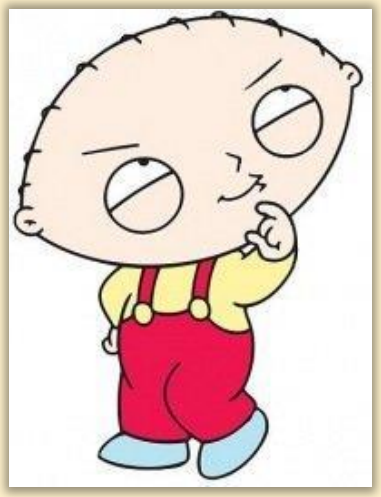
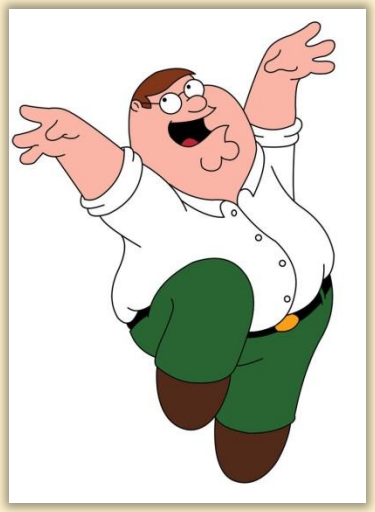
REGULAR EXPRESSIONS (REGEX)

DEFINITION



A regular expression, regex or regexp (sometimes called a rational expression) is, in theoretical computer science and formal language theory, a sequence of characters that define a search pattern. Usually this pattern is then used by string searching algorithms for "find" or "find and replace" operations on strings. (c) wikipedia



Dummy definition: regular expression is a pattern describing a certain amount of text

DEFINITION

REGEX Engine - is a piece of software that can process regular expressions, trying to match the pattern to the given string.

USE CASES

VALIDATION

TEXT STATISTICS

MACHINE LERNING

HISTORY

- 1943, *Warren McCulloch* and *Walter Pitts* “A logical calculus of the ideas immanent in nervous activity”
- 1956, Stephen Kleene “Representation of events in nerve nets and finite automata” – born of regex algebra
- 1968, *Ken Thompson* article “Regular Expression Search Algorithm” With code and prose he described a Regular Expression compiler that creates IBM 7094 object code.
- He also implemented Kleene’s notation in the editor QED. The aim was that the user could do advanced pattern matching in text files. The same feature appeared later on in the editor ed.
- Late 80’s Larry Wall Perl programming language helped regex to become more popular or even mainstream.

To search for a Regular Expression in ed you wrote g/<regular expression>/p The letter *g* meant global search and *p* meant print the result. The command — g/re/p — resulted in the standalone program grep, released in the fourth edition of Unix 1973.

HISTORY

A *Kleene algebra* is a structure $\mathcal{K} = (K, +, \cdot, ^*, 0, 1)$ where

- $(K, +, \cdot, 0, 1)$ is an idempotent semiring
- The * operation satisfies

$$1 + xx^* \leq x^*$$

$$1 + x^*x \leq x^*$$

$$b + ax \leq x \longrightarrow a^*b \leq x$$

$$b + xa \leq x \longrightarrow ba^* \leq x$$

The axioms say that a^*b is the unique least solution to $b + ax \leq x$

PCRE – QUICK REFERENCE - CHARACTERS

Charakter	Legend	Example	Sample Match
\d	Most engines: one digit from 0 to 9	**file_\d\d**	file_25
\w	Most engines: "word character": ASCII letter, digit or underscore	**\w-\w\w\w**	A-b_1
\s	Most engines: "whitespace character": space, tab, newline, carriage return, vertical tab	**a\sb\sc**	a b c
\D	One character that is not a digit as defined by your engine's \d	**\D\D\D**	ABC
\W	One character that is not a word character as defined by your engine's \w	**\W\W\W\W\W**	*-+=)
\S	One character that is not a whitespace character as defined by your engine's \s	**\S\S\S\S**	Yoyo
.	Any character except line break	**a.c**	abc
****	Escapes a special character	**\.*\+\? \$\^\/**	.*+? $^/\

PCRE – QUICK REFERENCE - GROUPS & SETS

Logic	Legend	Example	Sample Match
[…]	One of the characters in the brackets	**[AEIOU]**	One uppercase vowel
[x-y]	One of the characters in the range from x to y	**[A-Z]+**	GREAT
[…]	One of the characters in the brackets	**[AB1-5w-z]**	One of either: A,B,1,2,3,4,5,w,x,y,z
[x-y]	One of the characters in the range from x to y	**[-~]+**	Characters in the printable section of the ASCII table.
[^x]	One character that is not x	**[^a-z]{3}**	A1!
[^x-y]	One of the characters not in the range from x to y	**[^ -~]+**	Characters that are not in the printable section of the ASCII table.
(…)	Capturing group	**A(nt|pple)**	Apple (captures "pple")

PCRE – QUICK REFERENCE - QUANTIFIERS

Quantifier	Legend	Example	Sample Match
|	Alternation / OR operand	**22|33**	33
?	Once or none	**plurals?**	plural
{3}	Exactly three times	**\D{3}**	ABC
{2,4}	Two to four times	**\d{2,4}**	156
{2,4}?	Makes quantifiers "lazy"	**\w{2,4}?**	ab in abcd
{3,}	Three or more times	**\w{3,}**	regex_tutorial
+	The + (one or more) is "greedy"	**\d+**	12345
+?	Makes quantifiers "lazy"	**\d+?**	1 in 12345
*	The * (zero or more) is "greedy"	**A***	AAA
***?**	Makes quantifiers "lazy"	**A*?**	empty in AAA

PCRE – QUICK REFERENCE - ANCHORS AND BOUNDARIES

Anchor	Legend	Example	Sample Match
^	Start of string or start of linedepending on multiline mode. (But when [^inside brackets], it means "not")	**^abc .***	abc (line start)
$	End of string or end of linedepending on multiline mode. Many engine-dependent subtleties.	**.*? the end$**	this is the end
\A	Beginning of string (all major engines except JS)	**\Aabc[\d\D]***	abc (string... ...start)
\z	Very end of the string Not available in Python and JS	**the end\z**	this is...\n...the end
\Z	End of string or (except Python) before final line break Not available in JS	**the end\Z**	this is...\n...the end\n
\b	Word boundary Most engines: position where one side only is an ASCII letter, digit or underscore	**Bob.*\bcat\b**	Bob ate the cat
\B	Not a word boundary	**c.*\Bcat\B.***	copycats

PCRE – QUICK REFERENCE - LOOKAROUNDS

Lookaround	Legend	Example	Sample Match
(?=…)	Positive lookahead	**(?=\d{10})\d{5}**	01234 in 0123456789
(?<=…)	Positive lookbehind	**(?<=\d)cat**	cat in 1cat
(?!…)	Negative lookahead	**(?!theatre)the\w+**	theme
(?<!…)	Negative lookbehind	**\w{3}(?<!mon)ster**	Munster

VALIDATION

1. The password must have between six and ten word characters \w
2. It must include at least one lowercase character [a-z]
3. It must include at least **three** uppercase characters [A-Z]
4. It must include at least one digit \d

\A(?=\w{6,10}\z)(?=.*[a-z])(?=(?:.*[A-Z]){3})(?=.*\d)

\A(?=\w{6,10}\z)(?=.?*[a-z])(?=(?:.*[A-Z]){3})(?=.*\d)

\A(?=\w{6,10}\z)(?=[^a-z]*[a-z])(?=(?:[^A-Z]*[A-Z]){3})(?=\D*\d)

\A(?=[^a-z]*[a-z])(?=(?:[^A-Z]*[A-Z]){3})(?=\D*\d)\w{6,10}

INTERESTING

Lookaround helps in real life!

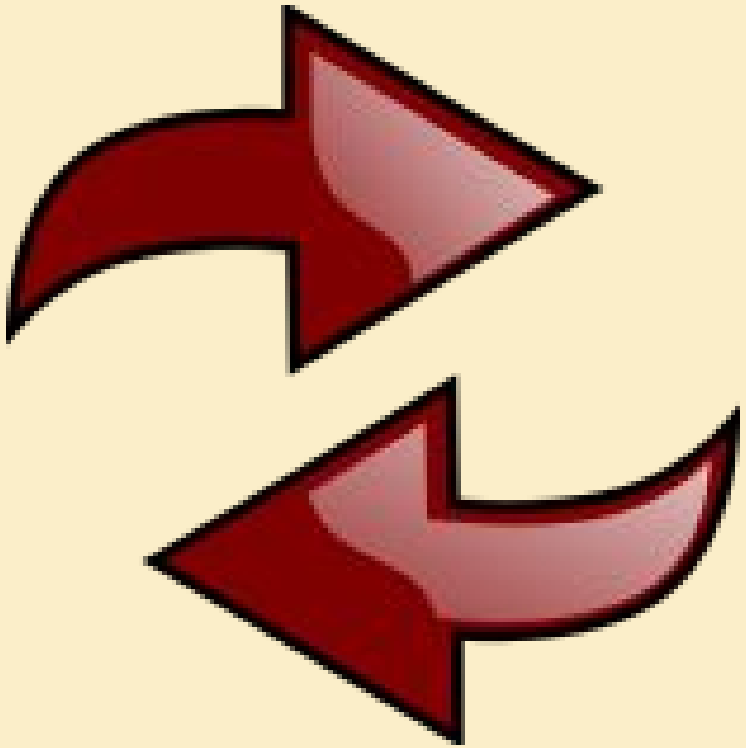
java(?!script)
javascript (?!is the best language)
javascript ?(?!is the best language)
javascript(?! ?is the best language)
javascript(?!.+is the best.+)

INTERESTING

It can be swapped

(?<!.)text = ^text
text(?!.) = text$

(?<![\s\S])text = \Atext
text(?![\s\S]) = \ztext



INTERESTING

How to deal with this : _12_

_(\d{2})_

(?<=_)\d{2}(?=_)

(?<=_(?=\d{2}_))\d{2} **- back to the future**

_\K\d{2}(?=_) **- \K means delete all from matched buffer**

INTRESTING

You need more gold!

Super greedy quantifiers:

++

***+**

5++5

CATASTROPHIC BACKTRACKING

History about REGEX backtracking

(...(...(...)))

CATASTROPHIC BACKTRACKING

(?R) - recursion to rule them all?

CATASTROPHIC BACKTRACKING

```
/\(([^)(]+|(?R))+\)/
```

(You shall not pass (not even try)

WORK MECHANISM

Finite State Automaton - POSIX

Backtracking Engine - Perl, PCRE

CATASTROPHIC BACKTRACKING

How to proceed with this problem

Atomic groups (?> .....) - no backtracking no problems

/\((?>[^)(]+|(?R))+\)/

and of course: GREED IS GOOD

/\(([^)(]++|(?R))++\)/

USEFULL LINKS:

- http://www.rexegg.com/regex-recursion.html
- http://www.phpliveregex.com/
- https://regex101.com/

THANKS FOR WATCHING