Dr. Sarah Abraham
University of Texas at Austin
Computer Science Department

Regular Expressions

Elements of Graphics
CS324e
What are Regular Expressions?

- Describe a set of strings based on common characteristics each string shares
- Used for searching, editing, and manipulating text and data
- Very flexible system for pattern matching
- Supported by many languages including Python and Java
- Note that the regex patterns will be the same but setup is language dependent
String Literals

❖ Most basic regular expression
❖ Regex engine matches literal character or string to the first occurrence of the character or string in input
❖ If the literal is He and the input string is “Hello, He-Man”, what was the returned match?
❖ Hello, He-Man
❖ Literals are case-sensitive by default (He != he)
Metacharacters allow for more flexible searches

Reserved for specific uses in a regex engine

The metacharacters are:

- \, ^, $, ., |, ?, *, +, -, (, ), [, ], {, }

Must use backslash before any of these characters to match them as literals

- e.g. 1+1 is not the same as 1\+1

Note that different languages handle backslashes differently, so always check specification if something is not working as expected
Parentheses

- Used for grouping characters or regular expressions
- Can be nested within other regular expressions
- Used for both searches and substitutions

(cats), (cat)s and (cats) all match “cats”
Backslash

❖ “Escape” character
  ❖ Reverts metacharacters to literals, gives special meaning to literals
❖ Control characters
  ❖ Specify difficult to type characters (\n is newline, \t is tab)
❖ Convenience escape sequences
  ❖ Specify character classes (\d matches digits, \D matches not digits, \s matches whitespace, \w matches word characters, \b matches word boundaries)
❖ Substitution special characters
  ❖ Substitutes subexpression matches based on match position (\1 … \9) or change upper/lower case (\U, \u, \L, \l)
Caret

- Specifies “anchor” at the start of a line
- Anchors denote specific position within search text
- \(^c\) matches to “cat”
- \(^{(\text{at})}\) has no match in “cat”
- Specifies negation of following characters
- \([^{0–9}\)]\) matches any non-digit sequence
Dollar

- Specifies “anchor” at the end of a line
- `(at)$` matches to “cat”
- `(at)$` has no match in “where is the cat?”
**Dot**

- Matches a single character of any kind except line breaks
- What is considered a line break varies across systems (\n is always a new line, \r is sometimes a new line)
- .a. matches “bat” and “lag” and “ a “
- Powerful metacharacter that can lead to unexpected matches if not careful
Vertical Pipe

- Separates series of alternatives
- Chooses between options for matching
- Similar to OR operation in boolean logic
- \((at \mid ba)\) matches "bats and cats"
- Note that the engine returns bats rather than bats or bats because "ba" matches first (engine is "eager" to return first first match)
Question Mark

- Makes proceeding regex token optional
  - Either 0 or 1 of that token is present
- `colour` matches "color" and "colour"
- Jan(uary)? matches "Jan" and "January"
Star and Plus

- Specify how often preceding regular expression should match
  - * matches regex 0 or more times
  - + matches regex 1 or more times
  - Similar to ? which matches regex 0 or 1 times
- \d*\.txt matches “file.txt” and “file01.txt”
- \d+\.txt matches “file01.txt”
- Note that \d?\.txt matches “file.txt” and “file01.txt”
Square Brackets

- Defines a character class that matches a single character
- `gr[ae]y` matches “gray” and “grey”
- `\b(c[aeiou]t)\b` matches “cat” and “cot” and “cut”
- Match is negated when `^` follows `[`
  - `\b(c[^aei]t)\b` matches “cot” and “cut”
  - `\b(c[^aei]t)\b` does not match “cat”
Minus

- Indicates a range in a character class
- \[a–z\] matches any lower case character in the alphabet
- \[0–9\] matches any digit
- \[A–Za–z\] matches any lower or upper case character in the alphabet
- – before a character in brackets indicates a match for “-“
- \[–0–9\] matches to “-“ or any digit
Curly Braces

- Defines a range quantifier for the preceding regular expression
- \((expr)\{m,n\}\) tries to match the expression between \(m\) and \(n\) times
- \((iss)\{1,2\}\) matches “miss” and “Mississipi”
- \((iss)\{2\}\) matches “Mississipi”
- \((iss)\{2,3\}\) matches “Mississipi”
Regexes in Python

- Uses `re` module: `import re`
- `re.search` finds first pattern within a string
- `re.findall` finds all non-overlapping patterns within a string
- Exceptions generated as `re.error` if unable to compile or use a regular expression
Regexes in Java

- Uses `java.util.regex` API
- Three main classes:
  - `Pattern` provides a compiled regular expression
  - `Matcher` performs match operations of `Pattern` object against input string
  - `PatternSyntaxException` reports syntax errors in `Pattern` object
Regexes in Processing

- Based on Java
- Regex matching handled within String class
  - `match(string, regex)` returns matching groups as a String array
  - Groups specified with sets of parentheses
  - `matchAll(string, regex)` returns all matching groups as a 2D String array
Reference

- Online reference and tutorials
  - [http://www.regular-expressions.info](http://www.regular-expressions.info)

- Online reference

- Online tool for building and testing
  - [http://regexr.com/](http://regexr.com/)