Regex lookahead, lookbehind and atomic groups

Ask Question

315

300

I found these things in my regex body but I haven't got a clue what I can use them for. Does somebody have examples so I can try to understand how they work?

```
(?!) - negative lookahead
(?=) - positive lookahead
(?<=) - positive lookbehind
(?<!) - negative lookbehind

(?>) - atomic group
```

regex lookaround

edited Oct 5 '15 at 17:14
grenierm5
186 3 13

asked Jun 4 '10 at 10:56
Spidfire
2,784 5 22 31

locked by Samuel Liew ♦ Jul 3 '18 at 1:30

This question's answers are a collaborative effort: if you see something that can be improved, just edit the answer to improve it! *No additional answers can be added here*

18 Why doesn't the regex website have some simple table like this? Instead they have blocks of text explaining only. regular-expressions.info/lookaround.html – Whitecat Aug 22 '16 at 17:30

3 @Whitecat Try: regex101.com regexr.com – Andrew Mar 28 '17 at 14:18

3 Answers

595

Examples

Given the string `foobarbarfoo` :

```
bar(?=bar)     finds the 1st bar ("ba
bar(?!bar)     finds the 2nd bar ("ba
(?<=foo)bar    finds the 1st bar ("ba
(?<!foo)bar    finds the 2nd bar ("ba
```

You can also combine them:

```
(?<=foo)bar(?=bar)    finds the 1st b
```

Definitions

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.

Some resources

- http://www.regular-expressions.info/lookaround.html
- http://www.rexegg.com/regex-lookarounds.html

edited Nov 27 '18 at 19:23
neaumusic
4,916 2 26 48

answered Jun 4 '10 at 11:06
skyfoot
11.5k 6 41 67

1 What do you mean by "finds the second bar" part? There is only one bar in the expression/string. Thanks – ziggy Feb 8 '14 at 11:22

1 @ziggy the string being tested is "foobarbarfoo". As you can see there are two foo and two bar in the string. – skyfoot Feb 12 '14 at 10:56

@ziggy try to go to pythex.org and play a little bit about it. you will understand it totally – stanleyli Mar 30 '15 at 19:09

Place two bars side by side, like, `barbar` in the text on which these regexs will be tried. – Obi Wan - PallavJha May 31 '17 at 13:08

3 Can someone explain when one may need an atomic group? If I only need to match with the first alternative, why would I want to give multiple alternatives? – arviman Aug 9 '17 at

12:27

198

Lookarounds are zero width assertions. They check for a regex (towards right or left of the current position - based on ahead or behind), succeeds or fails when a match is found (based on if it is positive or negative) and discards the matched portion. They don't consume any character - the matching for regex following them (if any), will start at the same cursor position.

Read regular-expression.info for more details.

- Positive lookahead:

Syntax:

```
(?=REGEX_1)REGEX_2
```

Match only if REGEX_1 matches; after matching REGEX_1, the match is discarded and searching for REGEX_2 starts at the same position.

example:

```
(?=[a-z0-9]{4}$)[a-z]{1,2}[0-9]{2,3}
```

REGEX_1 is `[a-z0-9]{4}$` which matches four alphanumeric chars followed by end of line.
REGEX_2 is `[a-z]{1,2}[0-9]{2,3}` which matches one or two letters followed by two or three digits.

REGEX_1 makes sure that the length of string is indeed 4, but doesn't consume any characters so that search for REGEX_2 starts at the same location. Now REGEX_2 makes sure that the string matches some other rules. Without look-ahead it would match strings of length three or five.

- Negative lookahead

Syntax:

```
(?!REGEX_1)REGEX_2
```

Match only if REGEX_1 does not match; after checking REGEX_1, the search for REGEX_2 starts at the same position.

example:

```
(?!.*\bFWORD\b)\w{10,30}$
```

The look-ahead part checks for the `FWORD` in the string and fails if it finds it. If it doesn't find `FWORD` , the look-ahead succeeds and the following part verifies that the string's length is between 10 and 30 and that it contains only word characters `a-zA-Z0-9_`

Look-behind is similar to look-ahead: it just looks behind the current cursor position. Some regex flavors like javascript doesn't support look-behind assertions. And most flavors that support it (PHP, Python etc) require that look-behind portion to have a fixed length.

- Atomic groups basically discards/forgets the subsequent tokens in the group once a token matches. Check this page for examples of atomic groups

edited Aug 24 '16 at 13:04
mike
3,160 3 25 53

answered Jun 4 '10 at 11:23
Amarghosh
48.5k 10 78 112

following your explanation, does not seem to work in javascript, /(?=source)hello/.exec("source...hummhellosource") = null. Is your explanation correct? – Helin Wang Jun 1 '13 at 17:47

@HelinWang That explanation is correct. Your regex expects a string that is both source and hello at the same time! – Amarghosh Jun 4 '13 at 11:54

@jddxf Care to elaborate? – Amarghosh Oct 4 '16 at 5:19

@Amarghosh I agree with "They check for a regex (towards right or left of the current position - based on ahead or behind), succeeds or fails when a match is found (based on if it is positive or negative) and discards is positive or negative) and discards the matched portion.". So lookahead should check for a regex towards right of the current position and the syntax of positive lookahead should be x(?=y) – jddxf Oct 5 '16 at 11:28

@Amarghosh would `(?=REGEX_1)REGEX_2` only match if `REGEX_2` comes *after* `REGEX_1` ? – aandis May 22 '18 at 11:50

0

Grokking lookaround rapidly.
How to distinguish lookahead and lookbehind? Take 2 minutes tour with me:

```
(?=) - positive lookahead
(?<=) - positive lookbehind
```

Suppose

```
    A  B  C #in a line
```

Now, we ask B, Where are you?
B has two solutions to declare it location:

One, B has A ahead and has C bebind
Two, B is ahead(lookahead) of C and behind (lookhehind) A.

As we can see, the behind and ahead are opposite in the two solutions.
Regex is solution Two.

edited Apr 15 '18 at 6:30

answered Apr 4 '18 at 15:08
JawSaw
4,188 1 16 34