**Examples**

Given the string foobarbarfoo:

bar(?=bar) finds the 1st bar ("bar" which has "bar" after it)

bar(?!bar) finds the 2nd bar ("bar" which does not have "bar" after it)

(?<=foo)bar finds the 1st bar ("bar" which has "foo" before it)

(?<!foo)bar finds the 2nd bar ("bar" which does not have "foo" before it)

You can also combine them:

(?<=foo)bar(?=bar) finds the 1st bar ("bar" with "foo" before it and "bar" after it)

**Definitions**

**Hoyana eken thama name eka denne () nathi eken look (ahead or behind kiyala) for example a(?=b) a issarahata ena b pitipassara ena ewain a ganna(so “a” issarahata ena ewa ganna nisa – lookahead kiyala danawa ).**

Look ahead positive (?=)

Find expression A where expression B follows:

A(?=B)

Look ahead negative (?!)

Find expression A where expression B does not follow:

A(?!B)

Look behind positive (?<=)

Find expression A where expression B precedes:

(?<=B)A

Look behind negative (?<!)

Find expression A where expression B does not precede:

(?<!B)A

Atomic groups (?>)

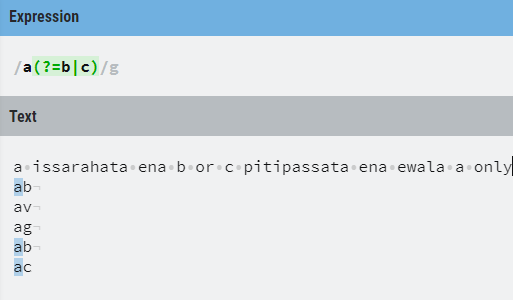
An atomic group exits a group and throws away alternative patterns after the *first* matched pattern inside the group (backtracking is disabled).

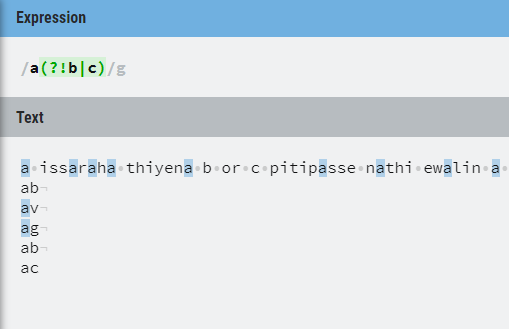
* (?>foo|foot)s applied to foots will match its 1st alternative foo, then fail as s does not immediately follow, and stop as backtracking is disabled

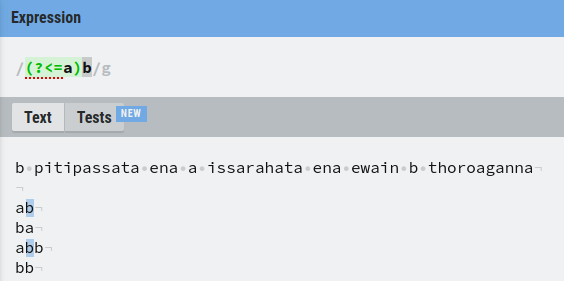
A non-atomic group will allow backtracking; if subsequent matching ahead fails, it will backtrack and use alternative patterns until a match for the entire expression is found or all possibilities are exhausted.

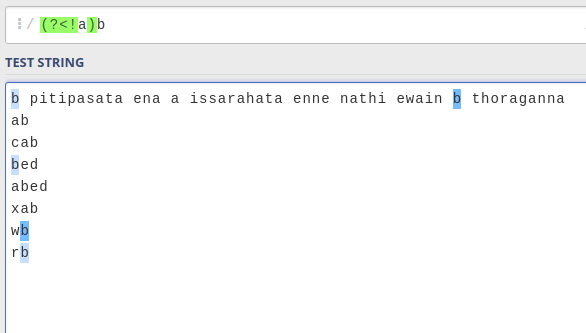
* (foo|foot)s applied to foots will:
  1. match its 1st alternative foo, then fail as s does not immediately follow in foots, and backtrack to its 2nd alternative;
  2. match its 2nd alternative foot, then succeed as s immediately follows in foots, and stop.

**Hamathissema () warahan athule thiyena eka ganne na**

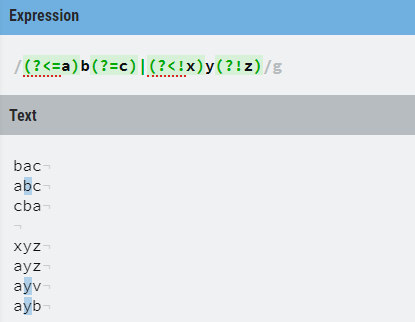


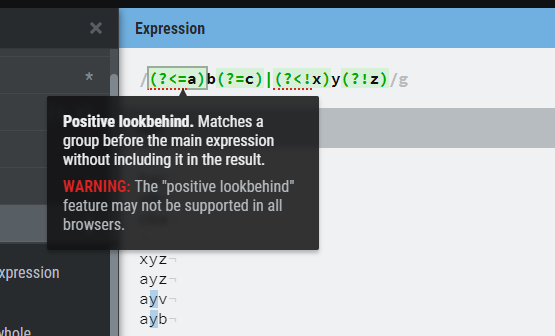


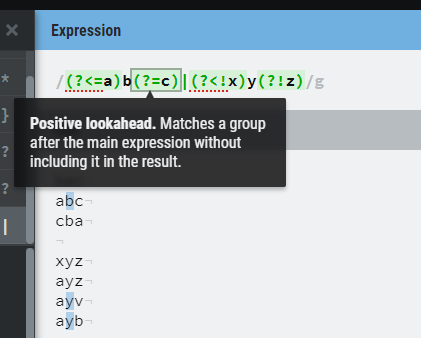


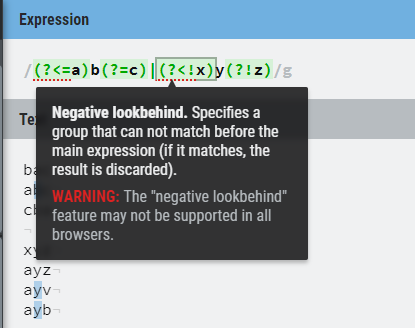


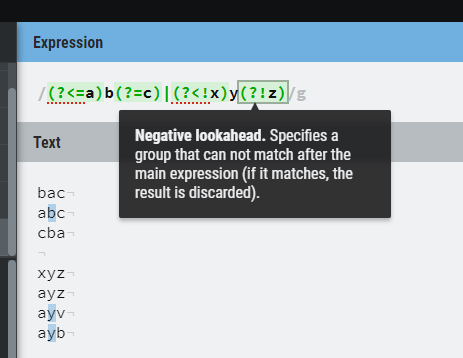
**ALL IN ONE**







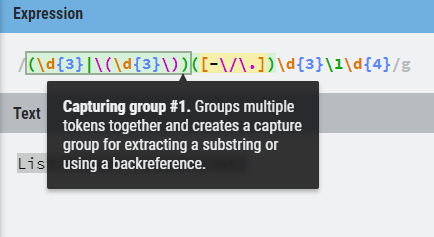




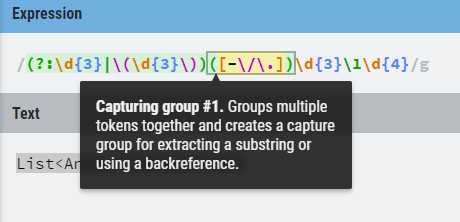
**(?: )=froget me non capturing groups 😊**

It means **only group** but do not remember the grouped part.

By default ( ) tells the regex engine to remember the part of the string that matches the pattern between it. But at times we just want to group a pattern without triggering the regex memory, to do that we use (?: in place of (

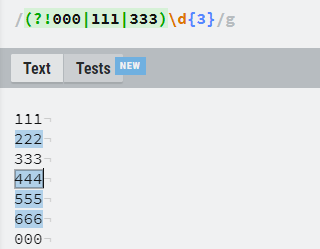


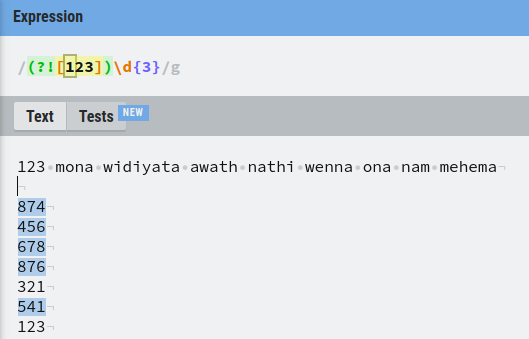
**Ignore the group by using ?:**



**Use negative look ahead as a and gate**

Example:- Select 3 digit no without 111 or 000 or 333





harinam api danna one Look behind positive (?<=)

Find expression A where expression B precedes:

(?<=B)A

ekane but ai uda api lookahead eka danne

reason eka thama (lookahead eke ahead part eka nathi nisa okkoma scan karala balanawa)

\*ahanna kagen hari

**interesting things**

**(?=foo) will not capture "foo". Any look-around assertion (negative- and positive look ahead & behind) will not capture, but only check the presence (or absence) of text. problem**

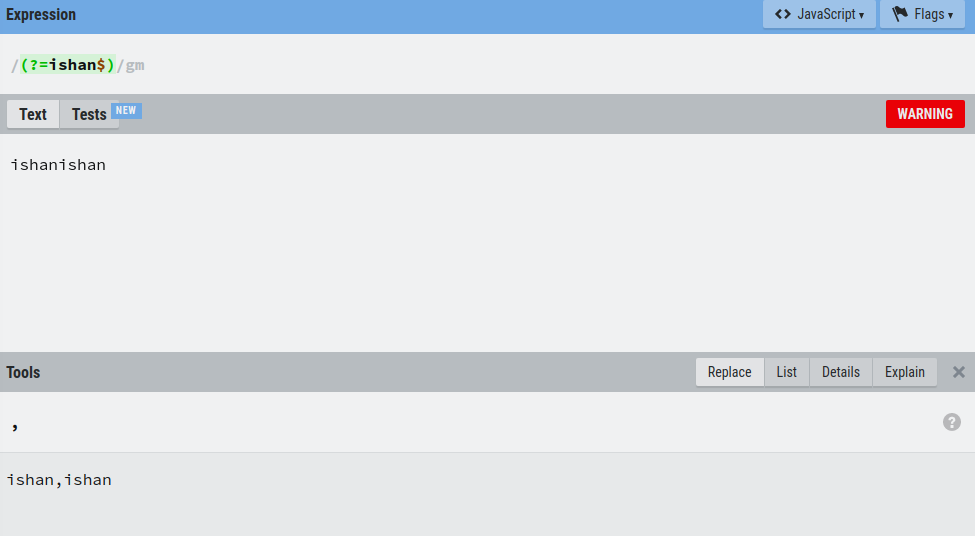
**?**

https://stackoverflow.com/questions/11435436/will-a-lookahead-in-regular-expressions-always-not-capture-or-does-it-depend

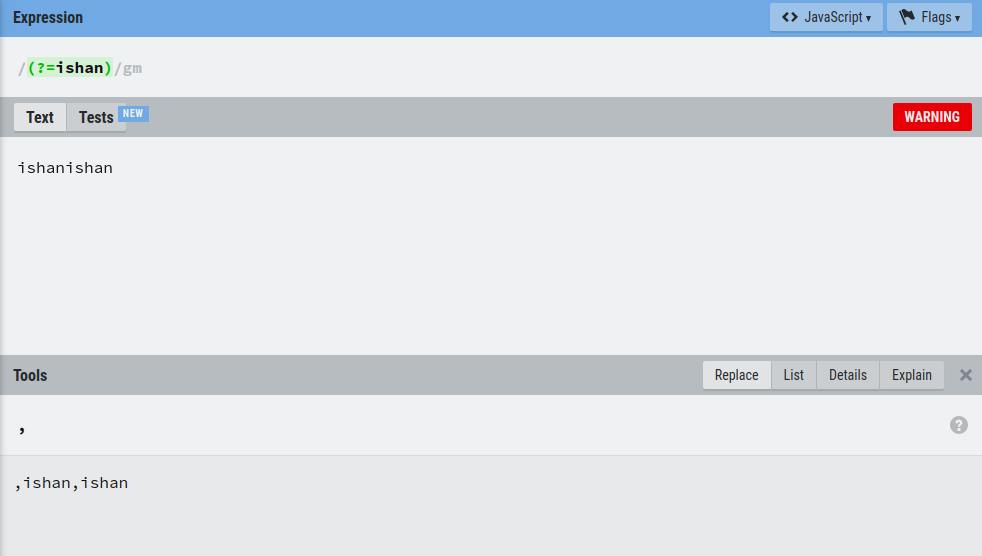
123456 tp 123,456 trick

https://stackoverflow.com/questions/2901102/how-to-print-a-number-with-commas-as-thousands-separators-in-javascript

actually we can make a grouping regex by combining replace.



We put $ because it make (,ishan,ishan) adding , at begining



Xor gate only foo or bar not foobar,barfoo or foofoo (great quection) !

foo: true

bar: true

foofoo: false

barfoo: false

foobarfoo: false

barbar: false

barfoofoo: false

**(?<!foo|bar)(foo|bar)(?!foo|bar)**



<https://stackoverflow.com/questions/247167/exclusive-or-in-regular-expression>

And gate hadana hati

[https://stackoverflow.com/questions/469913/regular-expressions-is-there-an-and-operator/469939#469939](https://stackoverflow.com/questions/469913/regular-expressions-is-there-an-and-operator/469939" \l "469939)

You need to use lookahead as some of the other responders have said, but the lookahead has to account for other characters between its target word and the current match position. **For example:**

**(?=.\*word1)(?=.\*word2)(?=.\*word3)**

The .\* in the first lookahead lets it match however many characters it needs to before it gets to "word1". Then the match position is reset and the second lookahead seeks out "word2". Reset again, and the final part matches "word3"; since it's the last word you're checking for, it isn't necessary that it be in a lookahead, but it doesn't hurt.

In order to match a whole paragraph, you need to anchor the regex at both ends and add a final .\* to consume the remaining characters. Using Perl-style notation, that would be:

(?=.\*WORD1) - there must be WORD1 somewhere in the string

(?=.\*WORD2) - there must be WORD2 somewhere in the string

First we take simple one

(?=a)(ab) kiyann 1expression eka issarahata thiyenawanam witharak deweni ekata yana kiyala

Eka include nokara .but meka special ?<= neme api use

Kale ?= .so apita repeated ly danna wenawa ab check karannanam issaraha nathuwa one thanakanam

?=.\*a danna wenawa

Full search and gate

Like I must find text with a and b and c

With all permutations

Abc

Bac

Cab

(?=.\*a)(?=.\*b)(?=.\*c).\*

1 a thiyenawada balanawa kohehari word eke thiyenawa nam ilaga expression ekata yanawa

2 b thiyenawada balanawa kohehari word eke

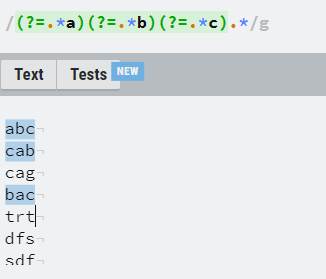
nam ilaga expression ekata yanawa

3 c thiyenawada balanawa kohehari word eke

nam ilaga expression ekata yanawa

123 ok nam .\* execute wenawa abc|bac okkoma permutation denna bari nisa thama .\* denne

Kalin check wela ena nisa awulak na



Regex to find non repeat

Matcher m= Pattern.compile("(\\w)(\\1)\*").matcher("aaabbcccdeeef");

ArrayList<String> firstNonrepeat=new ArrayList();

while (m.find())if(m.group().toString().length()==1)firstNonrepeat.add(m.group());

System.out.println(firstNonrepeat.get(0));