characters, spaces and symbols (!,@,#,$,%,&).

A screenshot of a matrix script

Description automatically generated

To decode the script, Neo needs to read each column and select only the alphanumeric characters and connect them. Neo reads the column from top to bottom and starts reading from the leftmost column.

If there are symbols or spaces between two alphanumeric characters of the decoded script, then Neo replaces them with a single space ' ' for better readability.

Neo feels that there is no need to use 'if' conditions for decoding.

Alphanumeric characters consist of: [A-Z, a-z, and 0-9].

**Input Format:** The first line contains space-separated integers N (rows) and M (columns) respectively.  
The next N lines contain the row elements of the matrix script.

**Output Format:** Print the decoded matrix script.

**Sample Input:** 7 3

Tsi

h%x

i #

sM

$a

#t%

ir!

**Sample Output:** This is Matrix# %!

**Explanation:** The decoded script is:

This$#is% Matrix# %!

Neo replaces the symbols or spaces between two alphanumeric characters with a single space   ' ' for better readability.

So, the final decoded script is:

This is Matrix# %!

Ans: import re

n,m=list(map(int,input().split()))

l=[]

for \_ in range(n):l.append(input())

ttl=list(zip(\*l))

s=''

for i in ttl:s+=''.join(i)

print(re.sub(r'(?<=[a-zA-Z0-9])[!@#$%&\s]+(?=[a-zA-Z0-9])',' ',s))

* You and Fredrick are good friends. Yesterday, Fredrick received N credit cards from **ABCD Bank**. He wants to verify whether his credit card numbers are valid or not. You happen to be great at regex so he is asking for your help!

A valid credit card from **ABCD Bank** has the following characteristics:  
  
► It must start with a 4, 5 or 6.  
► It must contain exactly 16 digits.  
► It must only consist of digits (0-9).  
► It may have digits in groups of 4, separated by one hyphen **"-"**.  
► It must NOT use any other separator like '  ' , '\_', etc.  
► It must NOT have 4 or more consecutive repeated digits.

**Examples**:

**Valid Credit Card Numbers**

4253625879615786

4424424424442444

5122-2368-7954-3214

**Invalid Credit Card Numbers**

42536258796157867 #17 digits in card number → Invalid

4424444424442444 #Consecutive digits are repeating 4 or more times → Invalid

5122-2368-7954 - 3214 #Separators other than '-' are used → Invalid

44244x4424442444 #Contains non digit characters → Invalid

0525362587961578 #Doesn't start with 4, 5 or 6 → Invalid

**Input Format**: The first line of input contains an integer N.  
The next N lines contain credit card numbers.

**Output Format:** Print 'Valid' if the credit card number is valid. Otherwise, print 'Invalid'. Do not print the quotes.

**Sample Input:** 6

4123456789123456

5123-4567-8912-3456

61234-567-8912-3456

4123356789123456

5133-3367-8912-3456

5123 - 3567 - 8912 - 3456

**Sample Output:** Valid

Valid

Invalid

Valid

Invalid

Invalid

Comments:

1. [re.match(pattern, string) checks for a match only at the beginning of the string](https://stackoverflow.com/questions/58774029/differences-between-re-match-re-search-re-fullmatch). [It returns a match object if the beginning of the string matches the pattern, otherwise it returns None](https://stackoverflow.com/questions/58774029/differences-between-re-match-re-search-re-fullmatch).
2. [re.fullmatch(pattern, string) checks for a match over the entire string](https://stackoverflow.com/questions/58774029/differences-between-re-match-re-search-re-fullmatch). [It returns a match object if the whole string matches the pattern, otherwise it returns None](https://stackoverflow.com/questions/58774029/differences-between-re-match-re-search-re-fullmatch).

Ans: import re

p1=r'[456]\d{3}\-?\d{4}\-?\d{4}\-?\d{4}'

p2=r'(\d)\1{3,}'

for \_ in range(int(input())):

     card\_num=input()

     print("Valid") if (re.fullmatch(p1,card\_num) and not re.search(p2, card\_num.replace('-',''))) else print('Invalid')

* The NumPy module also comes with a number of built-in routines for linear algebra calculations. These can be found in the sub-module linalg.

[**linalg.det**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.det.html): The linalg.det tool computes the determinant of an array.

print numpy.linalg.det([[1 , 2], [2, 1]]) #Output : -3.0

[**linalg.eig**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.eig.html): The linalg.eig computes the eigenvalues and right eigenvectors of a square array.

vals, vecs = numpy.linalg.eig([[1 , 2], [2, 1]])

print vals #Output : [ 3. -1.]

print vecs #Output : [[ 0.70710678 -0.70710678]

# [ 0.70710678 0.70710678]]

[**linalg.inv**](http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.inv.html): The linalg.inv tool computes the (multiplicative) inverse of a matrix.

print numpy.linalg.inv([[1 , 2], [2, 1]]) #Output : [[-0.33333333 0.66666667]

# [ 0.66666667 -0.33333333]]

Other routines can be found [here](http://docs.scipy.org/doc/numpy/reference/routines.linalg.html)

**Task:** You are given a square matrix A with dimensions NXN. Your task is to find the determinant. Note: Round the answer to 2 places after the decimal.

**Input Format:** The first line contains the integer N.  
The next N lines contains the N space separated elements of array A.

**Output Format:** Print the determinant of A.

**Sample Input:** 2

1.1 1.1

1.1 1.1

**Sample Output:** 0.0

Ans: import numpy as np

A=[] for \_ in range(int(input())):A.append(list(map(float,input().split())))

print(round(np.linalg.det(A),2))