$ cd reference / Regular expression syntax

Regular expression syntax (JavaScript)

This is the syntax of JavaScript's RegExp. Most of it is shared with PCRE, Python and Java; the differences that bite are noted where they apply.

[01] Characters and classes

SYNTAX	MATCHES
`.`	any character except line terminators; any character at all with the s flag
`\d \D`	ASCII digit 0-9 / anything else
`\w \W`	ASCII letter, digit or underscore / anything else
`\s \S`	whitespace, including Unicode spaces and line breaks / anything else
`[abc]`	one of a, b, c
`[^abc]`	any character except a, b, c, including line breaks
`[a-z0-9_-]`	ranges; a hyphen is literal at the start or end
`\p{L} \p{Lu} \p{N}`	Unicode letter / uppercase letter / number; needs u or v
`\p{Script=Greek}`	characters of one script; needs u or v
`\t \n \r`	tab, line feed, carriage return
`\x41 \u0041 \u{1F600}`	by code; the braced form needs u
`\. \* \\`	a literal dot, asterisk, backslash

[02] Anchors and boundaries

SYNTAX	MATCHES
`^`	start of input; start of any line with m
`$`	end of input; end of any line with m
`\b`	between a \w and a non-\w character; based on the ASCII \w
`\B`	any position that is not \b

[03] Quantifiers

GREEDY	LAZY	REPEATS
`*`	`*?`	0 or more
`+`	`+?`	1 or more
`?`	`??`	0 or 1
`{3}`	`{3}?`	exactly 3
`{2,}`	`{2,}?`	2 or more
`{2,5}`	`{2,5}?`	2 to 5

Greedy quantifiers take as much as they can and give back; lazy ones take as little as possible. On `<b>x</b> <b>y</b>`, `<b>.*</b>` matches the whole string and `<b>.*?</b>` matches only the first element. JavaScript has no possessive quantifiers and no atomic groups.

[04] Groups and references

SYNTAX	MEANING
`(abc)`	capturing group, numbered from 1 by its opening parenthesis
`(?:abc)`	group without capturing
`(?<year>\d{4})`	named group, available as match.groups.year
`\1`	backreference to group 1
`\k<year>`	backreference to a named group
`a|b`	alternation; the leftmost alternative that matches wins, not the longest

[05] Lookaround

SYNTAX	ASSERTS THAT WHAT FOLLOWS OR PRECEDES
`(?=abc)`	follows: abc
`(?!abc)`	follows: not abc
`(?<=\$)`	precedes: a dollar sign
`(?<!-)`	precedes: not a minus

Lookaround consumes nothing. `\d+(?=px)` finds `12` in `12px`. JavaScript allows lookbehind of any length, unlike Python and PCRE, which require a fixed or bounded width.

[06] Flags

FLAG	PROPERTY	EFFECT
`g`	global	find all matches; exec and test advance lastIndex
`i`	ignoreCase	case-insensitive; Unicode case folding with u
`m`	multiline	^ and $ match at line breaks
`s`	dotAll	. also matches line breaks
`u`	unicode	code points instead of UTF-16 units; enables \p{} and \u{}; stricter syntax
`v`	unicodeSets	like u, plus set operations such as [\p{L}--[a-z]]
`y`	sticky	match only at lastIndex
`d`	hasIndices	adds start and end indices of each group

[07] Replacement patterns

TOKEN	INSERTS
`$1`	group 1
`$<name>`	a named group
`$&`	the whole match
`$``	the text before the match
`$'`	the text after the match
`$$`	a literal dollar sign

```
'2026-10-07'.replace(/(\d+)-(\d+)-(\d+)/, '$3.$2.$1')   // '07.10.2026'
'a-b-c'.replaceAll('-', '$$')                           // 'a$b$c'
```

[08] Common traps

A global regex keeps state. `const re = /a/g; re.test("a"); re.test("a")` returns `true` and then `false`, because the second call starts at `lastIndex` 1.

Nested quantifiers such as `(a+)+`, `(\w*)*` or `(a|a)*` can backtrack exponentially when the input almost matches. Rewrite them so that each part of the input can be matched in only one way: `a+` instead of `(a+)+`.

Inside a string literal every backslash must be doubled: `new RegExp("\\d+")`. To match user input literally, escape it first; `RegExp.escape` is in recent engines, and elsewhere use `s.replace(/[.*+?^${}()|[\]\\]/g, "\\$&")`.

[09] Sources

[1] ECMAScript: RegExp objects

[2] MDN: regular expressions guide

[3] MDN: RegExp.escape() and browser support

[4] MDN: replacement patterns