

Reguläre Ausdrücke

Inhaltsverzeichnis

Reguläre Ausdrücke.....	1
Besondere Zeichen.....	1
Vordefinierte Zeichen.....	2
Modifier.....	2
Einige Beispiele.....	3
Verwendung mit PHP.....	4
Preg_match.....	4
str_replace.....	4
ereg.....	4
eregi.....	5
ereg_replace.....	5
eregi_replace.....	5
Weitere Beispiele zum Üben.....	5

Online Tester: <http://regexr.com/>

Reguläre Ausdrücke

Reguläre Ausdrücke sind genau definierte Suchmuster. Mit Hilfe dieser Suchmuster können beispielsweise Variableninhalte durchsucht werden. Reguläre Ausdrücke sehen auf den ersten Blick verwirrend aus, folgen aber einer genau definierten Syntax.

```

1 if (preg_match('/<img /', $html)){
2     //Tag für ein Bild gefunden
3 }

```

Besondere Zeichen

Zeichen	Grund	Beispiel
.	Der Punkt stimmt in regulären Ausdrücken für ein beliebiges anderes Zeichen überein.	<code>/.ein/</code> mein, dein, Wein
*	Das Sternzeichen stimmt für <i>0 oder mehr Vorkommen</i> des davor stehenden Zeichens überein.	<code>/*.ein/</code>
+	Das Pluszeichen ähnelt dem Stern, stimmt aber für <i>ein oder mehr Vorkommen</i> des davor stehenden Zeichens überein.	<code>/.+ein/</code> Pein, Schwein, Stein

?	Das Fragezeichen stimmt ansonsten für <i>kein oder einmaliges Vorkommen</i> des davor stehenden Zeichens überein.	<code>/aus?/</code> aus, au
()	Um * oder + auf größere Elemente anzuwenden, wird die Zeichenfolge in () geschrieben.	<code>/(abc)*/</code> abc, abcabc, abcabcabc,...
\	Übliches Escape Zeichen	<code>/(<\/?>b)/</code> , <code>/19\.1./</code> 19.12, 19.11, 19.1A, 19.1%, ...
[]	Eckige Klammern begrenzen eine Zeichenklasse.	<code>/[aeiou]+/</code> Ausdrücke mit Vorkommen eines Vokals
[^]	Verneinung der Zeichenklasse.	<code>/[^aeiou]+/</code> Ausdrücke OHNE Vorkommen eines Vokals
{ }	Geschweifte Klammern bedeuten eine Wiederholungs-Angabe für Zeichen. <code>{min, max}</code> <code>{min}</code>	<code>/x{3,5}/</code> passt auf zwischen 3 und 5 'x' in Folge xxx, xxxx, xxxxxx <code>/[0-9]{2,4}\./</code> 12, 14, 123, 9876, ...
^	Am Beginn des Wortes	<code>/^(abc)+/</code> liefert z.B: abcHans, abcabcHaus
\$	Am Ende des Wortes	<code>/(abc)+\$/</code> liefert z.B: Hansabc, Hansabcabc
	Für ein ganzes Wort	<code>/^(abc)+\$/</code> liefert z.B: abc, abcabc

Vordefinierte Zeichen

Zeichen	Bedeutung	Beispiel
<code>\d</code>	eine Ziffer [0-9]	<code>\d\d</code> 12, 45, 77, ...
<code>\D</code>	keine Ziffer [^0-9]	
<code>\w</code>	ein Buchstabe, eine Ziffer oder der Unterstrich [a-zA-Z_0-9]	<code>[\w]{1,2}</code> af, AB, 1a, ...
<code>\W</code>	kein Buchstabe, keine Zahl und kein	

	Unterstrich [^\w]	
\s	Whitespace	
\S	alle Zeichen außer Whitespace [^\s]	

Gieriges Verhalten vs. Genügsames Verhalten

An folgendem Beispiel erklärt:

Der Ausdruck `1935675349` wird mit `1.*9` bearbeitet. Gefunden wird der Gesamte Ausdruck `1935675349`. Da zwischen 1 und 9 alles stehen darf.

So einen Ausdruck nennt man gierig (englisch *greedy*), weil er alles markiert.

Ein genügsamer Ausdruck (englisch *non-greedy*) würde bereits beim ersten 9er abbrechen. Abhilfe schafft hier der Ausdruck `1.*?9`. Dieser liefert nur noch `19`!

Warum?

<code>*</code>	=	beliebig viel Zeichne
<code>*?</code>	=	beliebig viel Zeichne kommen 1x oder gar nicht vor, und zwar zwischen 1 und dem ersten vorkommenden 9

Der Operator `*?` wird daher als *non-greedy Operator* bezeichnet.

Modifier

Zeichen	Bedeutung	Beispiel
<code>i</code>	Bei regulären Ausdrücken wird standardmäßig zwischen Groß- und Kleinschreibung unterschieden - dieser Modifier schaltet dies aus. Das hängt aber stark von der Verwendeten Sprache ab.	<code>/h?aus/i</code> Haus, haus, HAUS, aus, Aus,...

Einige Beispiele

Regulärer Ausdruck	Ergebnis
<code>/.aus/</code>	aus Haus Maus saus ...
<code>/aus?/</code>	aus au ...
<code>/a./</code>	ab an ... (ein beliebiges Zeichen hinter 'a', außer \n)
<code>/a+/</code>	aa aaaaa ... (ein oder beliebig viele 'a')
<code>/a*/</code>	a, aa, aaaaa ... (kein oder beliebig viele 'a')
<code>/Ha.s/</code>	Haus, Hans, HaWe, ...
<code>/Ha.+s/</code>	Haus, Hans, Hannes ... (ein oder beliebig viele beliebige Zeichen, außer \n)

<code>/Ha.*s/</code>	Haus, Hans, Hannes, Has ... (kein oder beliebig viele beliebige Zeichen, außer \n)
<code>/Ha.?s/</code>	Haus, Hans, Has ...
<code>/x{10,20}/</code>	passt auf zwischen 10 und 20 x in Folge
<code>/x{10}/</code>	passt auf 10 und mehr x in Folge
<code>/x.{2}y/</code>	xxxxy, xaby, xACy, x15y ... (zwei beliebige Zeichen zwischen 'x' und 'y', außer \n)
<code>\< * [img] [^\>]* [src] *= * [\"'] {0,1} ([^\\"'] >] *)</code>	findet den src Anteil von Images in html Code heraus src="./images/img1.png"
<code>/[\\\/]*\$/</code>	Directory Separator am Ende suchen /etc/config/ oder C:\Dokumente aber nicht /etc/config,D:\Spiele

Mail me!

```
/[\.a-z0-9_-]+@[\.a-z0-9-]+\.[a-z]{2,4}/i
```

Eine eMail-Adresse darf nur aus Buchstaben, Ziffern, Bindestrichen, Unterstrichen, Punkten und einem "@" als Trenner bestehen.

IP-Adressen

```
([0-9]{1,3}\.){3}[0-9]{1,3}
```

Hierbei handelt es sich um einen einfachen IP-Adressen-Finder. Er sucht dreistellige Zahlen, die durch drei Punkte getrennt sind → 192.168.123.123, 127.0.0.1 bzw.

IP4

```
^(25[0-5]|2[0-4]\d|[0-1]?\d?\d)(\.(25[0-5]|2[0-4]\d|[0-1]?\d?\d)){3}$
```

IP6

```
^(?:[0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}$
```

IP6 compressed

```
^((?:[0-9A-Fa-f]{1,4}(?::[0-9A-Fa-f]{1,4})*)?::((?:[0-9A-Fa-f]{1,4}(?::[0-9A-Fa-f]{1,4})*)?))$
```

Verwendung mit PHP

Preg_match

int **preg_match** (string Suchmuster, string Zeichenkette [, array &Treffer]

Liefert die Anzahl der gefundenen Übereinstimmungen zurück. Wird noch ein Array angegeben, werden alle Übereinstimmungen im Array abgespeichert

```
1 if (preg_match('/';
2 str_replace('/<img /', '<href', $html);
```

ereg

int **ereg** (string Suchmuster, string Zeichenkette [, array ®s])

Sucht in Zeichenkette unter Berücksichtigung der Groß- und Kleinschreibung nach Übereinstimmungen mit dem regulären Ausdruck, der in Suchmuster angegeben wurde.

Wenn Übereinstimmungen mit eingeklammerten Teilzeichenketten von Suchmuster gefunden werden und die Funktion mit dem dritten Argument regs aufgerufen wurde, werden die Übereinstimmungen in den Elementen des Arrays regs gespeichert. \$regs[1] enthält dann die Teilzeichenkette der ersten Klammer, \$regs[2] die Teilzeichenkette der zweiten usw. \$regs[0] enthält bei Übereinstimmung mit Zeichenkette eine Kopie der kompletten Zeichenkette.

```
1 if (ereg ("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})", $date, $regs)) {
2     echo "$regs[3].$regs[2].$regs[1]";
3 } else {
4     echo "Ungültiges Datumsformat: $date";
5 }
```

eregi

Genauso wie **ereg**, nur OHNE Unterscheidung der Groß und Kleinschreibung.

ereg_replace

```
string ereg_replace ( string Suchmuster, string Ersatz, string
Zeichenkette )
```

Diese Funktion durchsucht Zeichenkette nach Übereinstimmungen mit Suchmuster und ersetzt dann den übereinstimmenden Text durch Ersatz.

Zurückgegeben wird die geänderte Zeichenkette, was auch bedeuten könnte, dass die ursprüngliche Zeichenkette zurückgegeben wird, wenn es keine zu ersetzenden Übereinstimmungen gibt.

```
1 $string = "Das ist ein Test";
2 echo ereg_replace(" ist", " war", $string);
```

eregi_replace

Genauso wie **ereg_replace**, nur OHNE Unterscheidung der Groß und Kleinschreibung.

Weitere Beispiele zum Üben

RegEx: 1st\s?yellow

RegEx: rates.+low

RegEx: m[o0]rt-?g[a@]ge

RegEx: wall\s?street

RegEx: Hot deal.? Adobe Creative Suite

RegEx: Hot Sale on all.* products at .+ Softshop

RegEx: ([a-z]\\W)\\$([a-z]\\W)

RegEx: s[o0]ftw@re

RegEx: s0ftw[a@]re

RegEx: \s{8}
 RegEx: .{200}
 RegEx: up to 90\W off at .* Electronics
 RegEx: proz[a@].?c
 RegEx: turn.*\\$.*into
 RegEx: meeting.*at [0-9]{2}-[0-9]{2}
 RegEx: x[a@]n[a@].?x
 RegEx: soft?\s?tabs
 RegEx: doctor.*(visit|hassle)
 RegEx: (^|\s)loans?(\\s|\$)
 RegEx: it.?s me.,+[0-9]{5}.+from [a-z]{3}
 RegEx: (^|\s)cunts?(\\s|\$)
 RegEx: (^|\s)debt(\\s|\$)
 RegEx: software and plugins.*cheapest prices
 RegEx: on.?line.dat(ing|e)
 RegEx: diet (pills|meds)
 RegEx: listing (in|on|at) (google|yahoo).*ref
 RegEx: free (computer|desktop pc|meds|siminar|extension|postcard)
 RegEx: (college|diploma|mba|buy)(-|\\s)degree
 RegEx: Name u(nd|\\.) Adresse zu.{0,2} jeder Tel
 RegEx: your website (in|on|to) search
 RegEx: sex\\s?life
 RegEx: on.*day at [0-9]{2}-[0-9]{2}
 RegEx: [oq]em software
 RegEx: c[il1!\\[\\]\\\\\\v][il1!\\[\\]\\\\\\v]ck here
 RegEx: g[il1!\\[\\]\\\\\\v]n[a@][il1!\\[\\]\\\\\\v] [s\\\$]oftw[a@]re[s\\\$]
 RegEx: reg[a@][il1!\\[\\]\\\\\\v][il1!\\[\\]\\\\\\v][s\\\$]
 RegEx: p.?r.?e.?[s\\\$].?c.?r.?[il1!\\[\\]\\\\\\v]?.?p
 RegEx: v[a@][il1!\\[\\]\\\\\\v][il1!\\[\\]\\\\\\v].?um
 RegEx: med[il1!\\[\\]\\\\\\v]c[a@]l.*(directory|s[a@]le|consult|news|shock)
 RegEx: v[il1!\\[\\]\\\\\\v]\\\\?.?[a@]-?g-?r[a-z]?a
 RegEx: v[a@]g[il1!\\[\\]\\\\\\v]n[a@]
 RegEx: h.?y.?d.?r.?o.?c.?o.?d.?o.?n.?e
 RegEx: pleas.+partner
 RegEx: partner.+pleas
 RegEx: (ecstasy|health|joy) time
 RegEx: [a@]nt[il1!\\[\\]\\\\\\v]d[o0]te
 RegEx: small.?cap
 RegEx: pre[-\\s]?approv
 RegEx: (^|\s)skin(\\s|\$)
 RegEx: \\w[\\W_]\\w[\\W_]\\w[\\W_]\\w[\\W_]\\w
 RegEx: v[\\W_]?.[\\W_]?[a@][\\W_]g[\\W_]r[\\W_]?[a@]
 RegEx: med[il1!\\[\\]\\\\\\v].?c[a@]t[il1!\\[\\]\\\\\\v][o0]n
 RegEx: internet (patch|upgrade|update)
 RegEx: en[il1!\\[\\]\\\\\\v]+[a@]rge
 RegEx: ch[o0][il1!\\[\\]\\\\\\v]e[s\\\$]ter[o0][il1!\\[\\]\\\\\\v]
 RegEx: (^|\s)med([s\\\$]?\\s+|[il1!\\[\\]\\\\\\v]c\\w+).*[o0]n\\?|[il1!\\[\\]\\\\\\v][il1!\\[\\]\\\\\\v]ne
 RegEx: (^|\s)med([s\\\$]?\\s+|[il1!\\[\\]\\\\\\v]c\\w+).*[s\\\$]t[o0]re
 RegEx: (^|\s)med([s\\\$]?\\s+|[il1!\\[\\]\\\\\\v]c\\w+).*[o0]vern[il1!\\[\\]\\\\\\v]ght
 RegEx: (^|\s)med([s\\\$]?\\s+|[il1!\\[\\]\\\\\\v]c\\w+).*d[il1!\\[\\]\\\\\\v][s\\\$]tre[s\\\$][s\\\$]

RegEx: [s\\$]ter[o0][il1!\[\]\W]d[s\\$]

RegEx: [s\\$]t0(x|re)

RegEx: (v\W)?.?[il1!\[\]\W]c[o0]d[il1!\[\]\W]n

RegEx: only \\${0-9}{3} for Adobe Creative Suite

RegEx: c[il1!\[\]\W][a@][il1!\[\]\W][il1!\[\]\W].?[s\\$]