

What do 'lazy' and 'greedy' mean in the context of regular expressions?

▲ Could someone explain these two terms in an understandable way?

432

regex

regex-greedy

non-greedy



116

edited Nov 16 '17 at 0:27



smci

16.4k

6

81

111

asked Feb 20 '10 at 6:17



ajsie

30.2k

92

247

362

5 See also stackoverflow.com/questions/3075130/... – polygenelubricants Aug 24 '10 at 11:43

11 Answers

▲ Greedy will consume as much as possible. From <http://www.regular-expressions.info/repeat.html> we see the example of trying to match HTML tags with `<.+>`. Suppose you have the following:

549

```
<em>Hello World</em>
```



You may think that `<.+>` (`.` means *any non newline character* and `+` means *one or more*) would only match the `` and the ``, when in reality it will be very greedy, and go from the first `<` to the last `>`. This means it will match `Hello World` instead of what you wanted.

Making it lazy (`<.+?>`) will prevent this. By adding the `?` after the `+`, we tell it to repeat *as few times as possible*, so the first `>` it comes across, is where we want to stop the matching.

I'd encourage you to download [RegExr](http://www.regextester.com), a great tool that will help you explore Regular Expressions - I use it all the time.

edited Sep 6 '17 at 19:39



answered Feb 20 '10 at 6:22



By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

- 2 so if you use greedy will u have 3 (1 element + 2 tags) matches or just 1 match (1 element)? – [ajsie](#) Feb 20 '10 at 6:27
- 9 It would match only 1 time, starting from the first < and ending with the last >. – [Sampson](#) Feb 20 '10 at 6:28
- 3 But making it lazy would match twice, giving us both the opening and closing tag, ignoring the text in between (since it doesn't fit the expression). – [Sampson](#) Feb 20 '10 at 6:29
- 55 [regex101.com](#) is like a jsfiddle for regex. – [nackjicholson](#) Oct 27 '14 at 5:18
- 6 Just to add that there is a greedy way to go about it, too: <[^>]+> [regex101.com/r/IW0cY6/1](#) – [alanbuchanan](#) Jun 15 '15 at 12:57

2

Best shown by example. String. 192.168.1.1 and a greedy regex `\b.+ \b` You might think this would give you the 1st octet but is actually matches against the whole string. WHY!!! Because the `+` is greedy and a greedy match matches every character in '192.168.1.1' until it reaches the end of the string. This is the important bit!!! Now it starts to backtrack one character at a time until it finds a match for the 3rd token (`\b`).

If the string a 4GB text file and 192.168.1.1 was at the start you could easily see how this backtracking would cause an issue.

To make a regex non greedy (lazy) put a question mark after your greedy search e.g `*? ?? +?` What happens now is token 2 (`+?`) finds a match, regex moves along a character and then tries the next token (`\b`) rather than token 2 (`+?`). So it creeps along gingerly.

answered Mar 12 '18 at 10:54



[Jason Alcock](#)

21 1

3

Greedy means it will consume your pattern until there are none of them left and it can look no further.

Lazy will stop as soon as it will encounter the first pattern you requested.

One common example that I often encounter is `\s*-\s*?` of a regex `([0-9]{2}\s*-\s*[0-9]{7})`

The first `\s*` is classified as greedy because of `*` and will look as many white spaces as possible after the digits are encountered and then look for a dash character "-". Where as the second `\s*?` is lazy because of the present of `*?` which means that it will look the first white space character and stop right there.

answered Feb 6 '18 at 15:41

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).



2



Greedy matching. The default behavior of regular expressions is to be greedy. That means it tries to extract as much as possible until it conforms to a pattern even when a smaller part would have been syntactically sufficient.

Example:

```
import re
text = "<body>Regex Greedy Matching Example </body>"
re.findall('<.*>', text)
#> ['<body>Regex Greedy Matching Example </body>']
```

Instead of matching till the first occurrence of '>', it extracted the whole string. This is the default greedy or 'take it all' behavior of regex.

Lazy matching, on the other hand, 'takes as little as possible'. This can be effected by adding a `?` at the end of the pattern.

Example:

```
re.findall('<.*?>', text)
#> ['<body>', '</body>']
```

If you want only the first match to be retrieved, use the search method instead.

```
re.search('<.*?>', text).group()
#> '<body>'
```

Source: [Python Regex Examples](#)

edited Jan 21 '18 at 5:41

answered Jan 21 '18 at 5:35



Selva

1,184 12 16



'Greedy' means match longest possible string.

'Lazy' means match shortest possible string.

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

edited Nov 16 '17 at 0:27

answered Feb 20 '10 at 6:19



smci

16.4k

6

81

111



slebetman

73.5k

16

99

125

73 Brilliant, so lazy will stop as soon as the condition I is satisfied, but greedy means it will stop only once the condition I is not satisfied any more? – Andrew S Feb 23 '14 at 21:27

3 For all people reading the post: greedy or lazy quantifiers by themselves won't match the longest/shortest possible substring. You would have to use either a [tempered greedy token](#), or use non-regex approaches. – Wiktor Stribiżew Oct 15 '16 at 21:29

3 @AndrewS Don't be confused by the double | in the example. It's rather lazy will match the shortest possible substring while greedy will match the longest possible. Greedy `h.+1` matches `'he1o1'` in `'he1o1o'` but the lazy `h.+?1` matches `'he1'` . – v.shashenko Mar 21 '17 at 16:38

3 @FloatingRock: No. `x?` means `x` is optional but `+?` is a different syntax. It means stop looking after you find something that matches - lazy matching. – slebetman Apr 14 '17 at 12:56

1 @FloatingRock: As for how you differentiate the different syntax, simple: `?` means optional and `+?` means lazy. Therefore `\+?` means `+` is optional. – slebetman Apr 14 '17 at 12:57

86

Greedy quantifier	Lazy quantifier	Description
<code>*</code>	<code>*?</code>	Star Quantifier: 0 or more
<code>+</code>	<code>+</code>	Plus Quantifier: 1 or more
<code>?</code>	<code>??</code>	Optional Quantifier: 0 or 1
<code>{n}</code>	<code>{n}?</code>	Quantifier: exactly n
<code>{n,}</code>	<code>{n,}?</code>	Quantifier: n or more
<code>{n,m}</code>	<code>{n,m}?</code>	Quantifier: between n and m

Add a `?` to a quantifier to make it ungreedy i.e lazy.

Example:

test string : `stackoverflow`

greedy reg expression : `s.*o` output: `stackoverflow`

lazy reg expression : `s.*?o` output: `stackoverflow`

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).



- 2 is not ?? equivalent to ? . Similarly , isn't {n}? equivalent to {n} – [Breaking Benjamin](#) Sep 2 '16 at 8:07
- 2 @BreakingBenjamin: no ?? is not equivalent to ? , when it has a choice to either return 0 or 1 occurrence, it will pick the 0 (lazy) alternative. To see the difference, compare `re.match('(f)?(.*)', 'food').groups()` to `re.match('(f)??(.*)', 'food').groups()` . In the latter, `(f)??` will not match the leading 'f' even though it could. Hence the 'f' will get matched by the second '.*' capture group. I'm sure you can construct an example with '{n}?' too. Admittedly these two are very-rarely-used. – [smci](#) Nov 16 '17 at 0:42

As far as I know, most regex engine is greedy by default. Add a question mark at the end of quantifier will enable lazy match.

12

As @Andre S mentioned in comment.

- Greedy: Keep searching until condition is not satisfied.
- Lazy: Stop searching once condition is satisfied.

Refer to the example below for what is greedy and what is lazy.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Test {
    public static void main(String args[]){
        String money = "100000000999";
        String greedyRegex = "100(0*)";
        Pattern pattern = Pattern.compile(greedyRegex);
        Matcher matcher = pattern.matcher(money);
        while(matcher.find()){
            System.out.println("I'm greedy and I want " + matcher.group() + " dollars.
This is the most I can get.");
        }

        String lazyRegex = "100(0*?)";
        pattern = Pattern.compile(lazyRegex);
        matcher = pattern.matcher(money);
        while(matcher.find()){
            System.out.println("I'm too lazy to get so much money, only " +
matcher.group() + " dollars is enough for me");
        }
    }
}
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

The result is:

I'm greedy and I want 100000000 dollars. This is the most I can get.

I'm too lazy to get so much money, only 100 dollars is enough for me

edited Nov 10 '16 at 1:49

answered Nov 9 '16 at 16:39



[Gearon](#)

5,096 3 20 40

3 I really like your example. – [Xatenev](#) Mar 13 '17 at 11:46

try to understand the following behavior:

-2

```
var input = "0014.2";

Regex r1 = new Regex(@"\d+.{0,1}\d+");
Regex r2 = new Regex(@"\d*.{0,1}\d*");

Console.WriteLine(r1.Match(input).Value); // "0014.2"
Console.WriteLine(r2.Match(input).Value); // "0014.2"

input = " 0014.2";

Console.WriteLine(r1.Match(input).Value); // "0014.2"
Console.WriteLine(r2.Match(input).Value); // " 0014"

input = " 0014.2";

Console.WriteLine(r1.Match(input).Value); // "0014.2"
Console.WriteLine(r2.Match(input).Value); // ""
```

answered Oct 30 '16 at 6:31



[FrankyHollywood](#)

837 8 13

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

9

Taken From www.regular-expressions.info

Greediness: Greedy quantifiers first tries to repeat the token as many times as possible, and gradually gives up matches as the engine backtracks to find an overall match.

Laziness: Lazy quantifier first repeats the token as few times as required, and gradually expands the match as the engine backtracks through the regex to find an overall match.

answered Oct 19 '14 at 8:34



[Suganthan Madhavan Pillai](#)

3,002 7 34 55

49

Greedy means your expression will match as large a group as possible, lazy means it will match the smallest group possible. For this string:

abcdefghijklmc

and this expression:

a.*c

A greedy match will match the whole string, and a lazy match will match just the first abc .

edited Feb 20 '10 at 6:30

answered Feb 20 '10 at 6:19



[Carl Norum](#)

180k 22 355 431

+1 [regular-expressions.info/repeat.html](http://www.regular-expressions.info/repeat.html) – [Sampson](#) Feb 20 '10 at 6:19

From [Regular expression](#)

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).



By using a lazy quantifier, the expression tries the minimal match first.

answered Feb 20 '10 at 6:21



[Adriaan Stander](#)

136k 25 248 262

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).