# 1 Allgemeine Informationen und Vorbemerkungen

- **In diesem Dokument werden folgende Abkürzungen verwendet:**
  - `P:` RegEx-Pattern
  - `m:` Zeichenfolge (das Pattern trifft auf den markierten Teil zu (match))
  - `(!)` Achtung!
- Regular Expressions werden immer zeichenweise interpretiert. Symbole repräsentieren i.d.R. einzelne Zeichen oder eine Gruppe von Zeichen.
- Einige Symbole haben in unterschiedlichen Kontexten unterschiedliche Bedeutungen.
- **Runde Klammern** werden verwendet, um Teile des regulären Ausdrucks in eine Gruppe zusammenzufassen, z.B. um anschließend Symbole wie `?` auf die gesamte Gruppe anwenden zu können:
  - `Hello? World` (das `?` bezieht sich **nur auf das „o“**, *matcht also „Hello World“ und „Hell World“*)
  - `(Hello )?World` (das `?` bezieht sich **auf den gesamten Ausdruck** in der Klammer, *matcht also „Hello World“ und „World“*)
- **Gleichzeitig** erzeugen runde Klammern sog. „**capture groups**“, d.h.: der Teil eines Strings, auf den das Pattern in den Klammern zutrifft, wird separat in Variablen (`$1`, `$2`, `$3`, ...) abgelegt, um später darauf zurückgreifen zu können.
  - **Beispiel:** In der folgenden Dateiliste sollen die Datei*namen* vereinheitlicht, die Datei*endungen* jedoch beibehalten werden:
    - **Liste:** `bild1.jpg`, `bild2.gif`, `bild3.png`
    - **`P: ^.+\.(jpg|gif|png)$`**
    - **Erklärung:** Anfang der Zeichensequenz (`^`), beliebige Zeichenfolge *(hier: der Dateiname)* (`.+`), Das Zeichen „.“ (`\.`), eine der Zeichenketten `jpg`, `gif` oder `png` (`(jpg|gif|png)`), Ende der Zeichensequenz (`$`))
    - Je nach dem, ob `jpg`, `gif` oder `png` zutraf, liegt in `$1` jetzt `jpg`, `gif` oder `png`.
    - Mit folgendem Pattern **ersetzen**: `newname.$1`
    - **Ergebnis**: `newname.jpg`, `newname.gif`, `newname.png`
- **capturing groups** werden auch für Backreferences benötigt (`\1`, `\2`, `\3`, ...). Backreferences verhindern, dass gleiche Zeichensequenzen mehrmals getippt werden müssen; folgende Patterns sind identisch:
  - **`P: (larry@google.com).+?(larry@google.com)`**
  - **`P: (larry@google.com).+?(\1)`**
- Um nach einem Zeichen zu suchen, das gleichzeitig ein Symbol ist, muss das Zeichen mit einem Backslash (`\`) escaped werden. **Beispiel:** **`P: vier\.`** `|` `m: vier.` `|` `m: vier!`
- Ein regulärer Ausdruck setzt sich zusammen aus dem Pattern und den Flags: `/<pattern>/<flags>` (z.B. `/^la/gm`)
- Einige Symbole tragen die Attribute `greedy` oder `lazy`. Sie geben an, ob das Symbol, wenn ihm eine Begrenzung gegeben ist, nur bis zum 1. Vorkommen dieser Begrenzung (`lazy`) oder bis zum letzten Vorkommen der Begrenzung (`greedy`) reicht.
  **Beispiel:**
  - **`P: H.+G`** `|` `m: H&G` `|` `m: H&G&G` (Das Symbol `+` ist `greedy`)
  - **`P: H.+?G`** `|` `m: H&G` `|` `m: H&G&G` (Das Symbol `+?` ist `lazy`)
- Vielen Symbolen kann eine Anzahl angehängt werden: **`P: \w{5}`** `|` `m: aBcDe`
  Es ist auch möglich, einen Bereich als Anzahl anzugeben (n mal bis m mal):
  **`P: \w{5,10}`** `|` `m: aBcDeFg` `|` `m: aBcDeFgHiJkL`
  Oder als Bereich mit offenem Ende (n mal oder öfter):
  **`P: \w{2,}`** `|` `m: a` `|` `m: ab` `|` `m: abababababababab`
- Unter http://gskinner.com/RegExr/ können Regular Expressions in einer Flash-Anwendung getestet werden. Die Anwendung kann auch als AIR-App lokal installiert werden.

# 2 Symbole

| Symbol | Beschreibung/Voraussetzung für match | Beispiel |
|---|---|---|
| `.`<br>(any character) | irgendein Zeichen<br>(außer Zeilenumbrüchen wenn `dotall` (`s`) nicht gesetzt) | **P: Hello World.**<br>m: Hello World.<br>m: Hello Worlds<br>m: Hello World |
| `\w`<br>(word characters) | Trifft zu auf „word character“<br>(alphanumerische Zeichen und Unterstrich) | **P: \w{5}**<br>m: abc_1<br>m: EFabc_D |
| `\W`<br>(!word characters) | Trifft zu auf nicht-„word characters“<br>(alles **außer** alphanumerischen Zeichen und Unterstrich) | **P: \W{5}**<br>m: ‹›@€§<br>m: $%‹›@€§ |
| `\d`<br>(digit characters) | Trifft zu auf numerische Zeichen (0-9) | **P: \d{5}**<br>m: 13557<br>m: 86843063 |
| `\D`<br>(!digit characters) | Trifft zu auf alles **außer** numerischen Zeichen (0-9) | **P: \D{5}**<br>m: fj&ds<br>m: jk§!ffa! |
| `\s`<br>(whitespace characters) | Trifft zu auf alle Arten von Leerzeichen/Whitespace<br>(spaces ( ), tabs (`\t`), line breaks (`\r`, `\n`)) | **P: \s{5}**<br>m: \t\t \t\t<br>m: \t\r\n\r\n\t\r\n \r\n |
| `\S`<br>(!whitespace characters) | Trifft auf alle nicht-Whitespace-Zeichen zu<br>(alles **außer** spaces ( ), tabs (`\t`), line breaks (`\r`, `\n`)) | **P: \S{5}**<br>m: abcde<br>m: abcde fghij |
| `?`<br>(0 oder 1) | 0 oder 1 Vorkommen | **P: (Hello )?World**<br>m: Hello World<br>m: World |
| `*` (greedy)<br>`*?` (lazy)<br>(0 oder mehr) | 0 oder mehr Vorkommen | **P: g*world**<br>m: gggworld<br>m: gworld<br>m: world (!) |
| `+` (greedy)<br>`+?` (lazy)<br>(1 oder mehr) | 1 oder mehr Vorkommen | **P: g+world**<br>m: gggwolrd<br>m: gworld<br>m: world (!) |
| `\|`<br>(oder) | oder („alternation“) | **P: (ht\|f)tp://**<br>m: http://<br>m: ftp:// |
| `^`<br>(Anfang) | Repräsentiert den Anfang der Zeichenkette<br>(wenn `multiline` (`m`) gesetzt: auch Anfang der Zeile) | **P: ^la**<br>m: la<br>m: bla |
| `$`<br>(Ende) | Repräsentiert das Ende der Zeichenkette<br>(wenn `multiline` (`m`) gesetzt: auch Ende der Zeile) | **P: \.(jpg\|gif)$**<br>m: coolFile.jpg.jpg<br>m: coolFile.jpg.gif |
| `[‹chars›]`<br>z.B. `[aeiou]`<br>z.B. `[a-zA-Z]`<br>z.B. `[^abc]` *(not abc)* | Eines der Zeichen in den eckigen Klammern | **P: H[ae]llo**<br>m: Hallo<br>m: Hello<br>m: Hollo |
| `(?=‹chars›)`<br>(positive lookahead) | Die Zeichenfolge *‹chars›* ist **hinter dem davor stehenden Teil des Patterns zwingend erforderlich**, zählt aber nicht zum Treffer. | **P: larry(?=@google\.com)**<br>m: larry@google.com<br>m: larry@yahoo.com |
| `(?!‹chars›)`<br>(negative lookahead) | Die Zeichenfolge *‹chars›* **darf nicht nach dem davor stehenden Teil des Pattern existieren**, zählt aber nicht zum Treffer. | **P: larry(?!@google\.com)**<br>m: larry@google.com<br>m: larry@yahoo.com |
| `(?‹=‹chars›)`<br>(positive lookbehind) | Die Zeichenfolge *‹chars›* ist **vor dem nachstehenden Teil des Patterns zwingend erforderlich**, zählt aber nicht zum Treffer. | **P: (?‹=larry@)google\.com**<br>m: larry@google.com<br>m: sergey@google.com |
| `(?‹!‹chars›)`<br>(negative lookbehind) | Die Zeichenfolge *‹chars›* **darf nicht vor dem nachstehenden Teil des Pattern existieren**, zählt aber nicht zum Treffer. | **P: (?‹!larry@)google\.com**<br>m: larry@google.com<br>m: sergey@google.com |
| `(?:‹…›)`<br>(non-capturing group) | Das Hinzufügen von `?:` am Anfang einer runden Klammer macht diese zu einer non-capturing group.<br>(non-capturing groups erfordern weniger Rechenleistung) | **P: (?:Hello )?World**<br>m: Hello World<br>m: World |

# 3 Flags

| Flag (k) | Flag (l) | Beschreibung/Effekt nach dem Setzen des Flags |
|---|---|---|
| g | global | Hört nicht nach dem 1. Treffer auf zu suchen |
| i | ignoreCase | Achtet nicht auf Groß- und Kleinschreibung |
| s | dotall | . trifft auch auf Zeilenumbrüche zu |
| m | multiline | ^ und $ bedeuten auch „Anfang/Ende einer Zeile“ |
| x | extended | Leerräume im Pattern werden ignoriert, wenn sie nicht escaped sind oder sich innerhalb einer Zeichenkette befinden.<br>Zeichen, die außerhalb einer Zeichenklasse zwischen nicht-escaped # stehen, werden einschließlich dem nächsten Zeilenumbruch ignoriert.<br>Das ermöglicht es, kompliziertere Pattern mit Kommentaren zu versehen. |

Beispiel für ein kommentiertes Pattern, Flag `x` wird gesetzt (Sinn: trifft auf valide amerikanische Telefonnummern zu):

```
1  /^
2  (?:1-?)?     # 1- or 1 or nothing
3  \(?\d{3}\)? # look for the prefix, which is optional
4  [\s-]?       # look for a space, dash, or nothing
5  \d{3}        # three digits after the prefix (area code)
6  [\s-]?       # look for a space, dash, or nothing
7  \d{4}        # the 4 digit line number
8  $/mx
```

(trifft z.B. auf 1-615-555-1234 zu)

# 4 Conditional Expressions

Syntax: `P: (?(condition)yes-pattern|no-pattern)`

Beispiel:
```
P: ^(?(?=h)hog|(cog|log))
m: hog
m: zog
m: cog
m: log
```

(Cheat sheet auf Basis des tuts+-Kurses „Regular Expressions: Up and Running“, https://tutsplus.com/course/regular-expressions-up-and-running/)