Literal Characters

The most basic regular expression consists of a single literal character, such as `a`. It matches the first occurrence of that character in the string. If the string is `Jack is a boy`, it matches the `a` after the `J`. The fact that this `a` is in the middle of the word does not matter to the regex engine. If it matters to you, you will need to tell that to the regex engine by using word boundaries. We will get to that later.

This regex can match the second `a` too. It only does so when you tell the regex engine to start searching through the string after the first match. In a text editor, you can do so by using its "Find Next" or "Search Forward" function. In a programming language, there is usually a separate function that you can call to continue searching through the string after the previous match.

Similarly, the regex `cat` matches `cat` in `About cats and dogs`. This regular expression consists of a series of three literal characters. This is like saying to the regex engine: find a `c`, immediately followed by an `a`, immediately followed by a `t`.

Note that regex engines are case sensitive by default. `cat` does not match `Cat`, unless you tell the regex engine to ignore differences in case.

Special Characters

Because we want to do more than simply search for literal pieces of text, we need to reserve certain characters for special use. In the regex flavors discussed in this tutorial, there are 12 characters with special meanings: the backslash `\`, the caret `^`, the dollar sign `$`, the period or dot `.`, the vertical bar or pipe symbol `|`, the question mark `?`, the asterisk or star `*`, the plus sign `+`, the opening parenthesis `(`, the closing parenthesis `)`, the opening square bracket `[`, and the opening curly brace `{`, These special characters are often called "metacharacters". Most of them are errors when used alone.

If you want to use any of these characters as a literal in a regex, you need to escape them with a backslash. If you want to match `1+1=2`, the correct regex is `1\+1=2`. Otherwise, the plus sign has a special meaning.

Note that `1+1=2`, with the backslash omitted, is a valid regex. So you won't get an error message. But it doesn't match `1+1=2`. It would match `111=2` in `123+111=234`, due to the special meaning of the plus character.

If you forget to escape a special character where its use is not allowed, such as in `+1`, then you will get an error message.

Most regular expression flavors treat the brace `{` as a literal character, unless it is part of a repetition operator like `a{1,3}`. So you generally do not need to escape it with a backslash, though you can do so if you want. But there are a few exceptions. Java requires literal opening braces to be escaped. Boost and std::regex require all literal braces to be escaped.

`]` is a literal outside character classes. Different rules apply inside character classes. Those are discussed in the topic about character classes. Again, there are exceptions. std::regex and Ruby require closing square brackets to be escaped even outside character classes.

All other characters should not be escaped with a backslash. That is because the backslash is also a special character. The backslash in combination with a literal character can create a regex token with a special meaning. E.g. `\d` is a shorthand that matches a single digit from `0` to `9`.

Escaping a single metacharacter with a backslash works in all regular expression flavors. Some flavors also support the `\Q…\E` escape sequence. All the characters between the `\Q` and the `\E` are interpreted as literal characters. E.g. `\Q*\d+*\E` matches the literal text `*\d+*`. The `\E` may be omitted at the end of the regex, so `\Q*\d+*` is the

same as `\Q*\d+*\E`. This syntax is supported by the JGsoft engine, Perl, PCRE, PHP, Delphi, Java, both inside and outside character classes. Java 4 and 5 have bugs that cause `\Q...\E` to misbehave, however, so you shouldn't use this syntax with Java. Boost supports it outside character classes, but not inside.

Special Characters and Programming Languages

If you are a programmer, you may be surprised that characters like the single quote and double quote are not special characters. That is correct. When using a regular expression or grep tool like PowerGREP or the search function of a text editor like EditPad Pro, you should not escape or repeat the quote characters like you do in a programming language.

In your source code, you have to keep in mind which characters get special treatment inside strings by your programming language. That is because those characters are processed by the compiler, before the regex library sees the string. So the regex `1\+1=2` must be written as `"1\\+1=2"` in C++ code. The C++ compiler turns the escaped backslash in the source code into a single backslash in the string that is passed on to the regex library. To match `c:\temp`, you need to use the regex `c:\\temp`. As a string in C++ source code, this regex becomes `"c:\\\\temp"`. Four backslashes to match a single one indeed.

See the tools and languages section of this website for more information on how to use regular expressions in various programming languages.

Make a Donation

Did this website just save you a trip to the bookstore? Please make a donation to support this site, and you'll get a **lifetime of advertisement-free access** to this site!

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