* 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 = []

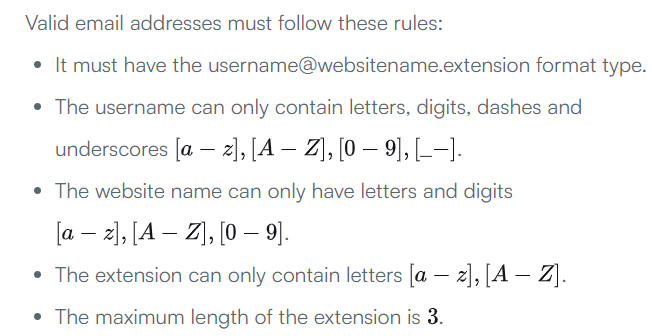
     for i in range(int(input())):

        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")