Viết chương trình nhập vào 2 số nguyên a, b. Tính tổng các số chẵn trong đoạn [a, b] hoặc [b, a]

Input:

a b trên cùng một dòng, cách nhau dấu cách.

Ví dụ: 5 12 hoặc 30 -8

Output:

Tổng các số chẵn tìm được. Ví dụ: 30

Constrains:

các biến kiểu nguyên

**For example:**

| **Input** | **Result** |
| --- | --- |
| 9 1 | 20 |

Answer:

a, b = [int(so) for so in input().split()]

if a > b:

i = a

a = b

b = i

tong = 0

for so in range(a, b+1):

if so %2 == 0:

tong += so

print(tong)

Viết chương trình nhập vào 2 số nguyên a, b. Tính trung bình cộng các số chẵn trong đoạn [a, b] hoặc [b, a].

Input:

a b trên cùng một dòng, cách nhau dấu cách.

Ví dụ: 5 12 hoặc 30 -8

Output:

Trung bình cộng các số chẵn tìm được với độ chính xác 2 chữ số thập phân. Ví dụ: 30.00

Constrains:

+ các biến a, b kiểu nguyên.

+ trung bình cộng có độ chính xác 2 chữ số thập phân.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 9 1 | 5.00 |

Answer:

a, b = [int(so) for so in input().split()]

if a > b:

i = a

a = b

b = i

tong = 0

count = 0

for so in range(a, b+1):

if so %2 == 0:

tong += so

count += 1

tbc = tong / count

print('{:.2f}'.format(tbc))

Viết chương trình nhập vào 4 số nguyên a, b, x, y. Tính trung bình cộng các số chẵn trong đoạn [a, b], hoặc [b, a], tính trung bình cộng các số lẻ trong đoạn [x, y], hoặc [y, x].

Input:

- a, b trên cùng một dòng, cách nhau dấu cách.

- x, y trên cùng một dòng, cách nhau dấu cách.

Ví dụ:

5 12

30 -8

Output:

Dòng 1: Trung bình cộng các số chẵn tìm được hoặc "NO" nếu không tính được.

Dòng 1: Trung bình cộng các số lẻ tìm được hoặc "NO" nếu không tính được.

Ví dụ:

30.00

NO

Constrains:

+ các biến a, b, x, y kiểu nguyên.

+ trung bình cộng có độ chính xác 2 chữ số thập phân.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 9 1  88 88 | 5.00  NO |

Answer:

a, b = [int(num\_ab) for num\_ab in input().split()]

x, y = [int(num\_xy) for num\_xy in input().split()]

if x > y:

z = x

x = y

y = z

if a > b:

c = a

a = b

b = c

tong\_ab = 0

count\_ab = 0

for num\_ab in range(a, b+1):

if num\_ab %2 == 0:

tong\_ab += num\_ab

count\_ab += 1

if count\_ab == 0:

print('NO')

else:

tbc\_ab = tong\_ab / count\_ab

print('{:.2f}'.format(tbc\_ab))

tong\_xy = 0

count\_xy = 0

for num\_xy in range(x, y+1):

if num\_xy %2 != 0:

tong\_xy += num\_xy

count\_xy += 1

if count\_xy == 0:

print('NO')

else:

tbc\_xy = tong\_xy / count\_xy

print('{:.2f}'.format(tbc\_xy))

Nhập một số nguyên chỉ điểm ở thang 10.

In ra xếp loại dựa trên điểm như sau:

* Nếu điểm từ 0~3 in ra **Kem**
* Nếu là 4 in ra **Yeu**
* Nếu từ 5~6 in ra **TB**
* Nếu từ 7~8 in ra **Kha**
* Nếu từ 9~10 in ra **Gioi**
* Nếu điểm không thuộc phạm vi nào bên trên thì in ra **Error**

**Input Format**

* Một số nguyên chỉ điểm (**d**) theo thang 10.

**Constraints**

* 0 <= **d** <= 10

**Output Format**

* In ra một trong các phân loại: **Kem**, **Yeu**, **TB**, **Kha**, **Gioi**
* Hoặc in ra thông báo lỗi: **Error**
* In đúng chữ hoa thường, không in thêm bất kì kí tự nào khác.

Answer:

d = int(input())

if 0 <= d <= 10 :

if 0 <= d <= 3:

print('Kem')

if d == 4:

print('Yeu')

if 5 <= d <= 6:

print('TB')

if 7 <= d <= 8:

print('Kha')

if 9 <= d <= 10:

print('Gioi')

else:

print('Error')

Viết chương trình thực hiện:

* Nhập hai số nguyên cách nhau bởi dấu cách.
* In ra bảng cửu chương trong phạm vi hai số đó.

**Input Format**

* Hai số nguyên dương **a**, **b** cách nhau đúng một dấu cách.

**Constraints**

* 1 <= **a**,**b** <= 9

**Output Format**

* Mỗi phép nhân in trên 1 dòng, trên dòng không có dấu cách.
* Phép nhân đầu tiên và cuối cùng lần lượt là nhân với 1 và nhân với 9.
* Sau phép nhân (dòng) cuối cùng là 1 dòng trống.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 2 3 | 2x1=2  2x2=4  2x3=6  2x4=8  2x5=10  2x6=12  2x7=14  2x8=16  2x9=18  3x1=3  3x2=6  3x3=9  3x4=12  3x5=15  3x6=18  3x7=21  3x8=24  3x9=27 |

Answer:

a, b = [int(x) for x in input().split()]

for i in range(a,b+1):

for j in range(1, 10):

print("{}x{}={}".format(i,j,i\*j))

Nhập 1 số nguyên dương **n** chỉ độ dài cạnh hình tam tác.

In ra hình tam giác gồm các kí tự \* với độ dài 3 cạnh là **n**, cụ thể như sau:

* Dòng 1: in ra (n-1) kí tự ⎵ và (1) cặp kí tự \*⎵
* Dòng 2: in ra (n-2) kí tự ⎵ và (2) cặp kí tự \*⎵
* ...
* Dòng n: in ra (0) kí tự ⎵ và (n) cặp kí tự \*⎵
* Sau dòng cuối dùng (dòng n) là 1 dòng trống

Với ⎵ là kí tự trắng.

**Input Format**

Một số nguyên dương. Vd: 6

**Constraints**

Không có.

**Output Format**

Như mô trả trên.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 6 | \*  \* \*  \* \* \*  \* \* \* \*  \* \* \* \* \*  \* \* \* \* \* \* |

Answer:

n=int(input())

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

for j in range(1, n-i+1):

print(" ", end='')

for k in range(1, i+1):

print("\* ", end='')

print("\n", end='')

Cho một dãy số nguyên. Hãy chia dãy thành hai phần: phần bên trái gồm các số âm sắp xếp theo thứ tự giảm dần, phần bên phải gồm các số không âm sắp xếp theo thứ tự tăng dần.

* Đầu vào: Dãy các số nguyên cách nhau bởi dấu cách
* Đầu ra: Dãy sau khi đã sắp xếp

**For example:**

| **Input** | **Result** |
| --- | --- |
| 5 3 2 -1 -4 0 | -1 -4 0 2 3 5 |

Answer:

n = input().split()

a = []

b = []

for i in range(0,len(n)):

if(int(n[i])<0):

a.append(n[i])

else:

b.append(n[i])

def sapxep(t):

for i in range(0,len(t)):

for j in range(i,len(t)):

if(t[i]>t[j]):

tg = t[i]

t[i] = t[j]

t[j] = tg

return t

sapxep(a)

sapxep(b)

c = a+b

for i in range(0,len(c)):

print(c[i],end=' ')

Bảng cửu chương khuyết là bảng cửu chương mà với mỗi số a chỉ có các phép nhân với các số không nhỏ hơn a (Xem ví dụ). Viết chương trình thực hiện:

* Nhập hai số nguyên cách nhau bởi dấu cách.
* In ra bảng cửu chương trong phạm vi hai số đó.

**Input Format**

* Hai số nguyên dương **a**, **b** cách nhau đúng một dấu cách.

**Constraints**

* 1 <= **a**, **b** <= 9

**Output Format (in ra bảng cửu chương dạng khuyết)**

* Mỗi phép nhân in trên 1 dòng, trên dòng không có dấu cách.
* Sau phép nhân (dòng) cuối cùng là 1 dòng trống.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 2 3 | 2x2=4  2x3=6  2x4=8  2x5=10  2x6=12  2x7=14  2x8=16  2x9=18  3x3=9  3x4=12  3x5=15  3x6=18  3x7=21  3x8=24  3x9=27 |

Answer:

a,b=[int(n) for n in input().split()]

for i in range(a,b+1,1):

for j in range(i,10,1):

print(str(i)+"x"+str(j)+"="+str(i\*j))

Nhập vào một số nguyên dương **n**. In **n** năm nhuận tiếp theo sau năm nhuận gần nhất là 2016. Năm nhuận là năm thỏa mãn a) hoặc b)

a) Nó lặp lại 4 năm 1 lần, nhưng không lặp lại mỗi 100 năm. (chia hết cho 4 nhưng không chia hết cho 100)

b) Năm nhuận cũng lặp lại mỗi 400 năm. (chia hết cho 400)

**Input Format**

* Một số nguyên dương - **n**.

**Constraints**

* **n** > 0

**Output Format**

* In **n** năm nhuận sau năm 2016 trên 1 dòng, cách nhau 1 dấu cách.

**For example:**

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 5 | 5 | 2020 2024 2028 2032 2036 |

Answer:

n = int(input())

def nam\_nhuan(n):

nam = 2016

ket\_qua = []

while len(ket\_qua) < n:

nam += 4

if nam % 400 == 0:

ket\_qua.append(nam)

continue

if nam % 100 != 0:

ket\_qua.append(nam)

continue

return ket\_qua

print(' '.join([str(x) for x in nam\_nhuan(n)]))

Viết chương trình giải hệ phương trình bậc nhất 2 ẩn

a1x+b1y=c1

a2x+b2y=c2

Input: Các hệ số a1, b1, c1 và a2, b2, c2 cách nhau dấu cách, trên 2 dòng. Ví dụ:

2 3 4

7 8 9

Output: Nếu có nghiệm thì in ra nghiệm x, y cách nhau dấu cách, các nghiệm có độ chính xác 5 chữ số thập phân. Ví dụ: -1.12345 2.12345

Nếu vô nghiệm in ra chuỗi VN

Nếu vô số nghiệm in ra chuỗi VSN

Constraints: các hệ số và các nghiệm kiểu float

**For example:**

| **Input** | **Result** |
| --- | --- |
| -425.000000 -54.000000 -236.000000  -337.000000 143.000000 -306.000000 | 0.63657 -0.63969 |

Answer:

a1, b1, c1 = [float(x) for x in input().split()]

a2, b2, c2 = [float(x) for x in input().split()]

D = a1\*b2 - a2\*b1

Dx = c1\*b2 - c2\*b1

Dy = a1\*c2 - a2\*c1

if D == 0:

if Dx == Dy == 0:

print('VSN')

else:

print('VN')

else:

x = Dx/D

y = Dy/D

print('{0:.5f} {1:.5f}'.format(x, y))

Viết chương trình nhập vào 3 số nguyên a, b, c. Tìm ước số chung lớn nhất của 3 số trên nếu chúng là các số nguyên dương. Nếu không, thông báo "DL sai."

Input: a, b, c là số nguyên cách nhau dấu cách.

Output: Ước số chung lớn nhất của a, b, c, hoặc thông báo "DL sai."

Constrains: các số kiểu int

**For example:**

| **Input** | **Result** |
| --- | --- |
| 18 34 86 | 2 |

Answer:

a, b, c = [int(x) for x in input().split()]

def ucln(a, b):

if b == 0:

return a

return ucln(b, a % b)

if a <= 0 or b <= 0 or c <= 0:

print("DL sai.")

else:

print(ucln(ucln(a, b), c))

Nhập vào 3 số thực a, b, c. Hãy kiểm tra xem 3 số thực trên có tạo thành 3 cạnh của tam giác hay ko? Nếu có tính và in ra chu vi, diện tích tam giác. Nếu không thông báo "DL sai"

Input:

- a, b, c cách nhau dấu cách.

Output:

- In ra thông báo "DL Sai" hoặc in ra:

- Dòng 1: "Dien tich tam giac: X"

- Dòng 2: "Chu vi tam giac: Y"

Với X, Y là diện tích, chu vi của tam giác.

Constraints:

- các giá trị là số thực, độ chính xác 2 chữ số thập phân.

**For example:**

| **Input** | **Result** |
| --- | --- |
| -459.00 -33.00 -166.00 | DL Sai |

Answer:

from math import \*

from math import sqrt

a,b,c= [float(x) for x in input().split()]

if (a+b>=c) and (b+c>=a) and (a+c>=b):

cv= a+b+c

p=(a+b+c)/2

s=sqrt(p\*(p-a)\*(p-b)\*(p-c))

print ("Dien tich tam giac: ", "{:.2f}".format(s))

print ("Chu vi tam giac: ", "{:.2f}".format(cv))

else:

print ("DL Sai")

Viết chương trình nhập vào một mảng các số nguyên. Sắp xếp tăng dần các phần tử lẻ trong mảng. Các phần tử chẵn không thay đổi vị trí ban đầu.

Input:

+ Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

+ Dòng thứ hai nhập vào N số nguyên cách nhau dấu cách.

Output:

+ Dòng thứ nhất in ra số phần tử của mảng (N)

+ Dòng thứ hai in ra các phần tử của mảng kết quả cách nhau dấu cách.

Constrains:

+ Các phần tử trong mảng là các số nguyên

**For example:**

| **Input** | **Result** |
| --- | --- |
| 11  169 228 108 212 214 205 145 31 77 211 241 | 11  31 228 108 212 214 77 145 169 205 211 241 |

Answer:

a = int(input())

list = [int(x) for x in input().split()]

def sort(list):

for i in range(len(list) - 1):

if list[i] % 2 == 0:

continue

for j in range(i, len(list)):

if list[j] % 2 == 0:

continue

if list[i] > list[j]:

temp = list[i]

list[i] = list[j]

list[j] = temp

return list

print(a)

print(' '.join(str(x) for x in sort(list)))

Viết chương trình nhập vào một mảng các số nguyên. Sắp xếp tăng dần các phần tử chẵn trong mảng. Các phần tử lẻ không thay đổi vị trí ban đầu.

Input:

+ Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

+ Dòng thứ hai nhập vào N số nguyên cách nhau dấu cách.

Output:

+ Dòng thứ nhất in ra số phần tử của mảng (N)

+ Dòng thứ hai in ra các phần tử của mảng kết quả cách nhau dấu cách.

Constrains:

+ Các phần tử trong mảng là các số nguyên

**For example:**

| **Input** | **Result** |
| --- | --- |
| 11  169 224 228 108 212 205 214 145 77 211 241 | 11  169 108 212 214 224 205 228 145 77 211 241 |

Answer:

a = int(input())

list = [int(x) for x in input().split()]

def sort(list):

for i in range(len(list) - 1):

if list[i] % 2 != 0:

continue

for j in range(i, len(list)):

if list[j] % 2 != 0:

continue

if list[i] > list[j]:

temp = list[i]

list[i] = list[j]

list[j] = temp

return list

print(a)

print(' '.join(str(x) for x in sort(list)))

Viết hàm nhập vào mảng gồm n số thực, hàm in mảng đó ra màn hình.

Viết chương trình nhập và in ra một mảng các số thực có sử dụng hàm ở trên.

Input:

Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

Dòng thứ hai nhập vào N số thực cách nhau dấu cách, độ chính xác 3 chữ số thập phân.

Output:

Dòng thứ nhất in ra thông báo "Mang ban dau"

Dòng thứ 2 in ra số phần tử của mảng (N)

Dòng thứ 3 in ra các phần tử của mảng cách nhau dấu cách, độ chính xác 3 chữ số thập phân.

Constrains:

N kiểu số nguyên

Mảng kiểu số thực

**For example:**

| **Input** | **Result** |
| --- | --- |
| 7  756.000 165.000 263.000 -33.000 548.000 689.000 350.000 | Mang ban dau  7  756.000 165.000 263.000 -33.000 548.000 689.000 350.000 |

Answer:

def nhap\_mang():

a = int(input())

mang = [float(x) for x in input().split()]

return a, mang

def in\_mang(a, mang):

print('Mang ban dau')

print(a)

print(' '.join('{:.3f}'.format(x) for x in mang))

a, mang = nhap\_mang()

in\_mang(a, mang)

Viết các hàm nhập vào mảng gồm n số thực, in ra mảng, sắp xếp mảng số thực tăng dần.

Viết chương trình (sử dụng các hàm trên) thực hiện nhập, sắp xếp mảng và in kết quả ra màn hình.

Input:

+ Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

+ Dòng thứ hai nhập vào N số thực cách nhau dấu cách, độ chính xác 3 chữ số thập phân.

Output:

+ Dòng thứ nhất in ra thông báo "Mang sap xep"

+ Dòng thứ hai in ra các phần tử của mảng đã được sắp xếp, cách nhau dấu cách, độ chính xác 3 chữ số thập phân.

Constrains: Các phần tử trong mảng là các số thực, độ chính xác 3 chữ số thập phân.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 5  302.000 744.000 -302.000 744.000 32.000 | Mang sap xep  -302.000 32.000 302.000 744.000 744.000 |

Answer:

def nhap\_mang():

a = int(input())

mang = [float(x) for x in input().split()]

return a, mang

def in\_mang(a, mang):

print('Mang sap xep')

print(' '.join('{:.3f}'.format(x) for x in sorted(mang)))

a, mang = nhap\_mang()

in\_mang(a, mang)

Viết các hàm nhập vào mảng gồm n số thực, hàm in mảng đó ra màn hình, sắp xếp mảng số thực tăng dần.

Sử dụng các hàm ở trên viết chương trình nhập vào một mảng gồm N số thực, sắp xếp và in ra kết quả ra màn hình.

Input:

Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

Dòng thứ hai nhập vào N số thực cách nhau dấu cách, độ chính xác 3 chữ số thập phân.

Output:

+ Dòng thứ nhất in ra số phần tử của mảng (N)

+ Dòng thứ hai in ra các phần tử của mảng đã được sắp xếp, cách nhau dấu cách, độ chính xác 3 chữ số thập phân.

Constrains: Các phần tử trong mảng là các số thực, độ chính xác 3 chữ số thập phân.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 5  302.000 744.000 -302.000 744.000 32.000 | 5  -302.000 32.000 302.000 744.000 744.000 |

Answer:

def nhap\_mang():

a = int(input())

mang = [float(x) for x in input().split()]

return a, mang

def in\_mang(a, mang):

print(a)

print(' '.join('{:.3f}'.format(x) for x in sorted(mang)))

a, mang = nhap\_mang()

in\_mang(a, mang)

Viết các hàm nhập vào mảng gồm n số thực, hàm tìm vị trí phần tử lớn nhất trong mảng. Nếu có nhiều hơn một phần tử lớn nhất thì lấy phần tử có vị trí nhỏ nhất.

Sử dụng các hàm ở trên viết chương trình nhập một mảng các số thực. Sau đó in ra vị trí và giá trị phần tử lớn nhất trong mảng.

Input:

+ Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

+ Dòng thứ hai nhập vào N số thực cách nhau dấu cách, độ chính xác 3 chữ số thập phân.

Output:

+ In ra thông báo "Phan tu lon nhat co vi tri x, co gia tri y"

Trong đó: x, y là vị trí và giá trị của phần tử lớn nhất trong mảng.

Constrains: Các phần tử trong mảng là các số thực, độ chính xác 3 chữ số thập phân.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 5  302.000 744.000 -302.000 744.000 32.000 | Phan tu lon nhat co vi tri 1, co gia tri 744.000 |

Answer:

def nhap():

a = int(input())

array = [float(x) for x in input().split()]

return a, array

def vitricuamax(array: list):

gia\_tri\_max = max(array)

po\_max = array.index(gia\_tri\_max)

print('Phan tu lon nhat co vi tri {0}, co gia tri {1:.3f}'.format(

po\_max, gia\_tri\_max))

n, array = nhap()

vitricuamax(array)

Viết chương trình nhập vào một chuỗi ký tự. In ra số lần xuất hiện từng ký tự trong chuỗi.

Input: Chuỗi ký tự không bao gồm các ký tự điều khiển.

Output: Gồm nhiều dòng, mỗi dòng in ra X: Y

Trong đó: X là ký tự, Y là số lần xuất hiện của X trong chuỗi ban đầu. Thứ tự in ra tuân theo mã ASCII của ký tự.

Constrains: Chuỗi ký tự không bao gồm các ký tự điều khiển, có độ dài không quá 100.

Ví dụ:

Input: Chao Cac Ban

Output:

 : 2 //dấu cách xuất hiện 2 lần

B: 1

C: 2

a: 3

c: 1

h: 1

n: 1

o: 1

**For example:**

| **Input** | **Result** |
| --- | --- |
| Chao Cac Ban | : 2  B: 1  C: 2  a: 3  c: 1  h: 1  n: 1  o: 1 |

Answer:

string = input()

string\_arr = [x for x in string]

count\_table = dict.fromkeys(string\_arr, 0)

for char in string\_arr:

if char in count\_table:

count\_table[char] += 1

for item in sorted(count\_table.items()):

print('{}: {}'.format(item[0], item[1]))

Viết chương trình nhập vào một chuỗi gồm các chữ cái và dấu cách. Chuẩn hóa chuỗi vừa nhập thành dạng tên riêng.

Input: chuỗi ký tự bao gồm chữ cái, dấu cách.

Output: chuỗi ký tự ở dạng tên riêng.

Constrains: chuỗi ký tự chỉ chứa các chữ cái, dấu cách, độ dài không quá 255.

**For example:**

| **Input** | **Result** |
| --- | --- |
| NGuyen VAn AnH | Nguyen Van Anh |

Answer:

rr = input().split()

tenr = []

for i in rr:

ten =""

for n in i:

if n.isalpha():

ten = ten + n

ten = ten.lower()

ten = ten.capitalize()

if ten.isalpha():

tenr.append(ten)

print(" ".join(tenr))

Viết chương trình nhập vào một chuỗi ký tự chỉ gồm chữ cái, chữ số. Nếu các chữ cái, chữ số trong chuỗi xuất hiện nhiều hơn một lần thì xóa ở lần xuất hiện thứ 2, 3, ... In chuỗi kết quả ra màn hình.

Input: chuỗi ký tự có độ dài <=100.

Output: in ra chuỗi ký tự sao cho mỗi ký tự chỉ xuất hiện không quá một lần.

Constrains: chuỗi ký tự chỉ chứa chữ cái, chữ số, độ dài không quá 100.

**For example:**

| **Input** | **Result** |
| --- | --- |
| Chaocacbansinhvien | Chaocbnsive |

Answer:

string = input()

out = ''

for char in string:

if char not in out:

out += char

print(out)

Viết chương trình nhập vào 2 số nguyên a, b. In ra các số nguyên tố trong đoạn [a, b] nếu a<=b hoặc ngược lại. Nếu trong đoạn này không có số nguyên tố thì in ra thông báo "Khong co". Biết số nguyên tố là số nguyên dương >=2 và chỉ chia hết cho 1 và chính nó.

Input: a, b là số nguyên cách nhau dấu cách.

Output: các số nguyên tố tìm được viết trên một dòng, cách nhau dấu cách. Hoặc thông báo "Khong co".

Constrains: các số kiểu int

Example 1:

+ Input: -9 1

+ Output: Khong co

Example 2:

+ Input:  10 3

+ Output: 3 5 7

Example 3:

+ Input: 3 10

+ Output: 3 5 7

**For example:**

| **Input** | **Result** |
| --- | --- |
| -9 -33 | Khong co |
| 18 28 | 19 23 |

Answer:

a, b= [int(x) for x in input().split()]

if a > b:

j = a

a = b

b = j

def kiemtra(n):

if n <= 1:

return False

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

if n % i == 0 :

return False

return True

dem = 0

for i in range(a, b+1):

if kiemtra(i) == True:

print(i, end=" ")

dem = dem + 1

if dem == 0:

print("Khong co")

Cho một dãy số tự nhiên a và một số nguyên b. Viết chương trình tìm số lớn nhất chia hết cho b.

* Input: Mỗi test bao gồm hai dòng. Dòng đầu tiên là dãy a, các phần tử cách nhau bởi một khoảng trắng. Dòng thứ hai chứa số b.
* Output: Số lớn nhất chia hết cho b, hoặc 0 (nếu không tìm thấy số nào chia hết cho b)

**For example:**

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 1 3 8 4 9 2 10  2 | 10 |

Answer:

mang = [int(x) for x in input().split()]

a = int(input())

out = 0

for so in mang:

if so % a == 0 and so > out:

out = so

print(out)

Cho một số nguyên n, hãy viết chương trình kiểm tra xem n có chia hết cho 3 không?

* Đầu vào: số n
* Đầu ra: 'true' nếu n chia hết cho 3, 'false' nếu ngược lại.
* Ràng buộc: 10−100≤n≤1010010−100≤n≤10100

**For example:**

| **Input** | **Result** |
| --- | --- |
| 6 | true |
| -7 | false |

Answer:

n = int(input())

if n % 3 == 0:

print ("true")

else:

print ("false")

Viết chương trình nhập vào 2 số nguyên a, b. Tính tổng các số lẻ trong đoạn [a, b] hoặc [b, a]

Input:

a b trên cùng một dòng, cách nhau dấu cách.

Ví dụ: 5 12 hoặc 30 -8

Output:

Tổng các số lẻ tìm được. Ví dụ: 30

Constrains:

các biến kiểu nguyên

**For example:**

| **Input** | **Result** |
| --- | --- |
| 9 1 | 25 |

Answer:

a, b = [int(so) for so in input().split()]

if a > b:

i = a

a = b

b = i

tong = 0

for so in range(a, b+1):

if so %2 != 0:

tong += so

print(tong)

Viết chương trình giải phương trình (pt) ax+b=0.

Với **a**, **b** là hai số thực nhập từ bàn phím.

**Input Format**

* Hai số thực cách nhau dấu 1 cách.

**Constraints**

* Không có.

**Output Format**

* Nếu pt vô nghiệm, in ra **VN**.
* Nếu pt có 1 nghiệm, in ra nghiệm ở định dạng số thực với độ chính xác 2 chữ số sau dấu chấm thập phân.
* Nếu pt có vô số nghiệm, in ra **VSN**.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 1.00 2.00 | -2.00 |

Answer:

a, b = [float(x) for x in input().split()]

if a == 0:

if b == 0:

print("VSN")

else:

print("VN")

else:

c= -b / a

print('{0:.2f}'.format(c))

Viết chương trình nhập vào 4 số nguyên a, b, x, y. Tính trung bình cộng các số chẵn trong đoạn [a, b], hoặc [b, a], tính trung bình cộng các số lẻ trong đoạn [x, y], hoặc [y, x].

Input:

- a, b trên cùng một dòng, cách nhau dấu cách.

- x, y trên cùng một dòng, cách nhau dấu cách.

Ví dụ:

5 12

30 -8

Output:

Dòng 1: Trung bình cộng các số chẵn tìm được hoặc "NO" nếu không tính được.

Dòng 1: Trung bình cộng các số lẻ tìm được hoặc "NO" nếu không tính được.

Ví dụ:

30.00

NO

Constrains:

+ các biến a, b, x, y kiểu nguyên.

+ trung bình cộng có độ chính xác 2 chữ số thập phân.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 9 1  88 88 | 5.00  NO |

Answer:

a, b = [int(num\_ab) for num\_ab in input().split()]

x, y = [int(num\_xy) for num\_xy in input().split()]

if x > y:

z = x

x = y

y = z

if a > b:

c = a

a = b

b = c

tong\_ab = 0

count\_ab = 0

for num\_ab in range(a, b+1):

if num\_ab %2 == 0:

tong\_ab += num\_ab

count\_ab += 1

if count\_ab == 0:

print('NO')

else:

tbc\_ab = tong\_ab / count\_ab

print('{:.2f}'.format(tbc\_ab))

tong\_xy = 0

count\_xy = 0

for num\_xy in range(x, y+1):

if num\_xy %2 != 0:

tong\_xy += num\_xy

count\_xy += 1

if count\_xy == 0:

print('NO')

else:

tbc\_xy = tong\_xy / count\_xy

print('{:.2f}'.format(tbc\_xy))

Cho n khối hộp chữ nhật, hãy tìm khối hộp có thể tích lớn nhất.

* Đầu vào:
  + Dòng đầu tiên là số n
  + n dòng tiếp theo chứa thông tin của n khối hộp chữ nhật, mỗi dòng gồm ba số cách nhau bởi dấu cách là các số đo chiều dài, chiều rộng, và chiều cao của khối hộp.
* Đầu ra: bốn số cách nhau bởi dấu cách là số đo chiều dài, chiều rộng, chiều cao, và thể tích của khối hộp chữ nhật có thể tích lớn nhất. Kết quả hiển thị với độ chính xác 1 chữ số sau dấu phẩy.
* Ràng buộc:
  + n là số nguyên dương.
  + Số đo các chiều của một khối hộp là số thực dương.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3  2.0 6.4 5.0  2.6 3.2 8.0  3.1 1.1 6.0 | 2.6 3.2 8.0 66.6 |

Answer:

n = int(input())

hcns = []

for i in range(n):

hinh\_chu\_nhat = [float(x) for x in input().split()]

hcns.append(hinh\_chu\_nhat)

biggest\_hcn = max(

hcns, key=lambda hcn: hcn[0]\*hcn[1]\*hcn[2])

dai, rong, cao = biggest\_hcn

print('{0:.1f} {1:.1f} {2:.1f} {3:.1f}'.format(dai, rong, cao, dai\*rong\*cao))

Viết chương trình giải phương trình bậc 2 ax2+bx+c =0. Với hệ số a, b, c được nhập từ bàn phím.

Yêu cầu: Xét tất cả các trường hợp.

Input:

a, b, c là các số thực được nhập từ bàn phím

Output:

+ Nếu vô nghiệm ghi VN

+ Nếu vô số nghiệm ghi VSN

+ Nếu có 1 nghiệm ghi giá trị nghiệm với độ chính xác 3 chữ số thập phân

+ Nếu có 2 nghiệm ghi giá trị nghiệm với độ chính xác 3 chữ số thập phân, các nghiệm cách nhau dấu cách. Nghiệm nhỏ hơn đứng trước.

Ví dụ:

Input: 3 5 -8

Output:

-2.670 1.000

Constrains:

a, b, c là các số thực

**For example:**

| **Input** | **Result** |
| --- | --- |
| -20.000 -1.000 4.000 | -0.473 0.423 |

Answer:

from math import sqrt

a, b, c = [float(x) for x in input().split()]

def ptb1(a, b):

if a == 0:

if b == 0:

print('VSN')

else:

print('VN')

else:

result = -b/a

print('{:.3f}'.format(result))

def ptb2(a, b, c):

delta = b\*\*2 - 4\*a\*c

if delta < 0:

print("VN")

elif delta == 0:

result = -b / (2\*a)

print('{0:.3f}'.format(result))

else:

result = []

result.append((-b + sqrt(delta)) / (2\*a))

result.append((-b - sqrt(delta)) / (2\*a))

result = sorted(result)

print('{0:.3f} {1:.3f}'.format(result[0], result[1]))

if a == 0:

ptb1(b, c)

else:

ptb2(a, b, c)

Viết chương trình nhập vào 2 số thực, tính tích và in kết quả ra màn hình.

Input: Hai số thực cách nhau dấu cách. Ví dụ 4.00 5.00

Output: In ra thông báo: Tich cua X \* Y = Z //X, Y là 2 số ở input, Z là tích của X, Y

Constrains: Các giá trị kiểu số thực, độ chính xác 2 chữ số thập phân

**For example:**

| **Input** | **Result** |
| --- | --- |
| 1 6 | Tich cua 1.00 \* 6.00 = 6.00 |

Answer:

x, y = [float(n) for n in input().split()]

z= x\*y

print("Tich cua {:.2f} \* {:.2f} = {:.2f}".format(x,y,x\*y))

Số nguyên a là một số chính phương nếu tồn tại số nguyên b sao cho b\*b = a. Hãy viết chương trình kiểm tra xem một số có phải số chính phương hay không?

* Đầu vào: số a
* Đầu ra: true nếu a là số chính phương, false nếu a không phải số chính phương
* Ràng buộc: a là số nguyên

**For example:**

| **Input** | **Result** |
| --- | --- |
| 9 | true |
| 7 | false |

Answer:

n = int(input())

check = False

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

if (i\*\*2 == n):

check = True

break

if (check == True):

print("true")

else:

print("false")

Một câu được gọi là đối xứng nếu sau khi loại bỏ các dấu cách và các kí tự đặc biệt (chỉ giữ lại chữ thường, chữ hoa và chữ số), rồi chuyển các chữ cái thành chữ thường thì viết theo thứ tự ngược lại vẫn được câu đó. Ví dụ các câu sau được gọi là các câu đối xứng:

* Stressed desserts!
* Now I see bees, I won.

Hãy viết chương trình kiểm tra xem một câu có phải là câu đối xứng hay không?

* Đầu vào: câu s nằm trên một dòng
* Đầu ra: 'true' nếu s là câu đối xứng, 'false' nếu ngược lại.

**For example:**

| **Input** | **Result** |
| --- | --- |
| Now I see bees, I won. | true |
| I tried my best to pass the exam! | false |

Answer:

def ktdx(input\_string: str):

return input\_string == input\_string[::-1]

def normalize(input\_string: str):

result = input\_string.replace(' ', '')

result = ''.join(letter for letter in result if letter.isalnum())

result = result.lower()

return result

input\_string = input()

print('true' if ktdx(normalize(input\_string)) else 'false')

Trong mật mã học, Caesar là hệ mã cổ điển được đặt theo tên của một vĩ nhân trong lịch sử La Mã - Gaius Julius Caesar. Hệ mã này sử dụng một số tự nhiên **k** làm khoá để mã hoá hoặc giải mã văn bản. Việc mã hoá hay giải mã được thực hiện bằng cách dịch chuyển vị trí các kí tự trong bảng chữ cái đi **k** vị trí. Ví dụ với bảng chữ cái **A gồm** 10 kí tự {'a', 'n', 'o', 'b', ' ', 't', 'i', 'm', 'e', 'd'}**,** và khoá **k** = 2:

* Việc mã hoá được thực hiện bằng cách dịch các chữ cái sang phải 2 vị trí. Kí tự n được mã hoá thành b, e được mã hoá thành a. Như vậy bản rõ '**no time to die**' sẽ được mã hoá thành '**b imedaim inea**'.
* Việc giải mã được thực hiện bằng cách dịch các chữ cái sang trái 2 vị trí. Kí tự b được giải mã thành n, a được giải mã thành e. Như vậy bản mã '**b imedaim inea**' sẽ được giải mã thành '**no time to die**'.

Hãy viết chương trình mã hoá và giải mã sử dụng hệ Caesar theo mô tả trên.

* Đầu vào:
  + Bảng chữ cái **A** gồm **n** kí tự viết liền nhau
  + Khoá **k**
  + Một xâu kí tự **s**
  + Một số nhị phân thể hiện yêu cầu cần thực hiện trên xâu **s**, với 0 là mã hoá, 1 là giải mã.
* Đầu ra: xâu kết quả của việc mã hoá / giải mã.
* Ràng buộc: 0 < **k** ≤≤ **n**

**For example:**

| **Input** | **Result** |
| --- | --- |
| anob timed  2  no time to die  0 | b imedaim inea |
| anob timed  2  b imedaim inea  1 | no time to die |

Answer:

bang\_chu\_cai = [chu\_cai for chu\_cai in input()]

khoa = int(input())

xau\_ky\_tu = input()

yeu\_cau = int(input())

def ma\_hoa(xau: str, bang\_chu\_cai: list, khoa: int):

cac\_chu\_cai = [chu\_cai for chu\_cai in xau]

xau\_ma\_hoa = []

for chu\_cai in cac\_chu\_cai:

xau\_ma\_hoa.append(

bang\_chu\_cai[(bang\_chu\_cai.index(chu\_cai) + khoa) % len(bang\_chu\_cai)]

)

return xau\_ma\_hoa

def giai\_ma(xau: str, bang\_chu\_cai: list, khoa: int):

cac\_chu\_cai = [chu\_cai for chu\_cai in xau]

xau\_giai\_ma = []

for chu\_cai in cac\_chu\_cai:

xau\_giai\_ma.append(

bang\_chu\_cai[(bang\_chu\_cai.index(chu\_cai) - khoa) % len(bang\_chu\_cai)]

)

return xau\_giai\_ma

if yeu\_cau == 0:

print("".join(ma\_hoa(xau\_ky\_tu, bang\_chu\_cai, khoa)))

else:

print("".join(giai\_ma(xau\_ky\_tu, bang\_chu\_cai, khoa)))

Cho một ma trận, hãy sắp xếp các phần tử trong từng hàng theo thứ tự tăng, giảm xen kẽ theo thứ tự từ trái qua phải. Ví dụ, nếu hàng thứ nhất sắp xếp theo chiều tăng dần thì hàng thứ hai sắp xếp theo chiều giảm dần.

* Đầu vào:
  + Dòng đầu tiên là hai số m, n đại diện cho số hàng và số cột của ma trận, cách nhau bởi dấu cách
  + m dòng tiếp theo, mỗi dòng gồm n số là các phần tử trên một hàng cách nhau bởi dấu cách
  + Dòng cuối cùng là một số nhị phân chỉ ra cách sắp xếp hàng đầu tiên. 0 là giảm dần, 1 là tăng dần.
* Đầu ra: ma trận đã sắp xếp, in ra trên m dòng, mỗi dòng gồm n phần tử cách nhau bởi dấu cách.
* Ràng buộc: các phần tử trong mảng là số nguyên

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 4  3 1 2 3  4 6 4 5  1 3 2 1  1 | 1 2 3 3  6 5 4 4  1 1 2 3 |
| 4 3  3 2 4  5 6 3  6 3 4  1 1 1  0 | 4 3 2  3 5 6  6 4 3  1 1 1 |

Answer:

m, n = [int(x) for x in input().split()]

mang = []

for i in range(m):

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

afl = bool(int(input()))

for i, line in enumerate(mang):

if i % 2 == 0:

print(" ".join([str(x) for x in sorted(line, reverse=not afl)]))

else:

print(" ".join([str(x) for x in sorted(line, reverse=afl)]))

Một xâu kí tự được gọi là đối xứng nếu viết theo chiều ngược lại ta vẫn được xâu đó. Cho một xâu kí tự, hãy tìm xâu con đối xứng dài nhất.

* Đầu vào: xâu s
* Đầu ra: xâu con đối xứng dài nhất của s, nếu có nhiều xâu như vậy, hãy in ra xâu xuất hiện đầu tiên theo thứ tự từ trái sang phải.
* Ràng buộc: xâu s có ít nhất một kí tự

**For example:**

| **Input** | **Result** |
| --- | --- |
| shissi white | issi |
| easy | e |

Answer:

def kt\_xaunguoc(input\_string: str):

return input\_string == input\_string[::-1]

def tim\_xau\_dx\_dainhat(input\_string: str):

xau\_dx\_dainhat = ''

for i in range(len(input\_string)):

for j in range(i, len(input\_string)):

if kt\_xaunguoc(input\_string[i:j+1]) and (len(input\_string[i:j+1]) > len(xau\_dx\_dainhat)):

xau\_dx\_dainhat = input\_string[i:j+1]

return xau\_dx\_dainhat

input\_string = input()

print(tim\_xau\_dx\_dainhat(input\_string))

Bạn hãy xây dựng các lớp cần thiết để có thể:

Viết chương trình nhập vào một danh sách gồm n nhân viên. Tìm nhân viên có hệ số lương thấp nhất. Nếu có nhiều nhân viên có hệ số lương thấp nhất bằng nhau thì kết quả là nhân viên đầu tiên trong danh sách. Biết thông tin của một nhân viên gồm: tên, mã nv, hệ số lương, phụ cấp

Input:

+ Dòng thứ nhất nhập vào n là số nhân viên

+ Dòng tiếp theo nhập vào thông tin của các nhân viên theo thứ tự mã, tên, hệ số lương, phụ cấp.

Output:

+ Dòng thứ nhất in ra thông báo "Nhan vien co he so luong thap nhat"

+ Dòng thứ hai in ra thông tin của nhân viên tìm được. Các thông tin cách nhau dấu cách, thứ tự mã, tên, hệ số lương, phụ cấp.

Lương tháng được tính bằng hệ số lương \* 2000000 + phụ cấp.

Constrains: 0<=n<=200, mã nhân viên, phụ cấp là các số nguyên dương, tên nhân viên không chứa dấu cách, các thông tin còn lại là số thực có độ chính xác 2 chữ số thập phân.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4  78 Anh 2.34 1000000  5 Vinh 3.30 500000  61 Nhung 2.67 700000  27 Trang 2.34 250000 | Nhan vien co he so luong thap nhat  78 Anh 2.34 1000000 |

Answer:

class NhanVien:

def \_\_init\_\_(self, ma, ten, hsl, pc) -> None:

self.ma = int(ma)

self.ten = ten

self.hsl = float(hsl)

self.pc = int(pc)

def in\_thong\_tin(self):

print("{} {} {:.2f} {}".format(self.ma, self.ten, self.hsl, self.pc))

n = int(input())

cac\_nhan\_vien = []

for i in range(n):

cac\_nhan\_vien.append(NhanVien(\*input().split()))

print("Nhan vien co he so luong thap nhat")

min(cac\_nhan\_vien, key=lambda nhan\_vien: nhan\_vien.hsl).in\_thong\_tin()

Một số được gọi là số mạnh mẽ nếu nó chia hết cho một số nguyên tố và chia hết cho bình phương của một số nguyên tố. Ví dụ, 6363 là số mạnh mẽ vì nó chia hết cho 77 và 3232. Hãy viết chương trình kiểm tra xem một số có phải số mạnh mẽ hay không?

* Đầu vào: một số a
* Đầu ra: true nếu a là số mạnh mẽ, false nếu ngược lại
* Ràng buộc: a là số tự nhiên

**For example:**

| **Input** | **Result** |
| --- | --- |
| 63 | true |
| 10 | false |

Answer:

a = int(input())

def kiem\_tra\_nguyen\_to(a):

if a <= 1:

return False

for i in range(2, int(a / 2)):

if a % i == 0:

return False

return True

def tim\_cac\_so\_nguyen\_to(a):

return [x for x in range(a + 1) if kiem\_tra\_nguyen\_to(x)]

def kiem\_tra(a):

cac\_so\_nguyen\_to = tim\_cac\_so\_nguyen\_to(a)

for x in cac\_so\_nguyen\_to:

if a % x == 0:

break

else:

return False

for x in cac\_so\_nguyen\_to:

if a % (x \*\* 2) == 0:

break

else:

return False

return True

if kiem\_tra(a):

print("true")

else:

print("false")

Viết chương trình nhập và in ra ma trận gồm n hàng, m cột các số thực.

Input: Dòng thứ nhất nhập vào n và m là số hàng và số cột của ma trận.

Dòng tiếp theo nhập vào các phần tử của ma trận gồm n hàng, m cột

Output:

Dòng thứ nhất in ra thông báo "Ma tran ban vua nhap"

Dòng thứ hai in ra số hàng và số cột cách nhau dấu cách

Các dòng tiếp theo in ra các phần tử của ma trận cách nhau dấu cách.

Constranins: 1<=n, m<=200, các phần tử trong ma trận là các số thực, có độ chính xác 2 chữ số thập phân.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 6 18  13.20 14.40 2.50 10.00 21.00 18.00 21.00 19.00 2.00 1.00 4.00 20.00 20.00 10.00 19.00 14.00 7.00 19.00  15.00 19.00 8.00 0.00 21.00 5.00 17.00 5.00 13.00 5.00 18.00 15.00 4.00 9.00 12.00 23.00 20.00 24.00  24.00 5.00 22.00 18.00 7.00 0.00 17.00 12.00 23.00 15.00 19.00 13.00 21.00 13.00 17.00 3.00 8.00 24.00  1.00 8.00 0.00 6.00 9.00 20.00 18.00 16.00 23.00 23.00 12.00 10.00 6.00 5.00 22.00 21.00 6.00 1.00  21.00 24.00 8.00 19.00 20.00 15.00 18.00 17.00 8.00 1.00 6.00 3.00 5.00 7.00 11.00 24.00 19.00 13.00  6.00 2.00 2.00 4.00 22.00 3.00 5.00 14.00 14.00 22.00 16.00 18.00 1.00 12.00 7.00 9.00 20.00 24.00 | Ma tran ban vua nhap  6 18  13.20 14.40 2.50 10.00 21.00 18.00 21.00 19.00 2.00 1.00 4.00 20.00 20.00 10.00 19.00 14.00 7.00 19.00  15.00 19.00 8.00 0.00 21.00 5.00 17.00 5.00 13.00 5.00 18.00 15.00 4.00 9.00 12.00 23.00 20.00 24.00  24.00 5.00 22.00 18.00 7.00 0.00 17.00 12.00 23.00 15.00 19.00 13.00 21.00 13.00 17.00 3.00 8.00 24.00  1.00 8.00 0.00 6.00 9.00 20.00 18.00 16.00 23.00 23.00 12.00 10.00 6.00 5.00 22.00 21.00 6.00 1.00  21.00 24.00 8.00 19.00 20.00 15.00 18.00 17.00 8.00 1.00 6.00 3.00 5.00 7.00 11.00 24.00 19.00 13.00  6.00 2.00 2.00 4.00 22.00 3.00 5.00 14.00 14.00 22.00 16.00 18.00 1.00 12.00 7.00 9.00 20.00 24.00 |

Answer:

m, n = [int(x) for x in input().split()]

ma\_tran = []

for i in range(m):

ma\_tran.append([float(x) for x in input().split()])

print("Ma tran ban vua nhap")

print(m, n)

for mang in ma\_tran:

print(" ".join("{:.2f}".format(x) for x in mang))

Viết chương trình nhập vào một ma trận gồm n hàng, m cột các số nguyên. Sắp xếp từng hàng của ma trận tăng dần. In kết quả ra màn hình.

Input:

- Dòng thứ nhất nhập vào n và m

- Dòng tiếp theo nhập vào các phần tử của ma trận

Output:

- Dòng thứ nhất in ra n và m cách nhau dấu cách

- Dòng tiếp theo in ra ma trận kết quả

Constrains: 1<=n, m<=100, các phần tử trong ma trận là số nguyên.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 6 6  17 23 9 3 0 9  12 2 21 12 23 7  12 2 2 13 7 6  7 24 22 12 11 20  4 5 7 24 2 11  15 23 20 22 20 24 | 6 6  0 3 9 9 17 23  2 7 12 12 21 23  2 2 6 7 12 13  7 11 12 20 22 24  2 4 5 7 11 24  15 20 20 22 23 24 |

Answer:

m, n = [int(x) for x in input().split()]

ma\_tran = []

for i in range(m):

ma\_tran.append([int(x) for x in input().split()])

print(m, n)

for mang in ma\_tran:

mang = sorted(mang)

print(" ".join(str(x) for x in mang))

Bạn hãy xây dựng các lớp cần thiết và thực hiện công việc sau.

Công ty Điện lực HP quản lý danh sách n khách hàng với các thông tin: mã khách hàng, tên khách hàng, chỉ số đầu kỳ, chỉ số cuối kỳ. Viết chương trình nhập vào một danh sách gồm n khách hàng. In thông tin khách hàng phải trả nhiều tiền nhất. Nếu có nhiều khách hàng phải trả nhiều tiền nhất thì in ra khách hàng có số thứ tự đầu tiên tìm thấy.

Input:

+ Dòng thứ nhất nhập vào n là số khách hàng

+ Dòng tiếp theo nhập vào thông tin của các khách hàng theo thứ tự tên, mã số khách hàng, chỉ số đầu kỳ, chỉ số cuối kỳ.

Output:

+ Dòng thứ nhất in ra thông báo: "Khach hang phai tra tien nhieu nhat: X". Với X là số thứ tự của khách hàng tìm được.

+ Dòng tiếp theo in ra thông tin của khách hàng tìm được. Các thông tin cách nhau dấu cách (thứ tự: mã khách hàng, tên, chỉ số đầu kỳ, chỉ số cuối kỳ, lượng điện tiêu thụ, số tiền phải trả.

+ Lượng điện tiêu thụ = chỉ số cuối kỳ - chỉ số đầu kỳ.

+ Số tiền phải trả được tính như sau:

- 100 số (kW) đầu tiên đơn giá là 1000 đồng.

- 100 số (kW) tiếp theo đơn giá là 1500 đồng.

- Từ 201số (kW) trở lên thì đơn giá là 2000 đồng.

Constrains: 1<=n<=200, tên thí sinh không chứa dấu cách, các thông tin còn lại là số nguyên.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3  Yen 919 55 363  Trang 99 10 643  Thu 5 72 82 | Khach hang phai tra tien nhieu nhat: 1  99 Trang 10 643 633 1116000 |

Answer:

class KhachHang:

def \_\_init\_\_(self, ten\_khach\_hang, ma\_khach\_hang, so\_dau\_ky, so\_cuoi\_ky) -> None:

self.ten\_khach\_hang = ten\_khach\_hang

self.ma\_khach\_hang = ma\_khach\_hang

self.so\_dau\_ky = int(so\_dau\_ky)

self.so\_cuoi\_ky = int(so\_cuoi\_ky)

self.luong\_dien\_tieu\_thu = self.so\_cuoi\_ky - self.so\_dau\_ky

def \_\_gioi\_han\_\_(self, gia\_tri, max\_gia\_tri):

if gia\_tri < max\_gia\_tri:

return gia\_tri

return max\_gia\_tri

def tinh\_tien(self):

so\_dien\_1 = self.\_\_gioi\_han\_\_(self.luong\_dien\_tieu\_thu, 100)

so\_dien\_2 = self.\_\_gioi\_han\_\_(

self.luong\_dien\_tieu\_thu, 200) - so\_dien\_1

so\_dien\_3 = self.luong\_dien\_tieu\_thu - so\_dien\_2 - so\_dien\_1

tong\_gia = so\_dien\_1 \* 1000 + so\_dien\_2 \* 1500 + so\_dien\_3 \* 2000

return tong\_gia

n = int(input())

cac\_khach\_hang = []

for i in range(n):

cac\_khach\_hang.append(KhachHang(\*input().split()))

khach\_hang\_max = max(

cac\_khach\_hang, key=lambda khach\_hang: khach\_hang.tinh\_tien())

print('Khach hang phai tra tien nhieu nhat: {}'. format(

cac\_khach\_hang.index(khach\_hang\_max)))

print('{} {} {} {} {} {}'.format(

khach\_hang\_max.ma\_khach\_hang,

khach\_hang\_max.ten\_khach\_hang,

khach\_hang\_max.so\_dau\_ky,

khach\_hang\_max.so\_cuoi\_ky,

khach\_hang\_max.luong\_dien\_tieu\_thu,

khach\_hang\_max.tinh\_tien()))

Viết các hàm nhập vào mảng gồm n số thực, tìm vị trí khóa key trong mảng. Nếu có nhiều hơn một phần bằng key thì lấy vị trí nhỏ nhất. Nếu không có khóa key trong mảng thì hàm trả về giá trị -1.

Sử dụng các hàm ở trên viết chương trình nhập một mảng các số thực. Sau đó nhập vào một số thực X, in ra vị trí của X trong mảng nếu tìm thấy. Ngược lại in ra thông báo " X not found".

Input:

+ Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

+ Dòng thứ hai nhập vào N số thực cách nhau dấu cách, độ chính xác 3 chữ số thập phân.

+ Dòng thứ ba nhập vào số thực X.

Output:

+ In ra thông báo "Phan tu X co vi tri y" hoặc "X not found"

Trong đó: X là khóa cần tìm, y là vị trí của X trong mảng.

Constrains: Các phần tử trong mảng là các số thực, độ chính xác 3 chữ số thập phân.

Examples:

+ Input:

5 //N

302.000 744.000 -302.000 744.000 32.000 //Mảng

320.000 //khóa X

+ Output: 320.000 not found

**For example:**

| **Input** | **Result** |
| --- | --- |
| 5  302.000 744.000 -302.000 744.000 32.000  32.000 | Phan tu 32.000 co vi tri 4 |

Answer:

def nhap():

n = int(input())

mang = [float(x) for x in input().split()]

key = float(input())

return n, mang, key

def vi\_tri\_key(mang: list, key: float):

for i in range(len(mang)):

if mang[i] == key:

return i

return -1

n, mang, key = nhap()

i = vi\_tri\_key(mang, key)

if i != -1:

print("Phan tu {0:.3f} co vi tri {1}".format(key, i))

else:

print("{:.3f} not found".format(key))

Dãy số tăng là dãy số mà trong đó mọi số đứng sau đều lớn hơn số đứng trước.

Viết hàm kiểm tra một mảng số thực gồm n phàn tử có tăng hay ko? Hàm nhận 2 tham số là mảng a và số phần tử. Hàm trả về 0 nếu dãy a ko tăng, 1 nếu ngược lại.

Viết chương trình (hàm main) nhập mảng **a** gồm **n** số thực và mảng b gồm m số thực. Với **n, m** là các số nguyên dương nhập từ bàn phím.

Kiểm tra phần tử mảng a, b có phải là một dãy số tăng hay không?

**Input Format**

Dữ liệu vào nằm trên 4 dòng:

* Dòng 1: một số nguyên dương chỉ số phần từ mảng a.
* Dòng 2: các phần tử mảng a, số thực, cách nhau bởi dấu cách.
* Dòng 3: một số nguyên dương chỉ số phần từ mảng b.
* Dòng 4: các phần tử mảng b, số thực, cách nhau bởi dấu cách.

**Constraints**

* Mảng nhập vào có ít nhất 1 phần tử và nhiều nhất là 500 phần tử. (1<=n, m<=500)
* Kiểu các phần tử mảng là số thực

**Output Format**

**Hai dòng: thông báo kết quả của mảng a và b tương ứng**

* Nếu dãy số tăng, in ra **TANG**
* Ngược lại, in ra **KHONG\_TANG**

**For example:**

| **Input** | **Result** |
| --- | --- |
| 5  2.00 4.50 6.12 8.30 10.99  5  2.00 4.50 6.15 10.95 8.00 | TANG  KHONG\_TANG |

Answer:

n = int(input())

a = [float(x) for x in input().split()]

m = int(input())

b = [float(x) for x in input().split()]

def kiemTra():

dem\_a = 0

dem\_b = 0

for i in range(len(a)-1):

if(a[i] < a[i+1]):

dem\_a = dem\_a + 1

else:

dem\_a = dem\_a - 1

if dem\_a == len(a) - 1:

print("TANG")

else:

print("KHONG\_TANG")

for j in range(len(b)-1):

if(b[j] < b[j+1]):

dem\_b = dem\_b + 1

else:

dem\_b = dem\_b - 1

if dem\_b == len(b) - 1:

print("TANG")

else:

print("KHONG\_TANG")

kiemTra()

Viết chương trình nhập vào một ma trận gồm n hàng, m cột các số nguyên. In ra vị trí của phần tử lớn nhất của từng hàng trong ma trận. Nếu trong hàng có nhiều phần tử lớn nhất thì in ra phần tử có chỉ số cột nhỏ nhất.

**Input:**

- Dòng thứ nhất nhập vào n và m

- Dòng tiếp theo nhập vào các phần tử của ma trận

**Output:** gồm n dòng, mỗi dòng in ra 3 giá trị x y Z cách nhau dấu cách

Trong đó: x là chỉ số hàng, y là chỉ số cột và Z là giá trị của phần tử lớn nhất tìm được.

**Constrains:**1<=n, m<=100, các phần tử trong ma trận là số nguyên.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 25 5  9 4 9 19 19  16 10 17 12 9  6 9 9 10 14  9 0 8 24 22  1 8 9 0 23  4 13 7 18 6  24 12 23 9 16  0 11 18 1 16  15 22 9 12 16  12 9 13 10 11  1 3 15 6 17  3 4 12 23 11  4 4 20 19 1  16 4 13 14 11  13 4 0 12 11  19 6 8 11 9  16 0 8 12 17  14 22 2 12 9  0 14 1 9 12  19 7 11 16 6  18 12 6 4 17  17 14 13 21 10  24 20 5 1 8  14 15 22 3 6  22 5 4 13 14 | 0 3 19  1 2 17  2 4 14  3 3 24  4 4 23  5 3 18  6 0 24  7 2 18  8 1 22  9 2 13  10 4 17  11 3 23  12 2 20  13 0 16  14 0 13  15 0 19  16 4 17  17 1 22  18 1 14  19 0 19  20 0 18  21 3 21  22 0 24  23 2 22  24 0 22 |

Answer:

m,n = [int(x) for x in input().split()]

mang=[]

for i in range(m):

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

def vt\_max(ds\_so):

so\_max = max(ds\_so)

for i in range(len(ds\_so)):

if ds\_so[i]==so\_max:

return i

for i in range(m):

vi\_tri\_max = vt\_max(mang[i])

max\_value = mang[i][vi\_tri\_max]

print("{0} {1} {2}".format(i,vi\_tri\_max,max\_value))

Viết chương trình nhập vào một mảng các số nguyên. Sắp xếp mảng đó tăng dần.

Input:

+ Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

+ Dòng thứ hai nhập vào N số nguyên cách nhau dấu cách.

Output:

+ Dòng thứ nhất in ra số phần tử của mảng (N)

+ Dòng thứ hai in ra các phần tử của mảng (đã được sắp xếp tăng dần) cách nhau dấu cách.

Constrains:

+ Các phần tử trong mảng là các số nguyên

**For example:**

| **Input** | **Result** |
| --- | --- |
| 11  169 224 228 108 212 214 205 145 77 211 241 | 11  77 108 145 169 205 211 212 214 224 228 241 |

Answer:

n = int(input())

N = [int(x) for x in input().split()]

for i in range(len(N)-1):

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

if N[i] > N[j]:

temp = N[i]

N[i] = N[j]

N[j] = temp

print(n)

print(" ".join("{}".format(x) for x in N))

Xây dựng các lớp cần thiết để có thể: Viết chương trình nhập vào một danh sách gồm n sinh viên (n<=100). In danh sách đó ra màn hình. Biết thông tin của một sinh viên gồm: tên, mã sv, điểm Toán, điểm Triết, điểm LT C

Input:

+ Dòng thứ nhất nhập vào n là số sinh viên

+ Dòng tiếp theo nhập vào thông tin của các sinh viên theo thứ tự tên, mã, điểm Toán, điểm Triết, điểm LT C.

Output:

+ Dòng thứ nhất in ra thông báo "Danh sach sinh vien"

+ Dòng thứ hai in ra số sinh viên trong danh sách

+ Các dòng tiếp theo, mỗi dòng in ra thông tin của từng sinh viên. Các thông tin cách nhau dấu cách (thứ tự tên, mã, điểm Toán, điểm Triết, điểm LT C), điểm trung bình in ở cuối mỗi dòng.

Constrains: 1<=n<=100, điểm các môn và điểm trung bình có độ chính xác 2 chữ số thập phân, mã sinh viên là các số nguyên dương, tên sinh viên không chứa dấu cách.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4  Anh 78 10.00 1.00 0.00  Vinh 5 7.00 5.00 8.00  Nhung 61 7.00 3.00 7.00  Trang 27 2.00 7.00 7.00 | Danh sach sinh vien  4  Anh 78 10.00 1.00 0.00 3.67  Vinh 5 7.00 5.00 8.00 6.67  Nhung 61 7.00 3.00 7.00 5.67  Trang 27 2.00 7.00 7.00 5.33 |

Answer:

class SV:

def \_\_init\_\_(self, ten, ma, dToan, dTriet, dLtC) -> None:

self.ten = ten

self.ma = ma

self.dToan = float(dToan)

self.dTriet = float(dTriet)

self.dLtC = float(dLtC)

self.dtb = ((self.dToan + self.dTriet + self.dLtC) / 3)

def xuat(self):

print("{} {} {:.2f} {:.2f} {:.2f} {:.2f}".format(self.ten,self.ma,self.dToan,self.dTriet,self.dLtC, self.dtb))

n = int(input())

cac\_sinh\_vien = []

for i in range(n):

cac\_sinh\_vien.append(SV(\*input().split()))

print("Danh sach sinh vien")

print(n)

for sinh\_vien in cac\_sinh\_vien:

sinh\_vien.xuat()

Viết chương trình nhập vào 2 ma trận X (có n1 hàng, m1 cột) và Y (có n2 hàng, m2 cột) các số nguyên. Tính tổng 2 ma trận đó và in kết quả ra màn hình. Nếu không tính được tổng 2 ma trận thì in ra thông báo "Du lieu vao sai"

Input:

+ Dòng thứ nhất nhập vào n1 và m1 là số hàng và số cột của ma trận X.

+ Dòng tiếp theo nhập vào các phần tử của ma trận X gồm n1 hàng, m1 cột

+ Dòng tiếp theo nhập vào n2 và m2 là số hàng và số cột của ma trận Y.

+ Dòng tiếp theo nhập vào các phần tử của ma trận Y gồm n2 hàng, m2 cột

Output:

+ Nếu tính được tổng:

- Dòng thứ nhất in ra thông báo "Ma tran tong"

- Các dòng tiếp theo in ra các phần tử của ma trận cách nhau dấu cách.

+ Nếu không tính được tổng:

- In ra thông báo "Du lieu vao sai"

Constranins: 1<=n, m<=100, các phần tử trong ma trận là các số nguyên.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 2 8  9 0 19 24 3 8 12 14  5 20 6 2 11 16 20 17  2 8  2 11 16 4 2 3 17 7  21 16 18 20 22 1 21 13 | Ma tran tong  11 11 35 28 5 11 29 21  26 36 24 22 33 17 41 30 |
| 5 6  10 6 16 1 7 1  12 17 21 10 16 23  12 9 9 4 22 7  23 0 13 23 17 4  9 5 8 0 23 4  6 3  23 6 0  13 16 5  5 17 12  11 2 24  3 2 18  21 23 15 | Du lieu vao sai |

Answer:

m1, n1 = [int(x) for x in input().split()]

ma\_tran\_1 = []

for i in range(m1):

ma\_tran\_1.append([int(x) for x in input().split()])

m2, n2 = [int(x) for x in input().split()]

ma\_tran\_2 = []

for j in range(m2):

ma\_tran\_2.append([int(x) for x in input().split()])

def cong(ma\_tran\_1, ma\_tran\_2):

hang = len(ma\_tran\_1)

cot = len(ma\_tran\_1[0])

result = [[0] \* cot for i in range(hang)]

for i in range(hang):

for j in range(cot):

result[i][j] = ma\_tran\_1[i][j] + ma\_tran\_2[i][j]

return result

if m1 == m2 and n1 == n2:

print("Ma tran tong")

ma\_tran = cong(ma\_tran\_1, ma\_tran\_2)

for i in range(m1):

print(" ".join( str(x) for x in ma\_tran[i]))

else:

print("Du lieu vao sai")

Xây dựng các lớp cần thiết để có thể: Viết chương trình nhập vào danh sách gồm n hình tròn trong mặt phẳng. In thông tin của hình tròn có tâm gần gốc tọa độ nhất ra màn hình. Nếu có nhiều hình tròn thỏa mãn thì kết quả là hình tròn có số thứ tự nhỏ nhất. Biết thông tin của hình tròn gồm mã hình tròn, bán kính, tọa độ tâm (x, y).

Input:

+ Dòng thứ nhất nhập vào số nguyên dương n

+ Dòng tiếp theo nhập vào thông tin của từng hình tròn. Thứ tự nhập vào là mã, bán kính, tâm x, tâm y.

Output:

+ In ra thông tin của hình tròn tìm được. Thứ tự in ra là mã, tọa độ x, tọa độ y, bán kính, diện tích, chu vi. Các thông tin cách nhau dấu cách. Giá trị số thực có độ chính xác 3 chữ số thập phân.

Constrains:

+ 1<=n<=1000

+ bán kính là số thực, tọa độ tâm là các số nguyên

+ Số pi =3.14159

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4  78 19.667 962 464  5 15.333 281 827  61 30.667 995 942  27 12.333 391 604 | 27 391 604 12.333 477.845 77.490 |

Answer:

from math import sqrt

PI = 3.14159

class HinhTron:

def \_\_init\_\_(self, ma, bk, tam\_x, tam\_y) -> None:

self.ma = float(ma)

self.bk = float(bk)

self.tam\_x = int(tam\_x)

self.tam\_y= int(tam\_y)

def dientich(self):

return self.bk \* self.bk \* PI

def chuvi(self):

return self.bk \* 2 \* PI

def khoang\_cach\_toi\_tam(self):

return sqrt((self.tam\_x\*self.tam\_x) + (self.tam\_y\*self.tam\_y))

def hinh\_tron\_gan\_tam\_nhat(cac\_hinh\_tron):

result = cac\_hinh\_tron[0]

for hinh\_tron in cac\_hinh\_tron:

if hinh\_tron.khoang\_cach\_toi\_tam() < result.khoang\_cach\_toi\_tam():

result = hinh\_tron

return result

n = int(input())

cac\_hinh\_tron= []

for i in range(n):

cac\_hinh\_tron.append(HinhTron(\*input().split()))

hinh\_tron\_can\_tim = hinh\_tron\_gan\_tam\_nhat(cac\_hinh\_tron)

print(

"{:.0f} {:.0f} {:.0f} {:.3f} {:.3f} {:.3f}".format(

hinh\_tron\_can\_tim.ma,

hinh\_tron\_can\_tim.tam\_x,

hinh\_tron\_can\_tim.tam\_y,

hinh\_tron\_can\_tim.bk,

hinh\_tron\_can\_tim.dientich(),

hinh\_tron\_can\_tim.chuvi(),

)

)

Giải bất phương trình **ax+b>0**, với **a** và **b** là 2 số nguyên nhập vào.

**Input Format**

Hai số nguyên, **a** và **b**, cách nhau 1 dấu cách.

**Constraints**

Không có.

**Output Format**

* Nếu bpt có vô số nghiệm, in ra **VSN**
* Nếu bpt vô nghiệm, in ra **VN**
* Nếu bpt có nghiệm, in ra nghiệm dạng **x>p** hoặc **x<p**

Trong đó, **p** là một giá trị thực với độ chính xác 2 chữ số sau dấu chấm.

**For example:**

| **Input** | **Result** |
| --- | --- |
| -2 4 | x<2.00 |

Answer:

a, b = [int(x) for x in input().split()]

if a == 0:

if b > 0:

print("VSN")

else:

print("VN")

else:

if a>0:

print("x>{:.2f}".format(-b/a))

else:

print("x<{:.2f}".format(-b/a))

Viết chương trình nhập vào một mảng các số nguyên. Tìm vị trí và giá trị của phần tử chẵn lớn nhất trong mảng. Nếu có nhiều số thỏa mãn thì kết quả là số có số thứ tự nhỏ nhất.

Input: Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

Dòng thứ hai nhập vào mảng gồm N số nguyên cách nhau dấu cách.

Output:

+ In ra thông báo "So chan lon nhat co vi tri X gia tri Y"//X là vị trí, Y là giá trị của phần tử tìm được.

+ Nếu trong mảng không có số chẵn thì thông báo "Khong co so chan trong mang".

Constrains: Các phần tử trong mảng là các số nguyên.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 13  163 195 75 79 148 148 237 41 175 147 24 122 6 | So chan lon nhat co vi tri 4 gia tri 148 |
| 11  881 2041 1671 2011 465 2483 141 2421 2371 1741 131 | Khong co so chan trong mang |

Answer:

n = int(input())

N = [float(x) for x in input().split()]

def vi\_tri(N):

vt = 0

for i in range(len(N)):

if N[i] % 2 == 0 and N[i] > N[vt]:

vt = i

return vt

return -1

vt = vi\_tri(N)

if vt == -1:

print("Khong co so chan trong mang")

else:

print("So chan lon nhat co vi tri {} gia tri {}",vt,N[vt])

Viết chương trình nhập và in ra ma trận gồm n hàng, m cột các số nguyên.

Input: Dòng thứ nhất nhập vào n và m là số hàng và số cột của ma trận.

Dòng tiếp theo nhập vào các phần tử của ma trận gồm n hàng, m cột

Output:

Dòng thứ nhất in ra thông báo "Ma tran ban dau"

Dòng thứ hai in ra số hàng và số cột cách nhau dấu cách

Các dòng tiếp theo in ra các phần tử của ma trận cách nhau dấu cách.

Constranins: 1<=n, m<=100, các phần tử trong ma trận là các số nguyên.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 6 18  13 14 2 10 21 18 21 19 2 1 4 20 20 10 19 14 7 19  15 19 8 0 21 5 17 5 13 5 18 15 4 9 12 23 20 24  24 5 22 18 7 0 17 12 23 15 19 13 21 13 17 3 8 24  1 8 0 6 9 20 18 16 23 23 12 10 6 5 22 21 6 1  21 24 8 19 20 15 18 17 8 1 6 3 5 7 11 24 19 13  6 2 2 4 22 3 5 14 14 22 16 18 1 12 7 9 20 24 | Ma tran ban dau  6 18  13 14 2 10 21 18 21 19 2 1 4 20 20 10 19 14 7 19  15 19 8 0 21 5 17 5 13 5 18 15 4 9 12 23 20 24  24 5 22 18 7 0 17 12 23 15 19 13 21 13 17 3 8 24  1 8 0 6 9 20 18 16 23 23 12 10 6 5 22 21 6 1  21 24 8 19 20 15 18 17 8 1 6 3 5 7 11 24 19 13  6 2 2 4 22 3 5 14 14 22 16 18 1 12 7 9 20 24 |

Answer:

m, n = [int(x) for x in input().split()]

matrix = []

for i in range(m):

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

print("Ma tran ban dau")

print(m, n)

for matran in matrix:

print(" ".join(str(x) for x in matran))

Viết hàm nhập vào mảng gồm n số nguyên, hàm in mảng đó ra màn hình.

Viết chương trình nhập và in ra một mảng các số nguyên có sử dụng hàm ở trên.

Input: Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

Dòng thứ hai nhập vào N số nguyên cách nhau dấu cách.

Output:

Dòng thứ nhất in ra thông báo "Mang ban dau"

Dòng thứ 2 in ra số phần tử của mảng (N)

Dòng thứ 3 in ra các phần tử của mảng cách nhau dấu cách.

Constrains: Các phần tử trong mảng là các số nguyên

**For example:**

| **Input** | **Result** |
| --- | --- |
| 12  20 11 -4 11 9 24 -15 9 12 1 13 0 | Mang ban dau  12  20 11 -4 11 9 24 -15 9 12 1 13 0 |

Answer:

n = int(input())

N = [int(x) for x in input().split()]

print("Mang ban dau")

print(n)

print(" ".join(str(x) for x in N))

Viết chương trình nhập vào một ma trận gồm n hàng, m cột các số nguyên. In ra vị trí của phần tử lớn nhất của từng hàng trong ma trận. Nếu trong hàng có nhiều phần tử lớn nhất thì in ra phần tử có chỉ số cột nhỏ nhất.

Input:

- Dòng thứ nhất nhập vào n và m

- Dòng tiếp theo nhập vào các phần tử của ma trận

Output: gồm n dòng, mỗi dòng in ra 3 giá trị x y Z cách nhau dấu cách

Trong đó: x là chỉ số hàng, y là chỉ số cột và Z là giá trị của phần tử lớn nhất tìm được.

Constrains: 1<=n, m<=100, các phần tử trong ma trận là số nguyên.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 6 21  5 0 18 19 12 19 4 7 0 9 20 22 13 11 7 4 0 23 16 19 11  6 17 9 10 2 18 23 4 6 8 7 20 18 13 13 3 5 3 2 17 5  4 20 16 10 5 10 10 16 11 24 12 13 9 3 19 4 23 0 14 14 9  20 23 15 13 17 2 17 1 11 24 12 24 0 5 0 14 8 4 0 11 16  21 23 24 0 10 24 4 4 8 18 22 18 13 15 24 15 17 4 10 10 14  24 11 14 1 9 14 3 13 7 15 22 19 11 21 13 14 1 1 15 21 23 | 0 17 23  1 6 23  2 9 24  3 9 24  4 2 24  5 0 24 |

Answer:

m, n = [int(x) for x in input().split()]

mang = []

for i in range(m):

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

def vt\_max(ds\_so):

so\_max = max(ds\_so)

for i in range(len(ds\_so)):

if ds\_so[i] == so\_max:

return i

for i in range(m):

vi\_tri\_max = vt\_max(mang[i])

value\_max = mang[i][vi\_tri\_max]

print("{0} {1} {2}".format(i, vi\_tri\_max, value\_max))

Nhập vào một số nguyên n. Kiểm tra xem số đó có là số hoàn hảo không. Biết số hoàn hảo là một số nguyên dương mà tổng các ước nguyên dương chính thức của nó (số nguyên dương bị nó chia hết ngoại trừ nó) bằng chính nó. Ví dụ số 6 là số hoàn hảo vì 6=1+2+3.

Input: Số nguyên n kiểu int được nhập từ bàn phím. Ví dụ: -30

Output: In ra thông báo "n la so hoan hao" hoặc "n khong la so hoan hao".

Ví dụ: "6 la so hoan hao"

Contrain: n kiểu int

**For example:**

| **Input** | **Result** |
| --- | --- |
| 2005 | 2005 khong la so hoan hao |

Answer:

n = int(input())

sum = 0

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

if(n%i==0):

sum = sum + i

if sum == n:

print("{} la so hoan hao".format(n))

else:

print("{} khong la so hoan hao".format(n))

Viết hàm kiểm tra một ma trận vuông có đối xứng qua đường chéo chính không.

Viết chương trình nhập vào hai ma trận vuông cấp N1, N2. Kiểm tra xem các ma trận có đối xứng qua đường chéo chính ko. In kết quả ra màn hình.

Input:

- Dòng thứ nhất nhập vào N1 (kích thước của ma trận thứ nhất)

- Các dòng tiếp theo nhập vào các phần tử của ma trận thứ nhất.

- Dòng tiếp theo nhập vào N2 (kích thước của ma trận thứ nhất)

- Các dòng tiếp theo nhập vào các phần tử của ma trận thứ hai.

Output:

- Dòng thứ nhất in ra thông báo "Ma tran 1: Co/Khong doi xung"

- Dòng thứ nhất in ra thông báo "Ma tran 2: Co/Khong doi xung"

Constrains: 1<=N, N2<=100, các phần tử trong ma trận là số nguyên.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 6  5 0 5 11 35 61  28 31 12 9 35 5  16 57 5 55 23 34  4 33 27 14 54 61  12 0 34 61 6 23  5 50 50 51 11 34  3  5 0 7  0 11 2  7 2 6 | Ma tran 1: Khong doi xung  Ma tran 2: Co doi xung |

Answer:

n1 = int(input())

ma\_tran\_1 = []

for i in range(n1):

ma\_tran\_1.append([int(x) for x in input().split()])

n2 = int(input())

ma\_tran\_2 = []

for i in range(n2):

ma\_tran\_2.append([int(x) for x in input().split()])

def kiem\_tra\_doi\_xung(ma\_tran) -> bool:

for i in range(len(ma\_tran)):

for j in range(i+1):

if ma\_tran[j][i] == ma\_tran[i][j]:

continue

else:

return False

return True

print("Ma tran 1: {} doi xung".format(

"Co" if kiem\_tra\_doi\_xung(ma\_tran\_1) else "Khong"))

print("Ma tran 2: {} doi xung".format(

"Co" if kiem\_tra\_doi\_xung(ma\_tran\_2) else "Khong"))

Hai số a và b gọi là một cặp số thân thiết khi tổng các ước của số này (không tính nó) bằng số kia. Ví dụ, 220 và 284 là một cặp số thân thiết, do tổng các ước của 220

1+2+4+5+10+11+20+22+44+55+110=284,1+2+4+5+10+11+20+22+44+55+110=284,

và tổng các ước của 284

1+2+4+71+142=220.1+2+4+71+142=220.

Hãy viết chương trình kiểm tra xem hai số có phải là cặp số thân thiết hay không?

* Đầu vào: Hai số a b cách nhau bởi dấu cách
* Đầu ra: 'true' nếu a và b là một cặp số thân thiết, 'false' nếu ngược lại.
* Ràng buộc: a, b là hai số nguyên dương

**For example:**

| **Input** | **Result** |
| --- | --- |
| 220 284 | true |
| 25 19 | false |

Answer:

a, b = [int(x) for x in input().split()]

def tinh\_tong\_uoc(a):

cac\_uoc = [x for x in range(1, int(a/2) + 1) if a % x == 0]

return sum(cac\_uoc)

tong\_uoc\_a = tinh\_tong\_uoc(a)

tong\_uoc\_b = tinh\_tong\_uoc(b)

if tong\_uoc\_a == b and tong\_uoc\_b == a:

print('true')

else:

print('false')

Phần tử đặc biệt trong một ma trận là phần tử xuất hiện trên tất cả các hàng và tất cả các cột. Hãy viết chương trình tìm các phần tử đặc biệt trong một ma trận.

* Đầu vào:
  + Dòng đầu tiên gồm hai số n, m cách nhau bởi dấu cách là số hàng và số cột của ma trận
  + n dòng tiếp theo, mỗi dòng chứa m số là các phần tử trong một hàng, cách nhau bởi dấu cách
* Đầu ra: các phần tử đặc biệt tìm được, cách nhau bởi dấu cách
* Ràng buộc: n, m là các số nguyên dương.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 3 4  0 1 2 3  1 2 4 1  2 6 1 2 | 1 2 |

Answer:

n, m = [int(x) for x in input().split()]

mang\_2\_chieu = []

for i in range(n):

mang\_2\_chieu.append([int(x) for x in input().split()])

def cac\_pt\_trong\_cac\_hang(mang\_2\_chieu):

cac\_set\_pt = []

for i in range(n):

set\_pt = set()

for j in range(m):

set\_pt.add(mang\_2\_chieu[i][j])

cac\_set\_pt.append(set\_pt)

return cac\_set\_pt

def cac\_pt\_trong\_cac\_cot(mang\_2\_chieu):

cac\_set\_pt = []

for i in range(m):

set\_pt = set()

for j in range(n):

set\_pt.add(mang\_2\_chieu[j][i])

cac\_set\_pt.append(set\_pt)

return cac\_set\_pt

def tim\_pt\_chung(cac\_set\_pt):

ket\_qua = cac\_set\_pt[0]

for i in range(1, len(cac\_set\_pt) - 1):

ket\_qua = ket\_qua & cac\_set\_pt[i]

return ket\_qua

pt\_db\_hang = tim\_pt\_chung(cac\_pt\_trong\_cac\_hang(mang\_2\_chieu))

pt\_db\_cot = tim\_pt\_chung(cac\_pt\_trong\_cac\_cot(mang\_2\_chieu))

pt\_db = pt\_db\_hang & pt\_db\_cot

print(' '.join([str(x) for x in pt\_db]))

Viết chương trình nhập vào một mảng các số nguyên. Liệt kê các phần tử lớn hơn hoặc bằng trung bình cộng của các phần tử trong mảng.

Input: Dòng thứ nhất nhập vào số nguyên N, 1 <= N <= 1000

Dòng thứ hai nhập vào mảng gồm N số nguyên cách nhau dấu cách.

Output:

+ Dòng thứ nhất in ra các phần tử tìm được, cách nhau dấu cách.

+ Dòng thứ hai in ra thông báo "So phan tu thoa man: X". Trong đó, X là số lượng các phần tử tìm được theo yêu cầu bài toán.

Constrains: Các phần tử trong mảng là các số nguyên.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 32  24 14 1 3 5 19 23 17 15 8 0 22 4 2 4 10 20 11 13 4 10 4 24 6 7 10 19 15 21 1 3 16 | 24 14 19 23 17 15 22 20 13 24 19 15 21 16  So phan tu thoa man: 14 |

Answer:

n = int(input())

mang = [int(x) for x in input().split()]

tbc = sum(mang)/len(mang)

count = 0

for so in mang:

if so >= tbc:

count += 1

print(so, end=' ')

print()

print('So phan tu thoa man: {}'.format(count))

Nhập vào 6 điểm (x1, y1), (x2, y2), (x3, y3), (x4, y4), (x5, y5), (x6, y6). Với xi, yi là các số thực. Hãy kiểm tra 3 điểm (x1, y1), (x2, y2), (x3, y3) và 3 điểm (x4, y4), (x5, y5), (x6, y6) có thẳng hàng hay không?

Input:

Dòng 1: x1 y1 x2 y2 x3 y3 cách nhau dấu cách

Dòng 1: x4 y4 x5 y5 x6 y6 cách nhau dấu cách

Output:

Nếu 3 điểm thẳng hàng thì in ra chữ YES; ngược lại in ra chữ NO

Constraints:

xi, yi kiểu float

**For example:**

| **Input** | **Result** |
| --- | --- |
| 219.000000 892.000000  -664.000000 244.000000  -781.000000 194.000000  585.000000 298.000000  585.000000 298.000000  585.000000 298.000000 | NO YES |

Answer:

class toado():

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

self.x=0

self.y=0

def nhap(self):

self.x,self.y=map(float,input().split())

def ktra(a,b,c):

if (c.y-a.y)\*(b.x-a.x)==(b.y-a.y)\*(c.x-a.x):

print("YES",end=" ")

else:

print("NO",end=" ")

A=toado()

A.nhap()

B=toado()

B.nhap()

C=toado()

C.nhap()

D=toado()

D.nhap()

E=toado()

E.nhap()

F=toado()

F.nhap()

ktra(A,B,C)

ktra(D,E,F)

Viết chương trình nhập vào 2 số nguyên a, b. In ra các số nguyên tố trong đoạn [a, b] hoặc [b, a]. Nếu trong đoạn này không có số nguyên tố thì in ra thông báo "Khong co". Biết số nguyên tố là số nguyên dương >=2 và chỉ chia hết cho 1 và chính nó.

Input: a, b là số nguyên cách nhau dấu cách.

Output: các số nguyên tố tìm được viết trên một dòng, cách nhau dấu cách. Hoặc thông báo "Khong co".

Constrains: các số kiểu int

Ví dụ 1:

+ Input

a= -30, b=-9

+ Output

Khong co

Ví dụ 2:

+ Input

a= 0, b=9

+ Output

2 3 5 7

Ví dụ 2:

+ Input

a= 9, b=0

+ Output

7 5 3 2

**For example:**

| **Input** | **Result** |
| --- | --- |
| -9 -33 | Khong co |
| 28 18 | 23 19 |

Answer:

def nguyento(a):

count=0

for i in range(2,a):

if a%i==0:

count+=1

if count==0:

return 0

else:

return 1

a,b=map(int,input().split())

if a<b:

nt=[]

for i in range(a,b+1):

if i>=2:

if nguyento(i)==0:

nt.append(i)

if len(nt)==0:

print("Khong co")

else:

for i in nt:

print("%d"%i,end=" ")

if a>b:

nt=[]

for i in range(b,a+1):

if i>=2:

if nguyento(i)==0:

nt.append(i)

if len(nt)==0:

print("Khong co")

else:

nt.reverse()

for i in nt:

print("%d"%i,end=" ")

Viết chương trình tính tổng 1 dãy các số nguyên.

Biết rằng số nguyên đầu tiên nhập vào là số lượng các giá trị, còn lại được nhập vào.

Chương trình cần đọc vào 1 giá trị tương ứng với mỗi lệnh scanf.

Dữ liệu nhập vào có thể là

5 100 200 300 400 500

trong đó 5 cho biết là có 5 số sau nó cần được tính tổng.

Dữ liệu in ra phải là:

1500

// Gợi ý giải thuật không dùng kiểu dl mảng

Nhập( n );

sum = 0;

Lặp lại n lần {

Nhập( x );

sum += x;

}

In( sum );

**Input Format**

* Số nguyên đầu tiên chỉ số lượng số cần tính tổng.
* Các số nguyên tiếp theo là các số cần được tính tổng.
* Hai số cách nhau 1 kí tự trắng.

**Constraints**

* Số lượng số cần tính tổng lớn hơn hoặc bằng 0.
* Số lượng các số nhập vào phải bằng số đầu tiên.

**Output Format**

* Số nguyên chỉ tổng các số.

**For example:**

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 5-so | 5 100 200 300 400 500 | 1500 |

Answer:

a=[]

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

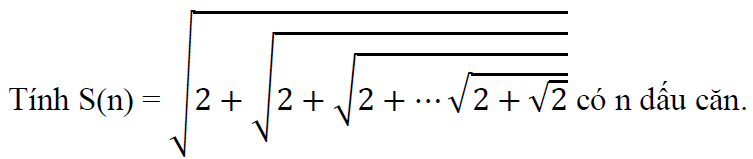
S=0

for i in range(1,a[0]+1):

S+=a[i]

print(S)

Viết chương trình nhập vào số nguyên n.  Tính tổng sau



Input:

n  là số nguyên.

Ví dụ: 5 hoặc -8

Output:

Tổng S(n) như trên với độ chính xác 2 chữ số. Hoặc thông báo "Error." nếu n là số âm.

Constrains:

n kiểu nguyên, s là số thực

**For example:**

| **Input** | **Result** |
| --- | --- |
| 9 | 2.00 |

Answer:

import math

n=int(input())

if n<=0:

print("Error.")

else:

S=0

for i in range(n):

S=math.sqrt(2+S)

print('%.2f'%S)