Day 8:
Regular Expressions, Part 2

Suggested Reading:
Learning Perl (4th Ed.),
Chapter 9: Processing Text with RegExps
Reminders

• Turn in homework at **START** of class
• Writing code is **fun**!
  – Write at least a little every day
  – The more you do, the easier it gets
• When in doubt, ask questions!
  – Zathras does not want you being confused
• Zathras warn, but no, no one listen to poor Zathras, no.
Testing Your Regex

• Use 'perl -ne' to try it out:
  – From the shell:
    $ perl -ne 'if (/expr/) { print; }'
  – For example, the "cat" expression:
    $ perl -ne 'if (/cat/) { print "$_"; }'
    abc
catalog
:catalog
cat
:cat
dog
Substituting with s///

- With the substitution operator `s/expr/subst/`, the matched expression can be modified

- Examples:
  - `s/abc/CBA/`
    - "abc" -> "CBA"
    - "123abc" => "123CBA"
    - "abcabc" => "CBAabc"
  - `s/\d+/+/`
    - "abc" -> "abc"
    - "123abc" => "+abc"
    - "234x123" => "+x123"
A better chomp with s///

- Some operating systems use "CR/LF" as line terminators, and these aren't cleaned up by chomp(). Here's a better chomp:

```perl
#!/usr/bin/env perl
use strict;
use warnings;

sub BetterChomp($) {
    my $v = shift;
    $v =~ s/\s+\r\n+$//;
    return $v;
}

my $lno = 0;
while(<>) {
    $lno++;
    $_ = BetterChomp($_);
    print "$lno:$_\n";
}
```
More Substitution

• Only one substitution will take place
• You can loop:

while( s/\d+/+/ ) { }

– "1234" -> "+++++
– "a12b34c" -> "a++b++c"

while( s/\d+/+/ ) { }

– "1234" -> "+
– "a12b34c" -> "a+b+c"
Global Substitution with \s///\g

• Use global modifier "g" to replace all instances:
  – s/\d+/+/g
    • "1234" -> "++++"
    • "a12b34c" => "a++b++c"
  – s/\d+/+/g
    • "1234" -> "+"
    • "a12b34c" => "a+b+c"
Capturing

• Individual items or groups of items matched in the regex can be "captured"
  – Can be used in the script
  – Can be used in the regular expression for further matching
Basic Capture Syntax

- \((\text{items})\)
  - $1, $2, \ldots$ populated with the captured items
  - Not modified if not matched
- `/^(\d+)/`

<table>
<thead>
<tr>
<th>String</th>
<th>Match?</th>
<th>$1</th>
</tr>
</thead>
</table>
| "123"      | Yes    | "123"
| "123x"     | Yes    | "123"
| "a123"     | No     | unchanged |

- `/:(\d+):/`

<table>
<thead>
<tr>
<th>String</th>
<th>Match?</th>
<th>$1</th>
</tr>
</thead>
</table>
| "foo:123:" | Yes    | "123"
| ":123"    | No     | unchanged |
| ":123:"   | Yes    | "123" |
More Capture Syntax

- Can capture multiple items
- `m|\^/([^S+])/([^S+])|`

<table>
<thead>
<tr>
<th>String</th>
<th>Match?</th>
<th>$1</th>
<th>$2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;/home/nick&quot;</td>
<td>Yes</td>
<td>&quot;home&quot;</td>
<td>&quot;nick&quot;</td>
</tr>
<tr>
<td>&quot;home/nick&quot;</td>
<td>No</td>
<td>unchanged</td>
<td>unchanged</td>
</tr>
</tbody>
</table>

- `^([^S+]):[^:]*:([^d+]):/`

<table>
<thead>
<tr>
<th>String</th>
<th>Match?</th>
<th>$1</th>
<th>$2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;nleroy:x:9795:&quot;</td>
<td>Yes</td>
<td>&quot;nleroy&quot;</td>
<td>&quot;9795&quot;</td>
</tr>
<tr>
<td>&quot;adesmet:x:3014:&quot;</td>
<td>Yes</td>
<td>&quot;adesmet&quot;</td>
<td>&quot;3014&quot;</td>
</tr>
<tr>
<td>&quot;:123:&quot;</td>
<td>No</td>
<td>unchanged</td>
<td>unchanged</td>
</tr>
</tbody>
</table>
Optional Captures

- Captures can be optional using "(items)?".
- `/\(No \ )?\(\w+\)/`

<table>
<thead>
<tr>
<th>String</th>
<th>Match?</th>
<th>$1</th>
<th>$2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;No user&quot;</td>
<td>Yes</td>
<td>&quot;No &quot;</td>
<td>&quot;user&quot;</td>
</tr>
<tr>
<td>&quot;nick&quot;</td>
<td>Yes</td>
<td>&quot;&quot;</td>
<td>&quot;nick&quot;</td>
</tr>
<tr>
<td>&quot;nick&quot;</td>
<td>No</td>
<td>Unchanged</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>
Capture Code Example

• Capture the title and name from a salutation

```perl
while( <STDIN> ) {
    if( /Dear ([a-z]+)\. (.*):/i ) {
        print "-> $2 is a $1\n";
    }
}
```

Dear Dr. Joe Smith:
-> Joe Smith is a Dr

Dear Mr. LeRoy:
-> LeRoy is a Mr

Dear Joe Smith:
Dear Dr. Joe Smith
Grouping: ()

- Grouping and capturing use the same syntax: ()
- With grouping, you can match an entire sequence as a single item
- Groups can be nested
Grouping: More Details

• Back to salutation example
  – The salutation starts with a greeting word which must start with an upper case letter:
    • ^([A-Z][S]*)
  – The title must begin with a upper case, and end in a ".", but is optional:
    • (([A-Z][S]*)\.)?
  – The salutation must end with a colon or a comma:
    • [:,]$
Grouping: More Details

• From our last slide, we have:
  – Greeting: ^([A-Z]\S*)
    • We get $1 from here
  – Title: (([A-Z]\S*)\.)?
    • Note: Nested groupings, generates $2 and $3
  – Punctuation: [:,]$ 
  – Name: (.*)

• Put it all together:
  – /^([A-Z]\S*)\s+([A-Z]\S*)\.?\s*(.\*)[,:]$/
Grouping Code Sample

• In perl:

```perl
while(<>){
    chomp;
    if (/^(\[A-Z]\S*)\s+\(([A-Z]\S*)\.)\s*\(\.*\)[,]\$/){
        print "<$1> <$2> <$3> <$4>\n";
    }
}
```

Dear Joe Smith:
<Dear> <> <> <Joe Smith>
Dear Mr. Joe Smith:
<Dear> <Mr.> <Mr> <Joe Smith>
Hello Dr. Joe Smith,
<Hello> <Dr.> <Dr> <Joe Smith>
Greetings Ms. Jane Doe:
<Greetings> <Ms.> <Ms> <Jane Doe>
Alternation

• "|" is used inside a group to specify alternate groups (think "or")
• /(Nick|Tim|Alan)/:
  – Match:
    • "Nick LeRoy" ($1="Nick")
    • "Tim Cartright" ($1="Tim")
    • "Alan De Smet" ($1="Alan")
  – Not: "nleroy", "cat", "adesmet"
• /^((Hello|Dear)\s+)?(\.*)/:
  – Match:
    • "Hello Mr. LeRoy" ($1="Hello")
    • "Mr. LeRoy" ($1="")
    • "Dear Nick" ($1="Dear")
Magic Incantation From Yesterday

• /(-\d{4})?/  
  – Why does this work?

• /\d{5}( -\d{4})?/  
  – "12345-4567"
  – "12345"
  – "12345-"
Matching Group in Regex

• Can be used to match items later in the regex
  – "\1", "\2", ... can be used to match previous captures / groups
  – /((\d)(\w+)(\1)/:
    • Match: "1hello1", "2Hello_2"
    • No: "1hello3"
  – /(([\[\[\[\[/]])(.+)(\1)/:
    • Match "|abc|", "/a|bcd/", "/a/b/c/", "/ab/c/"
    • Not: "|abc/", "abc"
    • Not: "/a|b|c\" because " +" is greedy
split()

• Used to split a string into pieces
• `split( /regex/, $line )`
  - `@list = split( /[:%]/, $line )`
    • "a:b:c" => [ "a", "b", "c" ]
    • "aa%bbb%ccc" => [ "aa", "bbb", "ccc" ]
    • "abc:123%def" => [ "abc", "123", "def" ]
• Use the "re" module

```python
#! /usr/bin/env python
import re
import sys

for line in sys.stdin.readlines():
    line = line.strip()
    print line
    m = re.match( "^([A-Z]\S*)\s+(([A-Z]\S*)\.)?\s*(.*)\[,.]$", line )
    if m is not None:
        print "<%s> <&s> <&s> <&s>" % \
        ( m.group(1), m.group(2), m.group(3), m.group(4) )
```
ruby

• Regex built in

```ruby
#!/usr/bin/env ruby
$stdin.each_line do |line|
  if line =~ /^([A-Z]\S*)\s+((([A-Z]\S*)\.)?\s*\1\2\3\4)$:/
    print "<#{1}> <#{2}> <#{3}> <#{4}>
  end
end
```