## Named Capturing Groups and Backreferences

Nearly all modern regular expression engines support numbered capturing groups and numbered backreferences. Long regular expressions with lots of groups and backreferences may be hard to read. They can be particularly difficult to maintain as adding or removing a capturing group in the middle of the regex upsets the numbers of all the groups that follow the added or removed group.

Python's re module was the first to offer a solution: named capturing groups and named backreferences. `(?P<name>group)` captures the match of `group` into the backreference "name". `name` must be an alphanumeric sequence starting with a letter. `group` can be any regular expression. You can reference the contents of the group with the named backreference `(?P=name)`. The question mark, P, angle brackets, and equals signs are all part of the syntax. Though the syntax for the named backreference uses parentheses, it's just a backreference that doesn't do any capturing or grouping. The HTML tags example can be written as `<(?P<tag>[A-Z][A-Z0-9]*)\b[^>]*>.*?</(?P=tag)>`.

The .NET framework also supports named capture. Microsoft's developers invented their own syntax, rather than follow the one pioneered by Python and copied by PCRE (the only two regex engines that supported named capture at that time). `(?<name>group)` or `(?'name'group)` captures the match of `group` into the backreference "name". The named backreference is `\k<name>` or `\k'name'`. Compared with Python, there is no P in the syntax for named groups. The syntax for named backreferences is more similar to that of numbered backreferences than to what Python uses. You can use single quotes or angle brackets around the name. This makes absolutely no difference in the regex. You can use both styles interchangeably. The syntax using angle brackets is preferable in programming languages that use single quotes to delimit strings, while the syntax using single quotes is preferable when adding your regex to an XML file, as this minimizes the amount of escaping you have to do to format your regex as a literal string or as XML content.

Because Python and .NET introduced their own syntax, we refer to these two variants as the "Python syntax" and the ".NET syntax" for named capture and named backreferences. Today, many other regex flavors have copied this syntax.

Perl 5.10 added support for both the Python and .NET syntax for named capture and backreferences. It also adds two more syntactic variants for named backreferences: `\k{one}` and `\g{two}`. There's no difference between the five syntaxes for named backreferences in Perl. All can be used interchangeably. In the replacement text, you can interpolate the variable `$+{name}` to insert the text matched by a named capturing group.

PCRE 7.2 and later support all the syntax for named capture and backreferences that Perl 5.10 supports. Old versions of PCRE supported the Python syntax, even though that was not "Perl-compatible" at the time. Languages like PHP, Delphi, and R that implement their regex support using PCRE also support all this syntax. Unfortunately, neither PHP or R support named references in the replacement text. You'll have to use numbered references to the named groups. PCRE does not support search-and-replace at all.

Java 7 and XRegExp copied the .NET syntax, but only the variant with angle brackets. Ruby 1.9 and supports both variants of the .NET syntax. The JGsoft flavor supports the Python syntax and both variants of the .NET syntax.

Boost 1.42 and later support named capturing groups using the .NET syntax with angle brackets or quotes and named backreferences using the \g syntax with curly braces from Perl 5.10. Boost 1.47 additionally supports backreferences using the \k syntax with angle brackets and quotes from .NET. Boost 1.47 allowed these variants to multiply. Boost 1.47 allows named and numbered backreferences to be specified with \g or \k and with curly braces, angle brackets, or quotes. So Boost 1.47 and later have six variations of the backreference syntax on top of the basic \1 syntax. This puts Boost in conflict with Ruby, PCRE, PHP, R, and JGsoft which treat \g with angle brackets or quotes as a subroutine call.

## Numbers for Named Capturing Groups

Mixing named and numbered capturing groups is not recommended because flavors are inconsistent in how the groups are numbered. If a group doesn't need to have a name, make it non-capturing using the `(?:group)` syntax. In .NET you can make all unnamed groups non-capturing by setting `RegexOptions.ExplicitCapture`. In Delphi, set `roExplicitCapture`. With XRegExp, use the `/n` flag. Perl supports `/n` starting with Perl 5.22. With PCRE, set `PCRE_NO_AUTO_CAPTURE`. The JGsoft flavor and .NET support the `(?n)` mode modifier. If you make all unnamed groups non-capturing, you can skip this section and save yourself a headache.

Most flavors number both named and unnamed capturing groups by counting their opening parentheses from left to right. Adding a named capturing group to an existing regex still upsets the numbers of the unnamed groups. In .NET, however, unnamed capturing groups are assigned numbers first, counting their opening parentheses from left to right, skipping all named groups. After that, named groups are assigned the numbers that follow by counting the opening parentheses of the named groups from left to right.

The JGsoft regex engine copied the Python and the .NET syntax at a time when only Python and PCRE used the Python syntax, and only .NET used the .NET syntax. Therefore it also copied the numbering behavior of both Python and .NET, so that regexes intended for Python and .NET would keep their behavior. It numbers Python-style named groups along unnamed ones, like Python does. It numbers .NET-style named groups afterward, like .NET does. These rules apply even when you mix both styles in the same regex.

As an example, the regex `(a)(?P<x>b)(c)(?P<y>d)` matches `abcd` as expected. If you do a search-and-replace with this regex and the replacement `\1\2\3\4` or `$1$2$3$4` (depending on the flavor), you will get `abcd`. All four groups were numbered from left to right, from one till four.

Things are a bit more complicated with the .NET framework. The regex `(a)(?<x>b)(c)(?<y>d)` again matches `abcd`. However, if you do a search-and-replace with `$1$2$3$4` as the replacement, you will get `acbd`. First, the unnamed groups `(a)` and `(c)` got the numbers 1 and 2. Then the named groups "x" and "y" got the numbers 3 and 4.

In all other flavors that copied the .NET syntax the regex `(a)(?<x>b)(c)(?<y>d)` still matches `abcd`. But in all those flavors, except the JGsoft flavor, the replacement `\1\2\3\4` or `$1$2$3$4` (depending on the flavor) gets you `abcd`. All four groups were numbered from left to right.

In PowerGREP, which uses the JGsoft flavor, named capturing groups play a special role. Groups with the same name are shared between all regular expressions and replacement texts in the same PowerGREP action. This allows captured by a named capturing group in one part of the action to be referenced in a later part of the action. Because of this, PowerGREP does not allow numbered references to named capturing groups at all. When mixing named and numbered groups in a regex, the numbered groups are still numbered following the Python and .NET rules, like the JGsoft flavor always does.

## Multiple Groups with The Same Name

The .NET framework and the JGsoft flavor allow multiple groups in the regular expression to have the same name. All groups with the same name share the same storage for the text they match. Thus, a backreference to that name matches the text that was matched by the group with that name that most recently captured something. A reference to the name in the replacement text inserts the text matched by the group with that name that was the last one to capture something.

Perl and Ruby also allow groups with the same name. But these flavors only use smoke and mirrors to make it look like the all the groups with the same name act as one. In reality, the groups are separate. In Perl, a backreference matches the text captured by the leftmost group in the regex with that name that matched something. In Ruby, a backreference matches the text captured by any of the groups with that name. Backtracking makes Ruby try all the groups.

So in Perl and Ruby, you can only meaningfully use groups with the same name if they are in separate alternatives in the regex, so that only one of the groups with that name could ever capture any text. Then backreferences to that group sensibly match the text captured by the group.

For example, if you want to match “a” followed by a digit 0..5, or “b” followed by a digit 4..7, and you only care about the digit, you could use the regex `a(?<digit>[0-5])|b(?<digit>[4-7])`. In these four flavors, the group named “digit” will then give you the digit 0..7 that was matched, regardless of the letter. If you want this match to be followed by c and the exact same digit, you could use `(?:a(?<digit>[0-5])|b(?<digit>[4-7]))c\k<digit>`

PCRE does not allow duplicate named groups by default. PCRE 6.7 and later allow them if you turn on that option or use the mode modifier `(?J)`. But prior to PCRE 8.36 that wasn’t very useful as backreferences always pointed to the first capturing group with that name in the regex regardless of whether it participated in the match. Starting with PCRE 8.36 (and thus PHP 5.6.9 and R 3.1.3) and also in PCRE2, backreferences point to the first group with that name that actually participated in the match. Though PCRE and Perl handle duplicate groups in opposite directions the end result is the same if you follow the advice to only use groups with the same name in separate alternatives.

Boost allows duplicate named groups. Prior to Boost 1.47 that wasn’t useful as backreferences always pointed to the last group with that name that appears before the backreference in the regex. In Boost 1.47 and later backreferences point to the first group with that name that actually participated in the match just like in PCRE 8.36 and later.

Python, Java, and XRegExp 3 do not allow multiple groups to use the same name. Doing so will give a regex compilation error. XRegExp 2 allowed them, but did not handle them correctly.

In Perl 5.10, PCRE 8.00, PHP 5.2.14, and Boost 1.42 (or later versions of these) it is best to use a branch reset group when you want groups in different alternatives to have the same name, as in `(?|a(?<digit>[0-5])|b(?<digit>[4-7]))c\k<digit>`. With this special syntax—group opened with `(?|` instead of `(?:`—the two groups named “digit” really are one and the same group. Then backreferences to that group are always handled correctly and consistently between these flavors. (Older versions of PCRE and PHP may support branch reset groups, but don’t correctly handle duplicate names in branch reset groups.)

## Make a Donation



Page URL: https://www.regular-expressions.info/named.html
Page last updated: 22 November 2019
Site last updated: 08 April 2020