ABSTRACT
Perl regular expressions allow you to locate a specific character or a specific string of characters in a text string. Many of the string processing tasks in SAS® can be done using Perl regular expressions. These tasks can also be performed using traditional string functions in SAS. But, Perl regular expressions can sometimes provide a much more compact solution to a much more complicated string manipulation task. Some examples are provided to illustrate the use of some regular expressions to find and manipulate some patterns in a text string.

INTRODUCTION
Perl regular expressions helps you to search for and extract single or multiple matching patterns from a character string in one step, as well as to make several substitutions in a string in one step.

The following is a basic example.

```
data _null_;  
pos = PRXPATCH('/world/', 'Hello world!'); 
   put pos=;  
   
txt = PRXCHANGE ('s/world/planet/', -1, 'Hello world! World!');  
   put txt=;  
run;  
```

Output:

```
pos = 7; 
txt = Hello planet! planet!;  
```

In this example, `/world/` specifies a Perl regular expression. The function PRXPATCH matches the Perl regular expression in the string ‘Hello world!’ and capture the position of the word ‘world’ in the string. PRXCHANGE substitutes ‘planet’ for the word ‘world’ in the string ‘Hello world! world!’ . The number -1 means it should do the replacements any number of times.

REGULAR EXPRESSIONS
Regular expressions can be written in many ways. Let us start with simple examples, and then go to more complex examples.

Example 1

In the following example, `/cat/` represents a Perl regular expression. PRMATCH function will match the regular expression in a string.
DATA _NULL_;  
INPUT STRING $35.;  
POS = PRXMATCH ('/cat/', STRING);  
IF POS > 0 THEN PUT STRING;  
DATALINES;  
CAT is an animal.  
There is a cat in the house.  
It is beautiful cat  
cat is lovely.  
It is a cat  
This is a cat.  
This is a caterpillar.  
It is mcat.  
;  
Output:  
There is a cat in the house.  
It is beautiful cat  
cat is lovely.  
It is a cat  
This is a cat.  
This is a caterpillar.  
It is mcat.  

Example 2  
In the code in Example1, if we replace the regular expression ‘/cat/’ by ‘/ cat /’, the output will be:  
There is a cat in the house.  
It is a cat  

Here the regular expression ‘ / cat /’ has a blank space in front and after ‘cat’. The regular expression ‘/ cat /’ is equivalent to ‘/\s*cat\s*/’. ‘\s’ matches a white space character.  

Example 3  
In the code in Example1, if we replace the regular expression ‘/cat/’ by ‘/^cat/’, the output will be:  
cat is lovely.  
In the regular expression, ‘^’ matches the beginning of the string.  

Example 4  
In the code in Example1, if we replace the regular expression ‘/cat/’ by ‘/\^cat | cat /’, the output will be:  
There is a cat in the house.  
It is a cat  
cat is lovely.  

In the above regular expression, ‘|’ specifies an ‘or’ condition.
Example 5

If we want the selection to be case insensitive, then we could use in example 4, the regular expression

`/ ^cat | cat /i` instead of `'/ ^cat | cat /'`.

Then the output will be

CAT is an animal.  
There is a cat in the house.  
cat is lovely.  
It is a cat

Example 6

In Example 5, to select also the strings with `‘cat.’`, we could use  
`/ ^cat | cat | cat\.\./i` instead of `'/ ^cat | cat /i'`. Here we need to use `‘\’`, escape character to mask the special character `‘.’`, which has a special meaning in regular expressions.

The output will be:

CAT is an animal.  
There is a cat in the house.  
cat is lovely.  
It is a cat  
This is a cat.

Example 7

We are still missing the sentence “It is beautiful                  cat” in the output. Since there is no space after the word ‘cat’ in this string, it was not selected. We can use `/ ^cat | cat | cat\.\./ cat\$/i` instead of `/ ^cat | cat | cat\$/i` in Example 6 to select sentences of this type. The $ operator matches the end of the string.

The output will be

CAT is an animal.  
There is a cat in the house.  
it is beautiful                  cat  
cat is lovely.  
It is a cat  
This is a cat.

Example 8

In the code in Example1, if we replace the regular expression `/cat/` by `/ cat|b /`, the output will be:
There is a cat in the house.
It is beautiful cat
This is a cat.
It is mcat.

'/b' matches word boundary. The sentence, 'This is a caterpillar.', was not matched.

Example 9
In the code in Example1, if we replace the regular expression '/cat/' by '/\bcat/', The sentence, 'It is mcat.', will not be matched.

Example 10
In this example, \d matches a digit 0 through 9.

```r
data _null_;
pos = prxmatch('/\d/', 'abc 5dc');

put pos = ;
run;
```

Output:

pos = 5;

Example 11
In example 8, if we replace “pos =prxmatch('/\d/', 'abc 5dc');” by ‘pos =prxmatch('/\d\d\d/', 'abc 5dc 587p');’, the output will be

pos = 9.

The regular expression \d\d\d/ matches a 3 digit number. The regular expression \d\d\d\d\d\d\d\d\d is equivalent to the regular expression \d{3}/
# Basic Syntax for Regular Expressions

The following table covers the basic syntax for regular expressions.

<table>
<thead>
<tr>
<th>Character</th>
<th>Behavior</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ ... /</td>
<td>Starting and ending regular expression delimiters</td>
<td>/x</td>
</tr>
<tr>
<td></td>
<td>Alternation</td>
<td></td>
</tr>
<tr>
<td>( ) Grouping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>Matches any character</td>
<td></td>
</tr>
<tr>
<td>\w</td>
<td>Matches a word character (alphanumeric and “.”)</td>
<td>/\w\w\w/ matches &quot;a_2&quot; but not &quot;a 2&quot;</td>
</tr>
<tr>
<td>\W</td>
<td>Matches a non-word character</td>
<td>/\W\W/ matches “?+”</td>
</tr>
<tr>
<td>\s</td>
<td>Matches a whitespace character</td>
<td>/\d\d\d/ matches “2 5&quot; but not “2a5”</td>
</tr>
<tr>
<td>\S</td>
<td>Matches a non-whitespace character</td>
<td></td>
</tr>
<tr>
<td>\d</td>
<td>Matches a digit character</td>
<td>/\d\d\d/ matches any three digit number</td>
</tr>
<tr>
<td>\D</td>
<td>Matches a non-digit character</td>
<td></td>
</tr>
<tr>
<td>[ ... ]</td>
<td>Matches a character in the brackets</td>
<td>[abf] Match any of a,b and f. [a-j] matches character in the range of a to j</td>
</tr>
<tr>
<td>[ ^... ]</td>
<td>Matches a character not in the brackets</td>
<td>[^abf] Match any character other than a,b, or f.</td>
</tr>
<tr>
<td>^</td>
<td>Matches beginning of line</td>
<td>/^cat/ matches any line beginning with cat</td>
</tr>
<tr>
<td>$</td>
<td>Matches end of line</td>
<td>/\cat$/ matches any line ending with cat</td>
</tr>
<tr>
<td>*</td>
<td>Matches 0 or more times</td>
<td>/\cat*/ matches c, cat, and catat</td>
</tr>
<tr>
<td>+</td>
<td>Matches 1 or more times</td>
<td>/\d+/ matches one or more digits</td>
</tr>
<tr>
<td>?</td>
<td>Matches 1 or 0 times</td>
<td>/\cat?/ matches c and cat but not catat.</td>
</tr>
<tr>
<td>{n}</td>
<td>Matches exactly n times</td>
<td>/\d{5}/ matches a 5 digit number</td>
</tr>
<tr>
<td>{n,m}</td>
<td>Matches at least n but not more than m times</td>
<td>/\d{3,}/ is equivalent to /\d\d\d+/</td>
</tr>
<tr>
<td>\</td>
<td>Overrides (escape) next metacharacters. Metacharacters are { } [ ] ( ) ^ $.</td>
<td>/{/ matches the character “{&quot;</td>
</tr>
<tr>
<td>\b</td>
<td>Matches word boundary</td>
<td></td>
</tr>
<tr>
<td>\B</td>
<td>Matches non-word boundary</td>
<td></td>
</tr>
</tbody>
</table>
SOME FUNCTIONS USED with PERL REGULAR EXPRESSIONS in SAS

We have already seen the application of the function PRXMATCH and PRXCHANGE. First, let’s look at the function PRXPARSE.

**PRXPARSE**
This function compiles a Perl regular expression that can be used for pattern matching.

Syntax: **PRXPARSE (Perl_regular_expression)**

Let’s look at an example.

**Example**

```sas
DATA _NULL_
  IF _N_ = 1 THEN PATTERN = PRXPARSE ('/cat/');
  RETAIN PATTERN;

INPUT STRING $30. ;
POS = PRXMATCH (PATTERN, STRING);
IF POS > 0 THEN PUT STRING;
DATALINES;
CAT is an animal.
There is a cat in the    house.
;
```

Output:

There is a cat in the    house.

In this example, the variable PATTERN contains a numeric pattern identifier returned by the PRXPARSE function. To avoid executing the function PRXPARSE for every iteration, IF _N_ = 1 … , and Retain statement are used here.

**PRXMATCH**

This function searches for a pattern and returns the position at which the pattern is found.

Syntax: **PRXMATCH(regular-expression-id or Perl-regular-expression)**

Many of the above examples use this function.
Example

data _null_;  
    pos = prxmatch('/\d/', 'abc 5dc');  

    put pos=;  
run;  

Output:  
pos=5;  

PRXCHANGE Function

It performs the pattern matching replacement.

Syntax: PRXCHANGE (Perl-regular-expression or regular-expression-id, times, source)

times  
    is a the number of times to search for a match and replace a matching pattern.

Source  
    is the string you want to search

Example

data switch;  
    date = "2012/05/17";  

    newdate = prxchange('s/(\d{4}).(\d{2}).(\d{2})/$2-$3-$1/', -1, date ) ;  

    put date= newdate=;  
run;  

Output:  
date=2012/05/17 newdate=05-17-2012  

In this example $1 contains what is matched within the first pair of parentheses. $2 contains what is matched within the second pair of parentheses, and so on.
CALL PRXSUBSTR

This returns the position and length of a substring that matches a pattern.

Syntax: CALL PRXSUBSTR(Regular-expression-id, string, position, length);

'Regular-expression-id' is the identification number returned by the PRXPARSE function
'string' is the string to be searched.
'position' is a numeric variable that will be assigned the starting position of the pattern
'length' is a numeric variable that will be assigned the length of the pattern

Example

data _null_; /* Use PRXPARSE to compile the Perl regular expression. */
patternID = prxparse('/world/'); /* Use PRXSUBSTR to find the position and length of the string. */
call prxsubstr(patternID, 'Hello world!', position, length);
put position= length=;
run;

Output:
position=7 length=5

Using the value of the variable position and the value of the variable length, one can extract the pattern in the string.

CALL PRXFREE

This frees memory that was allocated for a Perl regular expression.

Syntax: CALL PRXFREE (Regular-expression-id);

For other functions that are used with Perl regular expressions in SAS, refer to Cody's paper in reference 1, and/or the SAS documentation (version 9.2) on Perl regular expressions
AN APPLICATION

Here we first select all the lines that have ‘PO Box’ , ’P.O.Box’, ‘Box’, ’P O Box’, etc., and replace each one of them with ‘P O BOX’ in the strings using PRXCHANGE function. Then using PRXSUBSTR function, we find the position of the string made up with ‘P O Box’ and its number. Then we extract the string made up with ‘P O Box’ and its number.

In this example, ‘?’ stands for matching 0 or 1 time, ‘*’ is for matching 0 or more times, and ‘\s’ is for matching a white space character. The first ‘s’ in the argument of first PRXPARSE expression is for substitution.

DATA _NULL_;
RETAIN PATTERN PATTERN1;
  IF _N_ =1 THEN PATTERN =
   PRXPARSE ("s/P?\s*/s*O?\s*/s*BOX\s*/s*/P O BOX /i");

  /* This regular expression stands for substitution of
   ‘/P?\s*/s*O?\s*/s*BOX\s*/s*/’ with ‘P O BOX’ */

INPUT STRING $31. ;

CALL PRXCHANGE(PATTERN, 4, STRING); /* PRXCHANGE does the Replacement using PATTERN in the STRING */

  IF _N_ =1 THEN PATTERN1 =
   PRXPARSE ("/P O BOX\s*\d+/i");

CALL PRXSUBSTR(PATTERN1, STRING, POSITION, LENGTH); /* PRXSUBSTR returns the position and length of a substring that Matches PATTERN1 */

  IF POSITION >0 THEN DO;
MATCH =SUBSTR(STRING, POSITION, LENGTH);
PUT MATCH;
END;
DATALINES;
1250 Church street  P.O.BOX 495
P  O BOX 2235 15 North st.
P O BOX 20202020.
123 Mill st  P BOX 223
11 prospect ave  p o box
BOX 2282828
P Box 225
p box22152
pobox 2212
Pbox 2345
p. o. box. 256
;
run;
CONCLUSION
If we use traditional character functions to do the above string manipulations, the codes could be very long. Even though code may be compact when we use SAS regular expression, it may sometimes be more efficient to use traditional SAS character functions to deal with string manipulations.

If you want to learn more about Perl regular expressions in SAS, refer to Cody’s paper in reference 1, and/or the SAS documentation (version 9.2) on Perl regular expressions.

REFERENCES

ACKNOWLEDGMENTS
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.

DISCLAIMER
This paper reports the results of research and analysis undertaken by Census Bureau staff. It has undergone a more limited review by the Census Bureau than its official publications. This report is released to inform interested parties and to encourage discussion.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:

Selvaratnam Sridharma
Economic Planning and Coordination Division
U.S. Bureau of the Census
Address
Washington, DC 20233-6100
301-763-6774
Email: selvaratnam.sridharma@census.gov