Syntax von Regulären Ausdrücken

Reguläre Ausdrücke werden verwendet, um Textmuster zu beschreiben, nach welchen dann gesucht wird. Spezielle **Metazeichen** erlauben das Definieren von Bedingungen, beispielsweise soll ein bestimmter gesuchter String am Anfang oder am Ende einer Zeile vorkommen, oder ein bestimmtes Zeichen soll n mal vorkommen.

Einfache Treffer

Jedes einzelne Zeichen findet sich selbst, außer es sei ein **Metazeichen** mit einer speziellen Bedeutung.

Eine Sequenz von Zeichen findet genau dieses Sequenz im zu durchsuchenden String (Zielstring). Also findet das Muster (= reguläre Ausdruck) "bluh" genau die Sequenz "bluh" irgendwo im Zielstring.

Damit Du Zeichen, die üblicherweise als **Metazeichen** oder **Escape-Sequenzen** dienen, als ganz normale Zeichen ohne jede Bedeutung finden kannst, stelle so einem Zeichen einen "\" voran. Ein Beispiel: das Metazeichen "^" findet den Anfang des Zielstrings, aber "\^" findet das Zeichen "^" (Circumflex), "\\" findet also "\" etc.

Beispiele:

`foobar`	*findet den String 'foobar'*
`\^FooBarPtr`	*findet den String '^FooBarPtr'*

Escape-Sequenzen

Zeichen können auch angeben werden mittels einer **Escape-Sequenz**: "\n" findet eine neue Zeile, "\t" einen Tabulator etc. Etwas allgemeiner: \xnn, wobei nn ein String aus hexadezimalen Ziffern ist, findet das Zeichen, dessen ASCII Code gleich nn ist. Falls Du Unicode-Zeichen (16 Bit breit kodierte Zeichen) angeben möchtest, dann benutze '\x{nnnn}', wobei 'nnnn' – eine oder mehrere hexadezimale Ziffern sind.

`\xnn`	*Zeichen mit dem Hex-Code nn (ASCII-Text)*
`\x{nnnn}`	*Zeichen mit dem Hex-Code nnnn (ein Byte für ASCII-Text und zwei Bytes für Unicode-Zeichen*
`\t`	*ein Tabulator (HT/TAB), gleichbedeutend wie \x09*
`\n`	*Zeilenvorschub (NL), gleichbedeutend wie \x0a*
`\r`	*Wagenrücklauf (CR), gleichbedeutend wie \x0d*
`\f`	*Seitenvorschub (FF), gleichbedeutend wie \x0c*
`\a`	*Alarm (bell) (BEL), gleichbedeutend wie \x07*
`\e`	*Escape (ESC), gleichbedeutend wie \x1b*

Beispiele:

`foo\x20bar`	*findet 'foo bar' (beachte den Leerschlag in der Mitte)*
`\tfoobar`	*findet 'foobar', dem unmittelbar ein Tabulator vorangeht*

Zeichenklassen

Du kannst sogenannte **Zeichenklassen** definieren, indem Du eine Liste von Zeichen, eingeschlossen in eckige Klammern [], angibst. So eine Zeichenklasse findet genau **eines der aufgelisteten Zeichen** im Zielstring.

Falls das erste aufgelistete Zeichen, das direkt nach dem "[", ein "^" ist, findet die Zeichenklasse jedes Zeichen **außer** denjenigen in der Liste.

Beispiele:

`foob[aeiou]r`	*findet die Strings 'foobar', 'foober' etc. aber nicht 'foobbr', 'foobcr' etc.*
`foob[^aeiou]r`	*findet die Strings 'foobbr', 'foobcr' etc. aber nicht 'foobar', 'foober' etc.*

Innerhalb der Liste kann das Zeichen "-" benutzt werden, um einen **Bereich** von Zeichen zu definieren. So definiert a-z alle Zeichen zwischen "a" und "z" inklusive.

Falls das Zeichen "-" selbst ein Mitglied der Zeichenklasse sein soll, dann setze es als erstes oder letztes Zeichen in die Liste oder schütze es mit einem vorangestellten "\" (escaping). Wenn das Zeichen "]" ebenfalls Mitglied der Zeichenklasse sein soll, dann setze es als erstes Zeichen in die Liste oder escape es.

Beispiele:

`[-az]`	*findet 'a', 'z' und '-'*
`[az-]`	*findet 'a', 'z' und '-'*
`[a\-z]`	*findet 'a', 'z' und '-'*
`[a-z]`	*findet alle 26 Kleinbuchstaben von 'a' bis 'z'*
`[\n-\x0D]`	*findet eines der Zeichen #10, #11, #12 oder #13.*
`[\d-t]`	*findet irgendeine Ziffer, '-' oder 't'.*
`[]-a]`	*findet irgendein Zeichen von ']'..'a'.*

Metazeichen

Metazeichen sind Zeichen mit speziellen Bedeutungen. Sie sind die Essenz der regulären Ausdrücke. Es gibt verschiedene Arten von Metazeichen wie unten beschrieben.

Metazeichen - Zeilenseparatoren

`^`	*Beginn einer Zeile*
`$`	*Ende einer Zeile*
`\A`	*Anfang des Textes*
`\Z`	*Ende des Textes*
`.`	*irgendein beliebiges Zeichen*

Beispiele:

`^foobar`	*findet den String 'foobar' nur, wenn es am Zeilenanfang vorkommt*
`foobar$`	*findet den String 'foobar' nur, wenn es am Zeilenende vorkommt*
`^foobar$`	*findet den String 'foobar' nur, wenn er der einzige String in der Zeile ist*
`foob.r`	*findet Strings wie 'foobar', 'foobbr', 'foob1r' etc.*

Standardmäßig garantiert das Metazeichen "^" nur, dass das Suchmuster sich am Anfang des Zielstrings befinden muss, oder am Ende des Zielstrings mit dem Metazeichen "$". Kommen im Zielstring Zeilenseparatoren vor, so werden diese von "^" oder "$" nicht gefunden. Du kannst allerdings den Zielstring als mehrzeiligen Puffer behandeln, so dass "^" die Stelle unmittelbar nach, und "$" die Stelle unmittelbar vor irgendeinem Zeilenseparator findet. Du kannst diese Art der Suche mit dem Modifikator /m einstellen.

Die Metazeichen \A und \Z wirken wie ^ und $, nur dass bei Benutzung des Modifkators /m nicht mehrfach fündig werden.

Das "." Metazeichen findet standardmäßig irgendein beliebiges Zeichen, also auch Zeilenseparatoren. Wenn Du den Modifikator /s ausschaltest, dann findet '.' keine Zeilenseparatoren mehr.

"^" ist am Anfang des Eingabestrings, und, falls der Modifikator /m gesetzt ist, auch unmittelbar folgend einem Vorkommen von \x0D\x0A oder \x0A oder \x0D.

"$" ist am Anfang des Eingabestrings, und, falls der Modifikator /m gesetzt ist, auch unmittelbar vor einem Vorkommen von \x0D\x0A oder \x0A oder \x0D.

"." findet ein beliebiges Zeichen. Wenn Du aber den Modifikator /s benutzt, dann findet "." keine Zeilenseparatoren \x0D\x0A und \x0A und \x0D mehr.

Beachte, dass "^.*$" (was auch eine leere Zeile finden können sollte) dennoch nicht den leeren String innerhalb der Sequenz \x0D\x0A findet, aber es findet den Leerstring innerhalb der Sequenz \x0A\x0D.

Metazeichen – vordefinierte Klassen

`\w`	*ein alphanumerisches Zeichen inklusive "_"*
`\W`	*kein alphanumerisches Zeichen, auch kein "_"*
`\d`	*ein numerisches Zeichen*
`\D`	*kein numerisches Zeichen*
`\s`	*irgendein wörtertrennendes Zeichen (entspricht [\t\n\r\f])*
`\S`	*kein wörtertrennendes Zeichen*

Du kannst \w, \d und \s innerhalb Deiner selbstdefinierten Zeichenklassen benutzen.

Beispiele:

`foob\dr`	*findet Strings wie 'foob1r', "foob6r' etc., aber not 'foobar', 'foobbr' etc.*
`foob[\w\s]r`	*findet Strings wie 'foobar', 'foob r', 'foobbr' etc., aber nicht 'foob1r', 'foob=r' etc.*

Metazeichen – Wortgrenzen

`\b`	*findet eine Wortgrenze*
`\B`	*findet alles außer einer Wortgrenze*

Eine Wortgrenze (\b) ist der Ort zwischen zwei Zeichen, welcher ein \w auf der einen und ein \W auf der anderen Seite hat bzw. umgekehrt. \b bezeichnet alle Zeichen des \w bis vor das erste Zeichen des \W bzw. umgekehrt.

Metazeichen - Iteratoren

Jeder Teil eines regulären Ausdruckes kann gefolgt werden von einer anderen Art von Metazeichen – den **Iteratoren**. Dank dieser Metazeichen kannst Du die Häufigkeit des Auftretens des Suchmusters im Zielstring definieren. Dies gilt jeweils für das vor diesem Metazeichen stehenden Zeichen, das **Metazeichen** oder den **Teilausdruck**.

`*`	*kein- oder mehrmaliges Vorkommen ("gierig"), gleichbedeutend wie {0,}*
`+`	*ein- oder mehrmaliges Vorkommen ("gierig"), gleichbedeutend wie {1,}*
`?`	*kein- oder einmaliges Vorkommen ("gierig"), gleichbedeutend wie {0,1}*
`{n}`	*genau n-maliges Vorkommen ("gierig")*
`{n,}`	*mindestens n-maliges Vorkommen ("gierig")*
`{n,m}`	*mindestens n-, aber höchstens m-maliges Vorkommen ("gierig")*
`*?`	*kein- oder mehrmaliges Vorkommen ("genügsam"), gleichbedeutend wie {0,}?*
`+?`	*ein oder mehrmaliges Vorkommen ("genügsam"), gleichbedeutend wie {1,}?*
`??`	*kein- oder einmaliges Vorkommen ("genügsam"), gleichbedeutend wie {0,1}?*
`{n}?`	*genau n-maliges Vorkommen ("genügsam")*
`{n,}?`	*Mindestens n-maliges Vorkommen ("genügsam")*
`{n,m}?`	*mindestens n-, aber höchstens m-maliges Vorkommen ("genügsam")*

Falls eine geschweifte Klammer in einem anderen als dem eben vorgestellten Kontext vorkommt, wird es wie ein normales Zeichen behandelt.

Beispiele:

`foob.*r`	*findet Strings wie 'foobar', 'foobalkjdflkj9r' und 'foobr'*
`foob.+r`	*findet Strings wie 'foobar', 'foobalkjdflkj9r', aber nicht 'foobr'*
`foob.?r`	*findet Strings wie 'foobar', 'foobbr' und 'foobr', aber nicht 'foobalkj9r'*
`fooba{2}r`	*findet den String 'foobaar'*
`fooba{2,}r`	*findet Strings wie 'foobaar', 'foobaaar', 'foobaaaar' etc.*
`fooba{2,3}r`	*findet Strings wie 'foobaar', or 'foobaaar', aber nicht 'foobaaaar'*

Eine kleine Erklärung zum Thema "gierig" oder "genügsam". "Gierig" nimmt so viel wie möglich, wohingegen "genügsam" bereits mit dem ersten Erfüllen des Suchmusters zufrieden ist. Beispiel: 'b+' und 'b*' angewandt auf den Zielstring 'abbbbc' findet 'bbbb', 'b+?' findet 'b', 'b*?' findet den leeren String, 'b{2,3}?' findet 'bb', 'b{2,3}' findet 'bbb'.

Du kannst alle Iteratoren auf den genügsamen Modus mit dem Modifiktor /g umschalten.

Metazeichen - Alternativen

Du kannst eine Serie von **Alternativen** für eine Suchmuster angeben, indem Du diese mit einem "|" trennst. Auf diese Art findet das Suchmuster fee|fie|foe eines von "fee", "fie", oder "foe" im Zielstring – dies würde auch mit f(e|i|o)e ereicht.

Die erste Alternative beinhaltet alles vom ersten Muster-Begrenzer ("(", "[" oder natürlich der Anfang des Suchmusters) bis zum ersten "|". Die letzte Alternative beinhaltet alles vom letzten "|" bis zum nächsten Muster-Begrenzer. Aus diesem Grunde ist es allgemein eine gute Gewohnheit, die Alternativen in Klammern anzugeben, um möglichen Missverständnissen darüber vorzubeugen, wo die Alternativen beginnen und enden.

Alternativen werden von links nach rechts geprüft, so dass der Treffer im Zielstring aus den jeweils zuerst passenden Alternativen zusammengesetzt ist. Das bedeutet, dass Alternativen nicht notwendigerweise "gierig" sind. Ein Beispiel: Wenn man mit "(foo|foot)" im Zielstring "barefoot" sucht, so passt bereits die erste Variante. Diese Tatsache mag nicht besonders wichtig erscheinen, aber es ist natürlich wichtig, wenn der gefundene Text weiterverwendet wird. Im Beispiel zuvor würde der Benutzer nicht "foot" erhalten, wie er eventuell beabsichtigt hatte, sondern nur "foo".

Erinnere Dich auch daran, dass das "|" innerhalb von eckigen Klammern wie ein normales Zeichen behandelt wird, so dass [fee|fie|foe] dasselbe bedeutet wie [feio|].

Beispiele:

`foo(bar|foo)`	*findet die Strings 'foobar' oder 'foofoo'.*

Metazeichen - Teilausdrücke

Das Klammernkonstrukt (...) wird auch dazu benutzt, reguläre Teilausdrücke zu definieren.

Teilausdrücke werden nummeriert von links nach rechts, jeweils in der Reihenfolge ihrer öffnenden Klammer. Der erste Teilausdruck hat die Nummer 1, der gesamte reguläre Ausdruck hat die Nummer 0.

Beispiele:

`(foobar){8,10}`	*findet Strings, die 8, 9 oder 10 Vorkommen von 'foobar' beinhalten*
`foob([0-9]|a+)r`	*findet 'foob0r', 'foob1r' , 'foobar', 'foobaar', 'foobaar' etc.*

Metazeichen - Rückwärtsreferenzen

Die **Metazeichen** \1 bis \9 werden in Suchmustern als Rückwärtsreferenzen interpretiert. \<n> findet einen zuvor bereits gefundenen **Teilausdruck** #<n>.

Beispiele:

`(.)\1+`	*findet 'aaaa' und 'cc'.*
`(.+)\1+`	*findet auch 'abab' und '123123'*
`(['"]?)(\d+)\1`	*findet "13" (innerhalb "), oder '4' (innerhalb ') oder auch 77, etc.*

Modifikatoren

Modifikatoren sind dazu da, das Verhalten der regulären Suche zu verändern. Jeder der Modifikatoren kann im Suchmuster des regulären Ausdruckes mittels des Konstruktes (?...) eingebettet werden.

i
: Führe die Suche Schreibweisen unabhängig durch.

m
: Behandle den Zielstring als mehrzeiligen String. Das bedeutet, ändere die Bedeutungen von "^" und "$": Statt nur den Anfang oder das Ende des Zielstrings zu finden, wird jeder Zeilenseparator innerhalb eines Strings erkannt.

s

Behandle den Zielstring als einzelne Zeile. Das bedeutet, dass "." jedes beliebige Zeichen findet, sogar Zeilenseparatoren, die es normalerweise nicht findet.

g

Modifikator für den "Genügsam"-Modus. Durch die Anwendung werden alle folgenden Operatoren in den "Genugsam"-Modus. Standardmäßig sind alle Operatoren "gierig". Wenn also der Modifikator /g aus ist, dann arbeitet '+' wie '+?', '*' als '*?' etc.

x

Erweitert die Lesbarkeit des Suchmusters durch Leerraum und Kommentare.

Der Modifikator /x selbst braucht etwas mehr Erklärung. Er sagt, dass er allen Leerraum ignorieren soll, der nicht escaped oder innerhalb einer Zeichenklasse ist. Du kannst ihn benutzen, um den regulären Ausdruck in kleinere, besser lesbare Teile zu zerlegen. Das Zeichen # wird nun ebenfalls als Metazeichen behandelt und leitet einen Kommentar bis zum Zeilenende ein. Beispiel:

```
(
(abc) # Kommentar 1
  |   # Du kannst Leerzeichen zur Formatierung benutzen
(efg) # Kommentar 2
)
```

Dies bedeutet auch, wenn Du echten Leerraum oder das # im Suchmuster haben möchtest (außerhalb einer Zeichenklasse, wo sie unbehelligt von /x sind), dann muss der entweder escaped oder mit der hexadezimalen Schreibweise angegeben werden. Beides zusammen sorgt dafür, dass reguläre Ausdrücke besser lesbar werden.

Quellen-Hinweis:
Die Unterstützung regulärer Ausdrücke wurde mit Hilfe der Klasse TRegExpr des Programmierers Andrey Sorokin realisiert.