HOW - 194
An Introduction to Perl Regular Expressions
Ron Cody

Introduction
- Perl regular expressions allow you to describe a text pattern.
- Regular expressions that are available in SAS®9 use the same syntax as regular expressions in Perl, a scripting language for UNIX systems.
- You can search for that pattern in a SAS character variable.
- You can extract the text matching the pattern.
- You can replace a located pattern with replacement text.
Introduction (cont.)

Regular expressions use text and special characters (known as metacharacters) to describe patterns. For example:

<table>
<thead>
<tr>
<th>Regex</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cat/</td>
<td>text from a catalog</td>
</tr>
<tr>
<td>/\d\d/</td>
<td>numbers 234 to 567</td>
</tr>
<tr>
<td>/\d\d\d-\d\d\d-d{4}/</td>
<td>number 434-56-9878</td>
</tr>
</tbody>
</table>

Some Perl Metacharacters

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\d</td>
<td>matches any digit</td>
</tr>
<tr>
<td>\D</td>
<td>matches a non-digit</td>
</tr>
<tr>
<td>\w</td>
<td>matches any word character (any letter, digit, or underscore)</td>
</tr>
<tr>
<td>\W</td>
<td>matches any non-word character</td>
</tr>
<tr>
<td>\s</td>
<td>matches any white space character</td>
</tr>
<tr>
<td>\b</td>
<td>matches a word boundary</td>
</tr>
<tr>
<td>[xyz]</td>
<td>matches any one of the characters in brackets</td>
</tr>
<tr>
<td>[a-zA-Z]</td>
<td>matches any upper- or lowercase letter</td>
</tr>
<tr>
<td>. (period)</td>
<td>matches any single character</td>
</tr>
</tbody>
</table>
### Some Regex Examples

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>/^[XYZ]\d\d/</td>
<td>number Y89123</td>
</tr>
<tr>
<td>/\w\w\ \d/</td>
<td>word 9 or 10</td>
</tr>
<tr>
<td>/\w\w\ \d/</td>
<td>ab123 (no match)</td>
</tr>
<tr>
<td>/r[aeiou]n/i</td>
<td>Ronald Cody</td>
</tr>
<tr>
<td>/\w\w\w\s\w\w\w/</td>
<td>four letters</td>
</tr>
</tbody>
</table>

### Exercise 1
Some Perl Metacharacters (cont.)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`(</td>
<td>matches left parenthesis</td>
</tr>
<tr>
<td>`.</td>
<td>matches a period</td>
</tr>
<tr>
<td><code>\</code></td>
<td>matches a single backslash</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td>`^</td>
<td>matches the beginning of a line</td>
</tr>
<tr>
<td>`$</td>
<td>matches the end of a line</td>
</tr>
</tbody>
</table>

Examples:

/`Mrs\.|Mr\.`/ matches Mrs. or Mr.
/`\(\d\d\d\)`/ matches an area code
/`^x\d/` matches 'x123' but not 'ax123'

Exercise 2
Repetition Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>matches previous expression zero or more times</td>
</tr>
<tr>
<td>+</td>
<td>matches previous expression one or more times</td>
</tr>
<tr>
<td>?</td>
<td>matches previous expression zero or one times</td>
</tr>
<tr>
<td>{n}</td>
<td>matches previous expression n times</td>
</tr>
<tr>
<td>{n,m}</td>
<td>matches previous expression at least n times and not more than m times</td>
</tr>
</tbody>
</table>

Examples:
- `/cat*/` matches "ca" followed by 0 or more "t's"
- `/cat?/` matches "ca" followed by 0 or 1 "t"
- `/c(at)?/` matches "c" followed by 0 or 1 occurrences of "at"
- `/\d\d+/` matches 2 or more digits

Understanding the Repetition Operators

Regular Expression: `/ab*c/`  
* = Zero or more times

<table>
<thead>
<tr>
<th>String</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td>Yes</td>
</tr>
<tr>
<td>ac</td>
<td>Yes</td>
</tr>
<tr>
<td>abxxc</td>
<td>No</td>
</tr>
<tr>
<td>abbc</td>
<td>Yes</td>
</tr>
<tr>
<td>abbbc</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Understanding the Repetition Operators

Regular Expression: `/ab+c/`

+ = One or more times

<table>
<thead>
<tr>
<th>String</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td>Yes</td>
</tr>
<tr>
<td>ac</td>
<td>No</td>
</tr>
<tr>
<td>abxc</td>
<td>No</td>
</tr>
<tr>
<td>abbc</td>
<td>Yes</td>
</tr>
<tr>
<td>abbbc</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Exercise 3
Character Classes

- `[a-zA-Z]` All upper- and lowercase letters
- `[2-9]` The digits 2-9
- `[A-E0-9]` A to E and digits 0-9
- `[^xyz]` Not x or y or z
- `[^a-e-]` Not a to e or dash

Demonstrating Repetition Operators

```
/x\d{2,4} /
```

X followed by 2 to 4 digits and a blank

<table>
<thead>
<tr>
<th>String</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>X12</td>
<td>Yes</td>
</tr>
<tr>
<td>X123</td>
<td>Yes</td>
</tr>
<tr>
<td>X1234</td>
<td>Yes</td>
</tr>
<tr>
<td>X12345</td>
<td>No</td>
</tr>
<tr>
<td>X123A</td>
<td>No</td>
</tr>
</tbody>
</table>

Length of String is 10
The PRX Functions

The PRXPARSE Function

Function: PRXPARSE
Purpose: Compiles a regular expression
Syntax: PRXPARSE (expression)

expression is a Perl regular expression.

Examples:
Pattern = prxparse("/X\d+/")
(matches an uppercase X followed by one or more digits)
Note: To ignore case use prxparse("/X\d+/i")

RE = prxparse("/\(8\(00|66|77|88\)\)\d\(3\)-\d\(4\)/")
(matches a toll-free telephone number starting with 800, 866, 877, or 888)
The PREMATCH Function

Function: PRXMATCH

Purpose: Returns the position of a regular expression in a string

Syntax: PRXMATCH (return-code or "regex",value)

- return-code is the return code from the PRXPARSE function;
- regex is a Perl regular expression;
- value is a SAS character value.

If no pattern is found, the function returns a 0.

Examples:

Position = PRXMATCH ("//\d\d/","ABCX56");
Position = 4, start of the pattern

Position = PRXMATCH ("//\d\d/","Apple");
Position = 0, no pattern found

PRXMATCH is like a generalized FIND function
Task

You have a raw data file that contains phone numbers and you want to create a new temporary SAS data set that contains all toll free numbers (defined as numbers starting with 800, 866, 877, 888).

***Solution using only PRXMATCH;

data toll_free;
  input String $15.;
  if prxmatch
    ("/\8(00|66|77|88)\d{3}-\d{4}/", String) then output;
datalines;
(800)123-4555
(908)782-6562
(866)777-8888
(808)131-1311
(877)985-4848;

Toll-free Numbers
  String
(800)123-4555
(866)777-8888
(877)985-4848
***Solution using PRXPARSE and PRXMATCH;
data toll_free;
   input String $15.;

   retain re;
   if _n_ = 1 then
      re = prxparse("/^(8(00|66|77|88))\d{3}-\d{4}/");

   if prxmatch(re,String) gt 0 then output;
datalines;
(800) 123-4555
(908) 782-6562
(866) 777-8888
(808) 131-1311
(877) 985-4848 
;

Toll-Free Numbers
   string   re
(800) 123-4555   1
(866) 777-8888   1
(877) 985-4848   1

Task

Check that an ID is in the form of an X, Y or Z, followed by one or more digits.
data codes;
    Perl = "/[XYZ]\d+/";
input Id $; *Length of Id is 8;
    if prxmatch(Perl,Id) ne 0 then
        Match = 'Yes';
    else Match = 'No';
datalines;
X87
87X77
A567
Z88W
X1234567

First Try

<table>
<thead>
<tr>
<th>Perl Regular Expression</th>
<th>Id</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>/[XYZ]\d+/</td>
<td>X87</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>87X77</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>A567</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Z88W</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>X1234567</td>
<td>Yes</td>
</tr>
</tbody>
</table>

***Add a beginning of line anchor;***

data codes;
    Perl = "/^\[XYZ]\d+/";
input Id $;
    if prxmatch(Perl,Id) ne 0 then
        Match = 'Yes';
    else Match = 'No';
datalines;
X87
87X77
A567
Z88W
X1234567

Second Try

<table>
<thead>
<tr>
<th>Perl Regular Expression</th>
<th>Id</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>^[XYZ]\d+/</td>
<td>X87</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>87X77</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>A567</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Z88W</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>X1234567</td>
<td>Yes</td>
</tr>
</tbody>
</table>
***Add a blank at the end;
data codes;
Perl = "/^\[XYZ\]\d+ /";
input Id $;
if prxmatch(Perl,Id) ne 0 then
    Match = 'Yes';
else Match = 'No';
datalines;
X87
87X77
A567
Z88W
X1234567
;

### Third Try

<table>
<thead>
<tr>
<th>Perl Regular Expression</th>
<th>Id</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>/^[XYZ]\d+ /</td>
<td>X87</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>87X77</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>A567</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Z88W</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>X1234567</td>
<td>No</td>
</tr>
</tbody>
</table>

***Add a word boundary at the end;
data codes;
Perl = "/^\[XYZ\]\d+\b/";
input Id $;
if prxmatch(Perl,Id) ne 0 then
    Match = 'Yes';
else Match = 'No';
datalines;
X87
87X77
A567
Z88W
X1234567
;

### Fourth Try

<table>
<thead>
<tr>
<th>Perl Regular Expression</th>
<th>Id</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>/^[XYZ]\d+\b/</td>
<td>X87</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>87X77</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>A567</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Z88W</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>X1234567</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Exercise 4 - 7

Contact Information

Ron Cody
Email: ron.cody@gmail.com