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.
- **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`, `$3`, ...) abgelegt, um später darauf zurückgreifen zu können.
  - **Beispiel:** In der folgenden Dateiliste sollen die Datei*namen* vereinheitlicht, die Datei*endungen* jedoch beibehalten werden:
    - **Liste:** `bild1.jpg`, `bild2.gif`, `bild3.png`
    - **Pattern:** `^.+\.(jpg|gif|png)$`
    - 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!**
- Unter http://gskinner.com/RegExr/ können Regular Expressions in einer Flash-Anwendung getestet werden. Die Anwendung kann auch als AIR-App lokal installiert werden.
- 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 oder bis zum letzten Vorkommen der Begrenzung 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

2 Symbole

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

Symbol	Beschreibung/Voraussetzung für match	Beispiel
(?<!*<chars>*) (negative lookbehind)	Die Zeichenfolge *<chars>* **darf nicht vor dem nachstehenden Teil des Pattern existieren**, zählt aber nicht zum Treffer.	**P: (?<!larry@)google\.com** m: larry@google.com m: sergey@google.com
(?:<…>)	Das Hinzufügen von ?: am Anfang einer runden Klammer macht diese zu einer non-capturing-group. (non-capturing-groups erfordern weniger Rechenleistung)	**P: (?:Hello)?World** m: Hello World 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 „Anfang/Ende einer Zeile“, nicht „der Zeichenkette“.
x	extended	Leerräume im Pattern werden ignoriert, wenn sie nicht escaped sind oder sich innerhalb einer Zeichenkette befinden. Zeichen, die außerhalb einer Zeichenklasse zwischen nicht-escaped # stehen, werden einschließlich dem nächsten Zeilenumbruch ignoriert. Das ermöglicht es, kompliziertere Pattern mit Kommentaren zu versehen.

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

```
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/)