WITHYOU WITHME

Regex

Boolean	
|	A vertical bar separates alternatives. For example, gray|grey can match "gray" or "grey".
Grouping	
( )	Parentheses are used to define the scope and precedence of the operators (among other uses). For example, gray|grey and gr(a|e)y are equivalent patterns which both describe the set of "gray" or "grey".
Quantifiers	
?	The question mark indicates *zero or one* occurrences of the preceding element. For example, `colou?r` matches both "color" and "colour".
*****	The asterisk indicates *zero or more* occurrences of the preceding element. For example, `ab*c` matches "ac", "abc", "abbc", "abbbc", and so on.
+	The plus sign indicates *one or more* occurrences of the preceding element. For example, `ab+c` matches "abc", "abbc", "abbbc", and so on, but not "ac".
{n}	The preceding item is matched exactly *n* times.
{min,}	The preceding item is matched *min* or more times.
{,max}	The preceding item is matched up to *max* times.
{min,max}	The preceding item is matched at least *min* times, but not more than *max* times.
Wildcard	
.	The wildcard **.** matches any character.

WITHYOU WITHME

Metacharacter	Description
`^`	Matches the starting position within the string. In line-based tools, it matches the starting position of any line.
`.`	Matches any single character (many applications exclude newlines, and exactly which characters are considered newlines is flavor-, character-encoding-, and platform-specific, but it is safe to assume that the line feed character is included). Within POSIX bracket expressions, the dot character matches a literal dot. For example, `a.c` matches "abc", etc., but `[a.c]` matches only "a", ".", or "c".
`[ ]`	A bracket expression. Matches a single character that is contained within the brackets. For example, `[abc]` matches "a", "b", or "c". `[a-z]` specifies a range which matches any lowercase letter from "a" to "z". These forms can be mixed: `[abcx-z]` matches "a", "b", "c", "x", "y", or "z", as does `[a-cx-z]`. The `-` character is treated as a literal character if it is the last or the first (after the `^`, if present) character within the brackets: `[abc-]`, `[-abc]`. Note that backslash escapes are not allowed. The `]` character can be included in a bracket expression if it is the first (after the `^`) character: `[]abc]`.
`[^ ]`	Matches a single character that is not contained within the brackets. For example, `[^abc]` matches any character other than "a", "b", or "c". `[^a-z]` matches any single character that is not a lowercase letter from "a" to "z". Likewise, literal characters and ranges can be mixed.
`$`	Matches the ending position of the string or the position just before a string-ending newline. In line-based tools, it matches the ending position of any line.
`( )`	Defines a marked subexpression. The string matched within the parentheses can be recalled later (see the next entry, `\n`). A marked subexpression is also called a block or capturing group. **BRE mode requires** `\( \)`.
`\n`	Matches what the *n*th marked subexpression matched, where *n* is a digit from 1 to 9. This construct is vaguely defined in the POSIX.2 standard. Some tools allow referencing more than nine capturing groups. Also known as a backreference. **backreferences are only supported in BRE mode**
`*`	Matches the preceding element zero or more times. For example, `ab*c` matches "ac", "abc", "abbbc", etc. `[xyz]*` matches "", "x", "y", "z", "zx", "zyx", "xyzzy", and so on. `(ab)*` matches "", "ab", "abab", "ababab", and so on.
`{m,n}`	Matches the preceding element at least *m* and not more than *n* times. For example, `a{3,5}` matches only "aaa", "aaaa", and "aaaaa". This is not found in a few older instances of regexes. **BRE mode requires** `\{m,n\}`.

Resources:

Wikipedia

W3School Regex PHP

W3School Regex JS

W3School Regex Python

MDN Regex JS

Online tools:

Regexr.com

Regex101.com