In [ ]:

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

CREATING SETS

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


MODIFYING SETS

Using the add() function:

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

Using the update() function:

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

REMOVING ITEMS

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

In [ ]:

```
#sort dictionary by values
sorted(dictionary.items(), key = lambda kv:(kv[1], kv[0]))
```

In [ ]:

```
#Reduce :  It applies a rolling computation to sequential pairs of values in a list

from functools import reduce
product = reduce((lambda x, y: x * y), [1, 2, 3, 4])

# Output: 24
```

In [ ]:

```
To find count of repetition of consecutive element
from itertools import groupby
s = input()
for key,grp in groupby(s):
    length = len(list(grp))
    print(f"({key}, {length})")

Input:11223334
Output;

(1, 2)
(2, 2)
(3, 3)
(4, 1)
```

Regular Expression

In [ ]:

```
import re
DATA = "Hey, you - what are you doing here!?"
print re.findall(r"[\w']+", DATA)
# Prints ['Hey', 'you', 'what', 'are', 'you', 'doing', 'here']
```

In [24]:

```
import re
s = '@something.co1'
re.split('; |, |\@|\.', s)
```

Out[24]:

```
['', 'something', 'co1']
```

In [ ]:

```
group()
A group() expression returns one or more subgroups of the match.

Code
>>> import re
>>> m = re.match(r'(\w+)@(\w+)\.(\w+)','username@hackerrank.com')
>>> m.group(0)       # The entire match
'username@hackerrank.com'
>>> m.group(1)       # The first parenthesized subgroup.
'username'
>>> m.group(2)       # The second parenthesized subgroup.
'hackerrank'
>>> m.group(3)       # The third parenthesized subgroup.
'com'
>>> m.group(1,2,3)   # Multiple arguments give us a tuple.
('username', 'hackerrank', 'com')

groups()
A groups() expression returns a tuple containing all the subgroups of the match.
Code

>>> import re
>>> m = re.match(r'(\w+)@(\w+)\.(\w+)','username@hackerrank.com')
>>> m.groups()
('username', 'hackerrank', 'com')

groupdict()
A groupdict() expression returns a dictionary containing all the named subgroups of the
match, keyed by the subgroup name.
Code

>>> m = re.match(r'(?P<user>\w+)@(?P<website>\w+)\.(?P<extension>\w+)','myname@hackerra
nk.com')
>>> m.groupdict()
{'website': 'hackerrank', 'user': 'myname', 'extension': 'com'}


re.findall()
The expression re.findall() returns all the non-overlapping matches of patterns in a st
ring as a list of strings.
Code

>>> import re
>>> re.findall(r'\w','http://www.hackerrank.com/')
['h', 't', 't', 'p', 'w', 'w', 'w', 'h', 'a', 'c', 'k', 'e', 'r', 'r', 'a', 'n', 'k',
'c', 'o', 'm']
re.finditer()
The expression re.finditer() returns an iterator yielding MatchObject instances over al
l non-overlapping matches for the re pattern in the string.
Code

>>> import re
>>> re.finditer(r'\w','http://www.hackerrank.com/')
<callable-iterator object at 0x0266C790>
>>> map(lambda x: x.group(),re.finditer(r'\w','http://www.hackerrank.com/'))
['h', 't', 't', 'p', 'w', 'w', 'w', 'h', 'a', 'c', 'k', 'e', 'r', 'r', 'a', 'n', 'k',
'c', 'o', 'm']
```

In [ ]:

```
import re
for _ in range(int(input())):
    print(bool(re.match(r'^[-+]?[0-9]*\.[0-9]+$', input())))
-1.00
+4.54
True
True
```

In [34]:

```
import re
v = "aeiou"
c = "qwrtypsdfghjklzxcvbnm"
m = re.findall(r"(?<=[%s])([%s]{2,})[%s]" % (c, v, c), input(), flags = re.I)
print('\n'.join(m or ['-1']))
```

```
ee
Ioo
Oeo
eeeee
```

In [ ]:

```

```

Validate phone number

In [11]:

```
import re
num1 = '9587456281'
num2 = '1252478965'
if re.match(r'^[789](\d+)',num1):
    print(True)
else:
    print(False)
```

```
True
```

validate Email

In [85]:

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
import email.utils
import re
for _ in range(int(input())):
    namewithadd = input()
    name,email_id = email.utils.parseaddr(namewithadd)
    if re.match(r'^[a-z][\w_.-]+@[a-z]+\.[a-z]{1,3}$',email_id,flags = re.I):
        print(namewithadd)
Input:
    2
    DEXTER <dexter@hotmail.com>
    VIRUS <virus!@variable.:p>
Output:
    DEXTER <dexter@hotmail.com>
```

```
<_sre.SRE_Match object; span=(0, 22), match='D_O.S-HI-@hackerrank.c'>
```

validate color

In [94]:

```
s = 'color: #FfFdF8; background-color:#aef'
import re
print(re.findall(r'[\w: ]+#[a-z0-9]+',s,flags = re.I))
```

```
['color: #FfFdF8', 'color:#aef']
```

In [97]:

```
ss = 'B1CD102354'
print(re.findall(r'[0-9]',ss))
print(re.findall(r'[A-Z]',ss))
print(len(re.findall(r'[0-9]',ss)))
```

```
['1', '1', '0', '2', '3', '5', '4']
['B', 'C', 'D']
7
```

In [126]:

```
from itertools import groupby
sss = '5133-3367-8912-3456'
print(re.match(r'^[456][0-9]{3}(-){1}[0-9]{4}(-){1}[0-9]{4}(-){1}[0-9]{4}$',sss))
maxi = max([len(list(grp)) for key,grp in groupby(sss)])
print(maxi)
```

```
<_sre.SRE_Match object; span=(0, 19), match='5133-3367-8912-3456'>
2
```

In [ ]:

```
The re.sub() tool (sub stands for substitution) evaluates a pattern and, for each valid
match, it calls a method (or lambda).
The method is called for all matches and can be used to modify strings in different way
s.
The re.sub() method returns the modified string as an output.

example:
    import re

    #Squaring numbers
    def square(match):
        number = int(match.group(0))
        return str(number**2)

    print re.sub(r"\d+", square, "1 2 3 4 5 6 7 8 9")
output:
    1 4 9 16 25 36 49 64 81

    import re

    html = """
    <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>
    """

    print re.sub("(<!--.*?-->)", "", html) #remove comment

Output:
    <head>
    <title>HTML</title>
    </head>
    <object type="application/x-flash"
      data="your-file.swf"
      width="0" height="0">

      <param name="quality" value="high"/>
    </object>
```

In [38]:

```
import re
N = int(input())
for i in range(N):
    print(re.sub(r'(?<= )(&&|\|\|)(?= )', lambda x: 'and' if x.group() == '&&' else 'o
r',input()))
```

```
And
```

In [116]:

```
l1 = ['07895462130', '919875641230', '9195969878']
l2 = []
for x in l1:
    if len(x) == 10:
        l2.append('+91 '+x[0:6] + ' ' + x[5:])

    elif re.findall('(?<=91)(.*?)$',x):
        number = re.findall('(?<=91)(.*?)$',x)[0]
        print(x,number)
        l2.append('+91 '+number[0:6] + ' ' + number[5:])

    elif re.findall('(?<=\+91)(.*?)$',x):
        number = re.findall('(?<=\+91)(.*?)$',x)
        print(x,number)
        l2.append('+91 '+number[0:6] + ' ' + number[5:])

    elif re.findall('(?<=0)(.*?)$',x):
        number = re.findall('(?<=0)(.*?)$',x)[0]
        print(x,number)
        l2.append('+91 '+number[0:6] + ' ' + number[5:])

    else:
        pass
print(*sorted(l2))
```

```
07895462130 7895462130
919875641230 9875641230
+91 789546 62130 +91 919596 69878 +91 987564 41230
```

NUMPY

In [ ]:

```
The NumPy (Numeric Python) package helps us manipulate large arrays and matrices of num
eric data.

To use the NumPy module, we need to import it using:

import numpy
Arrays

A NumPy array is a grid of values. They are similar to lists, except that every element
of an array must be the same type.

import numpy

a = numpy.array([1,2,3,4,5])
print a[1]          #2

b = numpy.array([1,2,3,4,5],float)
print b[1]          #2.0
```

In [122]:

```
l1 = [1,2,3]
print(l1[::-1])
```

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

In [ ]:

```
shape:
The shape tool gives a tuple of array dimensions and can be used to change the dimensio
ns of an array.
(a). Using shape to get array dimensions
import numpy
my__1D_array = numpy.array([1, 2, 3, 4, 5])
print my_1D_array.shape     #(5,) -> 5 rows and 0 columns

my__2D_array = numpy.array([[1, 2],[3, 4],[6,5]])
print my_2D_array.shape     #(3, 2) -> 3 rows and 2 columns

(b). Using shape to change array dimensions
import numpy
change_array = numpy.array([1,2,3,4,5,6])
change_array.shape = (3, 2)
print change_array
#Output
[[1 2]
[3 4]
[5 6]]

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

import numpy
my_array = numpy.array([1,2,3,4,5,6])
print numpy.reshape(my_array,(3,2))
#Output
[[1 2]
[3 4]
[5 6]]

Transpose:
We can generate the transposition of an array using the tool numpy.transpose.
It will not affect the original array, but it will create a new array.
import numpy
my_array = numpy.array([[1,2,3],
                        [4,5,6]])
print numpy.transpose(my_array)
#Output
[[1 4]
 [2 5]
 [3 6]]

Flatten:
The tool flatten creates a copy of the input array flattened to one dimension.

import numpy
my_array = numpy.array([[1,2,3],
                        [4,5,6]])
print my_array.flatten()
#Output
[1 2 3 4 5 6]



Concatenate
Two or more arrays can be concatenated together using the concatenate function with a t
uple of the arrays to be joined:
```

```
import numpy
array_1 = numpy.array([1,2,3])
array_2 = numpy.array([4,5,6])
array_3 = numpy.array([7,8,9])
print numpy.concatenate((array_1, array_2, array_3))
#Output
[1 2 3 4 5 6 7 8 9]
If an array has more than one dimension, it is possible to specify the axis along which
multiple arrays are concatenated. By default, it is along the first dimension.
import numpy
array_1 = numpy.array([[1,2,3],[0,0,0]])
array_2 = numpy.array([[0,0,0],[7,8,9]])
print numpy.concatenate((array_1, array_2), axis = 1)
#Output
[[1 2 3 0 0 0]
 [0 0 0 7 8 9]]



zeros
The zeros tool returns a new array with a given shape and type filled with 's.
import numpy
print numpy.zeros((1,2))                    #Default type is float
#Output : [[ 0.  0.]]
print numpy.zeros((1,2), dtype = numpy.int) #Type changes to int
#Output : [[0 0]]

ones
The ones tool returns a new array with a given shape and type filled with 's.
import numpy
print numpy.ones((1,2))                    #Default type is float
#Output : [[ 1.  1.]]
print numpy.ones((1,2), dtype = numpy.int) #Type changes to int
#Output : [[1 1]]


identity
The identity tool returns an identity array. An identity array is a square matrix with
all the main diagonal elements as  and the rest as . The default type of elements is fl
oat.
import numpy
print numpy.identity(3) #3 is for  dimension 3 X 3
#Output
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]

eye
The eye tool returns a 2-D array with 's as the diagonal and 's elsewhere. The diagonal
can be main, upper or lower depending on the optional parameter . A positive  is for th
e upper diagonal, a negative  is for the lower, and a   (default) is for the main diago
nal.
import numpy
print numpy.eye(8, 7, k = 1)    # 8 X 7 Dimensional array with first upper diagonal 1.
#Output
[[ 0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.]]
print numpy.eye(8, 7, k = -2)   # 8 X 7 Dimensional array with second lower diagonal 1.
```

In [8]:

```
import numpy
array_1 = numpy.array([[1,2,3],[1,0,0]])
array_2 = numpy.array([[0,0,0],[7,8,9]])
print(numpy.concatenate((array_1, array_2), axis = 0))
```

```
[[1 2 3]
 [1 0 0]
 [0 0 0]
 [7 8 9]]
```

In [14]:

```
import numpy
#numpy.identity(4)
numpy.eye(4,3,k=0)
```

Out[14]:

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 0.]])
```

In [ ]:

```
```