Introduction to Regular Expressions

Just Enough to Make You Dangerous

Or: *Just Enough To Google It Later*

Slides online at nathanic.org/regex

CITx 2017, Nathan P. Stien

What are Regular Expressions?

They are a language for querying bodies of text.

Well, more like a *family* of languages.

The Plan

- Teach you some basics
- Show you some improvized demos
- Give you some applications
- Answer you some questions at any time

Language Basics

Literals

Most characters are literals that just match themselves.

Easier to list the NON-literals: `$^*+|()[].\/`

The expression `CITx` matches the string `"CITx"` and nothing else.

Escaping with Backslash

Special chars like `|` become literals `\|`

Some non-special chars become special:

- `\n` Newline
- `\t` Tab
- `\s` Any Whitespace
- `\d` Any Digit
- `\b` Word Break

(There are more but those are the best ones)

Alternatives

The pipe character `|` is the "or" operator.

`this|that` will match both `"this"` and `"that"`

Can be chained indefinitely:
`apples|oranges|bananas|kumquats|...`

Demo

Grouping with Parentheses

`th(is|at)` will match both `"this"` and `"that"`

Parens also create *capture groups* you can refer to in substitutions

Character Classes

`gr[ae]y` will match both `"gray"` and `"grey"`

`[aeiou]` will match any vowel, though it will never match `y`

(Not even sometimes.)

`[^aeiou]` will match *any* non-vowel, including

whitespace and Emoji ☕

Character Ranges

You can express a range of possible characters:

- `[a-z]` any lowercase
- `[a-zA-Z]` any alpha
- `[^0-9]` anything NOT a digit

Shorthand Character Ranges

- `\d [0-9]` (digits)
- `\w [a-zA-Z0-9_]` (word characters)
- `\s [ \t\r\n\f]` (space/separator characters)

There are more, but those are the main ones I use.

Anchors

Anchors allow you to reference certain parts of the text

- `^` is the beginning of the line
- `$` is the end of the line
- `\b` is a word boundary

Demo

Dots are Wild

`.` will match *any* character

Even  !

Demo

Quantifiers

Any subexpression can be repeated some number of times:

- `?` occurs 0 or 1 times
- `*` occurs 0 through ∞ times
- `+` occurs 1 through ∞ times
- `{x}` occurs exactly `x` times
- `{x,}` occurs `x` or more times
- `{x,y}` occurs `x` through `y` times

Demo

Dot Star: anything any number of times

`.*` will match ANY text of ANY length

The lazy man's subexpression


WITH GREAT POWER COMES GREAT...
WEEEEEEEEEEE!!!!!!!!!

Substitution

Reference *capture groups* with `\1`,`\2`, etc.

Replace `^The (.*)$` with `\1, The`

Demo

What did I not talk about?

- Other Predefined Character Classes
- Unicode Property Queries
- Negative and Positive Lookahead
- Lazy, Possessive, and Greedy Quantifiers
- Subquery Recursion

I almost never need that stuff.

Where Can I Use This Stuff?

Editors with RegEx Search/Replace

- Microsoft Word & co
- NotePad++
- Any programmer's editor or IDE
- Vim, Emacs, Sed, Grep, Ack, Ag, Awk, and pretty much any UNIX tool
- Bulk file rename tools like `rename` and `vidir`

(Most) Form Tools

Define validations for form fields in terms of regex

```
\(\d{3}\) \d{3}-\d{4}
```

But not in FormStack AFAICT :-(

SQL

```sql
UPDATE flexadmin.web_log
SET message = REGEXP_REPLACE(
    message,
    'pmt_method_exp_date: \d{4}',
    'pmt_method_exp_date: 9999'
) WHERE some_stuff = 'some other stuff'
```

Every Programming Language Ever

Even PeopleCode!

They mostly even use the same few libs like PCRE or java.util.RegEx

Search, replace, split, parse

On the Integrations Team, we use java.util.RegEx all the time

Where *Shouldn't* You Use Regex?

XML parsing, because that way lies madness

Any sufficiently nasty job where you can't read your own regexes after you're done

Reach for a specialized parsing lib for things like JSON, CSV, XML, etc.!

NOW YOU'RE
PLAYING
WITH POWER.

Bonus Slide: IRC Bot `@roll` InstaParse Grammar

```
<command>  := throw (<ws? '+' ws?> throw)* <ws*> comment?
<throw>    := die | const
die        := #'[0-9]*' <'d'> #'[0-9%]+'
const      := #'-?[0-9]+'
comment    := <#';\\s*'> #'.*$'
<ws>       := #'\\s+'
```