Inhaltsverzeichnis

1. Regular Expression: REGEX....1
1.1. Ziele....1
1.2. Reguläre Ausdrücke....1
1.2.1. Beispiel: indexOf – Einfache Textsuche....1
1.2.2. Beispiel: find and replace von Text-Mustern....2
1.2.3. Vergleich mit exaktem Muster....5
1.2.4. Zeichenklassen: [A-Z0-9]....5
1.2.5. Zeichenklassen: [a-z] [0-9] [a\-z][^a-zA-Z]....6
1.2.6. Beginn und Ende Operatoren: ^, $....7
1.2.7. Quantoren: ?, * , +....7
1.2.8. Gierige * und nicht gierige ? Quantoren....8
1.2.9. Gruppen....9
1.2.10. +Auf einzelne Gruppen zugreifen mit \ziffer....10
1.3. Zusammenfassung....10
1.3.1. Reguläre Ausdrücke für einzelne Zeichen....10
1.3.2. Reguläre Ausdrücke für Zeichenketten....11
1.3.3. Maskierung von Zeichen in regulären Ausdrücken....12
1.4. Übungsbeispiele....13
1.5. Beispiel: REGEX und MySQL....15

1. Regular Expression: REGEX

1.1. Ziele

☑ Optimales Finden und Verarbeiten von Texten (Matching)

☑ Quellen:

- ☐ http://www.proggen.org/doku.php?id=frameworks:qt:generalclasses:regex
- ☐ http://de.autohotkey.com/docs/misc/RegEx-QuickRef.htm
- ☐ http://www.sql-und-xml.de/regex/
- ☐ http://msdn.microsoft.com/en-us/library/bb982727.aspx
- ☐ http://de.selfhtml.org/perl/sprache/regexpr.htm
- ☐ http://www.pagecolumn.com/tool/regtest.htm (online testen)
- ☐ http://regexr.com/ (online testen, SUPER)

1.2. Reguläre Ausdrücke

1.2.1. Beispiel: indexOf – Einfache Textsuche

Erstellen Sie das **Java-Projekt: Regex-01-indexOf**
und ersetzen Sie die Fragezeichen:

```
/**
 * @author Anton Hofmann
 * @date 24.10.2020
 * @file IndexOf.java
```

```
 * @description Einfache Textsuche
 */

public class IndexOf {


     public static int indexOf(String text, String pattern,
                               int startIndex){
          int txtlen= text.length();
          int plen= pattern.length();
          boolean contains;

          for(int i=startIndex; i <= txtlen-plen; i++  ){
               contains=?????????;
               for (int j=0; j < plen && contains==true; j++){
                    if (text.charAt(??????) != pattern.charAt(????)){
                         contains=?????;
                    }
               }
               ?????????
          }

          ???????????;
     }

     public static void main(String[] args) {

          System.out.println(indexOf("hofmann", "nn", 0));

          System.out.println(indexOf("hofmann", "na", 0));
     }
}

// Ausgabe:
// 5
//-1
```

Nun eine etwas komplexere Aufgabenstellung:

1.2.2. Beispiel: find and replace von Text-Mustern

Gegeben ist ein sql-file, das in eine Oracle-Datenbank eingespielt werden soll.

Das Problem dabei ist, dass die Form der Datumsangabe (z.B: '30 June 2006 5:00 PM') dieses Einspielen verhindert. Die Datumsangaben müssen auf folgendes Format umgestellt werden:

str_to_date('30 June 2006 5:00 PM', '%d %M %Y %h:%i PM')

Hier die Datensätze (Auszug):

Ersetze

```
insert into Spiel values( 57,'Quarter-finals','30 June 2006 5:00 PM',
'Germany','Argentina','Berlin',72000 );
insert into Spiel values( 58,'Quarter-finals','30 June 2006 9:00 PM',
'Italy','Ukraine','Hamburg',50000  );
insert into Spiel values( 59,'Quarter-finals','1 July 2006 5:00 PM',
'England','Portugal','Gelsenkirchen',52000  );
insert into Spiel values( 60,'Quarter-finals','1 July 2006 9:00 PM',
'Brazil','France','Frankfurt',48000 );
insert into Spiel values( 61,'Semi-Finals','4 July 2006 9:00 PM',
'Germany','Italy','Dortmund',65000 );
```

durch

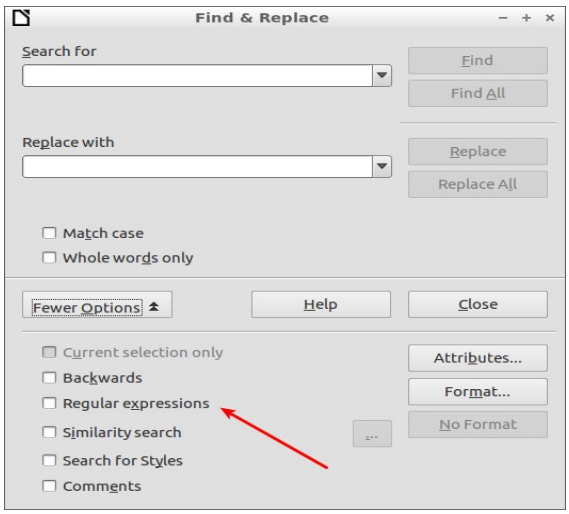
```
insert into Spiel values( 57,'Quarter-finals',str_to_date('30 June 2006 5:00
PM', '%d %M %Y %h:%i PM'),'Germany','Argentina','Berlin',72000 );

insert into Spiel values( 58,'Quarter-finals',str_to_date('30 June 2006 9:00
PM', '%d %M %Y %h:%i PM'),'Italy','Ukraine','Hamburg',50000  );

insert into Spiel values( 59,'Quarter-finals',str_to_date('1 July 2006 5:00
PM', '%d %M %Y %h:%i PM'), 'England','Portugal','Gelsenkirchen',52000);

insert into Spiel values( 60,'Quarter-finals',str_to_date('1 July 2006 9:00
PM', '%d %M %Y %h:%i PM'),'Brazil','France','Frankfurt',48000 );

insert into Spiel values( 61,'Semi-Finals',str_to_date('4 July 2006 9:00 PM',
'%d %M %Y %h:%i PM'),'Germany','Italy','Dortmund',65000 );
```

Lösung:

Da einfache Textsuche nicht möglich ist, (Jeder zu suchende String hat zwar den gleichen Aufbau aber besteht aus unterschiedlichen Zeichen)

verwenden wir das Suchen/Ersetzen eines Editors, der sogenannte „reguläre Ausdrücke“ verwendet.

Menü: edit->suchen&ersetzen (^H)



Wenn wir den Aufbau des zu ersetzenden Strings analysieren, fällt folgendes auf:

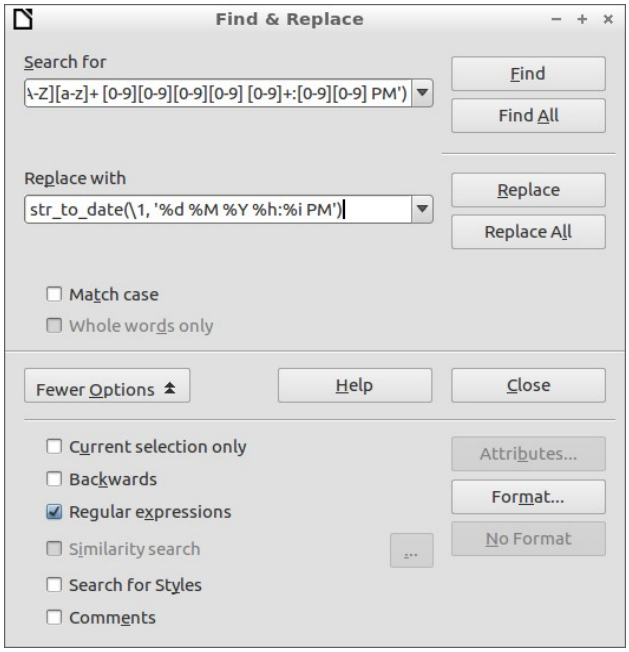
```
'30 June 2006 5:00 PM'
'1 July 2006 5:00 PM'
```

1. 1 oder 2 stellige Ziffern
2. Leerzeichen
3. mehrstellige Zeichen/Buchstaben (klein bzw. gross)
4. Leerzeichen
5. 4 stellige Ziffern
6. Leerzeichen
7. 1 oder 2 stellige Ziffer
8. :
9. 1 oder 2 stellige Ziffer
10. Leerzeichen
11. PM oder AM

Ein sogenannter REGEX-Ausdruck sieht dazu folgendermaßen aus:
('[0-9]+ [A-Z][a-z]+ [0-9][0-9][0-9][0-9] [0-9]+:[0-9][0-9] PM')

☑ **Aufgabe SUCHEN**:
geben sie den obigen REGEX in das Suchfeld (^H) ein, und starten Sie die Suche. Vergessen Sie nicht unter Optionen die Regular expression Checkbox zu selektieren.

☑ **Aufgabe SUCHEN&Ersetzen**:
gegeben Sie nun noch in das Replace with Feld folgenden Text ein und starten Sie das Ersetzen:
str_to_date(&, '%d %M %Y %h:%i PM') (bei libreoffice)
str_to_date($1, '%d %M %Y %h:%i PM') (bei cpp)



☑ **Zusammenfassung**:
Wenn Sie also wissen wollen, ob der eingegebene Text z.B.
- ☐ eine email-Adresse,
- ☐ ein Auto-Kennzeichen oder
- ☐ eine Fliesskommazahl,
- ☐ …

ist, dann können Sie dies durch die Angabe von **Textmustern** (wir sagen: reguläre Ausdrücke; englisch: **regular expressions**; kurz: REGEX) leicht feststellen.

Wir wollen in der Folge diese REGEX genauer kennen lernen.

1.2.3. Vergleich mit exaktem Muster

☑ Beispiel1:

```
Der Kandidat hat von den 50 Fragen nur 25 richtig beantwortet.
```

Wenn Sie im obigen Text die Zahlen durch 100 ersetzen wollen, wird es schwierig. Wir brauchen ein Regelwerk, um ein Muster/Pattern für ganze Zahlen anzugeben.

So etwas wie: **eine ganze Zahl besteht**

☑ **eventuell einem Vorzeichen und**

☑ **gefolgt von einer oder mehreren Ziffern.**

In der Folge lernen wir **Regeln zur Definition von Mustern** kennen.

☑ Beispiel2:

Hier zunächst ein Muster zur Erkennung von email-Adressen:

\b[A-Z0-9._%-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b

erscheint sehr kompliziert, aber vielleicht können wir am Ende dieses Kurses das Muster lesen.
(siehe: http://www.regular-expressions.info/regexbuddy/email.html)

1.2.4. Zeichenklassen: [A-Z0-9]

Zeichenklassen erlauben das Vorkommen von

mehreren **verschiedenen** Zeichen **AN EINER STELLE**.

Dazu werden die möglichen Zeichen zwischen eckige Klammern gesetzt:

Muster/regulärer Ausdruck	Übereinstimmung
[JL]ava	Das Wort beginnt mit **J ODER L** Ok: **Lava** oder. **Java** aber nicht: java bzw. lava

☑ Regular Expressions sind **case-sensitive**, d.h. unser vorheriger Ausdruck trifft nicht auf „lava“ oder „java“ zu. Das können wir leicht ändern, indem wir die kleinen Buchstaben ebenfalls in die Zeichenklasse aufnehmen:

☑ **[JLjl]ava**
ok: **Java** ODER **java** ODER **Lava** ODER **lava**

1.2.5. Zeichenklassen: [a-z] [0-9] [a\-z][^a-zA-Z]

Nun wollen wir einen regulären Ausdruck erstellen, der auf jede klein geschriebene Zeichenfolge mit 3 Buchstaben zutrifft (zB: akl, uzs, dfg, ...).

Nach unserem bisherigen Wissensstand würde das so aussehen:
`[abcdefghijklmnopqrstuvwxyz][abcdefghijklmnopqrstuvwxyz]`
`[abcdefghijklmnopqrstuvwxyz]`

Mit dem Bindestrich kann mein eine Abkürzung erreichen.

☑ 3 Kleinbuchstaben:
[a-z][a-z][a-z]

☑ Dieser Ausdruck trifft auf alle **3-stelligen Ziffernfolgen** zu.
[0-9][0-9][0-9]

☑ Wollen wir den Bindestrich als Teil der Zeichenklasse verwenden müssen wir davor einen Backslash setzen:

[a\-z] Hier wird kein Bereich angegeben, sondern eine Zeichenklasse mit den **3 einzelnen Buchstaben 'a', '-' und 'z'**.
D.h. **a** ODER **–** ODER **z**

☑ Indem wir ein Zirkumflex (**^**) an den Beginn einer Zeichenklasse stellen, **negieren** wir sie. Somit trifft die Zeichenklasse auf alle Zeichen, außer jenen die in ihr definiert sind.

[^a-zA-Z] trifft also auf ein Zeichen zu, das **kein Buchstabe** ist.

Für das Zirkumflex gilt das selbe wie für den Bindestrich: Soll es ein Zeichen der Zeichenklasse sein, muss ein Backslash davor gesetzt werden.

Da diese Zeichenklassen so gebräuchlich sind, gibt es Abkürzungen für sie:

Beschreibung	**Zeichenklasse**	**Abkürzung**
Buchstaben, Ziffern und Unterstrich	[a-zA-Z_0-9]	**\w**
Alles **außer** Buchstaben, Ziffern und Unterstrich	[^a-zA-Z_0-9]	**\W**
Ziffern	[0-9]	**\d**

Alles **außer** Ziffern	[^0-9]	**\D**
Whitespaces	[\n\t]	**\s**
Alles außer Whitespaces	[^ \n\t]	**\S**
Ein beliebiges Zeichen außer Newline	.	**.**

Merke:

Muster/regulärer Ausdruck	**Übereinstimmung**
[a-z][a-z][a-z]	Ok: **abc** oder **ujh** oder **rrr** usw.... aber nicht: Abc, 3fg, ab, a, ...
[0-9][0-9][0-9]	OK: **123** oder **333**, … aber nicht: 12, 00a, ...
[a\-z]	OK: **a** oder **-** oder **z** aber nicht: alles andere
[^a-zA-Z]	OK: **1** oder **!** oder / oder usw… aber nicht: b, G, ...
[0-9A-F]	OK: ??????????? aber nicht: ??????????

1.2.6. Beginn und Ende Operatoren: ^, $

☑ Fall 1:
Wir wissen, dass das **^ in einer Zeichenklasse** eine Art **Negation** darstellt.
Also **[^0-9]** bedeutet **keine Ziffern**.

☑ Fall 2:
Was ist aber, wenn das ^ ausserhalb einer Zeichenklasse eingesetzt wird?
☐ **^** bedeutet **am Beginn** und
☐ **$** bedeutet **am Ende**

Beispiele:
☑ **^abc** **abc**, **abc**defg, **abc**123, ...
☑ **abc$** **abc**, endsin**abc**, 123**abc**, …

1.2.7. Quantoren: ?, * , +

Wie oft kann/muss ein Zeichen bzw. eine Zeichenklasse vorkommen?

Quantor	**Bedeutung**
☑ **?**	Kann **0- oder 1-mal** oft vorkommen
☑ *****	Kann **0- oder beliebig** oft vorkommen
☑ **+**	Kann **1- oder beliebig** oft vorkommen
☑ **{n, m}**	Muss **mindestens n und maximal m** mal vorkommen

☑ **{n}** Muss **genau n** mal vorkommen
☑ **{n, }** Muss **mindestens n** mal vorkommen

Beispiel:

☑ 3 Kleinbuchstaben:
statt Statt **[a-z][a-z][a-z]** kürzer **[a-z]{3}**

Frage: Gib Beispiele an.

Muster	Übereinstimmung
[+-]?[0-9]+	OK: ???????????? nicht aber: ?????
[+-]?[0-9]	OK: ???????????? nicht aber: ?????

1.2.8. Gierige * und nicht gierige ? Quantoren

Die drei Quantoren ?, * und + haben die Eigenschaft, die längste mögliche Zeichenfolge abzudecken – das nennt sich gierig (engl. greedy).

☑ Beispiel 1:

In einem HTML-String sollen alle fett gesetzten Teile **einzeln** gefunden werden.

Gesucht ist also ein Muster, das im String

```
String string = "Echt <b>fett</b>. <b>Cool</b>!";

die Teilfolgen <b>fett</b> und <b>Cool</b> erkennt.
```

☑ Die gierige Variante:
.* findet:
fett. Cool

Das verwundert nicht, denn mit dem Wissen, dass * gierig ist, passt .* auf die Zeichenkette vom ersten bis zum letzten .

Die Lösung ist der Einsatz eines nicht gierigen Operators (auch »genügsam«, »non-greedy« genannt). In diesem Fall wird hinter dem Qualifizierer einfach ein Fragezeichen gestellt.

Gieriger Operator	Nicht gieriger Operator ?
X?	X?**?**
X*	X***?**
X+	X+**?**
X{n}	X{n}**?**
X{n,}	X{n,}**?**
X{n,m}	X{n,m}**?**

☑ Die NICHT gierige Variante:

.*? findet:
fett und
Cool

AUFGABE: Testen Sie das folg. Beispiel mit http://regexr.com

☑ Text: **<html><head><title>Test</title></head><body><h1>Test</h1></body></html>**

☐ Muster **^<.+>** findet
`<html><head><title>Test</title></head><body><h1>Test</h1></body></html>`

aber

☐ Muster **^<.+?>** findet
`<html>`
`<head>`
`...`

Das ? macht das Muster sozusagen genügsam.

1.2.9. Gruppen

Genau wie in der Mathematik mit den **Klammern ein geschlossener Ausdruck** gebildet wird, geschieht das auch hier.

Man kann also hiermit die **Zahl der Zeichen verändern, die auf einen Stern-, Plus**-Operator oder einen Alternativ-Operator wirken.

Außerdem kann man auf die **Indexe** derartiger Gruppen zurückgreifen.

☑ Gruppen definieren mit runden Klammern
Wir können Zeichenketten und -klassen auch zu einer Gruppe zusammenfassen. Dazu setzen wir sie einfach zwischen **runde Klammern**:

([pP]erl)

☑ **Alternativen** in Gruppen mit | angeben
Es ist ebenfalls möglich Bedingungen in Gruppen zu formulieren, dazu wird das in vielen Programmiersprachen als „oder“-verwendete Zeichen '|' innerhalb einer Gruppe geschrieben.
(Perl|Python) gefällt mir (sehr gut|nicht).

Dieses Muster passt auf folgende Texte:

Perl gefällt mir sehr gut.
Python gefällt mir sehr gut.
Perl gefällt mir nicht.
Python gefällt mir nicht.

☑ Qantoren und Gruppen:
Ein regulärer Ausdruck, der viele blabla matchen würde:
(bla)+

Dies würde auf alle Strings matchen, die mindestens einmal oder mehrmals den String "bla" enthalten. zB.

"**bla**", "**blablabla**" etc..

☑ Beispiel: Ein regulärer Ausdruck, mit zwei Alternativen Schreibweisen eines Strings wäre zum Beispiel:

bl(a|u)fasel

Dies würde auf "**blafasel**" oder "**blufasel**" matchen, aber nicht auf "blofasel".

1.2.10. +Auf einzelne Gruppen zugreifen mit \ziffer

☑ **ziffer**
Dieser Operator bezieht sich auf eine Gruppe (), die sich vor dem Operator befindet.

☑ Die Gruppen werden autom. durchnummeriert.
Beginnend ab \1 usw. kann dann auf die jeweilge Gruppe zugegriffen werden.

☑ Beispiel:
(bla)\1

Dies würde auf "blabla" matchen.

Die erste Gruppe matcht auf "bla" und der Back-Referenz Operator matcht auf das, auf was auch die referenzierte Gruppe (Nummer 1) gematcht hat.

☑ Ein regulärer Ausdruck, der interessant ist:
(.*)\1

Dies würde auf jeden String matchen, der aus genau zwei identischen Hälften zusammengesetzt ist, wie zB "XYZXYZ".

1.3. Zusammenfassung

Quelle:
☐ http://de.selfhtml.org/perl/sprache/regexpr.htm

1.3.1. Reguläre Ausdrücke für einzelne Zeichen

Anmerkung:
Das Zeichen / wird hier nur zur optischen Begrenzung des Textes verwendet.

№	Regulärer Ausdruck (Beispiel)	passt auf eine Zeichenkette, die (mindestens)
1.	**/a/**	**ein** 'a' enthält

2.	**/[ab]/**	**ein** 'a' **oder** ein 'b' enthält
3.	**/[A-Z]/**	**einen** Großbuchstaben enthält (passt nicht auf Umlaute)
4.	**/[0-9]/**	**eine** Ziffer enthält
5.	**/\d/**	**eine** Ziffer enthält - genau wie (4.)
6.	**/\D/**	**ein** Zeichen enthält, das **keine** Ziffer ist
7.	**/[-\d]/**	**ein Minuszeichen oder eine** Ziffer
8.	**/[\[\]]/**	**eine** eckige Klammer enthält
9.	**/[a-zA-Z0-9_]/**	**ein** Zeichen vom Typ Buchstabe (ohne Umlaute) oder vom Typ Ziffer oder einen Unterstrich enthält
10.	**/\w/**	**eins** der Zeichen vom Typ Buchstabe, vom Typ Ziffer oder einen Unterstrich enthält - (fast) genau wie (9.); ob Umlaute erkannt werden können, hängt von der Systemkonfiguration ab
11.	**/\W/**	**ein** Zeichen enthält, was weder Buchstabe noch Ziffer noch Unterstrich ist; ob Umlaute ausgeschlossen werden können, hängt von der Systemkonfiguration ab
12.	**/\r/**	**ein** Steuerzeichen für den Wagenrücklauf enthält
13.	**/\n/**	ein Steuerzeichen für den Zeilenvorschub enthält
14.	**/\t/**	ein Steuerzeichen für den Tabulator enthält
15.	**/\f/**	ein Steuerzeichen für den Seitenvorschub enthält
16.	**/\s/**	ein Leerzeichen oder ein Steuerzeichen aus (12.-15.) enthält
17.	**/\S/**	ein Zeichen enthält, das kein Leerzeichen oder Steuerzeichen aus (12.-15.) ist
18.	**/[^äöüÄÖÜ]/**	**ein** Zeichen enthält, das kein deutscher Umlaut (in der entsprechenden Zeichenkodierung) ist
19.	**/[^a-zA-Z]/**	ein Zeichen enthält, das kein Buchstabe ist (ohne Umlaute)

1.3.2. Reguläre Ausdrücke für Zeichenketten

?	0mal oder 1mal
*	0mal oder beliebig
+	1mal oder beliebig
\b	Wortgrenze
^	Beginn
$	Ende
.	beliebiges Zeichen außer newline '\n'

№	Regulärer Ausdruck (Beispiel)	Wirkung
1.	**/aus/**	passt auf '**aus**' - auch in 'H**aus**' oder 'M**aus**i'
2.	**/aus?/**	passt auf '**aus**' usw. - aber auch '**au**'

3.	**/a./**	passt auf '**ab**' und '**an**' (ein beliebiges Zeichen hinter 'a', außer \n)
4.	**/a+/**	passt auf '**a**' oder '**aa**' oder '**aaaaa**' (ein oder beliebig viele 'a')
5.	**/a*/**	passt auf '**a**' oder '**aa**' oder '**aaaaa**' (kein oder beliebig viele 'a')
6.	**/Ha.s/**	passt auf '**Haus**' oder '**Hans**' aber nicht 'Hannes'
7.	**/Ha.+s/**	passt auf '**Haus**' oder '**Hans**' oder '**Hannes**' (ein oder beliebig viele beliebige Zeichen, außer \n)
8.	**/Ha.*s/**	passt auf '**Has**' oder '**Haus**' oder '**Hans**' oder '**Hannes**' (kein oder beliebig viele beliebige Zeichen, außer \n)
9.	**/Ha.?s/**	passt auf '**Haus**' oder '**Hans**' oder '**Hase**' ...
10.	**/x{10,20}/**	passt auf zwischen 10 und 20 'x' in Folge
11.	**/x{10,}/**	passt auf 10 und mehr 'x' in Folge
12.	**/x.{2}y/**	passt auf '**xxxy**' oder '**xaby**' usw. (**zwei beliebige Zeichen** zwischen 'x' und 'y', außer \n)
13.	**/Hans\b/**	passt auf '**Hans**' aber nicht 'Hansel' (**Wortgrenze**)
14.	**/\baus/**	passt auf '**aus**' oder '**aussen**' aber nicht 'Haus' (**Wortgrenze**)
15.	**/\baus\b/**	passt auf '**aus**' aber nicht 'Haus' und auch nicht 'außen' (Wortgrenze)
16.	**/\baus\B/**	passt auf '**aussen**' aber nicht 'aus' und auch nicht 'Haus' (Wortgrenze und "negative" Wortgrenze)
17.	**/^Hans/**	passt auf '**Hans**' nur am **Anfang** des zu durchsuchenden Bereichs
18.	**/Hans$/**	passt auf '**Hans**' nur am **Ende** des zu durchsuchenden Bereichs
19.	**/^\s*$/**	passt auf Zeilen, die nur aus Leerzeichen und anderen Leerraumzeichen bestehen oder leer sind

1.3.3. Maskierung von Zeichen in regulären Ausdrücken

Da es bei regulären Ausdrücken einige Zeichen mit Sonderbedeutung gibt, müssen Sie solche Zeichen maskieren, wenn Sie nicht die Sonderbedeutung des Zeichens meinen. Das Maskierungszeichen ist in allen Fällen der **Backslash**.

Zeichen	Maskierung	Grund	Beispiel (Muster)
.	\.	Der Punkt steht in regulären Ausdrücken ansonsten für ein beliebiges anderes Zeichen.	**/Ende aus\./**
+	\+	Das Pluszeichen steht ansonsten für ein oder mehrmaliges Vorkommen des davorstehenden Zeichens.	**/\d\+\d/**
*	*	Das Sternzeichen steht ansonsten für kein, ein oder mehrmaliges Vorkommen des davorstehenden Zeichens.	**/char*/**

?	\?	Das Fragezeichen steht ansonsten für kein oder einmaliges Vorkommen des davorstehenden Zeichens.	**/Wie geht das\?/**
^	\^	Das Dach- oder Hütchensymbol kann ansonsten eine Zeichenklasse verneinen oder bei Zeichenketten angeben, dass das nachfolgende Suchmuster am Anfang des Suchbereichs vorkommen muss.	**/ein \^ über dem Kopf/**
$	\$	Das Dollarzeichen kann bei Zeichenketten angeben, dass das voranstehende Suchmuster am Ende des Suchbereichs vorkommen muss.	**/Preis (US-Dollar): \d*\$/**
|	\\|	Der Senkrechtstrich kann ansonsten alternative Ausdrücke auseinanderhalten.	**/find (.*) | sort/**
\	\\	Der Backslash würde ansonsten das nachfolgende Zeichen maskieren.	**/C:\\/**
( )	\(\)	Runde Klammern können ansonsten Teilausdrücke gruppieren und zum Merken einklammern.	**/\(Hinweis: (.*)\)/**
[]	\[\]	Eckige Klammern begrenzen ansonsten eine Zeichenklasse.	**/\$(.*)\[\d+\]/**
{ }	\{ \}	Geschweifte Klammern bedeuten ansonsten eine Wiederholungs-Angabe für davorstehende Zeichen.	**/ENV\{.*\}/**

1.4. Übungsbeispiele

Sind die in der folgenden Tabelle angegebenen Aussagen wahr oder falsch?

MUSTER	Match	no Match	(w)ahr od. (f)alsch	Lösung
blafasel	blafasel	blufasel		
bl.	bla, blu	bal, bul		
inter\.net	inter.net	inter-net		
bla*	bl, blaaa	ba, bar		
bla+	bla, blaa	bl, ba		
ball?	bal, ball	ba, bull		
z{3}	zzz	z, zzzz		
z{2,}	zz, zzzz	z		
z{2,3}	zz, zzz	z, zzzz		
[au]nd	and, und	ind, mnd		
[^au]nd	ind, mnd	and, und		
[a-e]	a, b, d, e	f, g, t		
(br)+	br, brbrbr	r, rrr, b		

and|und	and, und	ind, mnd		
^time	time out	last time		
time$	last time	time out		
.	b, c,	Fa,abc		
b.a	bla, bba, bra	faa, zaa		
b\.a	b.a			
bl*	b, bl, blll	ba, bla		
bl+	bl, bla, blll	b, ba		
n.+b	nabana	nbana		
bl?a	bla, ba	blla, blu		
´ga{3}nz´	gaaanz	ganz gaanz gaaaaaaaaanz		
´ga{2,}nz´	gaanz gaaaaaaanz	ganz		
´ga{2,5}nz´	gaanz gaaaaanz	ganz gaaaaaaaaanz		
bla|fasel	bla oder fasel			
bl(a|f)asel	blaasel und blfasel	blafasel		
[kflr]ind	kind, find, lind, und rind	sind , wind, blind		
[^kflr]ind	sind oder wind	kind, find, lind und rind		

☑ Frage:
Wenn Sie zum Beispiel ein Logfile untersuchen, und nur an Zeilen interessiert sind, in denen im vorderen Teil der String "error" vorkommt, und etwas weiter hinten der String "critical". Außerdem wissen Sie, dass zwischen den Strings andere Zeichen vorkommen, die aber beliebig und in beliebiger Anzahl erscheinen. Ein regulärer Ausdruck der Ihnen genau diese Zeilen herausfiltert sieht zum Beispiel so aus:

☑ Antwort:
error.*critical

☑ Merke:
.* bedeutet: jedes beliebige Zeichen beliebig oft oder sogar keinmal.

☑ Frage:
Wer ein Chat-Logfile untersucht, möchte vielleicht alle Zeilen, in denen das Wort "ganz" vorkommt finden. Nun ist es bei Chattern oft so, dass sie zum Beispiel schreiben "ich hab dich ggaaanz lieb". Um unabhängig von der Zahl der Wiederholungen diese Zeilen zu finden, könnte man folgenden regulären

Ausdruck einsetzten:

☑ Anwort:
g+a+n+z+

Er findet sowohl "ganz" als auch jede Kombination von "ggaaannzz".

☑ Frage:
wie lautet das Muster, um Sauersto**fff**lasche aber auch Sauersto**ff**lasche zu matchen?

☑ Antwort:
Sauerstofff?lasche

1.5. Beispiel: REGEX und MySQL

REGEX kann auch in Verbindung von Datenbanken verwendet werden.

http://dev.mysql.com/doc/refman/5.1/de/regexp.html