The Regex Cheat Sheet

The Regex Cheat Sheet · WebsiteSetup.org

Reading a pattern

A regular expression is a pattern for matching text. Read it left to right, one piece at a time. Most characters match themselves; a handful are special and control repetition, position, or grouping. So `\d{4}-\d{2}-\d{2}` reads as four digits, hyphen, two digits, hyphen, two digits. Test as you go at regex101.com and pick the right flavor.

Character classes

Token	Matches	Example
`.`	Any character except newline	`a.c` matches abc
`\d` / `\D`	Digit / non-digit	`\d\d` matches 42
`\w` / `\W`	Word char [A-Za-z0-9_] / non-word	`\w+` matches user_1
`\s` / `\S`	Whitespace / non-whitespace	`\s` matches a space
`[abc]`	Any one listed character	matches a, b, or c
`[^abc]`	Any character not listed	matches d, not a
`[a-z]`	Any character in a range	matches m
`[A-Z0-9]`	Combined ranges	matches G or 4

Anchors and boundaries

Token	Matches	Example
`^`	Start of string or line	`^Dear`
`$`	End of string or line	`end$`
`\b`	Word boundary	`\bcat\b` not category
`\B`	Non-boundary position	`\Bcat` inside bobcat
`\A` / `\z`	String start / end (PCRE/Python, not JS)	`\Aid`, `done\z`

Quantifiers (greedy vs lazy)

Quantifiers are greedy by default. Add `?` to make them lazy and match as few characters as possible. This is the top beginner gotcha.

Token	Matches	Example
`*`	Zero or more	`ab*` matches a, abbb
`+`	One or more	`ab+` matches ab

`?`	Zero or one	`colou?r`
`{n}`	Exactly n	`\d{4}` matches 2026
`{n,}`	n or more	`\d{2,}`
`{n,m}`	Between n and m	`\d{2,4}`
`*? +? ?? {n,m}?`	Lazy: as few as possible	`<.+?>` one tag

Groups and alternation

Token	Matches	Example
`(abc)`	Capturing group	`(ab)+`
`(?:abc)`	Non-capturing group	`(?:ab)+`
`(?<name>abc)`	Named capturing group	`(?<year>\d{4})`
`|`	Alternation, this or that	`cat|dog`
`\1`	Backreference to group 1	`(\w)\1`
`\k<name>`	Named backreference	`(?<q>['"]).*\k<q>`

Lookaround

Token	Matches	Example
`(?=...)`	Positive lookahead	`\d+(?= dollars)`
`(?!...)`	Negative lookahead	`foo(?!bar)`
`(?<=...)`	Positive lookbehind	`(?<=\$)\d+`
`(?<!...)`	Negative lookbehind	`(?<!\$)\d+`

Lookbehind is supported in JavaScript since ES2018.

Flags and modifiers

Token	Matches	Example
`g`	Global, find all (JS)	`/a/g`
`i`	Ignore case	`/cat/i`
`m`	Multiline: ^ and $ per line	`/^x/m`
`s`	Dotall: dot matches newline	`/a.b/s`
`x`	Extended (PCRE/Python, not JS)	comment a pattern
`u`	Unicode	`/\u{1F600}/u`

The g flag is JS-specific; PHP uses preg_match_all and similar functions instead.

Escaping and whitespace tokens

Token	Matches	Example
`\. \^ \$ \* \+ \? \( \)`	Escaped metacharacters, matched literally	`example\.com`
`\\`	Literal backslash	`C:\\temp`
`\t` / `\n` / `\r`	Tab / newline / carriage return	`\r\n` Windows EOL
`\uXXXX` / `\x{...}`	Unicode codepoint (JS / PCRE)	`\x{00e9}` matches e-acute

Real-world recipes

```
^[\w.+-]+@[\w-]+\.[\w.-]+$
# Email, pragmatic. No regex catches every valid address.
```

```
^https?:\/\/[^\s/$.?#].[^\s]*$
# URL with http or https.
```

```
^\+?[\d\s().-]{7,}$
# Phone, loose. Optional +, at least 7 chars.
```

```
^[a-z0-9-]+$
# Slug or username. Lowercase, digits, hyphens.
```

```
^#?([0-9a-fA-F]{3}|[0-9a-fA-F]{6})$
# Hex color. fff or ffffff, optional leading #.
```

```
^\d{4}-\d{2}-\d{2}$
# Date YYYY-MM-DD. Shape only, not a real calendar check.
```

```
find \s{2,}  ->  replace with a single space
# Collapse runs of whitespace.
```

```
VS Code: find (\w+)  ->  replace $1
# VS Code uses $1 in the replacement, not \1.
```

```
RewriteRule ^old/(.*)$ /new/$1 [R=301,L]
# .htaccess 301 redirect. Left side is regex; $1 reuses the path.
```

```
^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{8,}$
# Password: lower, upper, digit, min 8 chars, via lookaheads.
```

Flavors and common mistakes

- **JavaScript**: g flag for global; lookbehind since ES2018; no `\A`, `\z`, or x flag.
- **PCRE/PHP** (preg_*): has `\A`, `\z`, x flag, and `\x{...}`.
- **Python** (re): close to PCRE; has `\A`, `\Z`, verbose flag.
- **grep**: BRE by default, so escape `\+ \? \{ \}` or use `grep -E` for ERE.

Top gotchas: greedy `.*` grabs everything (use `.*?`); a dot does not match a newline without the s flag; always test in the correct flavor.