Given an array of integers, find the sum of its elements.

For example, if the array $ar = [1, 2, 3]$, $1 + 2 + 3 = 6$, so return 6.

Function Description

Complete the simpleArraySum function in the editor below. It must return the sum of the array elements as an integer.

simpleArraySum has the following parameter(s):

- ar: an array of integers

Input Format

The first line contains an integer, n, denoting the size of the array.

The second line contains n space-separated integers representing the array's elements.

Constraints

$0 < n, ar[i] \leq 1000$

Output Format

Print the sum of the array's elements as a single integer.

```python
#!/bin/python3
import math
import os
import random
import re
import sys
def simpleArraySum(ar):
    # Write your code here
    total = 0
    for i in ar:
        total += i
    return total
if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')
    ar_count = int(input().strip())
    ar = list(map(int, input().rstrip().split()))

    result = simpleArraySum(ar)
    fptr.write(str(result) + '\n')
    fptr.close()
```

Here is a sample line of code that can be executed in Python:

```
print("Hello, World!")
```

You can just as easily store a string as a variable and then print it to stdout:

```
my_string = "Hello, World!"
print(my_string)
```

The above code will print `Hello, World!` on your screen. Try it yourself in the editor below!

```
if __name__ == '__main__':
    my_string = "Hello, World!"
    print(my_string)
    #print("Hello, World!")
```

Task

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

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

```
if __name__ == '__main__':
    a = int(input())
    b = int(input())
    print(a+b)
    print(a-b)
    print(a*b)
```

Task

Given an integer, n, perform the following conditional actions:

- If n is odd, print `Weird`
- If n is even and in the inclusive range of 2 to 5, print `Not Weird`
- If n is even and in the inclusive range of 6 to 20, print `Weird`
- If n is even and greater than 20, print `Not Weird`

Input Format

A single line containing a positive integer, n.

Constraints

- $1 \leq n \leq 100$

Output Format

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

```
#!/bin/python3
import math
import os
import random
import re
import sys
if __name__ == '__main__':
    n = int(input().strip())
    if n%2!=0:
        print("Weird")
    else:
        if n>=2 and n<=5:
            print("Not Weird")
        elif n>=6 and n<=20:
            print("Weird")
        else:
            print("Not Weird")
```

Task

The provided code stub reads two integers, a and b, from STDIN.

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

No rounding or formatting is necessary.

Example

$a = 3$

$b = 5$

- The result of the integer division $3//5 = 0$.
- The result of the float division is $3/5 = 0.6$.

```
if __name__ == '__main__':
    a = int(input())
    b = int(input())
    print(a//b)
    print(a/b)
```

Task

The provided code stub reads and integer, n, from STDIN. For all non-negative integers $i < n$, print i^2.

Example

$n = 3$

The list of non-negative integers that are less than $n = 3$ is $[0, 1, 2]$. Print the square of each number on a separate line.

```
0
1
4
```

```
if __name__ == '__main__':
    n = int(input())
    for i in range(0,n):
        print(i**2)
```

or

```python
if __name__ == '__main__':
    n = int(input())
    for i in range(0,n):
        print(i*i)
```

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 $year$, the year to test.

Constraints

$1900 \leq year \leq 10^5$

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.

```python
def is_leap(year):
    leap = False
    if (year % 4 == 0) and (year % 100 != 0):
        leap = True
    elif (year % 100 == 0) and (year % 400 != 0):
        leap = False
    elif (year % 100 == 0) and (year % 400 == 0):
       leap = True
    else:
        leap = False
    return leap
year = int(input())
print(is_leap(year))
```

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

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

$123 \cdots n$

Note that "..." represents the consecutive values in between.

Example

$n = 5$

Print the string 12345.

Input Format

The first line contains an integer n.

Constraints

$1 \leq n \leq 150$

Output Format

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

```
if __name__ == '__main__':
    n = int(input())
    for i in range(1,n+1):
        print(i,end="")
```

Let's learn about list comprehensions! You are given three integers x, y and z representing the dimensions of a cuboid along with an integer n. Print a list of all possible coordinates given by (i, j, k) on a 3D grid where the sum of $i + j + k$ is not equal to n. Here, $0 \leq i \leq x; 0 \leq j \leq y; 0 \leq k \leq z$. Please use list comprehensions rather than multiple loops, as a learning exercise.

Example

$x = 1$

$y = 1$

$z = 2$

$n = 3$

All permutations of $[i, j, k]$ are:

$[[0, 0, 0], [0, 0, 1], [0, 0, 2], [0, 1, 0], [0, 1, 1], [0, 1, 2], [1, 0, 0], [1, 0, 1], [1, 0, 2], [1, 1, 0], [1,$

.

Print an array of the elements that do not sum to $n = 3$.

$[[0, 0, 0], [0, 0, 1], [0, 0, 2], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 2]]$

Sample Input 0

```
1
1
1
2
```

Sample Output 0

```
[[0, 0, 0], [0, 0, 1], [0, 1, 0], [1, 0, 0], [1, 1, 1]]
```

Explanation 0

Each variable x, y and z will have values of 0 or 1. All permutations of lists in the form $[i, j, k] = [[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]$. Remove all arrays that sum to $n = 2$ to leave only the valid permutations.

```
if __name__ == '__main__':
    x = int(input())
    y = int(input())
    z = int(input())
    n = int(input())
    ans = [[i,j,k] for i in range(x+1) for j in range(y+1) for k in range(
z+1) if (i+j+k)!=n]
    print(ans)
```

Given the participants' score sheet for your University Sports Day, you are required to find the runner-up score. You are given n scores. Store them in a list and find the score of the runner-up.

Input Format

The first line contains n. The second line contains an array $A[]$ of n integers each separated by a space.

Constraints

- $2 \leq n \leq 10$
- $-100 \leq A[i] \leq 100$

Output Format

Print the runner-up score.

Sample Input 0

```
5
2 3 6 6 5
```

Sample Output 0

```
5
```

Explanation 0

Given list is $[2, 3, 6, 6, 5]$. The maximum score is 6, second maximum is 5. Hence, we print 5 as the runner-up score.

```python
if __name__ == '__main__':
    n = int(input())
    arr = map(int, input().split())
    lst=list(arr)
    ordr = sorted(lst, reverse=True)
    for i in range(n):
        if ordr[i]==ordr[i+1]:
            pass
        else:
             print(ordr[i+1])
             break
```

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 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.

```
'''

Can be solved in 2 parts:

1st part:

1.Take name and score as input

2.Store name and score in nested list `students_grade`

3.Make a Unique_sorted_score_list that is unique(using set()) and
sorted(using sorted()) scores

4.Fetch second_lowest_score from Unique_sorted_score_list

2nd part:

1.create an empty list => low_student_list

2.for every student in `students_grade`

3.check if second_lowest_score match any student[score]

4.if Yes: append in student_list

5.iterate through `low_student_list` to print result

'''
```

```
if __name__ == '__main__':
    student_grade=[]
    for _ in range(int(input())):
        name = input()
        score = float(input())
        student_grade.append([name,score])
    scores=set([x[1] for x in student_grade])
    list_scores=list(scores)
    sorted_scores=sorted(list_scores)
    second_lowest=sorted_scores[1]
    low_final_list=[]
    for i in student_grade:
        if i[1]==second_lowest:
            low_final_list.append(i[0])
    for i in sorted(low_final_list):
        print(i)
```

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

```
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 $\frac{52+56+60}{3} \Rightarrow 56$

```python
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()
    sum=0
    average=float()
    for i in student_marks[query_name]:
        sum=sum+i
    average=sum/len(student_marks[query_name])
    print("%.2f"%average)
```

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

- 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]
```

Using split() method :

This function helps in getting multiple inputs from users. It breaks the given input by the specified separator. If a separator is not provided then any white space is a separator. Generally, users use a split() method to split a Python string but one can use it in taking multiple inputs.

```
# 'o' is inserted at index 3 (4th position)
vowel.insert(3, 'o')
```

```
if __name__ == '__main__':
    N = int(input())
    L=[]
    for i in range(N):
        cmd=input().split()
        if cmd[0] == "insert":
            L.insert(int(cmd[1]),int(cmd[2]))
        elif cmd[0] == "append":
            L.append(int(cmd[1]))
        elif cmd[0] == "pop":
            L.pop()
        elif cmd[0] == "print":
            print(L)
        elif cmd[0] == "remove":
            L.remove(int(cmd[1]))
        elif cmd[0] == "sort":
            L.sort()
        else:
            L.reverse()
```

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

Usage:

```
>> a = raw_input()
5 4 3 2
>> lis = a.split()
>> print (lis)
['5', '4', '3', '2']
```

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

```
>> newlis = list(map(int, lis))
>> print (newlis)
[5, 4, 3, 2]
```

Sets are an unordered collection of unique values. A single set contains values of any immutable data type.

CREATING SETS

```
>> myset = {1, 2} # Directly assigning values to a set
>> myset = set()  # Initializing a set
>> myset = set(['a', 'b']) # Creating a set from a list
>> myset
{'a', 'b'}
```

MODIFYING SETS

Using the add() function:

```
>> myset.add('c')
>> myset
{'a', 'c', 'b'}
>> myset.add('a') # As 'a' already exists in the set, nothing happens
>> myset.add((5, 4))
>> myset
{'a', 'c', 'b', (5, 4)}
```

Using the update() function:

```
>> myset.update([1, 2, 3, 4]) # update() only works for iterable object
>> myset
{'a', 1, 'c', 'b', 4, 2, (5, 4), 3}
>> myset.update({1, 7, 8})
>> myset
{'a', 1, 'c', 'b', 4, 7, 8, 2, (5, 4), 3}
>> myset.update({1, 6}, [5, 13])
>> myset
{'a', 1, 'c', 'b', 4, 5, 6, 7, 8, 2, (5, 4), 13, 3}
```

REMOVING ITEMS

Both the discard() and remove() functions take a single value as an argument and removes that value from the set. If that value is not present, discard() does nothing, but remove() will raise a KeyError exception.

```
>> myset.discard(10)
>> myset
{'a', 1, 'c', 'b', 4, 5, 7, 8, 2, 12, (5, 4), 13, 11, 3}
>> myset.remove(13)
>> myset
{'a', 1, 'c', 'b', 4, 5, 7, 8, 2, 12, (5, 4), 11, 3}
```

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

```
>> a = {2, 4, 5, 9}
>> b = {2, 4, 11, 12}
>> a.union(b) # Values which exist in a or b
{2, 4, 5, 9, 11, 12}
>> a.intersection(b) # Values which exist in a and b
{2, 4}
>> a.difference(b) # Values which exist in a but not in b
{9, 5}
```

The union() and intersection() functions are symmetric methods:

```
>> a.union(b) == b.union(a)
True
>> a.intersection(b) == b.intersection(a)
True
>> a.difference(b) == b.difference(a)
False
```

These other built-in data structures in Python are also useful.

Task

Given 2 sets of integers, M and N, print their symmetric difference in ascending order. The term symmetric difference indicates those values that exist in either M or N but do not exist in both.

Input Format

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

Output Format

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

Sample Input

```
STDIN       Function
-----       --------
4           set a size M = 4
2 4 5 9     a = {2, 4, 5, 9}
4           set b size N = 4
2 4 11 12   b = {2, 4, 11, 12}
```

Sample Output

```
5
9
11
12
```

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
m=int(input())
mset=set(map(int,input().split()))
n=int(input())
nset=set(map(int,input().split()))
mdef=mset.difference(nset)
ndef=nset.difference(mset)
output=mdef.union(ndef)
for i in sorted(list(output)):
    print(i)
```

Given an integer, n, and n space-separated integers as input, create a tuple, t, of those n integers. Then compute and print the result of $hash(t)$.

Note: hash() is one of the functions in the `__builtins__` module, so it need not be imported.

Input Format

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

Output Format

Print the result of $hash(t)$.

```
if __name__ == '__main__':
    n = int(input())
    integer_list = map(int, input().split())
    t=tuple(integer_list)
    print(hash(t))
```

There is an array of n integers. There are also 2 **disjoint sets**, A and B, each containing m integers. You like all the integers in set A and dislike all the integers in set B. Your initial happiness is 0. For each i integer in the array, if $i \in A$, you add 1 to your happiness. If $i \in B$, you add -1 to your happiness. Otherwise, your happiness does not change.
Output your final happiness at the end.

Note: Since A and B are sets, they have no repeated elements. However, the array might contain duplicate elements.

Constraints

$1 \leq n \leq 10^5$
$1 \leq m \leq 10^5$
$1 \leq Any\ integer\ in\ the\ input \leq 10^9$

Input Format

The first line contains integers n and m separated by a space.
The second line contains n integers, the elements of the array.
The third and fourth lines contain m integers, A and B, respectively.

Output Format

Output a single integer, your total happiness.

Sample Input

```
3 2
1 5 3
3 1
5 7
```

Sample Output

```
1
```

Explanation

You gain 1 unit of happiness for elements 3 and 1 in set A. You lose 1 unit for 5 in set B.

The element 7 in set B does not exist in the array so it is not included in the calculation.

Hence, the total happiness is $2 - 1 = 1$.

input() - takes input from user strip() - deletes white spaces from the begin and the end of input split(' ') - **splits input into elements of an list with (' ') being as separator**.

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
if __name__=="__main__":
    happiness=0
    n,m=map(int,input().strip().split(' '))
    arr=list(map(int,input().strip().split(' ')))
    good=set(map(int,input().strip().split(' ')))
    bad=set(map(int,input().strip().split(' ')))
    for i in arr:
        if i in good:
            happiness+=1
        elif i in bad:
            happiness-=1
    print(happiness)
```

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

It adds the element to the set and returns '`None`'.

Example

```
>>> s = set('HackerRank')
>>> s.add('H')
>>> print s
set(['a', 'c', 'e', 'H', 'k', 'n', 'r', 'R'])
>>> print s.add('HackerRank')
None
>>> print s
set(['a', 'c', 'e', 'HackerRank', 'H', 'k', 'n', 'r', 'R'])
```

Task

Apply your knowledge of the .add() operation to help your friend Rupal.

Rupal has a huge collection of country stamps. She decided to count the total number of distinct country stamps in her collection. She asked for your help. You pick the stamps one by one from a stack of N country stamps.

Find the total number of distinct country stamps.

Input Format

The first line contains an integer N, the total number of country stamps.

The next N lines contains the name of the country where the stamp is from.

Constraints

$0 < N < 1000$

Output Format

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

Sample Input

```
7
UK
China
USA
France
New Zealand
UK
France
```

Sample Output

```
5
```

Explanation

UK and France repeat twice. Hence, the total number of distinct country stamps is 5 (five).

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
n=int(input())
set1=set()
for i in range(n):
    set1.add(input())
print(len(set1))
```

.remove(x)

This operation removes element x from the set.

If element x does not exist, it raises a `KeyError`.

The .remove(x) operation returns `None`.

Example

```
>>> s = set([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> s.remove(5)
>>> print s
set([1, 2, 3, 4, 6, 7, 8, 9])
>>> print s.remove(4)
None
>>> print s
set([1, 2, 3, 6, 7, 8, 9])
>>> s.remove(0)
KeyError: 0
```

.discard(x)

This operation also removes element x from the set.

If element x does not exist, it **does not** raise a `KeyError`.

The .discard(x) operation returns `None`.

Example

```
>>> s = set([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> s.discard(5)
>>> print s
set([1, 2, 3, 4, 6, 7, 8, 9])
>>> print s.discard(4)
None
>>> print s
set([1, 2, 3, 6, 7, 8, 9])
>>> s.discard(0)
>>> print s
set([1, 2, 3, 6, 7, 8, 9])
```

.pop()

This operation removes and return an arbitrary element from the set.

If there are no elements to remove, it raises a `KeyError`.

Example

```
>>> s = set([1])
>>> print s.pop()
1
>>> print s
set([])
>>> print s.pop()
KeyError: pop from an empty set
```

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

For Example:

```
Www.HackerRank.com → wWW.hACKERrANK.COM
Pythonist 2 → pYTHONIST 2
```

Function Description

Complete the swap_case function in the editor below.

swap_case has the following parameters:

- string s: the string to modify

Returns

- string: the modified string

Input Format

A single line containing a string s.

Constraints

$0 < len(s) \leq 1000$

Sample Input 0

```
HackerRank.com presents "Pythonist 2".
```

Sample Output 0

```
hACKERrANK.COM PRESENTS "pYTHONIST 2".
```

```python
def swap_case(s):
    swap=s.swapcase()
    return swap

if __name__ == '__main__':
    s = input()
    result = swap_case(s)
    print(result)
```

In Python, a string can be split on a delimiter.

Example:

```
>>> a = "this is a string"
>>> a = a.split(" ") # a is converted to a list of strings.
>>> print a
['this', 'is', 'a', 'string']
```

Joining a string is simple:

```
>>> a = "-".join(a)
>>> print a
this-is-a-string
```

Task

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

Function Description

Complete the split_and_join function in the editor below.

split_and_join has the following parameters:

- string line: a string of space-separated words

Returns

- string: the resulting string

Input Format

The one line contains a string consisting of space separated words.

Sample Input

```
this is a string
```

Sample Output

```
this-is-a-string
```

```
def split_and_join(line):
    # write your code here
    line=line.split(" ")
    line="-".join(line)
    return line

if __name__ == '__main__':
    line = input()
    result = split_and_join(line)
    print(result)
```

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

> Hello `firstname` `lastname`! You just delved into python.

Function Description

Complete the print_full_name function in the editor below.

print_full_name has the following parameters:

- string first: the first name
- string last: the last name

Prints

- string: 'Hello $firstname\ lastname$! You just delved into python' where $firstname$ and $lastname$ are replaced with $first$ and $last$.

Input Format

The first line contains the first name, and the second line contains the last name.

Constraints

The length of the first and last names are each ≤ 10.

Sample Input 0

```
Ross
Taylor
```

Sample Output 0

```
Hello Ross Taylor! You just delved into python.
```

Explanation 0

The input read by the program is stored as a string data type. A string is a collection of characters.

```python
def print_full_name(first, last):
    # Write your code here
    print(f"Hello {first} {last}! You just delved into python.")

if __name__ == '__main__':
    first_name = input()
    last_name = input()
    print_full_name(first_name, last_name)
```

We have seen that lists are mutable (they can be changed), and tuples are immutable (they cannot be changed).

Let's try to understand this with an example.

You are given an immutable string, and you want to make changes to it.

```
>>> string = "abracadabra"
```

You can access an index by:

```
>>> print string[5]
a
```

What if you would like to assign a value?

```
>>> string[5] = 'k'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

How would you approach this?

- One solution is to convert the string to a list and then change the value.

```
>>> string = "abracadabra"
>>> l = list(string)
>>> l[5] = 'k'
>>> string = ''.join(l)
>>> print string
abrackdabra
```

Another approach is to slice the string and join it back.

```
>>> string = string[:5] + "k" + string[6:]
>>> print string
abrackdabra
```

Task

Read a given string, change the character at a given index and then print the modified string.

Function Description

Complete the mutate_string function in the editor below.

mutate_string has the following parameters:

- string string: the string to change
- int position: the index to insert the character at
- string character: the character to insert

Returns

- string: the altered string

Input Format

The first line contains a string, *string*.

The next line contains an integer *position*, the index location and a string *character*, separated by a space.

Sample Input

```
STDIN           Function
-----           --------
abracadabra     s = 'abracadabra'
5 k             position = 5, character = 'k'
```

Sample Output

```
abrackdabra
```

```
def mutate_string(string, position, character):
    pos = position+1
    Output = string[:position]+character+string[pos:]
    return Output
```

```
if __name__ == '__main__':
    s = input()
    i, c = input().split()
    s_new = mutate_string(s, int(i), c)
    print(s_new)
```

OR

```
def mutate_string(string, position, character):
    l=list(string)
    l[position]=character
    l1="".join(l)
    return l1

if __name__ == '__main__':
    s = input()
    i, c = input().split()
    s_new = mutate_string(s, int(i), c)
    print(s_new)
```

In this challenge, the user enters a string and a substring. You have to print the number of times that the substring occurs in the given string. String traversal will take place from left to right, not from right to left.

NOTE: String letters are case-sensitive.

Input Format
The first line of input contains the original string. The next line contains the substring.

Constraints

$$1 \leq len(string) \leq 200$$

Each character in the string is an ascii character.

Output Format
Output the integer number indicating the total number of occurrences of the substring in the original string.

Sample Input

ABCDCDC
CDC

Sample Output
2

Concept
Some string processing examples, such as these, might be useful.
There are a couple of new concepts:
In Python, the length of a string is found by the function len(s), where s is the string.
To traverse through the length of a string, use a for loop:

```
for i in range(0, len(s)):
    print (s[i])
```

A range function is used to loop over some length:

range (0, 5)
Here, the range loops over 0 to 4 , 5 is excluded .

>

```
def count_substring(string, sub_string):
    count=0
    for i in range(len(string)-len(sub_string)+1):
        if string[i:i+len(sub_string)]==sub_string:
            count+=1
    return count

if __name__ == '__main__':
    string = input().strip()
    sub_string = input().strip()

    count = count_substring(string, sub_string)
    print(count)
```

Python has built-in string validation methods for basic data. It can check if a string is composed of alphabetical characters, alphanumeric characters, digits, etc.

str.isalnum()

This method checks if all the characters of a string are alphanumeric (a-z, A-Z and 0-9).

```
>>> print 'ab123'.isalnum()
True
>>> print 'ab123#'.isalnum()
False
```

str.isalpha()

This method checks if all the characters of a string are alphabetical (a-z and A-Z).

```
>>> print 'abcD'.isalpha()
True
>>> print 'abcd1'.isalpha()
False
```

str.isdigit()

This method checks if all the characters of a string are digits (0-9).

```
>>> print '1234'.isdigit()
True
>>> print '123edsd'.isdigit()
False
```

str.islower()

This method checks if all the characters of a string are lowercase characters (a-z).

```
>>> print 'abcd123#'.islower()
True
>>> print 'Abcd123#'.islower()
False
```

str.isupper()

This method checks if all the characters of a string are uppercase characters (A-Z).

```
>>> print 'ABCD123#'.isupper()
True
>>> print 'Abcd123#'.isupper()
False
```

Task

You are given a string S

Your task is to find out if the string S contains: alphanumeric characters, alphabetical characters, digits, lowercase and uppercase characters.

Input Format

A single line containing a string S

Constraints

$0 < len(S) < 1000$

Output Format

In the first line, print `True` if S has any alphanumeric characters. Otherwise, print `False`.

In the second line, print `True` if S has any alphabetical characters. Otherwise, print `False`.

In the third line, print `True` if S has any digits. Otherwise, print `False`.

In the fourth line, print `True` if S has any lowercase characters. Otherwise, print `False`.

In the fifth line, print `True` if S has any uppercase characters. Otherwise, print `False`.

Sample Input

qA2
Sample Output

True
True
True
True
True

>

The any() function returns True if any item in an iterable are true, otherwise it returns False.

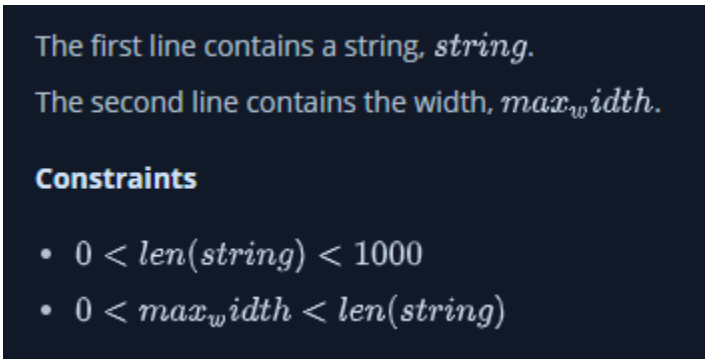
If the iterable object is empty, the any() function will return False.

The map() function executes a specified function for each item in an iterable. The item is sent to the function as a parameter.

map(*function*, *iterables*)

```
if __name__ == '__main__':
    s = input()
    print(any(map(str.isalnum, s)))
    print(any(map(str.isalpha, s)))
    print(any(map(str.isdigit, s)))
    print(any(map(str.islower, s)))
    print(any(map(str.isupper, s)))
```

You are given a string S and width w .
Your task is to wrap the string into a paragraph of width .
Function Description
Complete the wrap function in the editor below.
wrap has the following parameters:
string string: a long string
int max_width: the width to wrap to
Returns
string: a single string with newline characters ('\n') where the breaks should be
Input Format



Sample Input 0
ABCDEFGHIJKLIMNOQRSTUVWXYZ
4

Sample Output 0

ABCD
EFGH
IJKL
IMNO
QRST
UVWX
YZ

SOLUTION >

```
import textwrap
def wrap(string, max_width):
    for i in range(0,len(string)+1,max_width):
        result=string[i:i+max_width]
        if len(result)==max_width:
            print(result)
        else:
            return(result)
if __name__ == '__main__':
    string, max_width = input(), int(input())
    result = wrap(string, max_width)
    print(result)
```

You are asked to ensure that the first and last names of people begin with a capital letter in their passports. For example, alison heck should be capitalised correctly as Alison Heck.
Given a full name, your task is to capitalize the name appropriately.
Input Format
A single line of input containing the full name, .
Constraints
The string consists of alphanumeric characters and spaces.
Note: in a word only the first character is capitalized. Example 12abc when capitalized remains 12abc.
Output Format
Print the capitalized string, .
Sample Input
chris alan
Sample Output
Chris Alan

The `split()` method splits a string into a list.

The `replace()` method replaces a specified phrase with another specified phrase.

string.replace(*oldvalue, newvalue, count*)

SOLUTION >

```
#!/bin/python3
import math
import os
import random
import re
import sys
# Complete the solve function below.
def solve(s):
    for i in s.split():
        s=s.replace(i,i.capitalize())
    return s
if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')
    s = input()
    result = solve(s)
    fptr.write(result + '\n')
    fptr.close()
```

Regular expression (or RegEx)
A regular expression is a sequence of characters that define a search pattern. It is mainly used for string pattern matching.

Regular expressions are extremely useful in extracting information from text such as: code, log files, spreadsheets, documents, etc.
We can match a specific string in a test string by making our regex pattern . This is one of the simplest patterns. However, in the coming challenges, we'll see how well we can match more complex patterns and learn about their syntax.

Task
You have a test string . Your task is to match the string hackerrank. This is case sensitive.

Note
This is a regex only challenge. You are not required to write code.
You only have to fill in the regex pattern in the blank (_____).

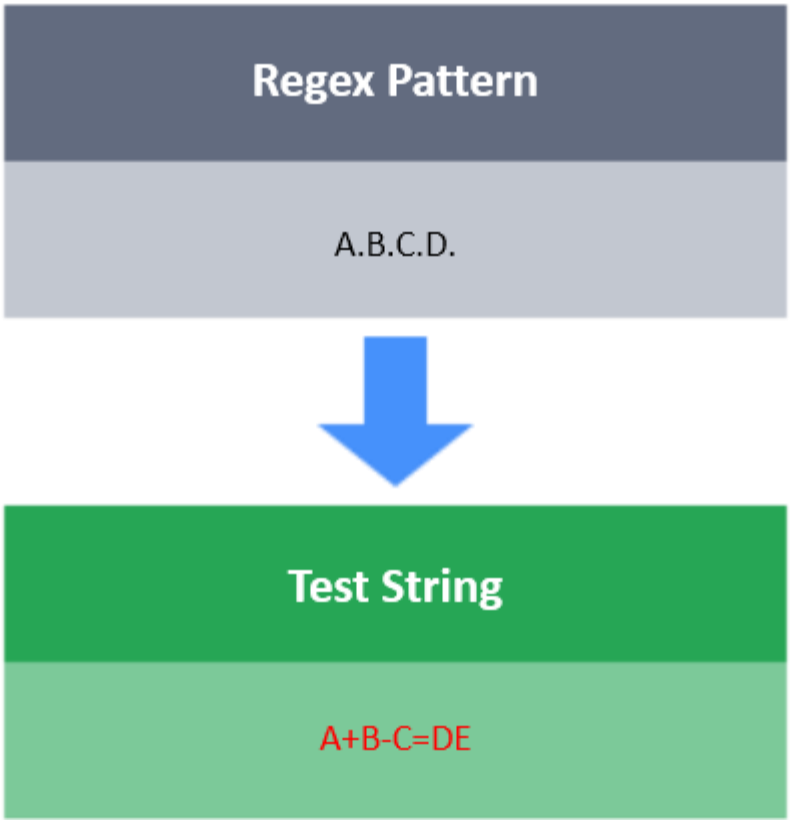
SOLUTION >

```
Regex_Pattern = r'hackerrank'    # Do not delete 'r'.
import re
Test_String = input()
match = re.findall(Regex_Pattern, Test_String)
print("Number of matches :", len(match))
```

---

dot

The dot (.) matches anything (except for a newline).



Note: If you want to match (.) in the test string, you need to escape the dot by using a slash \..
In Java, use \\. instead of \..
Task
You have a test string .
Your task is to write a regular expression that matches only and exactly strings of form: , where each variable can be any single character except the newline.
Note
This is a regex only challenge. You are not required to write any code.
You only have to fill in the regex pattern in the blank (_____).

SOLUTION >

```
# Because we want to match exact patterns in the string, we'll have
# The caret, ^, matches the start of the line.
# And similarly, the dollar sign, $, matches the end of the line.

# A dot matches anything except for a newline character.
# And to match a dot we need to escape it using a backslash, like this: \.

# So the following regex matches:
# A line that has 3 non-newline characters, followed by a dot, 3 non-newline
characters again,
# then a dot, 3 non-newline characters again a dot and then 3 non-newline characters
and nothing else.
regex_pattern = r"^...\....\....\....$" # Do not delete 'r'.
import re
import sys
test_string = input()
match = re.match(regex_pattern, test_string) is not None
print(str(match).lower())
```

The expression \d matches any digit [-].

The expression \D matches any character that is not a digit.

Task

You have a test string S. Your task is to match the pattern $xxXxxXxxxx$

Here x denotes a digit character, and X denotes a non-digit character.

```
Regex_Pattern = r"\d\d\D\d\d\D\d\d\d\d" # Do not delete 'r'.
import re
print(str(bool(re.search(Regex_Pattern, input()))).lower())
```

\s matches any whitespace character [\r\n\t\f].
\S matches any non-white space character.

You have a test string S. Your task is to match the pattern $XXxXXxXX$

Here, x denotes whitespace characters, and X denotes non-white space characters.

SOLUTION >

```
Regex_Pattern = r"\S\S\s\S\S\s\S\S" # Do not delete 'r'.
import re
print(str(bool(re.search(Regex_Pattern, input()))).lower())
```
