Regex and Text Manipulation Cheat Sheet

A comprehensive cheat sheet covering regular expressions and text manipulation techniques. Learn how to effectively search, extract, and modify text using various tools and methods.

Regex Basics

Metacharacters

`\`	Escapes the next character.
`^`	Matches the beginning of the string or line.
`$`	Matches the end of the string or line.
`.`	Matches any single character except newline.
`|`	Alternation (OR operator).
`()`	Grouping and capturing.
`[]`	Character class (matches any character within the brackets).
`{}`	Quantifier (specifies how many occurrences to match).
`*`, `+`, `?`	Quantifiers: `*` (0 or more), `+` (1 or more), `?` (0 or 1).

Character Classes

`\d`	Matches any digit (0-9).
`\D`	Matches any non-digit character.
`\w`	Matches any word character (a-z, A-Z, 0-9, _).
`\W`	Matches any non-word character.
`\s`	Matches any whitespace character (space, tab, newline).
`\S`	Matches any non-whitespace character.
`[abc]`	Matches a, b, or c.
`[^abc]`	Matches any character except a, b, or c.
`[a-z]`	Matches any lowercase letter.

Quantifiers

`*`	Matches 0 or more occurrences.
`+`	Matches 1 or more occurrences.
`?`	Matches 0 or 1 occurrence.
`{n}`	Matches exactly n occurrences.
`{n,}`	Matches n or more occurrences.
`{n,m}`	Matches between n and m occurrences (inclusive).

Advanced Regex

Grouping and Capturing

`()`	Creates a capturing group. The matched text within the parentheses can be referenced later.
`\1`, `\2`, etc.	Backreferences to the captured groups. `\1` refers to the first group, `\2` to the second, and so on.
(?:pattern)	Non-capturing group. Groups the pattern but doesn't capture the matched text.

Lookarounds

`(?=pattern)`	Positive lookahead. Matches if the pattern follows the current position, but doesn't include the pattern in the match.
`(?!pattern)`	Negative lookahead. Matches if the pattern does not follow the current position.
`(?<=pattern)`	Positive lookbehind. Matches if the pattern precedes the current position, but doesn't include the pattern in the match.
`(?<!pattern)`	Negative lookbehind. Matches if the pattern does not precede the current position.

Flags/Modifiers

`i`	Case-insensitive matching.
`g`	Global matching (find all matches instead of stopping after the first).
`m`	Multiline mode (`^` and `$` match the start and end of each line).
`s`	Dotall mode (`.` matches any character, including newline).

Text Manipulation Tools

Common Tools

grep: A command-line tool for searching text using regular expressions.
sed: A stream editor for performing text transformations using regular expressions.
awk: A programming language designed for text processing and data extraction.

Grep Examples

- `grep 'pattern' file.txt` - Searches for 'pattern' in file.txt.
- `grep -i 'pattern' file.txt` - Case-insensitive search.
- `grep -r 'pattern' directory/` - Recursive search in a directory.
- `grep -v 'pattern' file.txt` - Invert match (show lines that do NOT contain the pattern).
- `grep -E 'pattern1|pattern2' file.txt` - Search for either pattern1 or pattern2 (extended regex).

Sed Examples

- `sed 's/old/new/g' file.txt` - Replace all occurrences of 'old' with 'new' in file.txt.
- `sed 's/old/new/gi' file.txt` - Case-insensitive global replacement.
- `sed '/pattern/d' file.txt` - Delete lines containing 'pattern'.
- `sed 's/^/# /' file.txt` - Add a '#' at the beginning of each line.
- `sed 's/\.$/!/g' file.txt` - Replace a period at the end of a line with an exclamation mark.

Programming Languages

Python Regex

```python
import re

# Matching
pattern = r"hello"
string = "hello world"
match = re.search(pattern, string)
if match:
    print("Match found:", match.group())

# Replacing
new_string = re.sub(pattern, "goodbye",
string)
print(new_string)

# Splitting
words = re.split(r"\s+", string)
print(words)
```

JavaScript Regex

```javascript
// Matching
const pattern = /hello/;
const string = "hello world";
const match = string.match(pattern);
if (match) {
    console.log("Match found:",
match[0]);
}

// Replacing
const newString =
string.replace(pattern, "goodbye");
console.log(newString);

// Testing
const testResult = pattern.test(string);
console.log("Test result:", testResult);
```

Ruby Regex

```ruby
# Matching
pattern = /hello/
string = "hello world"
match = string.match(pattern)
if match
    puts "Match found: #{match[0]}"
end

# Replacing
new_string = string.gsub(pattern,
"goodbye")
puts new_string

# Splitting
words = string.split(/\s+/)
puts words
```