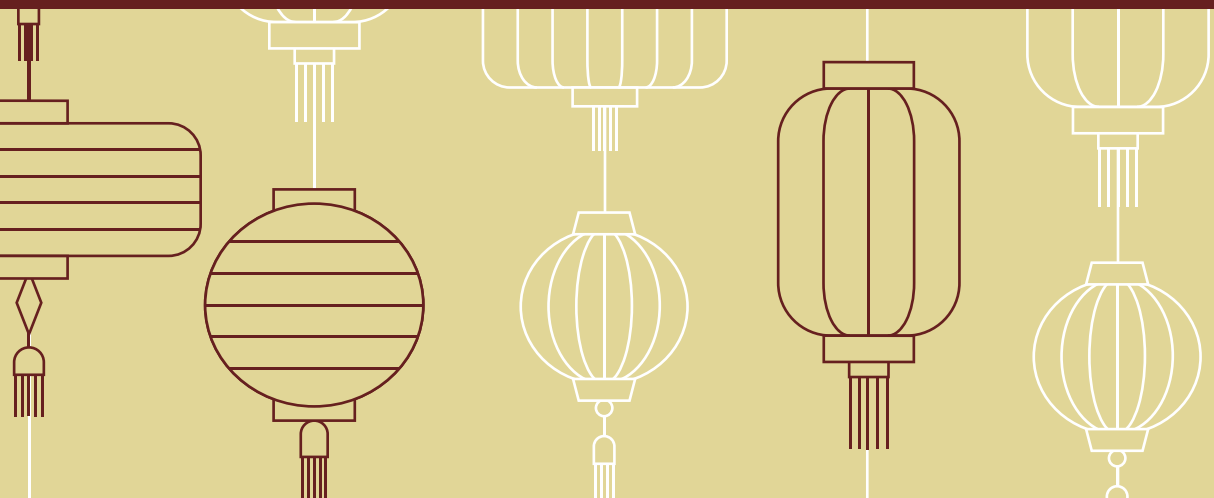

Regular Expressions (Regex) are patterns used to match character combinations in strings. They are extremely powerful for searching, replacing, and manipulating text.

1. Literals.

- a, b, c : Match the exact characters a, b, and c.

2. Meta Characters.

. Matches any single character except newline.

^ Matches the beginning of a string.

$ Matches the end of a string.

* Matches 0 or more repetitions of the preceding element.

+ Matches 1 or more repetitions of the preceding element.

? Matches 0 or 1 repetition of the preceding element.

[] Matches any one of the characters inside the brackets (character class).

| Acts as an OR operator.

() Groups patterns together.

Simple Matches

pattern : cat

Matches : The exact string "cat" in the text.

Any Character (.)

pattern : c.t

Matches : "cat", "cot", "cut", etc.

Negated Character classes :

- Match any character Not in the set.
- Example : [^abc] matches any character except "a", "b", or "c"!

Anchors

- Specify positions in the string.
- ^ matches the start of the string.
- $ Matches the end of the string.

Beginning (^) & End ($) of string :

pattern : ^cat

Matches : "cat" only if it appears at the beginning of the string.

pattern : cat$

Matches : "cat" only if it appears at the end of the string.

pattern : ^cat$

Matches : The exact string "cat".

Character classes ([]) :

Pattern : [cb]at

Matches : "cat" or "bat".

pattern : [a-z]

Matches : Any lowercase letter.

Define a set of characters to match.

Example : [abc] matches "a", "b", or "c".

predefined classes : \d (digits), \w (word characters), \s (whitespace).

Quantifiers (*, +, ?):

pattern : c a * t

Matches : "ct", "cat", "caat", "caaat", etc.

pattern : c a + t

Matches : "cat", "caat", "caaat", but Not "ct."

Pattern : ca ? t

Matches : "ct" and "cat".

- Specify the number of occurrences.
- * (0 or more), + (1 or more), ? (0 or 1), {n} (exactly n), {n,} (n or more), {n, m} (between n and m.)

Alteration (|):

pattern : cat | dog

Matches : "cat" or "dog"

Grouping (( ))

pattern : (abc)+

Matches : "abc", "abcabc", "abcabcabc", etc.

parentheses () create groups for capturing matched sub-patterns.

Non-Capturing Groups

- Group without capturing
- Syntax (?: ...).
- Example : (?: abc | def) matches "abc" or "def" without capturing.

Escape characters (\):

pattern : \d

Matches : Any digit (0-9).

pattern : \w

Matches : Any word character (Letters, digits, underscore)

pattern : \s

Matches : Any whitespace character
(Space, tab, newline)

**pattern : **

Matches : A literal backlash.

1. **Move Trailing Spaces to the Next Line.**

purpose: To ensure that any spaces at the end of a lines are moved to the Beginning of the next line.

Find: `( +)$`

This pattern matches one or more spaces to a new Line.

Replace: `\n$1`

This replacement moves the spaces to a new line.

2. Add a Newline After Every period.

purpose: To insert a newline after every period, effectively separating sentences onto new lines.

Find: (\.)

This pattern matches every period.

Replace: $1\n

This replacement keeps the period & adds a newline after it.

3. Split CamelCase Words into Separate Lines

purpose : To break up CamelCase words so that each segment appears on a new line.

Find : `([a-z])([A-Z])`

This pattern matches a Lowercase Letter followed by a Uppercase Letter.

Replace : `$1\n$2`

This replacement inserts a newline between the Lowercase & Uppercase Letters.

4. Move Comments to the Next Line

purpose : To move comments that are at the end of lines to their own separate lines

Find : `\s*(//.*)$`

This pattern matches zero or more whitespace characters followed by // and any subsequent characters at the end of a line.

Replace : `\n$1`

This replacement moves the comment to a new line.

5. Insert a Newline After Each Comma

purpose : To separate items in a comma-separated list onto new lines.

Find : ,

- Matches each comma.

Replace : ,\n

Adds a newline after each comma.