

# Reguläre Ausdrücke in Perl

## Die wichtigsten Elemente auf einen Blick

### Zeichen und Zeichenklassen

a	Zeichenliteral, hier „a“, trifft nur auf das genaue Zeichen zu.
.	Any-Zeichen, trifft auf ein beliebiges Zeichen <u>außer</u> \n zu.
[0-9a-f]	Klasse, trifft auf ein Zeichen aus 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f zu.
[^chars1-3]	Trifft auf ein beliebiges Zeichen <u>außer</u> c, h, a, r, s, 1, 2 oder 3 zu.

### Metazeichen-Äquivalenzklassen

\w	[a-zA-Z0-9_]	Ein „Wort-Zeichen“ (alphanumerische Zeichen plus _).
\d	[0-9]	Ein beliebiges numerisches Zeichen.
\s	[\t\n\r\f]	Ein beliebiges Whitespace-Zeichen.
\W	[^a-zA-Z0-9_]	Ein beliebiges Zeichen <u>außer</u> „Wort-Zeichen“ (Negation zu \w).
\D	[^0-9]	Ein beliebiges Zeichen <u>außer</u> numerische Zeichen (Negation zu \d).
\S	[^ \t\n\r\f]	Ein beliebiges Zeichen <u>außer</u> Whitespace-Zeichen (Negation zu \s).

### Quantoren

*	Das vorhergehende Zeichen tritt beliebig oft auf (0 bis unendlich).
+	Das vorhergehende Zeichen tritt mindestens einmal auf.
?	Das vorhergehende Zeichen tritt nicht oder genau einmal auf.
{n}	Das vorhergehende Zeichen tritt genau n-mal auf.
{n,m}	Das vorhergehende Zeichen tritt mindestens n- und höchstens m-mal auf.
{n,}	Das vorhergehende Zeichen tritt mindestens n-mal auf.
<Quantor>?	Bescheidenheitsquantor, nimmt <Quantor> die Gier. Siehe auch unten.

### Anker, Gruppierung und Alternation

^	Trifft auf den Beginn der Zeile zu.
\$	Trifft auf das Ende der Zeile zu.
(regex)	Gruppiert „regex“ zu einem regulären Unterausdruck.
	Alternation, logische Oder-Verknüpfung über gesamten (Unter-)Ausdruck.

### Operatoren

m/ausdruck/	Match-Operator, erkennt das Auftreten von „ausdruck“ (und extrahiert)
s/ausdruck/string/	Substitutionsoperator, ersetzt „ausdruck“ durch „string“.
tr/klasse1/klasse2/	Transliterationsoperator, ersetzt Zeichen in „klasse1“ durch „klasse2“.

### Operationsmodi für m/ausdruck/igsmo

i	Ignoriere Groß- und Kleinschreibung.
g	Globales Matching, setzt nach dem ersten Auftreten fort.
s	Behandle gesamte Eingabe als einen einzigen String.
m	Trenne Eingabe in Zeilen und wende Ausdruck auf jede Zeile einzeln an.
o	Kompiliere den regulären Ausdruck nur einmal. Vorsicht mit Variablen!

### Operationsmodi für s/ausdruck/string/igsmeo

i	Ignoriere Groß- und Kleinschreibung.
g	Globales Matching, setzt nach dem ersten Auftreten fort.
s	Behandle gesamte Eingabe als einen einzigen String.
m	Trenne Eingabe in Zeilen und wende Ausdruck auf jede Zeile einzeln an.
e	Führe „string“ als Perlcode aus.
o	Kompiliere den regulären Ausdruck nur einmal. Vorsicht mit Variablen!

### Operationsmodi für tr/klasse1/klasse2/ds

d	Lösche alle Zeichen, die gematcht aber nicht ersetzt wurden.
s	Fasse mehrfach auftretende Zeichen beim Ersetzen zusammen.

# Reguläre Ausdrücke in Perl

## Wichtig zu wissen

1. Reguläre Ausdrücke matchen, wenn nicht explizit anders angefordert, Zeichen und nicht Bytes. Dies ist üblicherweise nur bei Verwendung von locales und für Multibyte-Zeichensätze wie Unicode UTF-8, UTF-16, etc wichtig.
2. Möchte man innerhalb des regulären Ausdrucks die Zeichen `^`, `]`, `{`, `+`, `*`, `?`, `^`, `$` matchen, muss man diese mit Backslashes quoten, z.B. `\^`, `\\`, `\[`, `\]`, etc.
3. Möchte man innerhalb einer Zeichenklasse die Zeichen `[`, `]`, `-` oder `^` aufzählen, muss man diese mit Backslashes quoten, z.B. `\[`, `\-`, etc.
4. Im Einzeilen-Modus (s) trifft das Any-Zeichen „.“ auch auf `\n` zu.
5. Quantoren sind stets gierig, d.h. auf „babcabab“ matcht `/a.*a/` vom ersten bis zum letzten a, also „abcaba“. Um dies zu verhindern, versieht man Quantoren zusätzlich mit dem Bescheidenheitsquantor „?“, so matcht `/a.*?a/` nur vom ersten bis zum zweiten a, also „abca“.
6. Reguläre Ausdrücke verhalten sich wie double-quoted strings, daher sind dort alle Metazeichen erlaubt, die auch in ""-Strings verwendet werden können, z.B. `\n`, `\r`, etc.
7. Den Match-Operator `m//` kann man auch durch `//` abkürzen, den Transliterationsoperator `tr///` kann man auch als `y///` schreiben.

## Praktische Anwendung in Perl

### Einfacher Match

```
if($a =~ m/^\d+$/){...}
```

Führt Block nur dann aus, wenn in \$a nicht leer ist und ausschließlich aus Ziffern besteht.

### Match auf Laufvariable

```
print if(/^foo/i);
```

Gibt den String in der Laufvariable \$\_ aus, wenn dieser mit der Zeichenfolge „foo“ beginnt. Gross-/Kleinschreibung wird dabei ignoriert.

### Einfache Substitution

```
$a =~ s/(Tee|Leber)wurst/Käse/g;
```

Ersetzt sämtliche Vorkommen der Zeichenfolgen „Teewurst“ und „Leberwurst“ durch „Käse“.

### Extraktion von Daten per Match

```
if($a =~ /\w+\s+(\w+)\s*(.*)$/){  
    printf(“%s\t%s“, $2, $1);  
}
```

Falls der String in \$a mindestens zwei durch beliebig viel Whitespace getrennte Wörter enthält, lade das zweite Wort in \$1 und alles was den 2 Wörtern folgt in \$2, wobei dieser Rest auch leer sein kann und eventuell anführender Whitespace nicht mit in \$2 geladen wird.

### Laden von Daten per Match in ein Array

```
@b = ($a =~ /([\s]+wurst)\s*/gi);
```

Lade sämtliche Zeichenfolgen aus dem String \$a, die auf „wurst“ enden, in das Array @b. Die Zeichenketten sind dabei durch beliebigen Whitespace getrennt.

### Gruppierungsvariablen in Substitution verwenden

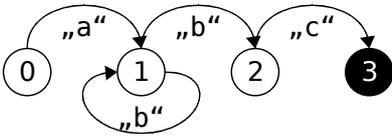
```
$a =~ s/(Fleisch|Leber)wurst/$1käse/g;
```

Ersetzt sämtliche Vorkommen der Zeichenfolgen „Fleischwurst“ und „Leberwurst“ durch „Fleischkäse“ bzw. „Leberkäse“.

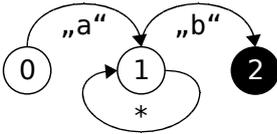
# Reguläre Ausdrücke in Perl

## Übersetzung in (nicht-deterministische) endliche Automaten

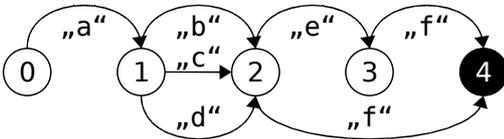
`^ab+c$`



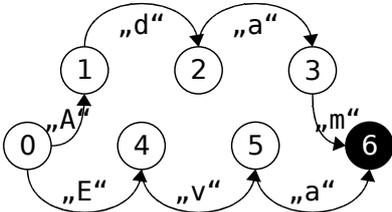
`^a.*b$`



`^a[bcd]e?f$`



`^Adam|Eva$`



`^Mein F(reu|ei)nd, der (ganz )*ho+he Baum\.$`

