

### Anchors (boundaries)

<code>^</code>	Start of string or line
<code>\$</code>	End of string or line
<code>\A</code>	Start of input (ignores 'm' flag)
<code>\Z</code>	End of input (ignores 'm' flag)
<code>\G</code>	End of the previous match
<code>\b</code>	Word boundary (any position preceded or followed - but not both - by a letter, digit or underscore)
<code>\B</code>	Non-word boundary

### Character and Sets

<code>\w</code>	Word	<code>[a-zA-Z0-9_]</code>
<code>\W</code>	Non-word	<code>[^a-zA-Z0-9_]</code>
<code>\d</code>	Digit	<code>[0-9]</code>
<code>\D</code>	Non-digit	
<code>\s</code>	Whitespace (Form-feed, tab, vertical-tab, new line, carriage return and space)	<code>[\f\t\x0b\n\r ]</code>
<code>\S</code>	Non-whitespace	
<code>\x</code>	Hexadecimal digit	<code>\x00=null; \x0d=\r; \x61-\x7a]=[a-z]</code>
<code>\O</code>	Octal digit	
<code>.</code>	Any character (except new line \n)	

### Groups

<code>(...)</code>	Capture group - captures a set of characters for a later expression
<code>(?:...)</code>	Non-capture group - groups an expression but does not capture. e.g. <code>/((?:foo fu)bar)/</code> matches "foobar" or "fubar" without "foo" or "fu" appearing as a captured subpattern
<code>(?=...)</code>	Lookahead - match on the characters following. e.g. <code>/ab(?=c)/</code> match "ab" only when followed by "c"
<code>(?!...)</code>	Negative lookahead - match on characters that aren't following. e.g. <code>/ab(?!c)/</code> match "ab" only when NOT followed by "c"
<code>(?&lt;=...)</code>	Positive look-behind assertion. e.g. <code>/(?&lt;=foo)bar/</code> matches "bar" when preceded by "foo"
<code>(?&lt;!=...)</code>	Negative look-behind assertion. e.g. <code>/(?&lt;!=foo)bar/</code> - matches "bar" when not preceded by "foo"
<code>(?#...)</code>	Comment e.g. <code>(?# This comment is ignored entirely)</code>

### Unicode character support

<code>\x0000-\xFFFF</code>	Unicode hexadecimal character set
<code>\x00-\xFF</code>	ASCII hexadecimal character set
<code>\cA-\cZ</code>	Control characters

Unicode is not fully supported on all platforms. JavaScript prior to ES6 for example allows ASCII hex but not full Unicode hex.

### Special Characters

<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>\v</code>	Vertical tab
<code>\f</code>	Form feed

### Quantifiers

<code>*</code>	Zero or more
<code>+</code>	One or more
<code>?</code>	Zero or One (i.e. optional)
<code>{n}</code>	Exactly 'n' (any number)
<code>{n,}</code>	Minimum ('n' or more)
<code>{n,m}</code>	Range ('n' or more, but less or equal to 'm')

### Flags (expression modifiers)

<code>/m</code>	Multi-line. (Makes <code>^</code> and <code>\$</code> match the start and end of a line respectively)
<code>/s</code>	Treat input as a single line. (Makes <code>!</code> match new lines as well)
<code>/i</code>	Case insensitive pattern matching.
<code>/g</code>	Global matching. (Don't stop after first match in a replacement function)
<code>/x</code>	Extended matching. (disregard white-space not explicitly escaped, and allow comments starting with <code>#</code> )

### Escape Characters

In regular expressions, the following characters have special meaning and must be escaped: `^ $ [ { ( ) < > . * \ + | ?`. Additionally the hyphen (`-`) and close square bracket (`]`) must be escaped when in an expression set (`[ . . . ]`).  
e.g. `/\(\d{3}\)\ \d{4}[\- ]\d{4}/` matches "(nnn) nnnn-nnnn" or "(nnn) nnnn nnnn" (where n is a numeric digit).



By **doublehelix**

Published 30th January, 2017.  
Last updated 25th February, 2020.  
Page 1 of 1.

Sponsored by **ApolloPad.com**  
Everyone has a novel in them. Finish Yours!  
<https://apollopad.com>