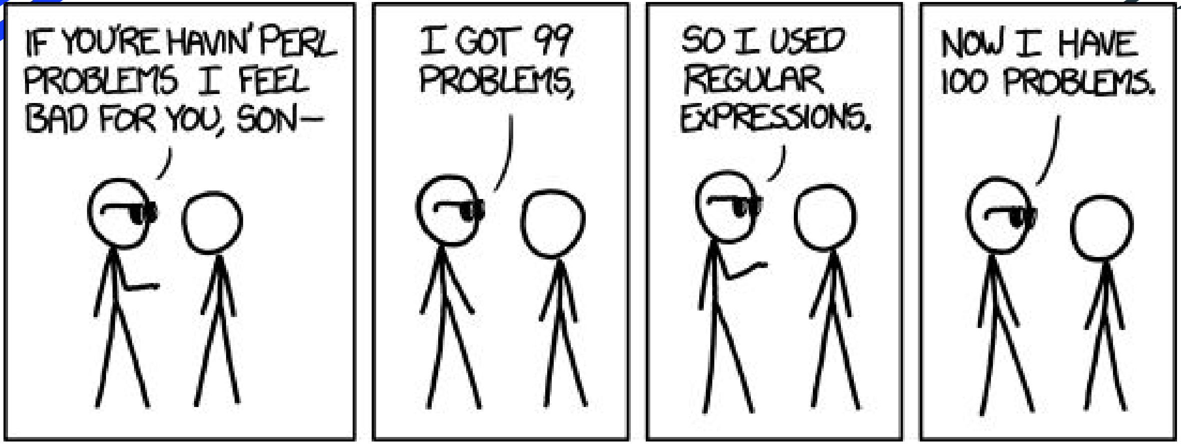
IF YOU'RE HAVIN' PERL PROBLEMS I FEEL BAD FOR YOU, SON—
I GOT 99 PROBLEMS,
SO I USED REGULAR EXPRESSIONS.
NOW I HAVE 100 PROBLEMS.

By: Brian Hodges

What are Regular Expressions

- Pattern matching sub-language
- Often called Regex or Regexp
  - I may call it RE on occasion after the Python module
- Available in many popular programming languages (not a comprehensive list)
  - JavaScript, Perl, Python, (Any scripting language really)
  - Some SQL flavors
  - C, Java (have some problems with raw strings)
- Use raw strings because of character escapes
  - r”\\” literal re backslash using raw string
  - “\\\\” literal re backslash using normal string (escapes the slashes in the string then the regex)
- Primarily use a flavor called Extended Regular Expressions

Matching

- Most characters match themselves
  - r“Hello” only matches “Hello”
- Special sets allow matching of multiple characters
- “.” Matches (almost) anything
  - Will not match a newline unless a flag is set
- “^” matches the beginning of a string and “$” matches the end
- “[...]” matches the set of characters inside
  - “[^...]” matches anything <u>but</u> the characters inside
  - “[A-Z]” matches the characters in a range (any capitalized letter in this case)
- “\d” matches a digit, “\w” matches a word character, “\s” matches a whitespace character
  - Capitalized versions invert the match, so “\S” matches non-whitespace characters
  - Word characters are generally “[A-Za-z0-9]”

Repetition

- Allow matching of several patterns in a row
- “.{N}” matches N characters
- “.{M,N}” matches a variable number of repetitions from M to N inclusive
  - You do not need to include both numbers, i.e. “.{3,}” matches at least 3 characters
- “.*” matches at least 0 characters, equivalent to “.{0,}”
- “.+” matches at least 1 characters, equivalent to “.{1,}”
- “.?” matches 0 or 1 characters, equivalent to “.{0,1}” (usually called maybe)
- “?” makes any repetition non-greedy, will match as few characters as possible
  - Can be confusing with things like: “\???” maybe matching a question mark, but preferably not

Grouping

- Allows repetition of patterns or selection of multiple subpatterns
- “(...)” basic capturing group
  - Each group can be retrieved from the match specified by number, 0 is always the whole match
- “(?:...)” non-capturing group
  - Cannot be retrieved, should be used when a group doesn’t need to be retrieved
- “(?=...)” lookahead
  - When placed at the end of a regex will only match things before the lookahead
  - Often need to be constant length (no variable repetition)
- “(?<=...)” lookbehind
  - Same as lookahead but at the beginning of the string
  - Both can be inverted: “(?!...)” for lookahead, “(?<!...)” for lookbehind
- “(?P<name>...)” named group
  - A capturing group that can be referred to by name, can make regexs more readable

Other Regex

- “a|b” matches a literal “a” <u>or</u> a literal “b”
- “\N” backreference to group N
  - Matches a previously matched group, must be a capturing group
- “(?P=name)” named backreference
  - Matches a named group
  - Syntax varies by language

Things you can do with Regular Expressions

- Match
  - Matches a string only if it begins with the regex
- Search
  - Finds the the first occurrence of the regex anywhere within the string
- Findall
  - Finds all non-overlapping occurrences of the regex in the string
- Split
  - Split the string by the regex
- Sub
  - Replace the regex within the string
  - (supports back references to allow reinserting part of the match)
- Compile
  - Makes an RE object that can be reused (more efficient)

Examples of Regular Expression uses (In Python of course)

- Find all phone numbers in a string
  - `re.findall(r'\b(?:\(\d{3}\) |\d{3}-)\d{3}-\d{4}\b', str)`
- Find all emails in a string
  - `re.findall(r'\b[\w\._%+-]+@[\w\.-]+\.[A-Za-z]{2,}\b', str)`
- Split by whitespace
  - `re.split(r'\s+', str)`
- Replace all whitespace with spaces
  - `re.sub(r'\s+', ' ', str)`
- Replace \r\n with \n
  - `re.sub(r'\r?\n', '\n', str)`
- Test for non-prime numbers (don’t do this)
  - `re.match(r'^.?$|^(..+?)\1+$', '1'*n)`

Resources

- Python’s re module documentation
    - This is what I used to write these slides
    - https://docs.python.org/3/library/re.html
- Regex Crosswords
    - Awesome site to learn how to read regular expressions
    - https://regexcrossword.com
- Regex 101
    - Regex syntax highlighting and checking
    - https://regex101.com