Regex

My Regex Background

- 5 Years ago (replace human text coding)
  - I mispronounce regex (!rejex)
  - Text analysis (extract patterns from transcripts)
- 2 R regex based packages

What you will learn/get today?

1. Basic regex syntax
2. Resources for learning regex
3. Examples and growth exercises

Agenda (~40 minutes)

1. Intro Regex (5 minutes)
2. Action Tools: DS Prog. Functions (5 mins)
3. Excercise #1: Action Tools (8 mins)
4. Regex Basics (10 mins)
5. Exercise #2: Basic Regex (remainder time)

Regex Intro

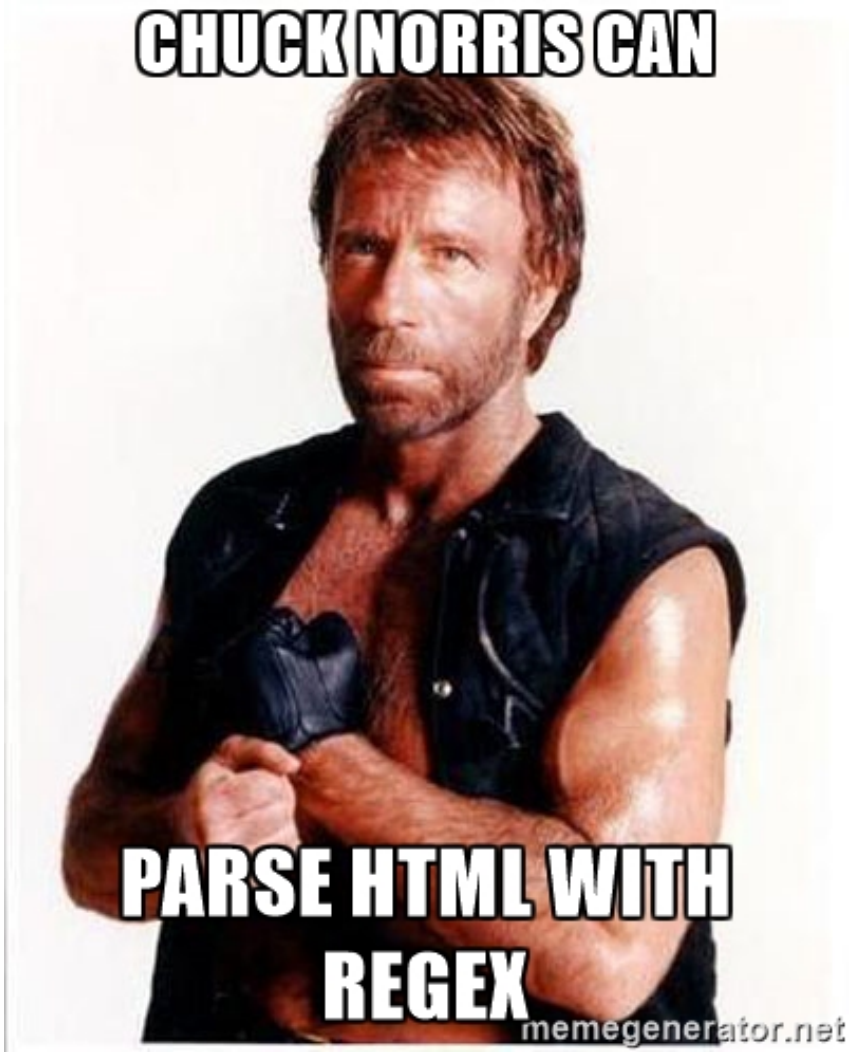
5 minutes

Additional Resources

- `http://www.regular-expressions.info/tutorial.html`
- `http://www.rexegg.com`
- `https://www.debuggex.com`

What's it good for?

- Munging
- Quantifying text data
- Categorizing
- Validating



What is regex?

patterns of characters that match, or fail to match, sequences of characters in text. (Watt, 2005, p. 2)

What is it for?

4 Actions

1. Matching (logical/counting)
2. Subbing
3. Splitting
4. Extracting

Action Tools

Data Science Programing Language Functions

R & Python 5 Minutes

Important Distinction

Slashes for special characters:

R - \\

Python - \

Language Functions for 4 Actions

Action	R: stringi	Python: pandas
Matching	`stri_detect_regex`	`.str.contains`
Counting	`stri_count_regex`	`.str.count`
Subbing	`stri_replace_all_regex`	`.str.replace`
Splitting	`stri_split_regex`	`.str.split`
Extracting	`stri_extract_all_regex`	`.str.extractall`

***Note**: There is a known bug in Pandas `.str.extractall` version 18.1*

Regular Characters

Most characters, including all letters and digits, are regular expressions that match themselves

Exercise #1: Action Tools

8 minutes

Regex Basics

10 minutes

Meta-Characters

Metacharacter	Meaning
^	Beginning of string
$	End of string
.	Any character
*	Match 0 or more times
+	Match 1 or more times
?	Match 0 or 1 times
|	Or
( )	Group
[ ]	Set of characters
{ }	Repetition modifier
	Escape quote or metacharacter

Escape 'em

Escape these (usually with a \)

Mental Excercise

Write a regular expression to match $1.00

Mental Excercise

Write a regular expression to match $1.00

\$1\.00

Character Classes

- []
- special
  - 0-9
  - A-Z
  - a-z
- meta-characters (you have no power here; only], \, ^, -)
- no ordering; e.g., [xy] == [yx]

```
[A-Za-z' ]
```

Character Class: Negation

`[^expressions_here]` - Everything except...

`[^0-9.-]`

Short Hand Character Class

Regex	Name	Action
\d	Digit	Match digits
\D	Non-Digit	Match non-digits
\w	Word	Match words
\W	Non-Word	Match non-words
\s	Whitespace	Match whitespace
\S	Non-Whitespace	Match non-whitespace

**Most Useful:* `\d` & `\s`

\w = [A-Za-z0-9_]

Quantifiers

Regex	Name	What it Does
`x?`	0-1 (Greedy)	Match 0-1 times greedy
`x??`	0-1 (Lazy)	Match 0-1 times lazy
`x*`	>= 0 (Greedy)	Match 0 or more times greedy
`x*?`	>= 0 (Lazy)	Match 0 or more times lazy
`x+`	>= 1 (Greedy)	Match 1 or more times greedy
`x+?`	>= 1 (Lazy)	Match 1 or more times lazy
`x{4}`	Exactly N	Match N times
`x{4,8}`	Min-Max	Match min-max times
`x{9,}`	> N	Match N or more times

Use with single chars or [] or ()

Quantifiers: Greedy vs. Lazy

```
gsub('\\(.*\\)', "<<OUT>>", "Look at (A) and then (B).")
```

```
## [1] "Look at <<OUT>>."
```

```
gsub('\\(.*?\\)', "<<OUT>>", "Look at (A) and then (B).")
```

```
## [1] "Look at <<OUT>> and then <<OUT>>."
```

Quantifiers: Greedy vs. Lazy

```
gsub('\\([^)]*\\)', "<<OUT>>", "Look at (A) and then (B).")
```

```
## [1] "Look at <<OUT>> and then <<OUT>>."
```

Boolean Or

| - pipe

a|e

Grouping

`()` - Group

- Order matters (unlike character classes `[]`)
- Useful with quantifiers

```
gr(a|e)y
(cat|dog|fish)
(?i)(hey jude ){2,}
```

Anchors & Boundaries

- ^ - begining
- $ - end
- \b - word boundary

*raw string (r'\b') for Python

```
^\w+
\w+[.?!]+$
\bread
\bread\b
```