# Reguläre Ausdrücke

# Reguläre Ausdrücke – Übersicht

- Einführung
- Grundlagen
- Währenddessen: Übungen

# Was sind Reguläre Ausdrücke?

Reguläre Ausdrücke sind Zeichenketten, die Mengen von Zeichenketten beschreiben.

# Was sind Reguläre Ausdrücke?

Man kann sie benutzen, um Zeichenketten zu manipulieren oder validieren.

# Was sind Reguläre Ausdrücke?

Viele Texteditoren (auch oXygen) verarbeiten reguläre Ausdrücke in der Funktion „Suchen und Ersetzen“.

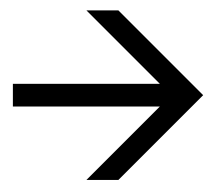
# Pattern Matching

Das ganze funktioniert via „Pattern Matching“. Reguläre Ausdrücke können als Filterkriterien in der Textsuche verwendet werden, indem der Text mit dem Muster des regulären Ausdrucks abgeglichen wird.

So ist es zum Beispiel möglich, alle Wörter in einem Text zu finden, die mit S beginnen und auf D enden, ohne die dazwischen liegenden Buchstaben oder deren Anzahl explizit angeben zu müssen.

# Beschreibung einer Menge von Zeichenketten

a



{"a"}

# Beschreibung einer Menge von Zeichenketten

a+

→

{"a", "aa", "aaa", "aaaa", ...}

# Beschreibung einer Menge von Zeichenketten

B(irn|(an){2})e

→

{"Birne", "Banane"}

# Reguläre Ausdrücke – Übersicht

- Standardzeichen
  - Buchstaben: A bis Z, a bis z
  - Ziffern: 0 bis 9
  - Symbole: !, @, #, %, &, usw.
- Reservierte Zeichen
  - ( ) [ ] \ ^ $ . | ? * +
  - Um reservierte Zeichen in einem String zu suchen, muss man sie mit einem Backslash (\) maskieren.

# Eine gute RegEx-Testseite

http://regex101.com

# Zeichenklassen

- [Aa]
- [^a] („^“ am Anfang einer Zeichenklasse negiert selbige)
- [A-Z] (Bindestriche sind Indikator für einen Bereich)
- [0-9]
- [A-Za-z0-9]
- [0-24] matcht „0“, „1“, „2“ und „4“
- „^“ ist ein reserviertes Zeichen und kann verschiedene Bedeutungen haben.
  - [^bgr] matcht alles außer „b“, „g“, „r“
  - [b^gr] matcht nur „b“, „^“, „g“, „r“

# Vordefinierte Zeichenklassen

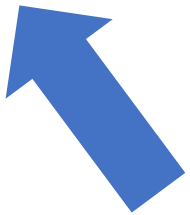
- \d (digit) = [0-9]
- \D = [^\d]
- \w (word character) =  ein Buchstabe, eine Ziffer oder der Unterstrich, also [a-zA-Z_0-9] (und je nach Implementation evtl. auch nicht-lateinische Buchstaben, z. B. Umlaute)
- \W = [^\w]
- \s (whitespace) = Space, Zeilenumbrüche, Tabulatorzeichen, vertikaler Whitespace (\v)
- \S = [^\s]
- . = alle Zeichen bis auf Zeilenumbrüche

# Pipes

- Eine Pipe matcht entweder den linken oder rechten Teil des Musters
  - rot|grün
  - Die Ampel ist (rot|grün).

# Pipes

- Eine Pipe matcht entweder den linken oder rechten Teil des Musters
  - rot|grün
  - Die Ampel ist (rot|grün).



Capturing Group
(zum Abgrenzen und Gruppieren eines Teils des regulären Ausdrucks)

# Quantoren

- {n} = Muster darf genau n mal vorkommen
- {n,m} = Muster darf mindestens n Mal und höchstens m Mal vorkommen.
- {n,} = Muster darf mindestens n Mal vorkommen.
- {0,m} = Muster darf höchstens m Mal vorkommen.
- * = {0,}
- + = {1,}
- ? = {0,1}

# Sonderzeichen

- ^ = Anfang der Zeichenkette (oder des Zeilenanfangs)
- $ = Ende der Zeichenkette (oder des Zeilenendes)
- \b = Wortgrenze

# Flags

Flags stehen am Ende des regulären Ausdrucks, getrennt vom restlichen Teil mit einem „/“

- i = case-insensitive
- s = Punkt frisst auch Zeilenumbrüche
- m = Die Zeichen ^ und $ matchen auch auf Zeilenanfänge bzw. -enden. Ohne den Modifikator passen sie nur auf Anfang und Ende der gesamten Zeichenkette.)
- g = Finde alle Vorkommnisse, nicht nur das erste Ergebnis in der Zeichenkette

# Cheatsheet

http://www.cbs.dtu.dk/courses/27610/regular-expressions-cheat-sheet-v2.pdf

# Übung 1

Trimme im Beispielcorpus den Text in <persName> mit <forename> und <surname> Kindern.

Bsp.: Boeckh_Plan.xml (aus dem Corpus „Berliner Intellektuelle“)

```
<respStmt>
    <resp>Edited by</resp>
    <persName ref="#sabine.seifert">
        <forename>Sabine</forename>
        <surname>Seifert</surname>
    </persName>
</respStmt>
```

Trimmen = Entfernen von Whitespaces am Anfang und Ende einer Zeichenkette
Whitespaces = Leerzeichen, Tabulatorzeichen, Zeilenumbrüche

# Übung 2

Finde in der Datei „Regex Übung Daten.txt“ valide Daten aus dem 20. und dem 21. Jahrhundert.

(Hinweis: Auf valide Tag-Monat-Kombinationen muss nicht geprüft werden. Daten wie der 30. Februar dürfen enthalten sein.)

# Übung 2

Finde in der Datei „Regex Übung Daten.txt“ valide Daten aus dem 20. und dem 21. Jahrhundert.

Mögliche Lösung:

```
^(0?[1-9]|[12][0-9]|3[01])\.(0?[1-9]|1[012])\.(19|20)\d\d
```

# Tipp 1

Oftmals führen viele verschiedene Wege ans Ziel. Dabei gibt es meistens allgemeinere (einfachere) und genauere (kompliziertere) Lösung.

# Tipp 2

Bei speziellen Problemen hilft oftmals googeln.

Vielen Dank für heute.
Bis morgen!