Python Experiments

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

Input Format

A single line containing a positive integer, .

Constraints

Output Format

Print Weird if the number is weird. Otherwise, print Not Weird.

Coding:

```
import math

import os

import random

import re

import sys
```

```
if __name__ == '__main__':
    n = int(input().strip())


if n%2 ==1 :
    print("Weird")
else:
    if n in range(2,6):
        print("Not Weird")
    elif n in range(6,21):
        print("Weird")
    else:
        print("Not Weird")
```

2). The provided code stub reads two integers from STDIN, and . Add code to print three lines where:

The first line contains the sum of the two numbers.

The second line contains the difference of the two numbers (first - second).

The third line contains the product of the two numbers.

Example

A=3

B=5

Print the following:

8

-2

15

Input Format

The first line contains the first integer, .

The second line contains the second integer, .

Constraints

1 greater than a greater than 10 power 10

1 greater than a greater than 10 power 10

Output Format

Print the three lines as explained above.

Sample Input 0

3

2

Sample Output 0

5

1

6

Explanation 0

Coding:

Print(a+b)

Print(a-b)

Print(a*b)

3). The provided code stub reads two integers, and , from STDIN.

Add logic to print two lines. The first line should contain the result of integer division, //
. The second line should contain the result of float division, / .

No rounding or formatting is necessary.

Example

The result of the integer division .

The result of the float division is .

Print:

0

0.6

Input Format

The first line contains the first integer, .

The second line contains the second integer, .

Output Format

Print the two lines as described above.

Sample Input 0

4

3

Sample Output 0

1

1.33333333333

Coding:

Print(a//b)

Print(a/b)

4). Task

The provided code stub reads and integer, , from STDIN. For all non-negative integers , print .

Example

The list of non-negative integers that are less than is . Print the square of each number on a separate line.

0

1

4

Input Format

The first and only line contains the integer, .

Constraints

Output Format

Print lines, one corresponding to each .

Sample Input 0

5

Sample Output 0

0

1

4

9

16

Coding:

n= int (input())

```
for i in range(n):
    print(i*i)
```

5). An extra day is added to the calendar almost every four years as February 29, and the day is called a leap day. It corrects the calendar for the fact that our planet takes approximately 365.25 days to orbit the sun. A leap year contains a leap day.

In the Gregorian calendar, three conditions are used to identify leap years:

The year can be evenly divided by 4, is a leap year, unless:

The year can be evenly divided by 100, it is NOT a leap year, unless:

The year is also evenly divisible by 400. Then it is a leap year.

This means that in the Gregorian calendar, the years 2000 and 2400 are leap years, while 1800, 1900, 2100, 2200, 2300 and 2500 are NOT leap years. Source

Task

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. It is only necessary to complete the is_leap function.

Input Format

Read , the year to test.

Constraints

Output Format

The function must return a Boolean value (True/False). Output is handled by the provided code stub.

Sample Input 0

1990

Sample Output 0

False

Explanation 0

1990 is not a multiple of 4 hence it's not a leap year.

Coding:

```
def is_leap(year):
    leap = False

    # Write your logic here

    if year % 4 ==0:
```

```
        if year % 100 == 0 :
            if year % 400 == 0 :
                return True
            else:
                return False
        else:
            return True
    else:
        return False

    return leap

year = int(input())
print(is_leap(year))
```

6). The included code stub will read an integer, , from STDIN.

Without using any string methods, try to print the following:

Note that "" represents the consecutive values in between.

Example

N=5

Print the string 12345 .

Input Format

The first line contains an n integer .

Constraints

Output Format

Print the list of integers from 1 through n as a string, without spaces.

Sample Input 0

3

Sample Output 0

123

Coding

```
if __name__ == '__main__':
    n = int(input())

    for i in range (1, n+1):
        print(i,end='')
```

problem:

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

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

Example

$records = [[\text{"chi"}, 20.0], [\text{"beta"}, 50.0], [\text{"alpha"}, 50.0]]$

The ordered list of scores is $[20.0, 50.0]$, so the second lowest score is 50.0. There are two students with that score: $[\text{"beta"}, \text{"alpha"}]$. Ordered alphabetically, the names are printed as:

```
alpha
beta
```

Input Format

The first line contains an integer, N, the number of students.

The $2N$ subsequent lines describe each student over 2 lines.

- The first line contains a student's name.

- The second line contains their grade.

Constraints

- $2 \leq N \leq 5$
- There will always be one or more students having the second lowest grade.

Output Format

Print the name(s) of any student(s) having the second lowest grade in. If there are multiple students, order their names alphabetically

Output Format

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

Sample Input 0

```
5
Harry
37.21
Berry
37.21
Tina
37.2
Akriti
41
Harsh
39
```

Sample Output 0

```
Berry
Harry
```

Explanation 0

There are 5 students in this class whose names and grades are assembled to build the following list:

```
python students = [['Harry', 37.21], ['Berry', 37.21], ['Tina', 37.2], ['Akriti', 41], ['Harsh', 39]]
```

The lowest grade of 37.2 belongs to Tina. The second lowest grade of 37.21 belongs to both Harry and Berry, so we order their names alphabetically and print each name on a new line.

Solution:

```
marksheet = []
scoresheet=[]

if __name__ == '__main__':
    for _ in range(int(input())):
        name = input()
        score = float(input())

        marksheet +=[[name, score]]
        scoresheet +=[score]

    x= sorted(set(scoresheet))[1]

    for n,s in sorted(marksheet):
        if s==x:
            print(n)
```

Problem:

Problem | Submissions | Leaderboard | Discussions | Editorial | Tutorial

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.

Example

marks key:value pairs are

'alpha': [20, 30, 40]

'beta': [30, 50, 70]

query_name = 'beta'

The **query_name** is 'beta'. beta's average score is $(30 + 50 + 70)/3 = 50.0$.

Input Format

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.

Constraints

- $2 \leq n \leq 10$
- $0 \leq marks[i] \leq 100$
- length of marks arrays $= 3$

Output Format

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

Sample Input 0

Output Format

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

Sample Input 0

```
3
Krishna 67 68 69
Arjun 70 98 63
Malika 52 56 60
Malika
```

Sample Output 0

```
56.00
```

Explanation 0

Marks for Malika are $\{52, 56, 60\}$ whose average is $\frac{52+56+60}{3} \Rightarrow 56$

Sample Input 1

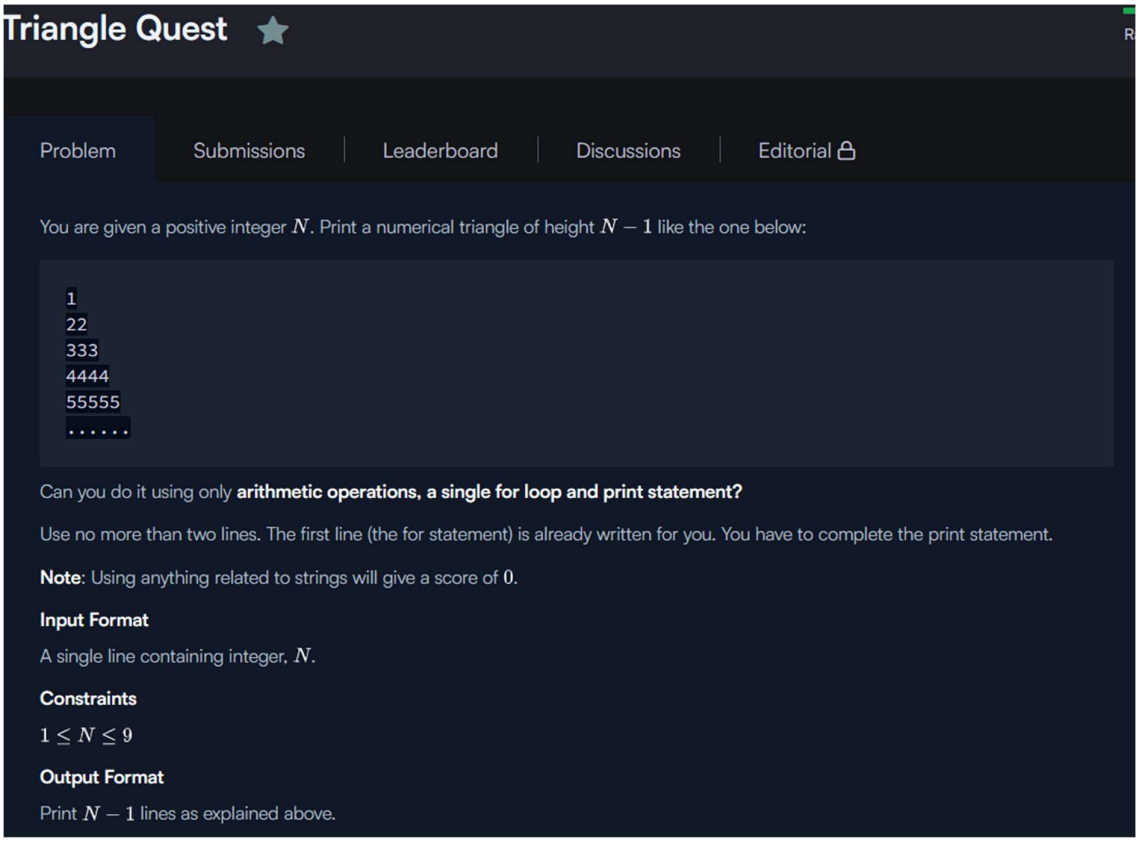
```
2
Harsh 25 26.5 28
Anurag 26 28 30
Harsh
```

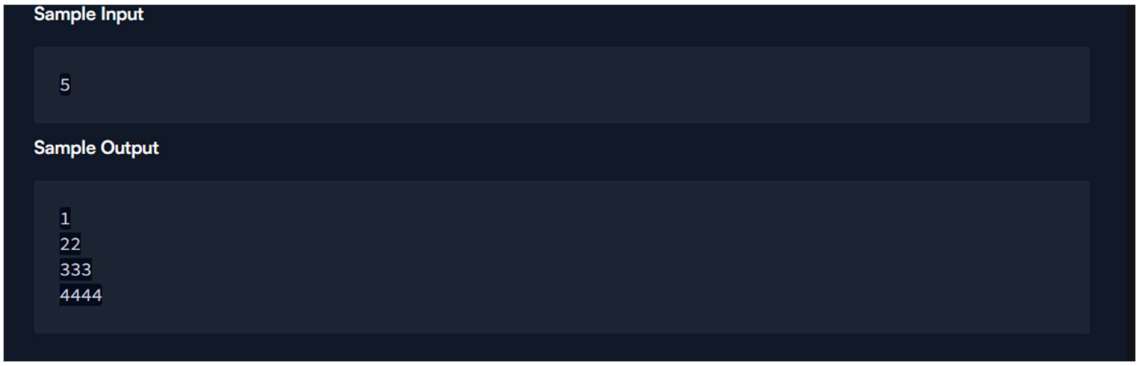
Sample Output 1

Solution:

```
from functools import reduce
if __name__ == '__main__':
    n = int(input())
    student_marks = {}
    for _ in range(n):
        name, *line = input().split()
        scores = list(map(float, line))
        student_marks[name] = scores
    query_name = input()
    a=student_marks[query_name]
    sum=reduce(lambda x,y,:x+y,a )
    average = sum/len(a)
    print(f'{average:0.2f}')
```

Problem:





Solution:

for I in range(1, int(input())): print(int(I*10**I/9))

Explanation:

This line of code is a concise way to generate a series of numbers where each number is formed by repeating the digit **i** a certain number of times. Let's break it down:

1. **for i in range(1, int(input())):** - This part of the code prompts the user to input a number (**input()**), converts it into an integer (**int(input())**), and then creates a loop that iterates from 1 up to (but not including) that input number.

2. **print(int(i * 10 ** i / 9))** - This line calculates the value to be printed within each iteration of the loop.
    - **i**: This represents the current value of the loop variable, ranging from 1 to the user's input number.
    - **10 ** i**: This raises 10 to the power of **i**, essentially appending **i** zeros to the number 1.
    - **i * 10 ** i**: This multiplies **i** with the result of **10 ** i**, creating a number with **i** repeated digits of **i**.
    - **i * 10 ** i / 9**: Dividing by 9 ensures that each number generated follows a pattern where the digit **i** repeats in the resulting number. This is a mathematical trick to generate numbers like 1, 22, 333, 4444, and so on.
    - **int(...)**: Finally, the result is converted to an integer before being printed.

Let's go through an example: if the user inputs 5, the loop will iterate over values of **i** from 1 to 4. For each value of **i**, it will calculate **i * 10 ** i / 9**:

- For **i = 1**: 1×101/9=111×101/9=11
- For **i = 2**: 2×102/9=222×102/9=22
- For **i = 3**: 3×103/9=3333×103/9=333
- For **i = 4**: 4×104/9=44444×104/9=4444

And these numbers will be printed respectively.

Problems:

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

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

Initialize your list and read in the value of n followed by n lines of commands where each command will be of the 7 types listed above. Iterate through each command in order and perform the corresponding operation on your list.

Example

$N = 4$

append 1

append 2

insert 3 1

print

Example

$N = 4$

append 1

append 2

insert 3 1

print

- append 1: Append 1 to the list, $arr = [1]$.
- append 2: Append 2 to the list, $arr = [1, 2]$.
- insert 3 1: Insert 3 at index 1, $arr = [1, 3, 2]$.
- print: Print the array.

  Output:

```
[1, 3, 2]
```

Input Format

The first line contains an integer, n, denoting the number of commands.

Each line i of the n subsequent lines contains one of the commands described above.

Constraints

- The elements added to the list must be integers.

Output Format

For each command of type `print`, print the list on a new line.

Sample Input 0

```
12
insert 0 5
insert 1 10
insert 0 6
print
remove 6
append 9
append 1
sort
print
pop
reverse
print
```

Sample Output 0

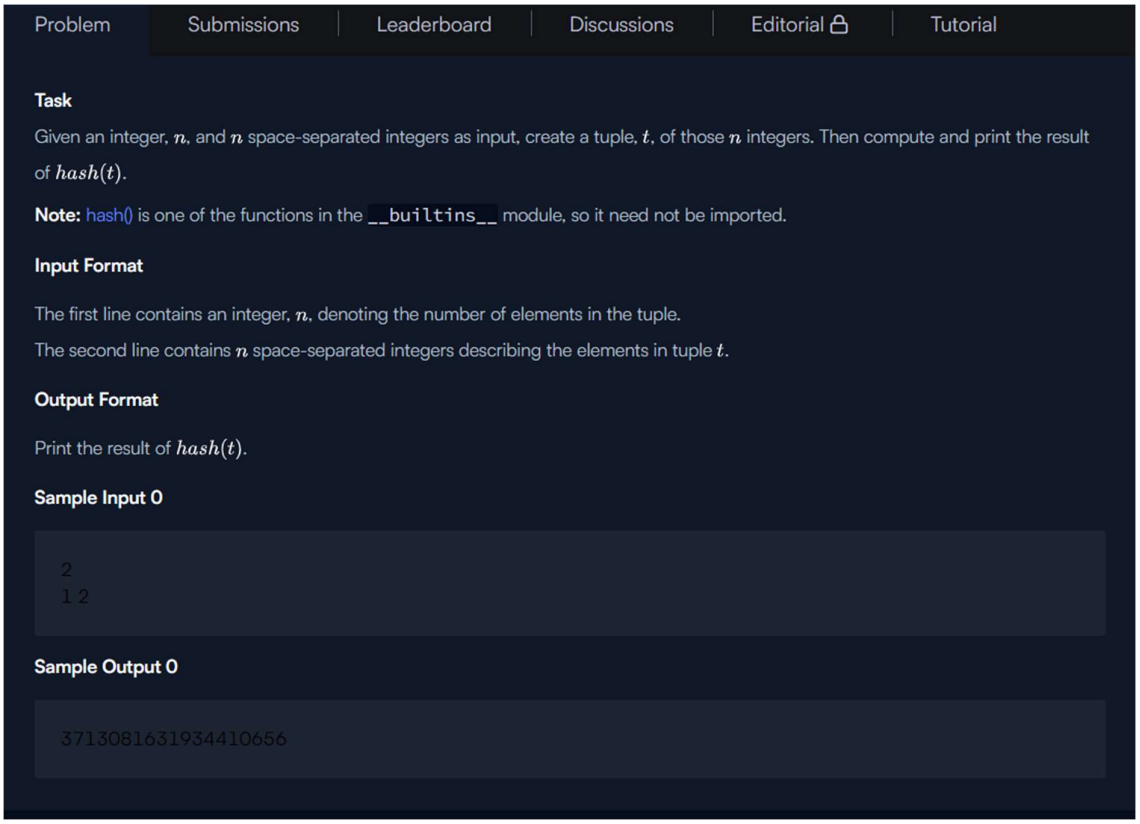
```
[6, 5, 10]
[1, 5, 9, 10]
[9, 5, 1]
```

Solution:

```
if __name__ == '__main__':
    N = int(input())  # Number of commands
    my_list = []  # Initialize an empty list

    # Loop through each command
    for _ in range(N):
        command = input().split()  # Split the command into words

        # If the command is 'insert'
        if command[0] == 'insert':
            i = int(command[1])
```

```
            e = int(command[2])
            my_list.insert(i, e)  # Insert integer e at position
i
        # If the command is 'print'
        elif command[0] == 'print':
            print(my_list)  # Print the list
        # If the command is 'remove'
        elif command[0] == 'remove':
            e = int(command[1])
            my_list.remove(e)  # Delete the first occurrence of i
nteger e
        # If the command is 'append'
        elif command[0] == 'append':
            e = int(command[1])
            my_list.append(e)  # Insert integer e at the end of t
he list
        # If the command is 'sort'
        elif command[0] == 'sort':
            my_list.sort()  # Sort the list
        # If the command is 'pop'
        elif command[0] == 'pop':
            my_list.pop()  # Pop the last element from the list
        # If the command is 'reverse'
        elif command[0] == 'reverse':
            my_list.reverse()  # Reverse the list
```

Problem:



Solution:

```
if __name__ == '__main__':
    n = int(input())
    integer_list = map(int, input().split())

    t = tuple(integer_list)
    print(hash(t))
```

Problem:

Dot and Cross

dot

The dot tool returns the dot product of two arrays.

```
import numpy

A = numpy.array([ 1, 2 ])
B = numpy.array([ 3, 4 ])

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

cross

The cross tool returns the cross product of two arrays.

```
import numpy

A = numpy.array([ 1, 2 ])
B = numpy.array([ 3, 4 ])

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

Task

You are given two arrays A and B. Both have dimensions of $N \times N$.

Your task is to compute their matrix product.

Input Format

The first line contains the integer N.

The next N lines contains N space separated integers of array A.

The following N lines contains N space separated integers of array B.

Output Format

Print the matrix multiplication of A and B.

Sample Input

```
2
1 2
3 4
1 2
3 4
```

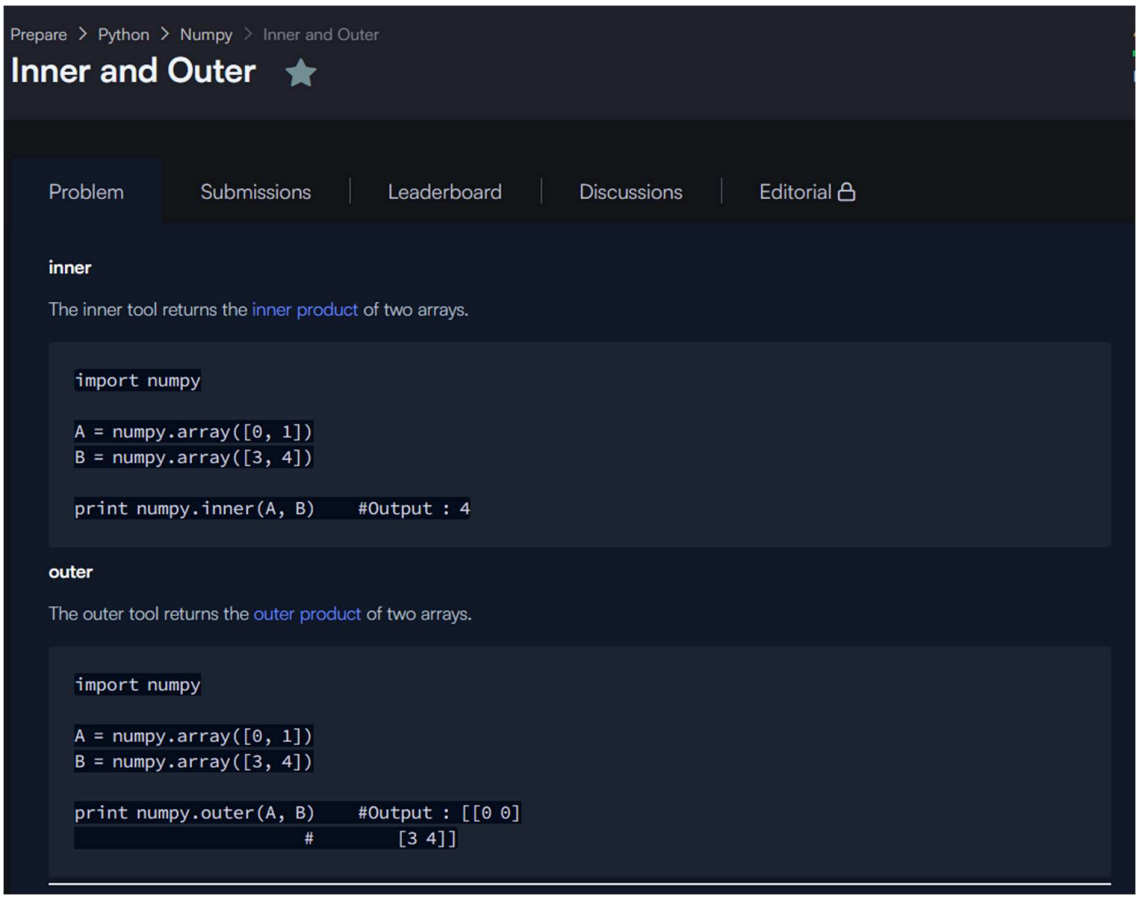
Sample Output

```
[[ 7 10]
 [15 22]]
```

Solution:

```
Import numpy as np

n=int(input())

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

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

Result= np.dot(A,B)

Print(Result)
```

Problem:



Task

You are given two arrays: A and B.

Your task is to compute their inner and outer product.

Input Format

The first line contains the space separated elements of array A.

The second line contains the space separated elements of array B.

Output Format

First, print the inner product.

Second, print the outer product.

Sample Input

```
0 1
2 3
```

Sample Output

```
3
[[0 0]
 [2 3]]
```

Solution:

Import numpy as np

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

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

Print(np.inner(A,B))

Print(np.outer(A,B))

Problems:

Polynomials

poly

The poly tool returns the coefficients of a polynomial with the given sequence of roots.

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

roots

The roots tool returns the roots of a polynomial with the given coefficients.

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

polyint

The polyint tool returns an antiderivative (indefinite integral) of a polynomial.

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

polyder

The polyder tool returns the derivative of the specified order of a polynomial.

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

polyval

The polyval tool evaluates the polynomial at specific value.

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

polyfit

The polyfit tool fits a polynomial of a specified order to a set of data using a least-squares approach.

```
print numpy.polyfit([0,1,-1, 2, -2], [0,1,1, 4, 4], 2)
#Output : [  1.00000000e+00   0.00000000e+00  -3.97205465e-16]
```

The functions polyadd, polysub, polymul, and polydiv also handle proper addition, subtraction, multiplication, and division of polynomial coefficients, respectively.

Task

You are given the coefficients of a polynomial P.

Your task is to find the value of P at point x.

Task

You are given the coefficients of a polynomial P.

Your task is to find the value of P at point x.

Input Format

The first line contains the space separated value of the coefficients in P.

The second line contains the value of x.

Output Format

Print the desired value.

Sample Input

```
1.1 2 3
0
```

Sample Output

```
3.0
```

Solution:

```
import numpy as np

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

x=float(input())

y = np.polyval(P,x)

print(y)
```

Problem:

Linear Algebra

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

linalg.det

The linalg.det tool computes the determinant of an array.

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

linalg.eig

The linalg.eig computes the eigenvalues and right eigenvectors of a square array.

```
vals, vecs = numpy.linalg.eig([[1 , 2], [2, 1]])
print vals                              #Output : [ 3. -1.]
print vecs                              #Output : [[ 0.70710678 -0.70710678]
                                        #         [ 0.70710678  0.70710678]]
```

linalg.inv

The linalg.inv tool computes the (multiplicative) inverse of a matrix.

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

Other routines can be found here

Task

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

Input Format

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

Output Format

Print the determinant of A.

Sample Input

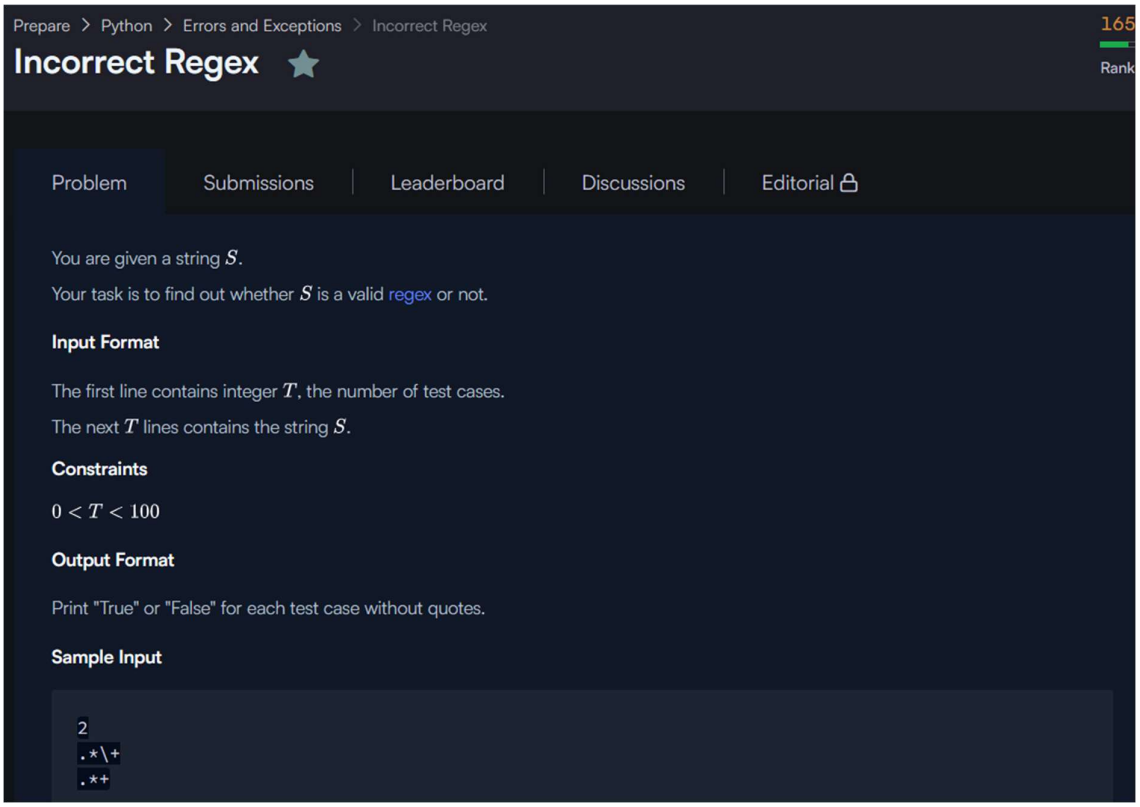
```
2
1.1 1.1
1.1 1.1
```

Sample Output

Solution:

```
import numpy as np

n=int(input())
A=[np.array(input().split(), dtype=float) for _ in range(n)]

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

Problem:



Solution:

```
import re

def is_valid_regex(pattern):
    try:
        re.compile(pattern)
        return True
    except re.error:
        return False

if __name__ == '__main__':
    n = int(input())  # Number of test cases
    for _ in range(n):
        pattern = input()  # Input pattern
        print(is_valid_regex(pattern))
```

Problem:

Problem | Submissions | Leaderboard | Discussions

In this challenge, the task is to debug the existing code to successfully execute all provided test files.

Python supports a useful concept of default argument values. For each keyword argument of a function, we can assign a default value which is going to be used as the value of said argument if the function is called without it. For example, consider the following increment function:

```
def increment_by(n, increment=1):
    return n + increment
```

The functions works like this:

```
>>> increment_by(6, 2)
7
>>> increment_by(4)
5
>>>
```

Debug the given function `print_from_stream` using the default value of one of its arguments.

The function has the following signature:

```
def print_from_stream(n, stream)
```

This function should print the first n values returned by `get_next()` method of `stream` object provided as an argument. Each of these values should be printed in a separate line.

This function should print the first n values returned by `get_next()` method of `stream` object provided as an argument. Each of these values should be printed in a separate line.

Whenever the function is called without the `stream` argument, it should use an instance of `EvenStream` class defined in the code stubs below as the value of `stream`.

Your function will be tested on several cases by the locked template code.

Input Format

The input is read by the provided locked code template. In the first line, there is a single integer q denoting the number of queries. Each of the following q lines contains a `stream_name` followed by integer n, and it corresponds to a single test for your function.

Constraints

- $1 \leq q \leq 100$
- $1 \leq n \leq 10$

Output Format

The output is produced by the provided and locked code template. For each of the queries `(stream_name, n)`, if the `stream_name` is `even` then `print_from_stream(n)` is called. Otherwise, if the `stream_name` is `odd`, then `print_from_stream(n, OddStream())` is called.

Sample Input 0

```
3
odd 2
even 3
odd 5
```

Sample Output 0

```
1
3
0
2
4
1
3
5
7
9
```

Explanation 0

There are 3 queries in the sample.

In the first query, the function `print_from_stream(2, OddStream())` is exectuted, which leads to printing values 1 and 3 in separated lines as the first two non-negative odd numbers.

In the second query, the function `print_from_stream(3)` is exectuted, which leads to printing values $2, 4$ and 6 in separated lines as the first three non-negative even numbers.

In the third query, the function `print_from_stream(5, OddStream())` is exectuted, which leads to printing values $1, 3, 5, 7$ and 9 in separated lines as the first five non-negative odd numbers.

Solution:

```
def print_from_stream(n, stream=None):
    if stream is None:
        stream = EvenStream()  # Instantiate EvenStream if stream
 is not provided
    for _ in range(n):
        print(stream.get_next())
```

Problem:
Detect HTML tags , Attributes and Attribute Values

Problem | Submissions | Leaderboard | Discussions | Editorial

You are given an HTML code snippet of N lines.

Your task is to detect and print all the HTML tags, attributes and attribute values.

Print the detected items in the following format:

```
Tag1
Tag2
-> Attribute2[0] > Attribute_value2[0]
-> Attribute2[1] > Attribute_value2[1]
-> Attribute2[2] > Attribute_value2[2]
Tag3
-> Attribute3[0] > Attribute_value3[0]
```

The `->` symbol indicates that the tag contains an attribute. It is immediately followed by the name of the attribute and the attribute value.

The `>` symbol acts as a separator of attributes and attribute values.

If an HTML tag has no attribute then simply print the name of the tag.

Note: Do not detect any HTML tag, attribute or attribute value inside the HTML comment tags (`<!-- Comments -->`). Comments can be multiline.

All attributes have an attribute value.

Input Format

The first line contains an integer N, the number of lines in the HTML code snippet.

Input Format

The first line contains an integer N, the number of lines in the HTML code snippet.

The next N lines contain HTML code.

Constraints

$0 < N < 100$

Output Format

Print the HTML tags, attributes and attribute values in order of their occurrence from top to bottom in the snippet.

Format your answers as explained in the problem statement.

Sample Input

```
9
<head>
<title>HTML</title>
</head>
<object type="application/x-flash"
 data="your-file.swf"
 width="0" height="0">
 <!-- <param name="movie" value="your-file.swf" /> -->
 <param name="quality" value="high"/>
</object>
```

Sample Output

```
head
title
object
-> type > application/x-flash
-> data > your-file.swf
-> width > 0
-> height > 0
param
-> name > quality
-> value > high
```

Explanation

- **head** tag: Print the head tag only because it has no attribute.
- **title** tag: Print the title tag only because it has no attribute.
- **object** tag: Print the object tag. In the next **4** lines, print the attributes type, data, width and height along with their respective values.
- **<!-- Comment -->** tag: Don't detect anything inside it.
- **param** tag: Print the param tag. In the next **2** lines, print the attributes name along with their respective values.

Solution:

```
from html.parser import HTMLParser

class MyHTMLParser(HTMLParser):
    def handle_starttag(self,tag,attrs):
        print(tag)
        for attr in attrs:
            print("-> " + attr[0]+ " > " +attr[1])

parser = MyHTMLParser()
for _ in range(int(input())):
    parser.feed(input())
```

Problem:
Validating UID:

Problem | Submissions | Leaderboard | Discussions | Editorial

ABCXYZ company has up to 100 employees.

The company decides to create a unique identification number (UID) for each of its employees.

The company has assigned you the task of validating all the randomly generated UIDs.

A valid UID must follow the rules below:

- It must contain at least 2 uppercase English alphabet characters.
- It must contain at least 3 digits (0 - 9).
- It should only contain alphanumeric characters (a - z, A - Z & 0 - 9).
- No character should repeat.
- There must be exactly 10 characters in a valid UID.

Input Format

The first line contains an integer T, the number of test cases.

The next T lines contains an employee's UID.

Output Format

For each test case, print 'Valid' if the UID is valid. Otherwise, print 'Invalid', on separate lines. Do not print the quotation marks.

Sample Input

```
2
B1CD102354
B1CDEF2354
```

Sample Output

```
Invalid
Valid
```

Explanation

B1CD102354: 1 is repeating → Invalid

B1CDEF2354: Valid

Solution:

```
import re

def valid_uid(uid):
    # Rule 1: It must contain at least 2 uppercase alphabet chara
cters
    if len(re.findall(r'[A-Z]', uid)) < 2:
        return False

    # Rule 2: It must contain at least 3 digits
```

```
    if len(re.findall(r'\d', uid)) < 3:
        return False

    # Rule 3: Should only contain alphanumeric characters
    if not re.match(r'^[a-zA-Z0-9]*$', uid):
        return False

    # Rule 4: No character should repeat
    if len(set(uid)) != len(uid):
        return False

    # Rule 5: Must be exactly 10 characters long
    if len(uid) != 10:
        return False

    return True

if __name__ == '__main__':
    n = int(input())

    for _ in range(n):
        uid = input().strip()
        if valid_uid(uid):
            print("Valid")
        else:
            print("Invalid")
```

Problem:
Validating Credit Card

Problem | Submissions | Leaderboard | Discussions | Editorial

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

A valid credit card from **ABCD Bank** has the following characteristics:

- ► It must start with a 4, 5 or 6.
- ► It must contain exactly 16 digits.
- ► It must only consist of digits (0-9).
- ► It may have digits in groups of 4, separated by one hyphen **"-"**.
- ► It must `NOT` use any other separator like ' ', '_', etc.
- ► It must `NOT` have 4 or more consecutive repeated digits.

Examples:

Valid Credit Card Numbers

```
4253625879615786
4424424424442444
5122-2368-7954-3214
```

Invalid Credit Card Numbers

```
42536258796157867      #17 digits in card number → Invalid
4424444424442444       #Consecutive digits are repeating 4 or more times → Invalid
5122-2368-7954 - 3214  #Separators other than '-' are used → Invalid
44244x4424442444       #Contains non digit characters → Invalid
0525362587961578       #Doesn't start with 4, 5 or 6 → Invalid
```

Input Format

The first line of input contains an integer N.

The next N lines contain credit card numbers.

Constraints

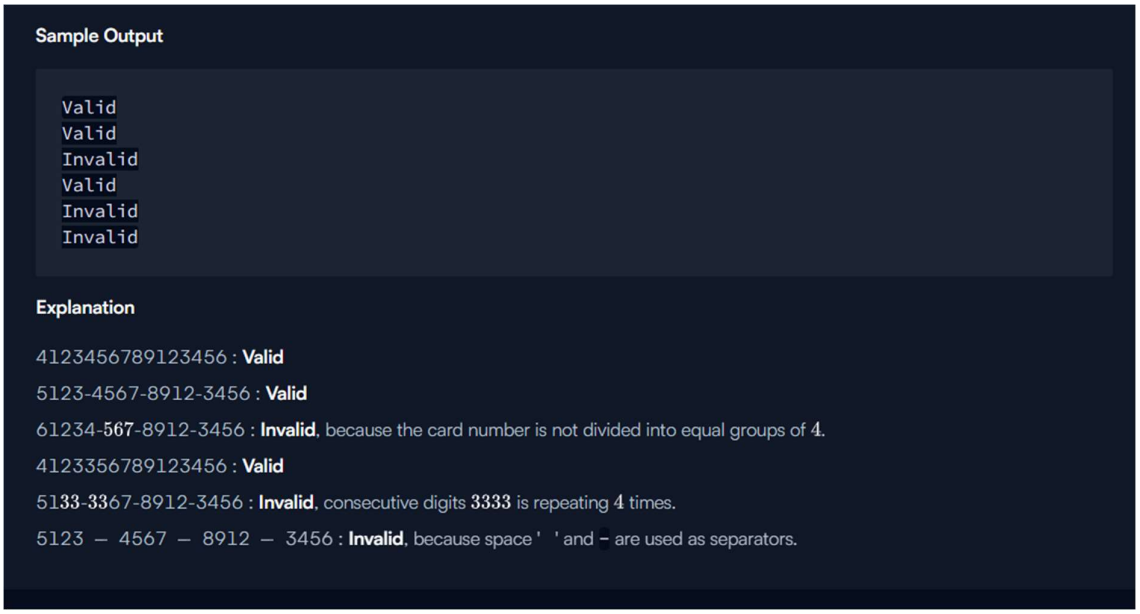
$0 < N < 100$

Output Format

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

Sample Input

```
6
4123456789123456
5123-4567-8912-3456
61234-567-8912-3456
4123356789123456
5133-3367-8912-3456
5123 - 3567 - 8912 - 3456
```



Sample Output

```
Valid
Valid
Invalid
Valid
Invalid
Invalid
```

Explanation

4123456789123456 : **Valid**

5123-4567-8912-3456 : **Valid**

61234-567-8912-3456 : **Invalid**, because the card number is not divided into equal groups of 4.

4123356789123456 : **Valid**

5133-3367-8912-3456 : **Invalid**, consecutive digits 3333 is repeating 4 times.

5123 - 4567 - 8912 - 3456 : **Invalid**, because space ' ' and - are used as separators.

Solution:

```
import re

for _ in range(int(input())):
    S = input()
    t = re.search(r"^[456]\d{15}$|^[456]\d{3}-\d{4}-\d{4}-
\d{4}$", S)
    if t:
        if re.search(r"(\d)(?:-?\1){3}", S.replace("-", "")):
            print("Invalid")
        else:
            print("Valid")
    else:
        print("Invalid")
```

Problem:

Validating Postal Cards

Problem | Submissions | Leaderboard | Discussions | Editorial

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

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

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

For example:

```
121426 # Here, 1 is an alternating repetitive digit.
523563 # Here, NO digit is an alternating repetitive digit.
552523 # Here, both 2 and 5 are alternating repetitive digits.
```

Your task is to provide two regular expressions `regex_integer_in_range` and `regex_alternating_repetitive_digit_pair`. Where:

`regex_integer_in_range` should match only integers range from 100000 to 999999 inclusive

`regex_alternating_repetitive_digit_pair` should find alternating repetitive digits pairs in a given string.

Both these regular expressions will be used by the provided code template to check if the input string P is a valid postal code using the following expression:

```
(bool(re.match(regex_integer_in_range, P))
and len(re.findall(regex_alternating_repetitive_digit_pair, P)) < 2)
```

Input Format

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

Output Format

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

Sample Input 0

```
110000
```

Sample Output 0

```
False
```

Explanation 0

1 1 0**0**0**0** : (0, 0) and (**0**, **0**) are two alternating digit pairs. Hence, it is an invalid postal code.

Note:

A score of 0 will be awarded for using '`if`' conditions in your code.

You have to pass all the testcases to get a positive score.

Solution:

```
regex_integer_in_range = r"^[1-9][0-9]{5}$" # Do not delete 'r'.
```

```
regex_alternating_repetitive_digit_pair = r"(\d)(?=\d\1)"    # Do
not delete 'r'.


import re
P = input()

print (bool(re.match(regex_integer_in_range, P))
and len(re.findall(regex_alternating_repetitive_digit_pair, P)) <
 2)
```

Hackkerrank Questions Certifications:

2. Python: String Transformation

There is a sentence that consists of space-separated strings of upper and lower case English letters. Transform each string according to the given algorithm and return the new sentence.

Each string should be modified as follows:

- The first character of the string remains unchanged
- For each subsequent character, say x, consider a letter preceding it, say y:
  - If y precedes x in the English alphabet, transform x to uppercase
  - If x precedes y in the English alphabet, transform x to lowercase
  - If x and y are equal, the letter remains unchanged

Example

sentence = "coOL dog"

The first letters of both words remain unchanged. Then, for the word "coOL" the first "o" is made uppercase because the letter preceding it, "c", comes earlier in the alphabet. Next, the case of the second "O" is unchanged because the letter preceding is also "o", and finally the "L" is made lowercase because the letter preceding it, "O", comes later in the alphabet. The second word, "dOg", is transformed according to the same rules. Return the resulting sentence 'cOOl dOg'.

Function Description

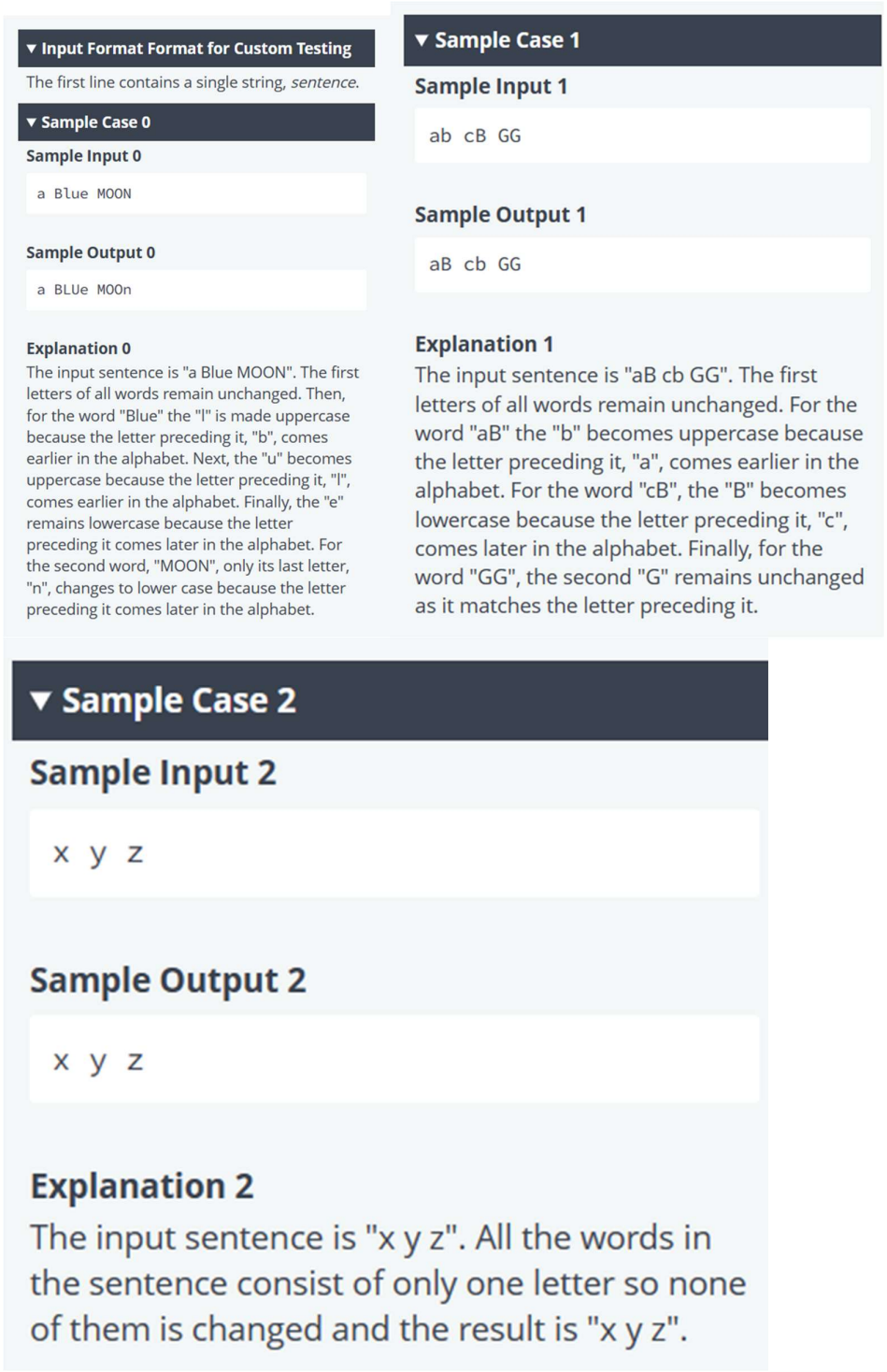
Complete the function *transformSentence* in the editor below. The function must return a string representing the resulting sentence.

transformSentence has the following parameter(s):

string sentence: the input sentence

Constraints

- 1 ≤ length of *sentence* ≤ 100
- It is guaranteed that the sentence consists of upper and lower case English letters and spaces.
- Each *sentence* starts with a letter, ends with a letter, and no two consecutive characters are spaces.



▼ Input Format Format for Custom Testing

The first line contains a single string, *sentence*.

▼ Sample Case 0

Sample Input 0

```
a Blue MOON
```

Sample Output 0

```
a BLUe MOOn
```

Explanation 0

The input sentence is "a Blue MOON". The first letters of all words remain unchanged. Then, for the word "Blue" the "l" is made uppercase because the letter preceding it, "b", comes earlier in the alphabet. Next, the "u" becomes uppercase because the letter preceding it, "l", comes earlier in the alphabet. Finally, the "e" remains lowercase because the letter preceding it comes later in the alphabet. For the second word, "MOON", only its last letter, "n", changes to lower case because the letter preceding it comes later in the alphabet.

▼ Sample Case 1

Sample Input 1

```
ab cB GG
```

Sample Output 1

```
aB cb GG
```

Explanation 1

The input sentence is "aB cb GG". The first letters of all words remain unchanged. For the word "aB" the "b" becomes uppercase because the letter preceding it, "a", comes earlier in the alphabet. For the word "cB", the "B" becomes lowercase because the letter preceding it, "c", comes later in the alphabet. Finally, for the word "GG", the second "G" remains unchanged as it matches the letter preceding it.

▼ Sample Case 2

Sample Input 2

```
x y z
```

Sample Output 2

```
x y z
```

Explanation 2

The input sentence is "x y z". All the words in the sentence consist of only one letter so none of them is changed and the result is "x y z".

Solution:

```
def transformSentence(sentence):
    # Write your code here
    sentence.strip()
    a = ''
```

```
b = 0

for x in sentence:
    if (b==0):
        a += x
        b += 1

    elif (x==' '):
        a += x
        b += 1
    else:
        y = sentence[b-1]
        if (y==' '):
            a += x
            b += 1

        elif(y.lower() > x.lower()):
            a += x.lower()
            b += 1

        elif(y.lower() < x.lower()):
            a += x.upper()
            b += 1

        else:
            a += x
            b += 1

return a
```