

# Regular Expressions for Technical Writers (tutorial)

tcworld conference 2016 - Stuttgart, Germany

Scott Prentice, Leximation, Inc.

modified 2017-05-13 (fixed typos)

# Introduction

— [ Scott Prentice, President of Leximation, Inc.

— [ Specializing in FrameMaker plugin development as well as structured FrameMaker conversions, consulting, and development. FrameMaker user/developer since 1991.

— [ Developed DITA-FMx, a FrameMaker plugin for efficient DITA authoring and publishing.

— [ Consulting for custom Help systems, creative/functional web applications, and EPUB solutions.

# Disclaimer

— [ This information is not exhaustive or complete

— [ Discuss regex features that may be most useful to technical writers

— [ Designed for beginners or infrequent regex users

— [ (However, some advanced topics are discussed)

# Audience participation

— [ Yes!

— [ Please ask questions when they come up

— [ Think of tasks you're faced with that might be solved with regular expressions

— [ I'll ask for those tasks in the Demo portion

# Regular expression?

- [ Regular expression, AKA “regex”

- [ Text string describing a search pattern

- [ Way beyond wildcards

- [ May also define a replacement string

- [ Replacement may contain content extracted from match

# Where can you use a regex?

— [ Many authoring tools provide regex support

— [ Most “serious” text editors

— [ Scripting languages like Perl, PHP, JavaScript, Python, Ruby

— [ Unix utilities like grep, sed, and awk

— [ Compiled programming languages like Java, C#, VB.NET

— [ Anything with a “regex engine”!

# Benefits

- [ Powerful searching

- [ Complex string replacements

- [ Intelligent modifications

- [ Text format conversions (this is huge)

- HTML or XML to CSV (or the other way around)

- HTML or XML cleanup

# But, better than wildcards?

— [ Yes. Much better.

— [ Wildcard search/replace is fine for simple patterns

— [ Regex is like a mini programming language

— [ Powerful syntax in very few characters



# Problems?

- [ Can appear very complex and overwhelming

- [ Regex syntax varies based on the “engine” and implementation

- [ Watch out for “greedy” matches

- [ Typically no “one right way” to do the same thing

- [ Some people say you shouldn't parse XML with a regex; as long as you understand the limitations it's fine

# Regex basics

Literal characters – **z, zorch, F00, F00**

Metacharacters – **\s, \S, \w, \W, \d, \D**

Anchors/boundaries – **^, \$, \b, \B**

Quantifiers – **\*, +, ?, {2}, {3,5}, {3,}**

Grouping – **., (...), (...|...), [...], [...-...], [^...]**

# Modifiers

— [ Common modifiers (options) in many tools

— g - global replace

— i - case insensitive match

— m - multiline mode (treats each line separately)

— s - single-line mode ("dot matches all", includes `\r\n`)

— x - free-spacing mode (comments follow "#")

— [ Inline use: `(?imsx)` enables, `(?-imsx)` disables

# Basic regex examples

Find the word .. "cat" (lowercase) – **\bcat\b**

.. "cat" or "dog" (lowercase) – **\b(cat|dog)\b**

.. "Cat" or "cat" – **\b[Cc]at\b**

.. "cat" followed by numbers – **\bcat[0-9]+\b**

.. that contains "cat" – **\Bcat\B**

.. that starts with "cat" – **\b[Cc]at\B**

# Backreferences / captures

— [ Backreferences match on an earlier group:

`class=(['' ])+?\1`

— [ Capture group uses content of matched group in replacement

— [ Tools use `\1` or `$1` to indicate captured string

— [ To get “number” count open parens from start (except non-capturing groups)

# Date regex examples

Match date in the form of yyyy-mm-dd or yyyy/mm/dd –  
`\b\d{4}[/-]\d\d[/-]\d\d\b` .. or  
`\b\d{4}([/-]\d\d?){2}\b` ..

Change the format of that date string to mm/dd/yyyy –  
m: `\b(\d{4})[/-](\d\d)[/-](\d\d)\b`  
r: `$2/$3/$1`

# Naturally "greedy"

— [ Regexes will typically match on as much as possible

— [ Need to add code for minimal match

— [ Match any char except ">" - `[^>]+`

— [ Use `?` for a minimal match - **this .\*? that**

— [ Use multiline mode (if possible) `(?m)`

# HTML/XML regex examples

— [ Extract tag name to **\$1** –

**<([\w-]+)[^>]\*>**

— [ Extract @class attribute value to **\$1** –

**<[\w-]+[^>]\***

**class\s?=\s?"([^\"]+)"[^>]\*>**

— [ Extract content from tag to **\$2** –

**<([\w-]+)[^>]\*>(.\*?)</\1>**



# Demo...

— [ Basic regex examples

— [ Date regex examples

— [ HTML/XML matching

— [ Questions?

# Where to start?

— [ Start simple, really simple .. get used to your editor

— [ Match on some literal characters

— [ Match on string of a specific length

— [ Try extracting and replacing portions of strings

— [ Use a text editor and match on some code, HTML, CSV, or whatever you're likely to encounter

# Tool-specific issues

— [ Adobe FrameMaker

— [ Adobe RoboHelp

— [ Microsoft Word

— [ MadCap Flare

— [ Oxygen XML

— [ Text editors and scripting languages

# General differences

— [ Text/code editors are line-based

— [ Authoring tools are paragraph-oriented

— [ Default may be single-line or multiline mode

— [ Not all modifiers are available in all tools (try inline)

— [ Use **\$1** or **\1** format for capture replacement match?

— [ Tool may or may not support backreferences

# FrameMaker (unstructured)

— [ Enable single-line mode with inline modifier **(?s)**

— [ Match: **\n** for EOL, **\x09** for line break (not **\r**),  
**\t** or **\x08** for tab

— [ Replace: **\r** or **\x09** for line break, **\x08** for tab

— [ Use **\$1** format for captured replacement value

# FrameMaker (structured)

— [ No single-line mode; inline modifiers not supported

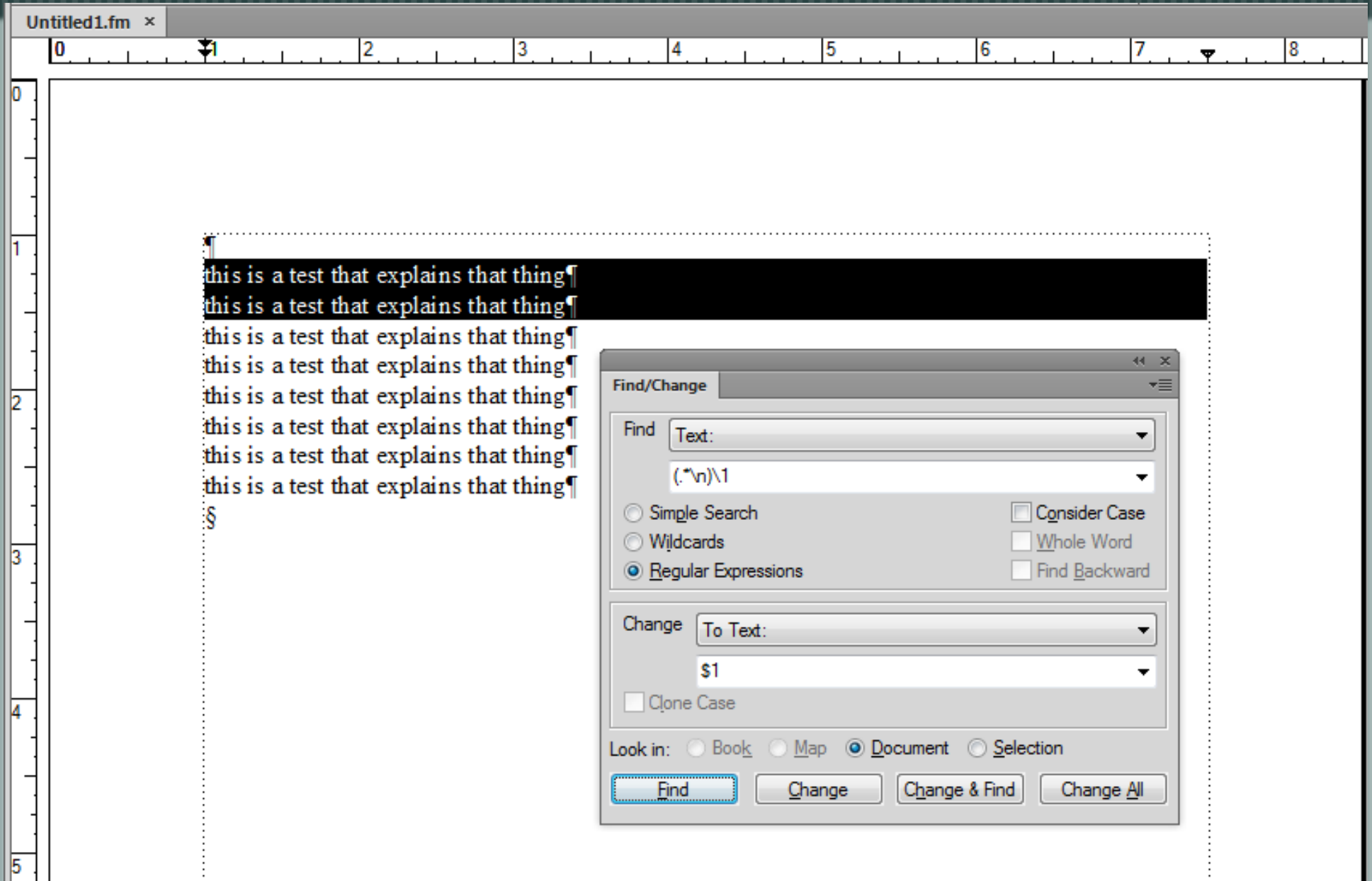
— [ Each node defines a “line” (match cannot span nodes)

— [ Use **\n** to match EOL (but that’s all it’ll match)

— [ Use **\$1** format for captured replacement value

— [ In XML View, use “Complex Expressions” option  
(limited features)

# FrameMaker



# RoboHelp

— [ Single-line mode is default in design view

— [ Multiline mode is default in source code view

— [ Inline modifiers not allowed, no capture group replacements

— [ Uses “Microsoft-style” regular expressions (??)

— [ Newline (**\n**) only matches in code view

— [ Supports find/replace in files



# RoboHelp

Rh Starter x Topic List x Chapter1 x Chapter2 x Chapter1.htm x

Design HTML

Document ▶

0 1 2 3 4 5 6


## Chapter One

This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content.

This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. For more information see ["Chapter Two" on page 3.](#)

- Bullet item one
- Bullet item two
  - Second paragraph in bullet item two
- Bullet item three

This is more simple content. This is more simple content. This is more simple content. This is more simple content. This is more simple content. This is more simple content. This is more simple content.



Find and Replace

Find Replace

Find: Show Advanced Filters

this .\*test

Look in:  
<Current Window>[Editor:Chapter1]

Hide Options

Files of Type: Text file types (\*.htm ; \*.html ; \*.t)

Match Case  Find in Source Code

Match Whole Word  Use: Regular Express

Direction: Forward

Find Next Find All

# MS Word

— [ Special MS hybrid regex/wildcard syntax; not “real”

— [ The **\*** matches anything except EOL (non-greedy),  
and **@** after a char or char class matches one or more

— [ Use **^13** to find a paragraph mark and replace with **^p**  
(replacing with **^13** can be bad)

— [ Find duplicate paras — **(\*^13)\1**

— [ Find duplicate “words” — **(<[a-zA-Z0-9]@>)\1**

# MS Word

The image shows a screenshot of the Microsoft Word application interface. The main window displays a document with several paragraphs of text. The text includes phrases like "how well it works. This is a test to see" and "Here's another para that's short and". At the bottom, there are three lines of text: "Para with dup dup words words to test test. Para with dup dup wor", "with dup dup words words to test test. Para with dup dup words w", and "dup dup words words to test test." The words "dup dup words words" in the first two lines are highlighted in blue.

Overlaid on the document is the "Find and Replace" dialog box. The "Find" tab is active. The "Find what:" field contains the wildcard search string: `(<[a-zA-Z]>) \1`. The "Options:" section is set to "Search Down, Use Wildcards". There is a checkbox for "Highlight all items found in:" set to "Current Selection". At the bottom of the dialog, there are "Close" and "Find Next" buttons.

Below the "Find" field, there is a "Search" section with a dropdown menu set to "Current Document All". Below this are several checkboxes: "Match case" (unchecked), "Find whole words only" (unchecked), "Use wildcards" (checked), "Sounds like" (unchecked), and "Find all word forms" (unchecked). Below the "Search" section is a "Find" section with "No Formatting" and "Format" buttons.

A dropdown menu is open from the "Format" button, listing various search criteria:

- Any Character
- Character in Range
- Beginning of Word
- End of Word
- Expression
- Not
- Num Occurrences
- Previous 1 or More
- 0 or More Characters
- Tab Character
- Comment Mark
- Caret Character
- Column Break
- Em Dash
- En Dash
- Graphic
- Manual Line Break
- Page / Section Break
- Nonbreaking Hyphen
- Nonbreaking Space
- Optional Hyphen

The status bar at the bottom of the window shows "Page 1 of 1", "30 of 385 Words", "English (US)", and "173%".

# Flare

— [ Best to use regexes in code view, seems unreliable in XML Editor view (search is done on underlying code)

— [ No single-line mode; inline modifiers not supported

— [ Use **\1** format for captured replacement value

— [ Supports find/replace in files

# Flare

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <html xmlns:MadCap="http://www.madcapsoftware.com/Schemas"
3 <head><title>Chapter Two</title>
4 <link href="testmap.css" rel="stylesheet" />
</head>
<body class="conbody">
  <h1 class="concepttitle">Chapter Two</h1>
  <p>This is a basic test of simple content. This is a basic
test of simple content. This is a basic test of simple content.
This is a basic test of simple content. This is a basic test of
simple content. This is a basic test of simple content. This is
a basic test of simple content. For more information see <MadCap:xref href="chapterone.htm#id59ddc450-81
  <ul>
    <li>
      <p>Bullet item one</p>
    </li>
    <li>
      <p>Bullet item two</p>
      <p>Second paragraph in bullet item two</p>
    </li>
    <li>
      <p>Bullet item three</p>
    </li>
  </ul>
  <p>This is more simple content. This is more simple content. This
is more simple content. This is more simple content. This is more
simple content. This is more simple content. </p>
  <div class="fig">
    <p>
      `

Replace with:  
`<\1 class="ctitle">`

Find in:  
(whole project)

File types:  
Topics

Find Options

- Match case
- Whole word
- Find in source code

Search type:  
Regular Expressions

Result Options

- Find Next
- Find Previous
- Skip File
- Find All
- Replace
- Replace All

# OxygenXML

- [ Enable single-line mode with “dot matches all” option
- [ Use **\1** format for captured replacement value
- [ Supports find/replace in files

# OxygenXML

The screenshot displays the OxygenXML editor interface. The main window shows an XML document titled "Untitled1.html\*" with the following content:

```
1 <!DOCTYPE html SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml">
3   <head>
4     <title>test</title>
5   </head>
6   <body>
7     <h1>chapter one</h1>
8     <p class="test">this is a test to see <b>how
9       a test to see how well it works. this is
10      to see how well it works. this is a test
11      how well it works. this is a test to see
12      well it works. this is a test to see how
13      it works. this is a test to see how well
14      works. </p>
15     <p>this is a test to see how well it works.
16       a test to see how well it works. this is
17       to see how well it works. this is a test
18       how well it works. this is a test to see
19       well it works. this is a test to see how
20       it works. this is a test to see how well
21       works. </p>
22     <p>this is a test to see how well it works.
23       a test to see how well it works. this is
24       to see how well it works. this is a test
25       how well it works. this is a test to see
26       well it works. this is a test to see how
27       it works. this is a test to see how well
28       works. </p>
29   </body>
30 </html>
```

The "Find/Replace" dialog box is open, showing the following settings:

- Find: `<\w+(.*?)>`
- Replace with: (empty)
- XPath: Type XPath expression
- Direction:  Forward,  Backward
- Scope:  All,  Only selected lines
- Options:  Case sensitive,  Incremental,  Wrap around,  Whole words only,  Regular expression,  Dot matches all,  Canonical equivalence
- Enable XML search options >>

The "Attributes" panel on the right shows the following table:

| Attribute | Value |
|-----------|-------|
| class     | test  |

The "Elements" panel at the bottom right shows a list of elements: cite, code, del, dfn, Transfo.., Entities, and Elements.

# TextWrangler

— [ Choose “grep” option to perform regex search/replace

— [ Enable single-line mode with inline modifier **(?s)**

— [ Use **\1** format for captured replacement value

— [ Supports find/replace in files



# TextWrangler

The screenshot shows the TextWrangler application window titled "(New Document)" with a file named "untitled text". The document contains several paragraphs of text. A "Find" dialog box is open, displaying a search pattern and replacement text.

**Find:** `(\d{4})-(\d{1,2})-(\d{1,2})`

**Replace:** `\2/\3/\1`

**Matching:**  Case sensitive  Entire word  Grep

**Search in:**  Selected text only  Wrap around

The dialog box also features navigation buttons: Next, Previous, Find All, Replace, Replace All, and Replace & Find.

The document text includes:

```
1 This is a test to see how well
2 a test to see how well it works
3 to see how well it works. This
4 how well it works. This is a te
5 it works.
6 This is a test to see how well
7 a test to see how well it works
8 to see how well it works. This
9 how well it works. This is a te
10 it works.
11 This is a test to see how well
12 a test to see how well it works
13 to see how well it works. This
14 how well it works. This is a te
15 it works.
16 Here's another para that's short and sweet. Some date 2016-1-3.
17 Here's another para that's short and sweet. Some date 2016-2-12.
18 Here's another para that's short and sweet. Some date 2016-12-31.
19 Here's another para that's short and sweet. Some date 2016-12-31.
20 Here's another para that's short and sweet. Some date 2016-12-31.
21 Here's another para that's short and sweet. Some date 2016-12-31.
22 Here's another para that's short and sweet. Some date 2016-12-31.
23 Para with dup dup words words to test test. Para with dup dup words words to test test.
24 Para with dup dup words words to test test. Para with dup dup words words to test test.
25 Para with dup dup words words to test test.
```

# Scripting with regexes

— [ Many languages provide regex modules

— [ Perform batch processing

— [ Easily repeat complex processing

— [ Perl and JavaScript are common

# Perl

— [ Tightly integrated into language

— [ Great for quick batch processing scripts

— [ Platform independent

— [ Find: `if ($str =~ m/\bcat\b/i) { ... }`

— [ Replace: `$str =~ s/\bcat\b/dog/g;`

# JavaScript

— [ Processing of HTML forms or other data

— [ search() - returns the position of the match (-1 if none)

```
var str = "Welcome to tcWorld";  
var pos = str.search(/tcworld/i);
```

— [ replace() - returns the new value

```
var ret = str.replace(/tcworld/ig,  
"Stuttgart");
```

# ExtendScript

— [ Scripting language in FrameMaker and RoboHelp

— [ Strip the full path and file name down to just the "name"

```
var doc = app.ActiveDoc;
```

```
var filename = doc.Name.replace
```

```
 (/^.*?([\^\]\+ )\.fm$/i, "$1");
```

# Regexes at the command line

— [ grep, sed, and awk are Unix utilities (Windows too)

— [ Often used together; pass output from one to another

— [ Can be used in shell scripts

— [ Wikipedia is a good place to start for more information on these utilities

# grep

— [ grep = Global Regular Expression Print

— [ Searches for regex matches in files (each line) or input

— [ Only scans each line, so not great for \*ML files

— [ Lists all DITA "task" files (recursive from "here")

**grep -rI "<task" \***

— [ Prints lines from CSV file with the specified pattern

**grep "\d{3}-[a-z]{4}" modes.csv**

# sed

— [ sed = Stream Editor

— [ One of the earliest tools to support regexes (1973)

— [ Line oriented matching or substitution

— [ Replace cat with dog in file

**sed -ie 's/\bcat\b/dog/g' file.txt**



# awk

— [ awk = surnames of creators (Aho, Weinberger, Kernighan)

— [ Programming language for processing text files

— [ Series of condition and action pairs

— [ Each line in text file is a “record” broken up into “fields”

— [ If condition matches in a line, the action is performed

— [ Fields are separated by whitespace, or as specified

# awk

— [ Print the lines that contain cat or Cat

```
awk '/\b[Cc]at\b/ {print $0}'  
dogs.txt
```

— [ Scan tab-delimited file for a name and print the first field  
from each "record"

```
awk 'BEGIN {FS="\t"} /[Jj]ohn/  
{print $1}' datafile.txt
```

# Demo...

— [ Simple Perl regex examples

— [ JavaScript regex examples

— [ grep, sed, awk examples

— [ Audience task examples?

— [ Questions?

# Resources

— [ RexEgg — [www.rexegg.com](http://www.rexegg.com)

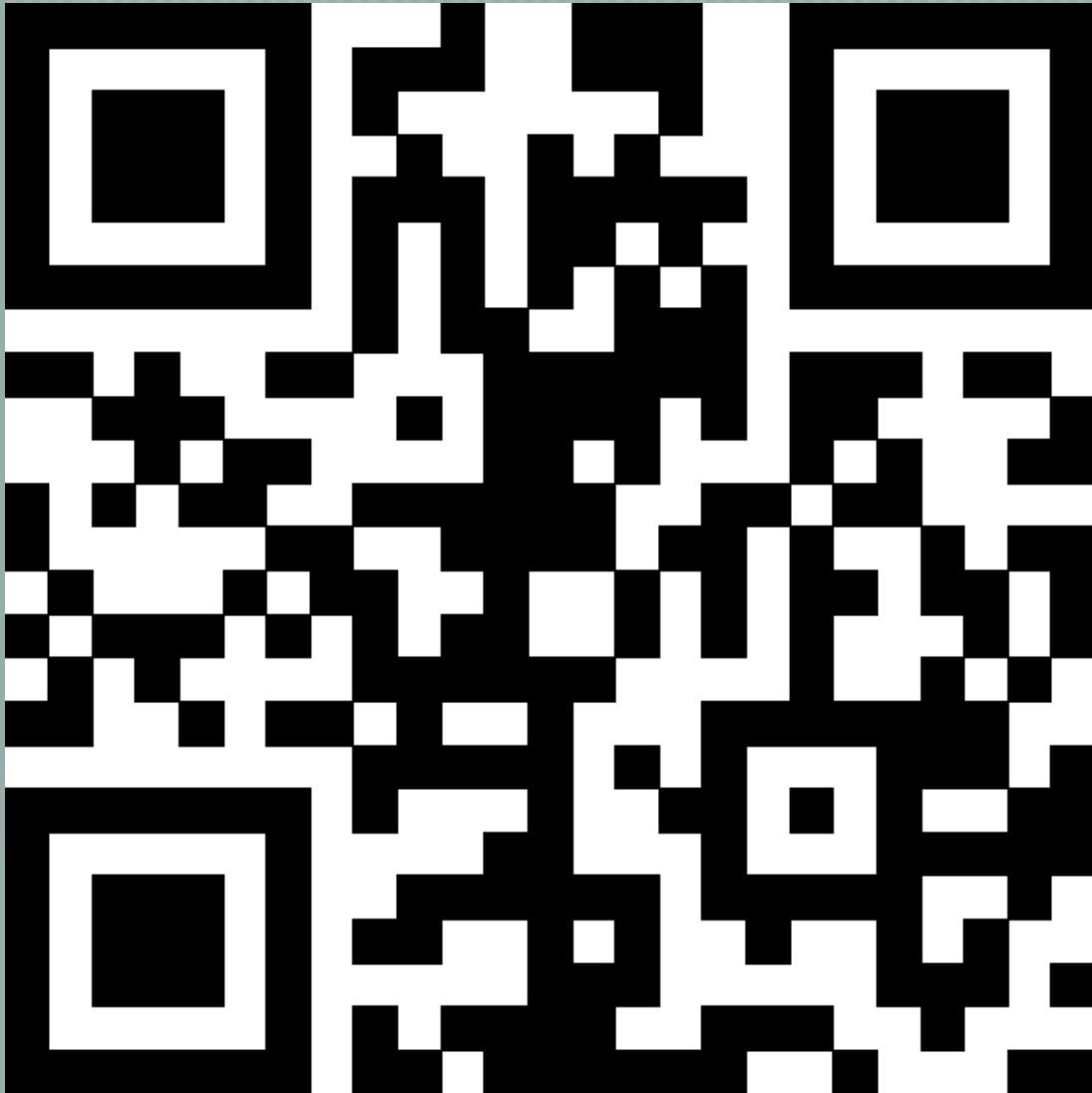
— [ Regular-Expressions.info — [www.regular-expressions.info](http://www.regular-expressions.info)

— [ Mastering Regular Expressions — O'Reilly

— [ Sample files — [www.leximation.com/downloads/regex-samples](http://www.leximation.com/downloads/regex-samples)

— [ Scott Prentice <[scott AT leximation.com](mailto:scott@leximation.com)> — [www.leximation.com](http://www.leximation.com)

# Feedback



**Your opinion is important!**

Please tell us what you thought of the lecture. We look forward to your feedback via smartphone or tablet.

**Scan the QR code  
or visit the URL:**

<http://ta17.honestly.de>

The feedback tool will be available even after the conference!