In [1]:

```
#Question 1: Detect Floating Point Number

a=int(input().strip())
for _ in range(a):
    answer=False
    try:
        stri=input().strip()
        N=float(stri)
        answer=True
        Num=int(stri)
        answer=False
    except:
        pass
    print(answer)
```

```
4
4.000
True
4.0o0
False
-1.00
True
+4.54
True
```

In [2]:

```
#Question 2: Re.split()

regex_pattern = r"[.,]+" # Do not delete 'r'.

import re
print("\n".join(re.split(regex_pattern, input())))
```

```
100,000,000.000
100
000
000
000
```

In [3]:

```
#Question 3: Group

import re
m = re.search(r'([A-Za-z0-9])\1+',input())
if m:
    print(m.group(1))
else:
    print(-1)
```

```
..12345678910111213141516171820212223
1
```

In [4]:

```
#Question 4: Re.find()

import re

vowels = 'aeiou'
consonants = 'qwrtypsdfghjklzxcvbnm'
match = re.findall(r'(?<=[' + consonants + '])([' + vowels + ']{2,})(?=[' + consonants +
'])', input().strip(), re.IGNORECASE)

if match:
    for i in match:
```

```
        print(i)
else:
    print(-1)
```

```
rabcdeefgyYhFjkIoomnpOeorteeeeet
ee
Ioo
Oeo
eeeee
```

In [5]:

```
#Question 5:  Re.start()

import re

string = input()
substring = input()

index = 0

match = re.search(substring,string)
if not match: print('(-1, -1)')
while match:
    print((match.start() + index , match.end() + index - 1))
    index += match.start() + 1
    match = re.search(substring, string[index:])
```

```
aaadaa
aa
(0, 1)
(1, 2)
(4, 5)
```

In [6]:

```
#Question 6: Regex Substitution

import re

def replace(match):
    sym = match.group(0)

    if sym == "&&":
        return "and"
    elif sym == "||":
        return "or"

N = int(input().strip())
for _ in range(N):
    print(re.sub(r'(?<= )(&&|\|\|)(?= )', replace, input()))
```

```
11
a = 1;
a = 1;
b = input();
b = input();
if a + b > 0 && a - b < 0:     start()
if a + b > 0 and a - b < 0:     start()
elif a*b > 10 || a/b < 1:     stop()
elif a*b > 10 or a/b < 1:     stop()
print set(list(a)) | set(list(b))
print set(list(a)) | set(list(b))
#Note do not change &&& or ||| or & or |
#Note do not change &&& or ||| or & or |
#Only change those '&&' which have space on both sides.
#Only change those '&&' which have space on both sides.
#Only change those '|| which have space on both sides.
#Only change those '|| which have space on both sides.
i
i
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-6-62ebf7b61b9c> in <module>
     13 N = int(input().strip())
     14 for _ in range(N):
---> 15     print(re.sub(r'(?<= )(&&|\|\|)(?= )', replace, input()))
     16

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in raw_input(self, prompt)
    858                 "raw_input was called, but this frontend does not support input r
equests."
    859             )
--> 860         return self._input_request(str(prompt),
    861             self._parent_ident,
    862             self._parent_header,

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in _input_request(self, prompt, ide
nt, parent, password)
    902             except KeyboardInterrupt:
    903                 # re-raise KeyboardInterrupt, to truncate traceback
--> 904                 raise KeyboardInterrupt("Interrupted by user") from None
    905             except Exception as e:
    906                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

```
start()
```

In [7]:

```
#Question 7: Validating Roman Numerals


thousand = "(?:(M){0,3})?"
hundred  = "(?:(D?(C){0,3})|(CM)|(CD))?"
ten      = "(?:(L?(X){0,3})|(XC)|(XL))?"
unit     = "(?:(V?(I){0,3})|(IX)|(IV))?"

regex_pattern = r"^"+ thousand + hundred + ten + unit + "$" # Do not delete 'r'.

import re
print(str(bool(re.match(regex_pattern, input()))))
```

```
CDXXI
True
```

In [8]:

```
#Question 8: Validating phone numbers

import re

N = int(input())

for _ in range(0, N):

    # should start with a 7, 8, or 9

    print('YES') if re.match(r'[789]\d{9}$', input()) else print('NO')
```

```
2
9587456281
YES
1252478965
NO
```

In [9]:

```
#Question 9: Validating and Parsing Email Addresses

import re
import email.utils
```

```
n = int(input())

match = r'^[a-z][\w\-\.]+@[a-z]+\.[a-z]{1,3}$'

for i in range(0, n):
    addr = email.utils.parseaddr(input())

    if re.search(match, addr[1]):
        print(email.utils.formataddr(addr))
```

```
2
DEXTER <dexter@hotmail.com>
DEXTER <dexter@hotmail.com>
VIRUS <virus!@variable.:p>
```

In [10]:

```
#Question 10: Hex Color Code

import re

for i in range(0, int(input())): # In this case the only valid color codes we want happen on lines that end in a semi colon
    match = re.findall(r"(\#[a-f0-9]{3,6})[\;\,\)]{1}", input(), re.IGNORECASE) #ignore case, since hex is both capital and lowercase
    if match:
        for j in list(match):
            print(j)
```

```
11
#BED
{
color: #FfFdF8; background-color:#aef;
#FfFdF8
#aef
font-size: 123px;
background: -webkit-linear-gradient(top, #f9f9f9, #fff);
#f9f9f9
#fff
}
#Cab
{
background-color: #ABC;
#ABC
border: 2px dashed #fff;
#fff
}
```

In [11]:

```
#Question 11: HTML Parser - Part 1


from html.parser import HTMLParser


class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print ('Start :', tag)
        for ele in attrs:
            print ('->', ele[0], '>', ele[1])

    def handle_endtag(self, tag):
        print ('End   :', tag)

    def handle_startendtag(self, tag, attrs):
        print ('Empty :', tag)
        for ele in attrs:
            print ('->', ele[0], '>', ele[1])
```

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

```
2
<html><head><title>HTML Parser - I</title></head>
Start : html
Start : head
Start : title
End   : title
End   : head
<body data-modal-target class='1'><h1>HackerRank</h1><br /></body></html>
Start : body
-> data-modal-target > None
-> class > 1
Start : h1
End   : h1
Empty : br
End   : body
End   : html
```

In [12]:

```
#Question 12: HTML Parser - Part 2

from html.parser import HTMLParser

class MyHTMLParser(HTMLParser):
    def handle_comment(self, data):
        if (len(data.split('\n')) != 1):
            print(">>> Multi-line Comment")
        else:
            print(">>> Single-line Comment")
        print(data.replace("\r", "\n"))
    def handle_data(self, data):
        if data.strip():
            print(">>> Data")
            print(data)

html = ""
for i in range(int(input())):
    html += input().rstrip() + "\n"

parser = MyHTMLParser()
parser.feed(html)
parser.close()
```

```
4
<!--[if IE 9]>IE9-specific content
<![endif]-->
<div> Welcome to HackerRank</div>
<!--[if IE 9]>IE9-specific content<![endif]-->
>>> Multi-line Comment
[if IE 9]>IE9-specific content
<![endif]
>>> Data
 Welcome to HackerRank
>>> Single-line Comment
[if IE 9]>IE9-specific content<![endif]
```

In [13]:

```
#Question 13: Detect HTML Tags, Attributes and Attribute Values

from html.parser import HTMLParser


N = int(input())

class MyHTMLParser(HTMLParser):

    def handle_starttag(self, tag, attrs):
```

```
        print(tag)
        [print('-> {} > {}'.format(*attr)) for attr in attrs]

html = '\n'.join([input() for x in range(0, N)])
parser = MyHTMLParser()
parser.feed(html)
parser.close()
```

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

In [14]:

```
#Question 14: Validating UID

import re

N = int(input())

# Declare the patterns
norepeats = r'(?!.*(.).*\1)'
uppercase = r'(?=(?:.*[A-Z]+){2,})'
digits = r'(?=(?:.*\d){3,})'
alphanumeric = r'[\w]{10}'
validity = [norepeats, uppercase, digits, alphanumeric]

for i in range(0, N):
    uid = input()
    print('Valid') if all([re.match(v, uid) for v in validity]) else print('Invalid')
```

```
2
B1CD102354
Invalid
B1CDEF2354
Valid
```

In [15]:

```
#Question 15: Validating Credit Card Numbers

import re

N = int(input())

# Starts with 4 or 5 or 6, consists of either 4 groups of 4 (split by a hyphen) or no gro
ups at all
pattern = r'[456]\d{3}(-?\d{4}){3}$'

# No digit repeats more than 4 times
norepeats = r'((\d)-?(?!(-?\2){3})){16}'

matcher = pattern, norepeats

for i in range(0, N):
```

```
    num = input()
    print('Valid') if all(re.match(m, num) for m in matcher) else print('Invalid')
```

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

In [16]:

```
#Question 16: Validating Postal Codes

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

```
110000
False
```

In [17]:

```
#Question 17: Matrix Script


#!/bin/python3

import math
import os
import random
import re
import sys




first_multiple_input = input().rstrip().split()

n = int(first_multiple_input[0])

m = int(first_multiple_input[1])

matrix = []

for _ in range(n):
    matrix_item = input()
    matrix.append(matrix_item)

x = ''.join(''.join(e) for e in zip(*matrix))
result = re.sub(r'([a-zA-Z0-9])([^a-zA-Z0-9]+)(?=[a-zA-Z0-9])', r'\1 ', x)
print (result)
```

```
7 3
Tsi
h%x
i #
sM
$a
```

```
#t%
ir!
This is Matr
```

In [ ]:

```
```