Today's Topic

• Homework 5 Review
• Perl regex
  – *probably the most valuable part of Perl...*
• Homework 6: Reading
• Homework 7: Exercises

**Reminder:** no class all next week
# Homework 5 Brief Review

## hw5data1.txt

<table>
<thead>
<tr>
<th>Rank</th>
<th>State</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>California</td>
<td>39,250,017</td>
</tr>
<tr>
<td>2</td>
<td>Texas</td>
<td>27,862,596</td>
</tr>
<tr>
<td>3</td>
<td>Florida</td>
<td>20,612,439</td>
</tr>
</tbody>
</table>

- Discard 1\(^{st}\) 3 lines
- For every 3:
  1. Discard 1\(^{st}\)
  2. Use 2\(^{nd}\) as key
  3. 3\(^{rd}\) is population

## hw5data2.txt

<table>
<thead>
<tr>
<th>State/territory</th>
<th>Rank</th>
<th>sq mi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alaska</td>
<td>1</td>
<td>665,384.04</td>
</tr>
<tr>
<td>Texas</td>
<td>2</td>
<td>268,596.46</td>
</tr>
<tr>
<td>California</td>
<td>3</td>
<td>163,694.74</td>
</tr>
</tbody>
</table>

- Discard 1\(^{st}\) 3 lines
- For every 3:
  1. Use 1\(^{st}\) as key
  2. Discard 2\(^{nd}\)
  3. 3\(^{rd}\) is area
Homework 5 Brief Review

• Use code template:

```perl
open($fh, $ARGV[0]) or die "$ARGV[0] not found!

while ($line = <$fh>) {
  ...
}
```

Useful code snippets:
• chomp $s;
• $s =~ tr//d;
• $s =~ s/^\s+|\s+$//g;

hw5data1.txt
• Discard 1st 3 lines
• For every 3:
  1. Discard 1st
  2. Use 2nd as key
  3. 3rd is population

hw5data2.txt
• Discard 1st 3 lines
• For every 3:
  1. Use 1st as key
  2. Discard 2nd
  3. 3rd is area

From lecture 7, let's also use our ability to make a hash table that contains sub-hash tables.
Homework 5 Brief Review

*Get organized, and start with ...*

```perl
1# hw5data1.txt 1: discard, 2: state, 3: pop
2# hw5data2.txt 1: state, 2: discard, 3: area
3# chomp $s;
4# $s =~ tr/,//d;
5# $s =~ s/\s+\s+/\s+/g;
6open($fh, $ARGV[0]) or die "$ARGV[0] not found!\n
7
8while ($line = <$fh>) {
9
10 }
```
Imagine that you have become a passionate fan of woodchucks and have recently learned that groundhog and woodchuck are different names for the same animal. Since you are writing a term paper on woodchucks, you now need to search through your paper for every occurrence of the term woodchuck and replace woodchucks with woodchucks (groundhogs). But you also need to replace singular woodchuck with woodchuck (groundhog). Instead of having to do this search twice, you would prefer to perform a single command for something like woodchuck with an optional final s. Or perhaps you might want to search for all the prices in some document; you might want to see all strings that look like $199 or $25 or $24.99 in order to automatically extract a table of prices. In this chapter we introduce the regular expression, the standard notation for characterizing text sequences. The regular expression is used for specifying text strings in all sorts of text processing and information extraction applications.
Homework 6

– Read up on the syntax of Perl Regular Expressions

– Online tutorials
  • http://perldoc.perl.org/perlrequick.html
  • http://perldoc.perl.org/perlretut.html
Perl Regular Expressions

• Perl regex matching:
  – $a =~ /foo/  (/.../ contains a regex)
  – can use in a conditional:
    • evaluates to true/false depending on what’s in $a
  – can also use as a statement:
    • variable $& contains the match

• Perl regex match and substitute:
  – $a =~ s/foo/bar/
  – s/...match... /...substitute... / contains two expressions
  – will modify $a by looking for a **single** occurrence of *match* and replacing that with *substitute*
  – s/...match... /...substitute... /g  global substitution
Perl Regular Expressions

• Most useful with the code template for reading in a file line-by-line:

```perl
open($txtfile,$ARGV[0]) or die "$ARGV[0] not found!\n";
while ($line = <$txtfile>) {
    do RE stuff with $line
}
```
Chapter 2: JM

<table>
<thead>
<tr>
<th>RE</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>/woodchucks/</td>
<td>“interesting links to woodchucks and lemurs”</td>
</tr>
<tr>
<td>/a/</td>
<td>“Mary Ann stopped by Mona’s”</td>
</tr>
<tr>
<td>/Claireorangy/</td>
<td>“Dagmar, my gift please,” Claire says,”</td>
</tr>
<tr>
<td>/DOROTHY/</td>
<td>“SURRENDER DOROTHY”</td>
</tr>
<tr>
<td>/!/</td>
<td>“You’ve left the burglar behind again!” said Nori</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>/[wW]oodchuck/</td>
<td>Woodchuck or woodchuck</td>
<td>“Woodchuck”</td>
</tr>
<tr>
<td>/[abc]/</td>
<td>‘a’, ‘b’, or ‘c’</td>
<td>“In uomini, in soldati”</td>
</tr>
<tr>
<td>/[1234567890]/</td>
<td>any digit</td>
<td>“plenty of 7 to 5”</td>
</tr>
</tbody>
</table>

**Figure 2.1** The use of the brackets [ ] to specify a disjunction of characters.

**character class**: Perl lingo
Chapter 2: JM

range: in ASCII table

<table>
<thead>
<tr>
<th>RE</th>
<th>Expansion</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>/[A-Z] /</td>
<td>an upper case letter</td>
<td>“we should call it ‘Drenched Blossoms’”</td>
<td></td>
</tr>
<tr>
<td>/[a-z] /</td>
<td>a lower case letter</td>
<td>“my beans were impatient to be hoed!”</td>
<td></td>
</tr>
<tr>
<td>/[0-9] /</td>
<td>a single digit</td>
<td>“Chapter 1: Down the Rabbit Hole”</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2 The use of the brackets [ ] plus the dash – to specify a range.

backslash lowercase letter for class
Uppercase variant for **all but** class

<table>
<thead>
<tr>
<th>RE</th>
<th>Expansion</th>
<th>Match</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>\d</td>
<td>[0–9]</td>
<td>any digit</td>
<td>Party_of_5</td>
</tr>
<tr>
<td>D</td>
<td>[^0–9]</td>
<td>any non-digit</td>
<td>Blue_moon</td>
</tr>
<tr>
<td>w</td>
<td>[a–zA–Z0–9_]</td>
<td>any alphanumeric/underscore</td>
<td>Daiyu</td>
</tr>
<tr>
<td>W</td>
<td>[^\w]</td>
<td>a non-alphanumeric</td>
<td>!!!!!</td>
</tr>
<tr>
<td>s</td>
<td>[\r\t\n\f]</td>
<td>whitespace (space, tab)</td>
<td>in_Concord</td>
</tr>
<tr>
<td>S</td>
<td>[^\s]</td>
<td>Non-whitespace</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.6 Aliases for common sets of characters.
Chapter 2: JM

<table>
<thead>
<tr>
<th>RE</th>
<th>Match (single characters)</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>^[A-Z]</td>
<td>not an upper case letter</td>
<td>“Oyfn prietchik”</td>
</tr>
<tr>
<td>^Ss</td>
<td>neither ‘S’ nor ‘s’</td>
<td>“I have no exquisite reason for’t”</td>
</tr>
<tr>
<td>^.</td>
<td>not a period</td>
<td>“our resident Djinn”</td>
</tr>
<tr>
<td>^e</td>
<td>either ‘e’ or ‘^’</td>
<td>“look up ^ now”</td>
</tr>
<tr>
<td>a^b</td>
<td>the pattern ‘a^b’</td>
<td>“look up a^b now”</td>
</tr>
</tbody>
</table>

**Figure 2.3** Uses of the caret `^` for negation or just to mean `^`. We discuss below the need to escape the period by a backslash.

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>/beg.n/</td>
<td>any character between <code>beg</code> and <code>n</code></td>
<td><code>begin</code>, <code>beg’n</code>, <code>begun</code></td>
</tr>
</tbody>
</table>

**Figure 2.5** The use of the period `. ` to specify any character.
the language of (certain) sheep, which consists of strings that look like the following:

Sheeptalk

- baa!
- baaa!
- baaaa!
- baaaaa!
- ...
Homework 7

• Do regex exercises 2.1 in JM (pg. 42)
• Ungraded, but please do them... I will review some of them next time ...
Another example

The regular expression engine provided by the PERL programming language is a powerful tool for defining and locating patterns in unstructured text. Unlike index-based approaches, this strategy does not impose a specific tokenization and thereby a predefined view of the basic entities contained in the corpus. As a consequence, it is possible to formulate patterns based on parts of words and patterns containing optional elements. For example, the expression `\$+ing\b` can be used to retrieve all words ending in `-ing`, or the pattern `\bmusick?\b` can be used to retrieve the spelling variants `music` and `musick`. The bridge version is searched character-by-character. In the search patterns, alpha-numeric characters are interpreted literally, except if they are preceded by a backslash character as in `\b`, which stands for a word-boundary, or `\s`, which stands for any character appearing on screen. Non-alphanumeric characters often have a non-literal interpretation, for example `?`, which, in the pattern `\bmusick?\b`, specifies that the character to its left may be present or

`\w` is a character class that matches any single word character (letters, digits, Unicode marks, and connector punctuation (like the underscore)).
Another example

• \b or \b{wb}

This matches a Unicode "Word Boundary", but tailored to Perl expectations. This gives better (though not perfect) results for natural language processing than plain \b (without braces) does. For example, it understands that apostrophes can be in the middle of words and that parentheses aren't (see the examples below). More details are at http://www.unicode.org/reports/tr29/.

• variables

| \` | Everything prior to matched string |
| \& | Entire matched string               |
| \' | Everything after to matched string  |
Another example

Note: global match in while-loop

```
This isn't a U.A.-approved sentence.
while( $s =~ m/b\{wb\}(\w.*\b\{wb\})/g ) {
  print "$&\n";
}
```
Unicode and \w

• \w is \[0–9A–Za–z_\]

Definition is expanded for Unicode:

```perl
use utf8;
use open qw(:std :utf8);

my $str = "school école École šola trường สกุล škole โรงเรียน";
@words = ($str =~ /\w+/g);
foreach $word (@words) { print "$word\n" }
```

 bash-3.2$ perl regex_utf.perl
 school
ecole
École
šola
trường
สกุล
škole
โรงเรียน

list context
Chapter 2: JM

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>an asterisk “*”</td>
<td>“K<em>A</em>P<em>L</em>A*N”</td>
</tr>
<tr>
<td>.</td>
<td>a period “.”</td>
<td>“Dr. Livingston, I presume”</td>
</tr>
<tr>
<td>?</td>
<td>a question mark</td>
<td>“Why don’t they come and lend a hand?”</td>
</tr>
<tr>
<td>\n</td>
<td>a newline</td>
<td></td>
</tr>
<tr>
<td>\t</td>
<td>a tab</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.8** Some characters that need to be backslashed.
Chapter 2: JM

- Precedence of operators
  - Example: Column 1 Column 2 Column 3 ...
  - `/Column [0-9]+ */`
  - `/ (Column [0-9]+ *)*/`
  - `/ house(cat(s|)])/`

- Perl:
  - in a regular expression the pattern matched by within the pair of parentheses is stored in designated variables $1$ (and $2$ and so on)

- Precedence Hierarchy:

```
Parenthesis     ()
Counters       *  +  ?  {}
Sequences and anchors  the `my end$
Disjunction    |
```
Perl Regular Expressions

http://perldoc.perl.org/perlretut.html

A shortcut: list context for matching

```
# extract hours, minutes, seconds
if ($time =~ /(\d\d):(\d\d):(\d\d)/) { # match hh:mm:ss format
    $hours = $1;
    $minutes = $2;
    $seconds = $3;
}
```

returns 1 (true) or "" (empty if false)

```
# extract hours, minutes, seconds
($hours, $minutes, $second) = ($time =~ /(\d\d):(\d\d):(\d\d)/);
```

returns a list