Verwendung:

- Strings: bearbeiten, suchen, ersetzen
- Zeichenketten validieren / auf Zusammensetzung prüfen
- Perl (hier sind die Reg Ex etwas besonderes)

- Bei web-Anwendungen (z.B. E-Mail Eingabefeld), in Shell-Skripten und in Editoren (suchen und ersetzen) verwendet

Ein-elementige Regex:

[1 2 3 4 5 6]
[1-6]
} 1 oder 2 oder 3 ... oder 6

⇒ Eckige Klammer: auflistung von Zeichen (immer nur einstellige Zeichen)

[1-3 5-9] Zwei Bereiche: 1-3 oder 5-9

Achtung: kein Leerzeichen zwischen 3 und 5

[1-12] Achtung: 1 bis 1 oder 2, also nicht 1 bis 12, da Zeichen einstellig sein müssen

[a-d] a oder b oder c oder d
[a-dA-D] a oder A oder ... oder d oder D

Mehr-elementige Regex

```
[1-9][a b]
```

=> 1a, 1b, 2a, 2b, ..., 9a, 9b
eine Ziffer 1-9 gefolgt von einem Buchstaben a oder b

```
[1-9][a-d A-D][2-4]
```

=> Kombinationen möglich, Zeichenkette muss der Reihe nach alle 3 Regeln für seine 3 Zeichen erfüllen.

Valide: 1c4, 9A3
Nicht valide: 0c4, 1e4, 9A5

Achtung: Verhalten bei Zeichenkette 1c4H hängt vom Interpreter und vom Einsatzzweck des Regex ab.

Optionen:

```
[1-9][0-9]?[0-9]?
```
1 bis 3-stellige Zahl

=> ? besagt: vorhergehendes Element kann vorkommen, muss aber nicht
([0-9] ist ein Element)

a{5} => aaaaa erfüllt diesen Regex

{n} ≙ genau n mal

a{1,3}b => ab, aab, aaab treffen zu

{min, max}

=> Wenn nur eine mindest- oder maximal-anzahl angegeben wird, muss auch nur diese Bedingung zutreffen, z.B.

[1-9][0-9]{2,} mindestens 3-stellig

[1-9][0-9]{,5} maximal 6-stellig

Beliebige Wiederholungen

Bsp: Telefonnummer 0 7131 1 9633 - 68
0180. 45 333. 68

Die Länge der Telefonnummer sei an dieser Stelle egal, wichtig sind nur die Zeichen die darin vorkommen.

[0-9/. \-]+ ← "mindestens 1 mal"

Leerzeichen ↑ ↑ das "-" muss maskiert werden, da es hier als "." und nicht als z.B. "1-3" verwendet wird

=> Das + hinter der Beschreibung des in der Zeichenliste vorkommenden Zeichens besagt, dass mindestens ein Zeichen diese Regel erfüllen muss.

=> Ein * verhält sich ähnlich, bis auf den Unterschied, dass auch kein Vorkommen erlaubt ist („beliebig oft")

Platzhalter

Bsp.: Suche nach allen Personen, die "Thomas" mit Vornamen und "Müller" mit Nachnamen heißen, unabhängig vom Zweitnamen

Thomas .* Müller

Leerzeichen

=> Der Punkt . steht für „ein beliebiges Zeichen", der Stern * sagt aus, dass es beliebig oft vorkommen darf.

Achtung: Zeilenumbrüche werden vom . nicht abgedeckt.
Der Punkt muss innerhalb eckiger Klammern [ ] nicht maskiert werden.

Negation

[^ a b] Nicht (a oder b)

Klammern

=> fassen Zeichen zusammen

Ba(na)*ne

=> Bane, Banane, Bananane, ...

Ba(na){2,5}ne => Anzahl begrenzt

Thomas (Marvin)? Müller

=> Thomas Müller, Thomas Marvin Marvin Müller

Ba(n|a)ne => Banne, Baane

↳ n oder a

=> Der senkrechte Strich ist ein logisches Oder

Das Wetter ist (gut | schlecht)

Modifikatoren

=> Das Verhalten des RegEx kann durch Modifikatoren beeinflusst werden. Diese sind Dialekt-abhängig aber häufig ähnlich.

Syntax: [Begrenzung] [Regex] [Begrenzung] [Modifikation]

Bsp.: /(ich | du)/i (JavaScript)

- i: Case-Insensitive, Groß- und Kleinschreibung ignorieren.
- s: Punkt wird multiline-fähig, das Wildcard-Zeichen . trifft auch auf Zeilenumbrüche zu.
- m: ^ und $ matchen auch auf Zeilenanfänge und -enden. Ohne diese Modifikation passen sie nur auf die gesamte Zeichenkette.

Verschachtelung

(VW (Golf | Polo) | Fiat (Punto | Panda))

=> akzeptiert nur 4 verschiedene Kombinationen

(1 0 1 0 1)+

=> Folge as Nullen und Einsen, in der maximal 2 Nullen oder Einsen aufeinander folgen

Einige Ausdrücke

Wir wollen HTML-Tags erkennen, die Links sind, also folgende Form haben:

<a href="https://...">

Vorschlag: <a href=".*". *>

Problem: Der zweite Punkt frisst die Klammer-Zu > ebenfalls und dazu noch die Zeichen dahinter.

Lösung 1: Punkt ersetzen

<a href="[^"]*" [^>]*>

Lösung 2: Punkt und Stern „bändigen"

<a href=".*?" .*?>

=> Das Fragezeichen sagt an dieser Stelle aus, dass nur so viele Zeichen wie nötig vom Punkt abgedeckt werden sollen.

Gruppen

Wir wollen aus dem vorherigen Beispiel die URL extrahieren.

```
<a href="(.*?)".*?>
```

↳ Gruppe

Je nach verwendetem Reg Ex Dialekt kann der Inhalt der Gruppe ausgelesen werden.

Achtung: Die Nummerierung der Gruppen erfolgt der Reihenfolge der öffnenden Klammern nach

Referenzen: Verweise auf Gruppen

```
([0-9]) [0-9]* \1
```

↑ Verweis auf Inhalt der 1. Klammer

=> Das Zeichen, das in der 1. Klammer gefunden wurde, muss sich an der Stelle \1 auch befinden

\0 ist der match des gesamten Reg Ex

Wortgrenzen

Bsp.: Suche nach "kai" in:

„Kai fährt nach Kaiserslautern"

kai => 2 Matches

\b kai \b => 1 Match

Wortanfang bzw. Wortende

Eine Wortgrenze tritt zwischen einem Wort-Zeichen und einem Nicht-Wort-Zeichen auf

Spezial-Zeichen

Immer ein Backslash und ein Buchstabe.

Großbuchstaben stehen immer für das Gegenteil des Kleinbuchstabens.

\w Wort-Zeichen, entspricht [a-zA-Z0-9]

\W Nicht-Wort-Zeichen, [^a-zA-Z0-9]

\d Ziffer („digit") [0-9]

\D [^0-9]

\b Wortende

\B kein Wortende

\s Whitespace-Zeichen: LF, CR, Space, Tab

\S kein Whitespace-Zeichen

\\ Backslash-Zeichen ("\")

Der Backslash wird zum Maskieren von Zeichen verwendet:

\. \+ * \(\) \[\]

\- \$ \|

Anfangs- und End-Zeichen

^1.3.2004$

Match nur, wenn der String hier anfängt

Match nur, wenn der String hier endet

1.3.2004 => Match

11.3.2004 => kein Match

\d$ => Matcht Strings die auf Ziffern enden

^\(=> Matcht Strings die mit "(" anfangen

Positive Lookaheads und Lookbehinds

Bsp.: Lange Ziffern-Folge

00563440825000332057911

=> Suche alle Folgen, die aus drei Ziffern ungleich 0 bestehen und von einer 0 an jeder Seite begrenzt werden.

Lösung: `(?<=0)[1-9]{3}(?=0)`

Erklärung:

- Mitte: `[1-9]{3}`

  Ziffernfolge von drei Ziffern ungleich Null

- Links: `(?<=0)`

  `?<=` => Sonderfunktion für „vornedran"

  Dies ist ein sog. "positive Lookbehind" (entgegen dem normalen Lese-Fluss)

  `?<=0` => Vornedran muss eine 0 sein

- Rechts: `(?=0)`

  `?=` => Sonderfunktion für Hinterdran

  Dies ist ein sog. „positive Lookahead"

  `?=0` => hinterdran muss eine 0 sein

\b (\d) \d* \1 \b

kann auch ausgedrückt werden als

(?<=\W) (\d) \d* \1 (?=\W)

Achtung: Lookbehinds und Lookaheads sind keine Gruppen, fallen also aus der Nummerierung heraus

Achtung: Lookbehinds dürfen keine unbekannte Anzahl an Zeichen haben, also kein:

*, +, {0,4}

Negative Lookaheads und Lookbehinds

(?!0) => negativer Lookahead

hier darf keine 0 folgen

Bsp.: \b F(?!eda\b).*\b

=> alle Wörter, die mit F anfangen, aber nicht „Feda" sind

(?<!0) => negativer Lookbehind

Hier darf keine Null davor stehen

Bsp.: .*(?<!Müll)eimer

=> alle Eimer, die keine Mülleimer sind