

`(?i)reg(ular expressions?|ex(p|es)?)`

WHENEVER I LEARN A  
NEW SKILL I CONCOCT  
ELABORATE FANTASY  
SCENARIOS WHERE IT  
LETS ME SAVE THE DAY.

OH NO! THE KILLER  
MUST HAVE FOLLOWED  
HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH  
THROUGH 200 MB OF EMAILS LOOKING FOR  
SOMETHING FORMATTED LIKE AN ADDRESS!



IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR  
EXPRESSIONS.



TAP  
TAP



# Regex can be a pain

- “Some people, when confronted with a problem, think ‘I know, I'll use regular expressions.’ Now they have two problems.”  
- Jamie Zawinski

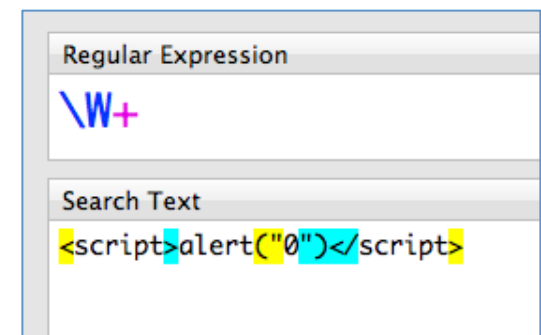
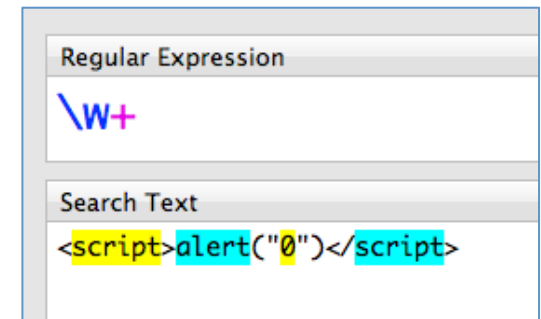
But...

# Regex is amazing

- How do you validate an email?

`\b[A-Z0-9._%+-]+\@[A-Z0-9.-]+\.[A-Z]{2,4}\b`

- How can you sanitize user input?
  - Only allow what matches: `\w+`
  - Strip what matches: `\W+`



# Regex (Regular Expressions)

- A sequence of characters that form a search pattern, mainly used for pattern matching strings
- Every character in a regular expression is either a meta-character with a special meaning or a regular character with a literal meaning
- Concept arose in the 1950s when American mathematician Stephen Kleene formalized the description of a “regular language”

# Basic concepts

- A regular expression (pattern) is an expression used to specify a set of strings
- Common operations used in construction:
  - Boolean “or”
    - A | to separate alternatives. `grey|gray` matches both `grey` and `gray`
  - Grouping
    - A `()` can be used to limit the scope of an operator.
    - `gr(a|e)y` matches `gray` and `grey` because `()` limits scope of `|`
  - Quantification
    - Specifies how often the preceding element is allowed to occur.
    - Common quantifiers are: `?` `(0|1)` `*` `(0|+)` `+` `(1+)`
      - `colou?r` matches `color` and `colour`
      - `ab*c` matches `ac`, `abc`, `abbc`, `abbbc`, etc.
      - `ab+c` matches `abc`, `abbc`, `abbbc`, etc. but not `ac`

# Example

- Match:
  - informatics
  - Information
- Match:
  - Handel
  - Haendel
  - Händel

# Answer

- `informati(cs|on)`
- What about:
  - `inFoRmation`
  - `(?i)informati(cs|on)`
- `H(ä|ae?)ndel`



# More syntax

- `.` Matches any single character
- `[ ]` Matches a single character or range of characters within the brackets
- `[^ ]` Matches a single character that is not contained within the brackets
- `^` Matches the starting position within the string
- `$` Matches the ending position of the string
- `( )` Defines a marked subexpression.  
The string matched can be recalled later with `\n` ( $n = 0+$ )
- `\n` Matches the  $n$ th marked subexpression matched where  $n$  is a digit from 1-9
- `{m,n}` Matches the preceding element at least  $m$  and not more than  $n$  times

# More examples

- `.at` Matches any three-character string ending with “at” - “hat”, “cat”, and “bat”
- `[hc]at` Matches “hat” and “cat”
- `[^b]at` Matches all strings matched by `.at` except “bat”
- `[^hc]at` Matches all strings matched by `.at` other than “hat” and “cat”
- `^[hc]at` Matches “hat” and “cat”, but only at the beginning of the string or line
- `[hc]at$` Matches “hat” and “cat”, but only at the end of the string or line
- `\[.\]` Matches any single character surrounded by “[” and “]”  
Such as “[a]” and “[b]”
- `s.*` Matches any number of characters preceded by `s`  
Such as “saw” and “seed”

# A shallow dip into how it works

When applying `cat` to `He captured a catfish for his cat.`, the engine tries to match the first token in the regex `c` to the first character in the match `H`. This fails. There are no other possible permutations of this regex, because it merely consists of a sequence of literal characters. So the regex engine tries to match the `c` with the `e`. This fails too, as does matching the `c` with the space. Arriving at the 4th character in the match, `c` matches `c`. The engine then tries to match the second token `a` to the 5th character, `a`. This succeeds too. But then, `t` fails to match `p`. At that point, the engine knows the regex cannot be matched starting at the 4th character in the match. So it continues with the 5th: `a`. Again, `c` fails to match here and the engine carries on. At the 15th character in the match, `c` again matches `c`. The engine then proceeds to attempt to match the remainder of the regex at character 15 and finds that `a` matches `a` and `t` matches `t`.

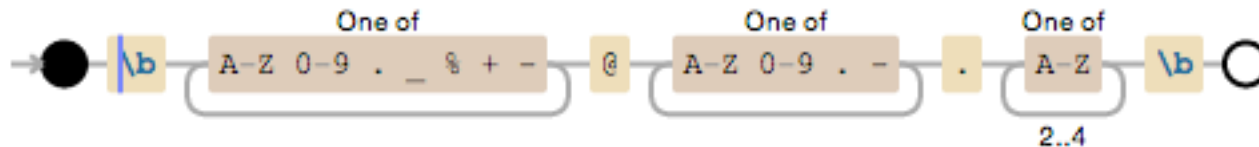
# Practice

	UB   IE   AW	[TUBE]*	[BORF].
[NOTAD]*			
WEL   BAL   EAR			

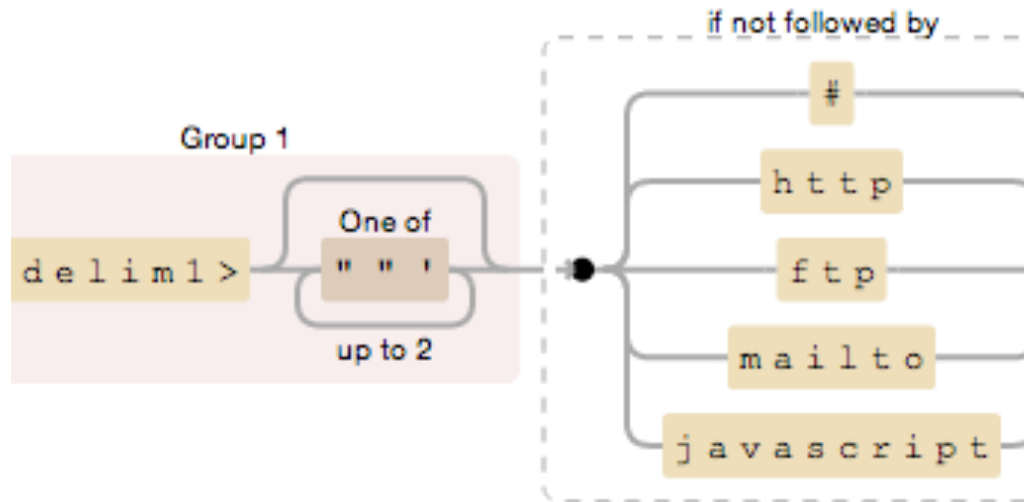
# Tools

- <https://www.debuggex.com/>
- Validating email addresses:

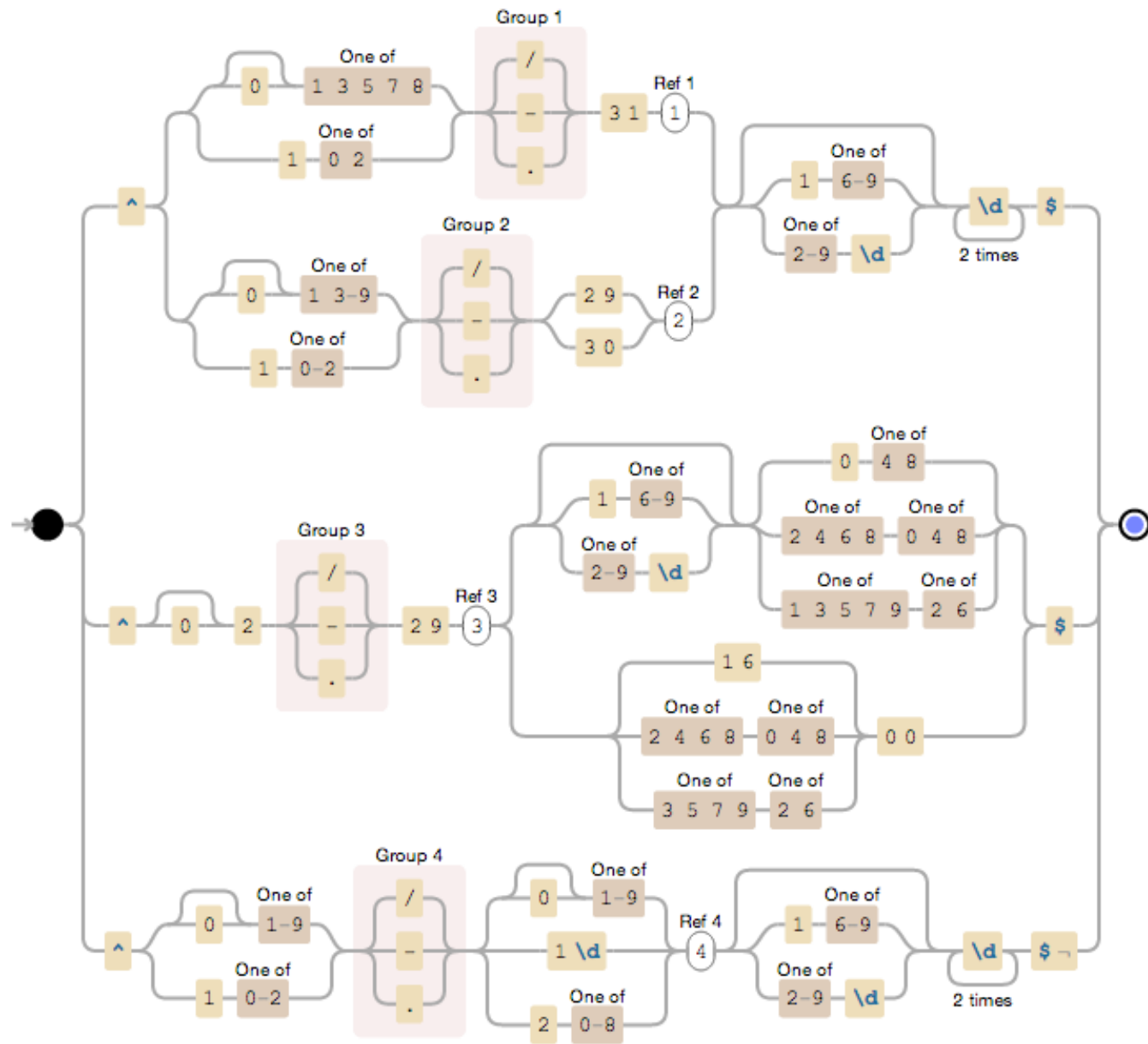
`\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b`



`(<delim1>["'"']{0,2})(?!#|http|ftp|mailto|javascript)`



^(?:((?:0?[13578]|1[02])(\\|\_|\\. )31)\\1|(?:0?[13-9]|1[0-2])(\\|\_|\\. )(?:29|30)\\2))(?:1[6-9]|[2-9]d)?\\d{2})\$|^((?:0?2(\\|\_|\\. )29\\3(?:1[6-9]|[2-9]d)?(?:0[48]|[2468][048]|[13579][26])|(?:16|[2468][048]|[3579][26])00))\$|^((?:0?[1-9])|(?:1[0-2])(\\|\_|\\. )(?:0?[1-9]|1d|2[0-8]))\\4(?:1[6-9]|[2-9]d)?\\d{2})\$



Ref 1

Ref 2

Ref 3

Ref 4



[illegible]

# Regex crossword

- Give it a shot: <http://regexcrossword.com/>

# Learn more

- <http://regex.learncodethehardway.org/book/>
- <http://nbviewer.ipython.org/url/norvig.com/ipython/xkcd1313-part2.ipynb?create=1>

# References

- [https://en.wikipedia.org/wiki/Regular\\_expression](https://en.wikipedia.org/wiki/Regular_expression)
- <https://programmers.stackexchange.com/questions/122440/how-do-regular-expressions-actually-work>
- <https://stackoverflow.com/questions/3622398/how-a-regex-engine-works>
- <http://www.regular-expressions.info/engine.html>
- <http://www.regular-expressions.info/tutorial.html>