Regular Expressions Cheatsheet

A quick reference guide to regular expressions (regex) in programming, covering syntax, common patterns, and usage examples.

Regex Basics & Metacharacters

Basic Matching

`literal`	Matches the literal character sequence. Example: `abc` matches 'abc'.
`.` (dot)	Matches any single character except newline. Example: `a.c` matches 'abc', 'adc', 'aec', etc.
`^`	Matches the beginning of the string. Example: `^abc` matches 'abcdef', but not 'defabc'.
`$`	Matches the end of the string. Example: `abc$` matches 'defabc', but not 'abcdef'.
`[]`	Character class: Matches any single character within the brackets. Example: `[abc]` matches 'a', 'b', or 'c'.
`[^]`	Negated character class: Matches any single character *not* within the brackets. Example: `[^abc]` matches any character except 'a', 'b', or 'c'.
`|`	Alternation: Matches either the expression before or after the `|`. Example: `cat|dog` matches 'cat' or 'dog'.

Quantifiers

`*`	Matches the preceding character or group zero or more times. Example: `ab*c` matches 'ac', 'abc', 'abbc', 'abbbc', etc.
`+`	Matches the preceding character or group one or more times. Example: `ab+c` matches 'abc', 'abbc', 'abbbc', etc., but not 'ac'.
`?`	Matches the preceding character or group zero or one time. Example: `ab?c` matches 'ac' or 'abc'.
`{n}`	Matches the preceding character or group exactly `n` times. Example: `ab{2}c` matches 'abbc'.
`{n,}`	Matches the preceding character or group `n` or more times. Example: `ab{2,}c` matches 'abbc', 'abbbc', 'abbbbc', etc.
`{n,m}`	Matches the preceding character or group between `n` and `m` times (inclusive). Example: `ab{2,4}c` matches 'abbc', 'abbbc', and 'abbbbc'.

Character Classes

`\d`	Matches any digit (0-9). Equivalent to `[0-9]`.
`\D`	Matches any non-digit character. Equivalent to `[^0-9]`.
`\w`	Matches any word character (alphanumeric and underscore). Equivalent to `[a-zA-Z0-9_]`.
`\W`	Matches any non-word character. Equivalent to `[^a-zA-Z0-9_]`.
`\s`	Matches any whitespace character (space, tab, newline, etc.).
`\S`	Matches any non-whitespace character.

Grouping and Backreferences

Grouping

`()`	Groups the enclosed pattern. Allows you to apply quantifiers or alternations to the entire group. Also captures the matched group for backreferencing.
`(?:)`	Non-capturing group. Groups the pattern but does *not* capture the matched group. Useful for performance or when you don't need the captured value.

Backreferences

`\1`, `\2`, etc.	Refers to the first, second, etc. captured group in the regex. Example: `(.)(.)\2\1` matches 'abba'.
`$1`, `$2`, etc. (in replacement strings)	Refers to the first, second, etc. captured group in the replacement string of a substitution operation.

Examples

Match a date in `YYYY-MM-DD` format:

```
\d{4}-\d{2}-\d{2}
```

Match an email address (simplified):

```
\w+@\w+\.\w+
```

Match HTML tags:

```
<[^>]+>
```

Anchors and Lookarounds

Anchors

`^`	Matches the beginning of the string (or line, in multiline mode).
`$`	Matches the end of the string (or line, in multiline mode).
`\b`	Matches a word boundary (the position between a word character and a non-word character).
`\B`	Matches a non-word boundary.

Lookarounds

`(?=pattern)`	Positive lookahead: Asserts that the pattern *follows* the current position, but does not consume the characters. Example: `\w+(?=\d)` matches 'abc' in 'abc123', but not 'abc' in 'abc def'.
`?!pattern`	Negative lookahead: Asserts that the pattern does *not* follow the current position. Example: `\w+(?!\d)` matches 'abc' in 'abc def', but not 'abc' in 'abc123'.
`(?<=pattern)`	Positive lookbehind: Asserts that the pattern *precedes* the current position, but does not consume the characters. Example: `(?<=\d)\w+` matches 'abc' in '123abc', but not 'abc' in 'abc def'.
`?<!pattern`	Negative lookbehind: Asserts that the pattern does *not* precede the current position. Example: `(?<!\d)\w+` matches 'abc' in 'abc def', but not 'abc' in '123abc'.

Flags/Modifiers

Common Flags

`i`	Case-insensitive matching. Example: `/abc/i` matches 'abc', 'ABC', 'aBc', etc.
`g`	Global matching. Finds all matches instead of stopping after the first.
`m`	Multiline mode. `^` and `$` match the beginning and end of each line (delimited by `\n`).
`s`	Dotall mode. Allows the `.` to match newline characters as well.
`x`	Verbose mode. Allows whitespace and comments in the regex pattern for better readability. Whitespace is ignored, and comments start with `#`.

Using Flags (Examples)

In Python:

```
import re

pattern = re.compile('abc', re.IGNORECASE)  # Case-insensitive
matches = pattern.findall('aBcAbC')
print(matches)  # Output: ['aBc', 'AbC']
```

In JavaScript:

```
const regex = /abc/i; // Case-insensitive
const matches = 'aBcAbC'.match(regex);
console.log(matches); // Output: ['aBc', index: 0, input: 'aBcAbC', groups: undefined]

const regexGlobal = /abc/gi; // Global and case-insensitive
const allMatches = 'aBcAbC'.match(regexGlobal);
console.log(allMatches); // Output: [ 'aBc', 'AbC' ]
```

In Ruby:

```
pattern = /abc/i  # Case-insensitive
matches = 'aBcAbC'.scan(pattern)
puts matches # Output: aBc
puts matches.count # Output: 2
```