Programming Task - 5

```
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
```

```
import re
m = re.findall(r"([A-Za-z0-9])\1+",input())
if m:
    print(m[0])
else:
    print(-1)
```

```
..12345678910111213141516171820212223
1
```

```
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']))
```

```
rabcdeefgyYhFjkIoomnpOeorteeeeet
ee
Ioo
Oeo
eeeee
```

```
S = input()
k = input()
import re
pattern = re.compile(k)
r = pattern.search(S)
if not r: print("(-1, -1)")
while r:
    print("({0}, {1})".format(r.start(), r.end() - 1))
    r = pattern.search(S,r.start() + 1)
```

```
aaadaa aa
aa
(0, 1)
(1, 2)
(4, 5)
(7, 8)
```

```
import re
[ print(\
    re.sub(r'(?<= )\|\|(?= )', "or", (\
    re.sub(r'(?<= )\&\&(?= )', "and", \
    input()))))) \
    for _ in range(int(input()))\
]
```

```
regex_pattern = r"" # Do not delete 'r'.
thousand = 'M{0,3}'
hundred = '(C[MD]|D?C{0,3})'
ten = '(X[CL]|L?X{0,3})'
digit = '(I[VX]|V?I{0,3})'
regex_pattern = r"%s%s%s%s$" % (thousand, hundred, ten, digit)
import re
print(str(bool(re.match(regex_pattern, input()))))
```

```
import re
for _ in range(int(input())):
    if re.match(r'[789]\d{9}$',input()):
        print('YES')
    else:
        print('NO')
```

```
import re
n = int(input())
for _ in range(n):
    x, y = input().split(' ')
    m = re.match(r'<[A-Za-z](\w|-|\.|_)+@[A-Za-z]+\.[A-Za-z]{1,3}>', y)
    if m:
        print(x,y)
```

```
import re
N=int(input())

for i in range(0,N):
    s=input()

    x=s.split()

    if len(x)>1  and  '{' not in x:
        x=re.findall(r'#[a-fA-F0-9]{3,6}',s)
        [print(i) for  i in x]
```

```
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])

MyParser = MyHTMLParser()
MyParser.feed(''.join([input().strip() for _ in range(int(input()))]))
```

```
from html.parser import HTMLParser
html = ""
class MyHTMLParser(HTMLParser):
  def handle_comment(self,data):
        if('\n' in data):
            print(">>> Multi-line Comment")
        else:
            print(">>> Single-line Comment")
        print(data)
def handle_data(self,data):
        if(data != '\n'):
            print(">>> Data")
            print(data)



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

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

```
from html.parser import HTMLParser
class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print(tag)
        [print('-> {} > {}'.format(*attr)) for attr in attrs]

html = '\n'.join([input() for _ in range(int(input()))])
parser = MyHTMLParser()
parser.feed(html)
parser.close()
```

```
import re

for _ in range(int(input())):
    u = ''.join(sorted(input()))
    try:
```

```
        assert re.search(r'[A-Z]{2}', u)
        assert re.search(r'\d\d\d', u)
        assert not re.search(r'[^a-zA-Z0-9]', u)
        assert not re.search(r'(.)\1', u)
        assert len(u) == 10
    except:
        print('Invalid')
    else:
        print('Valid')
```

```
import re
TESTER = re.compile(
    r"^"
    r"(?!.*(\d)(-?\1){3})"
    r"[456]"
    r"\d{3}"
    r"(?:-?\d{4}){3}"
    r"$")
for _ in range(int(input().strip())):
    print("Valid" if TESTER.search(input().strip()) else "Invalid")
```

```
regex_integer_in_range = r"_________"   # Do not delete 'r'.
regex_alternating_repetitive_digit_pair = r"_________"  # Do not delete 'r'.
print bool(re.match(r'^(?!(?:.*(.).\1.*){2,})(?!.*(.)(.)\2\3)[1-9]\d{5}$', raw_input()))


import re
P = input()

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

```
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)
encoded_string = "".join([matrix[j][i] for i in range(m) for j in range(n)])
pat = r'(?<=[a-zA-Z0-9])[^a-zA-Z0-9]+(?=[a-zA-Z0-9])'
print(re.sub(pat,' ',encoded_string))
```