Regular Expressions (Regex) Basics Cheatsheet

A quick reference guide to the fundamental concepts and syntax of regular expressions, covering patterns, metacharacters, and common use cases.

Character Matching

Basic Characters

`character`	Matches the literal character. For example, `a` matches 'a'.
`.` (dot)	Matches any single character except newline (`\n`).
`\d`	Matches any digit (0-9).
`\w`	Matches any word character (a-z, A-Z, 0-9, and underscore).
`\s`	Matches any whitespace character (space, tab, newline).
`\D`	Matches any non-digit character.
`\W`	Matches any non-word character.
`\S`	Matches any non-whitespace character.

Character Sets

`[abc]`	Matches any single character in the set (a, b, or c).
`[^abc]`	Matches any single character *not* in the set (anything but a, b, or c).
`[a-z]`	Matches any lowercase letter (a to z).
`[0-9]`	Matches any digit (0 to 9).
`[a-zA-Z0-9_]`	Matches any alphanumeric character or underscore (same as `\w`).
`[ ]`	Matches a space character inside a character set.

Quantifiers

Quantifier Basics

`*`	Matches the preceding character or group zero or more times.
`+`	Matches the preceding character or group one or more times.
`?`	Matches the preceding character or group zero or one time (optional).
`{n}`	Matches the preceding character or group exactly n times.
`{n,}`	Matches the preceding character or group n or more times.
`{n,m}`	Matches the preceding character or group between n and m times (inclusive).

Greedy vs. Lazy Matching

Greedy	By default, quantifiers are greedy, meaning they match the longest possible string.
Lazy (Reluctant)	Adding `?` after a quantifier makes it lazy, matching the shortest possible string. Example: `.*?`
Example	Given the string `'aabbbbcc'`, the regex `a.*b` will match `'aabbbb'` (greedy), while `a.*?b` will match `'aab'` (lazy).

Anchors and Grouping

Anchors

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

Grouping and Capturing

`()`	Groups characters together and captures the matched group. Example: `(abc)+` matches one or more occurrences of 'abc'.
`\1`, `\2`, etc.	Backreferences to captured groups. `\1` refers to the first captured group, `\2` to the second, and so on. Example: `(.)abc\1` matches 'zabcz'.
`(?:..)`	Non-capturing group. Groups characters together without capturing the matched group. Useful for applying quantifiers or alternations. Example: `(?:abc)+` matches one or more occurrences of 'abc' but doesn't capture the group.

Alternation

`|`	Matches either the expression before or after the `|` . Example: `cat|dog` matches either 'cat' or 'dog'.

Flags (Modes)

Common Flags

`i`	Case-insensitive matching. Matches both uppercase and lowercase letters.
`g`	Global matching. Finds all matches rather than stopping after the first.
`m`	Multiline mode. `^` and `$` match the start and end of each line, rather than the entire string.
`s`	Dotall mode. Allows the dot (`.`) to match newline characters as well.
`x`	Verbose mode. Allows whitespace and comments in the regex for better readability.

Using Flags

Flags are often specified at the end of the regex pattern, e.g., `/pattern/i` for case-insensitive matching.
In some languages, flags can be specified inline within the regex using the `(?flag)` syntax, e.g., `(?i)pattern` .