Regular Expressions Cheat Sheet

A concise reference for regular expressions (regex) syntax and usage, covering patterns, metacharacters, quantifiers, and common operations.

Regex Fundamentals

Basic Patterns

`abc`	Matches the literal sequence `abc`.
`[abc]`	Matches any single character: `a`, `b`, or `c`.
`[^abc]`	Matches any single character *except* `a`, `b`, or `c`.
`[a-z]`	Matches any lowercase letter from `a` to `z`.
`[0-9]`	Matches any digit from `0` to `9`.
`.`	Matches any single character (except newline).

Metacharacters

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

Anchors

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

Quantifiers and Grouping

Quantifiers

`*`	Matches the preceding element 0 or more times.
`+`	Matches the preceding element 1 or more times.
`?`	Matches the preceding element 0 or 1 time.
`{n}`	Matches the preceding element exactly n times.
`{n,}`	Matches the preceding element n or more times.
`{n,m}`	Matches the preceding element between n and m times (inclusive).

Grouping and Capturing

`()`	Groups the enclosed pattern. Captures the matched text for backreferencing.
(?:pattern)	Non-capturing group. Groups the pattern without capturing the matched text.
`|`	Acts as an 'or' operator. Matches either the pattern before or after the `|`.
`(?<name>...)`	Named capturing group. Matches `...` and stores it in the group named `name`.
`\1`, `\2`, ...	Backreferences to the captured groups. `\1` refers to the first captured group, `\2` to the second, and so on.

Greedy vs. Lazy Matching

By default, quantifiers are *greedy*, meaning they match as much as possible.

Add a `?` after a quantifier to make it *lazy*, matching as little as possible.

Example:

Given the string `<a><b></a></b>` and the pattern `<.*>`:

- Greedy: matches `<a><b></a></b>`
- Lazy: matches `<a>`

Advanced Regex Features

Lookarounds

`(?=pattern)`	Positive lookahead assertion. Ensures that the pattern is followed by `pattern`, but doesn't include `pattern` in the match.
`?!pattern`	Negative lookahead assertion. Ensures that the pattern is *not* followed by `pattern`.
`(?<=pattern)`	Positive lookbehind assertion. Ensures that the pattern is preceded by `pattern`, but doesn't include `pattern` in the match (not supported in all regex engines).
`(?<!pattern)`	Negative lookbehind assertion. Ensures that the pattern is *not* preceded by `pattern` (not supported in all regex engines).

Flags/Modifiers

`i`	Case-insensitive matching.
`g`	Global matching (find all matches, not just the first).
`m`	Multiline matching. `^` and `$` match the start and end of each line (as well as the start/end of the string).
`s`	Dotall. Allows `.` to match newline characters.

Conditional Regex

`(?(condition)then|else)` - Matches the `then` part if the `condition` is met, otherwise matches the `else` part. The `else` part can be omitted.

Common Regex Operations

Substitution

Replace matches of a pattern with a specified string.

Example (Python):

```python
import re

text = "The quick brown fox"
new_text = re.sub(r"\s+", "-", text)
print(new_text) # Output: The-quick-brown-fox
```

Splitting

Split a string into a list of substrings based on a regex delimiter.

Example (JavaScript):

```javascript
const text = "apple,banana,orange";
const fruits = text.split(/,/);
console.log(fruits); // Output: [ 'apple',
'banana', 'orange' ]
```

Validation

Verify that a string matches a specific format using regex.

Example (Java):

```java
import java.util.regex.Pattern;

String email = "test@example.com";
boolean isValid = Pattern.matches("[a-zA-Z0-
9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}",
email);
System.out.println(isValid); // Output: true
```