

# REGULAR EXPRESSIONS QUICK REFERENCE

## Basic Patterns

### Metacharacters

<code>.</code>	Any character (except newline)
<code>^</code>	Start of string / line
<code>\$</code>	End of string / line
<code>*</code>	0 or more of previous
<code>+</code>	1 or more of previous
<code>?</code>	0 or 1 of previous (optional)
<code>\</code>	Escape metacharacter

### Literal Matching

```
hello # matches "hello" exactly
a.c   # matches "abc", "a1c", "a-c", etc.
.txt  # matches literal ".txt"
```

## Character Classes

### Bracket Expressions

<code>[abc]</code>	Match a, b, or c
<code>[^abc]</code>	Match anything except a, b, c
<code>[a-z]</code>	Lowercase letter
<code>[A-Z]</code>	Uppercase letter
<code>[0-9]</code>	Digit
<code>[a-zA-Z0-9]</code>	Alphanumeric

### Shorthand Classes

<code>\d</code>	Digit [0-9]
-----------------	-------------

<code>\D</code>	Non-digit [^0-9]
<code>\w</code>	Word char [a-zA-Z0-9_]
<code>\W</code>	Non-word char
<code>\s</code>	Whitespace [ \t\n\r\f]
<code>\S</code>	Non-whitespace

## Quantifiers

### Greedy Quantifiers

<code>*</code>	0 or more (greedy)
<code>+</code>	1 or more (greedy)
<code>?</code>	0 or 1 (greedy)
<code>{n}</code>	Exactly n times
<code>{n,}</code>	n or more times
<code>{n,m}</code>	Between n and m times

### Lazy Quantifiers

<code>*?</code>	0 or more (lazy / non-greedy)
<code>+?</code>	1 or more (lazy)
<code>??</code>	0 or 1 (lazy)
<code>{n,m}?</code>	Between n and m (lazy)

Lazy quantifiers match as few characters as possible

### Greedy vs Lazy

```
<.+> # greedy: "bold"
<.+?> # lazy:  ""
```

## Anchors

<code>^</code>	Start of string (or line with m flag)
<code>\$</code>	End of string (or line with m flag)
<code>\b</code>	Word boundary
<code>\B</code>	Non-word boundary
<code>\A</code>	Start of string (not affected by m)
<code>\Z</code>	End of string (not affected by m)

### Anchor Examples

```
^Hello # starts with "Hello"
world$ # ends with "world"
\bword\b # "word" as whole word
\Bword\B # "word" inside another word
```

## Groups & Alternation

### Capturing Groups

```
(abc) # capture group: match "abc"
(a|b|c) # alternation: a or b or c
(cat|dog) # match "cat" or "dog"
(\d{3})-(\d{4}) # groups: "123-4567"
```

### Group Types

<code>(pattern)</code>	Capturing group
<code>(?:pattern)</code>	Non-capturing group
<code>(?P&lt;name&gt;pat)</code>	Named group (Python)
<code>(?&lt;name&gt;pat)</code>	Named group (JS, .NET)
<code>\1 \2</code>	Backreference to group 1, 2
<code>a b</code>	Alternation: a or b

# REGULAR EXPRESSIONS QUICK REFERENCE (continued)

## Lookahead & Lookbehind

<code>(?=pattern)</code>	Positive lookahead
<code>(?!pattern)</code>	Negative lookahead
<code>(?&lt;=pattern)</code>	Positive lookbehind
<code>(?&lt;!pattern)</code>	Negative lookbehind

## Lookaround Examples

```
\d+(?= USD)    # digits followed by " USD"
\d+(?! USD)    # digits NOT followed by " USD"
(?<=\$)\d+     # digits preceded by "$"
(?:# digits NOT preceded by "$"
```

Lookarounds match a position without consuming characters

## Common Patterns

<code>\d{1,3}(\.\d{1,3}){3}</code>	IPv4 address (basic)
<code>[\w.+-]+@[ \w-]+\.[ \w. ]+</code>	Email (basic)
<code>https?://[ \w. / \- ? &amp; # = ]+</code>	URL (basic)
<code>\(? \d{3} \)? [- . \s]? \d{3} [- . \s]? \d{3}</code>	Phone number
<code>\d{4}-\d{2}-\d{2}</code>	Date (YYYY-MM-DD)
<code>#[0-9a-fA-F]{6}</code>	Hex color code

These are simplified patterns; production use may need stricter validation

## Flags

<code>g</code>	Global: find all matches, not just first
<code>i</code>	Case-insensitive matching
<code>m</code>	Multiline: <code>^ / \$</code> match line boundaries
<code>s</code>	Dotall: <code>.</code> matches newline too
<code>x</code>	Verbose: ignore whitespace, allow comments
<code>u</code>	Unicode: full Unicode support

## Flag Usage by Language

```
/pattern/gi    # JavaScript
re.compile(r"pat", re.I | re.M) # Python
grep -iE "pattern" # grep (extended)
```