PharmaSUG China 2019 - Paper CC-073

Four steps to get a quick start with Perl Regular Expressions in SAS®

Qin Ni, Paul Burmenko, Everest Clinical Research, Shanghai, China

ABSTRACT

Many languages co-exist in the ecosystem of your SAS® software. The Perl language element is one of them. The knowledge of the basics of regular expressions (PRX functions) will definitely sharpen programmers' programming skills, especially in string manipulations and locating patterns in text strings.

Learning regular expressions starts with understanding of character classes and metacharacters. Metacharacters are a system of symbols to describe a text pattern to read text. This can be an obvious reason to get cold feet, but do not fear!

This paper will describe the steps on how to learn Perl Regular Expression in SAS 9.4. We will cover: 1. Several simple metacharacters and their combinations. 2. Learn how to represent patterns using metacharacters, and how to use online tools (e.g. https://regex101.com) to test your syntax. 3. Start with PRXMATCH and PRXCHANGE to understand their usage compared with functions like INDEX or SUBSTR or FINDC. 4. Practice with date imputation using PRX functions.

The four steps provided here will enable anyone to learn this topic with ease and confidence.

INTRODUCTION

There are many published papers describing different features of Perl regular expressions in SAS® and how it can increase a programmer's skillset. But there are rarely articles that introduce a simple and easy to understand ways to learn Perl regular expressions in SAS. In this article, we focus on a step by step learning method to achieve our final goal—manipulating data with PRX functions.

SETP 1 MASTER THE METACHARACTERS

Learning regular expressions (a.k.a. regex) starts with understanding of character classes and metacharacters. Metacharacters are a system of symbols to describe a text pattern to process text. What are metacharacters? We summarized and extracted the most important metacharacters here (Table 1) for beginners to start with. Once you get familiar with these, you can try to combine them and create patterns for matching strings.

Table 1. Basic Metacharacters for Beginners

Metacharacters	Descriptions
/…/	Specifies the default delimiters of Perl regular expressions
(…)	Indicate a grouping.

Metacharacters	Descriptions
[...]	Specifies a character set that matches any one of the enclosed characters
[^…]	Matches any character that is not in the enclosed parentheses.
^	Matches the position at the beginning of the input string
$	Matches the position at the end of the input string.
\d	Matches a digit character that is equivalent to [0−9].
\D	Matches a non-digit character that is equivalent to [^0−9].
\w	Matches any word character or alphanumeric character, including the underscore
\W	Matches any non-word character or non-alphanumeric character, and excludes the underscore
\s	Matches a space
|	Specifies the OR condition. For example, the construct x|y matches either x or y
\/	Matches the forward slash

STEP 2 POWERFUL ONLINE TOOL HELPS TO TEST YOUR SYNTAX

It is said that Perl regular expressions are "write only." What does it mean? You can become quite accomplished at writing regular expressions after some practice, but reading them, is still quite difficult, even to read those written by yourself. But with the online tool (https://regex101.com), the pattern expression becomes much easier for any beginners. Let's take a look at the website tool much closer (Figure 1 & Figure 2). It allows you to put the regex in and also to input another string for testing. The most wonderful thing is: you can see the explanation of your regex just appear on the right side in an easy to read format.

Figure 1. A date pattern testing with regex101 online tool

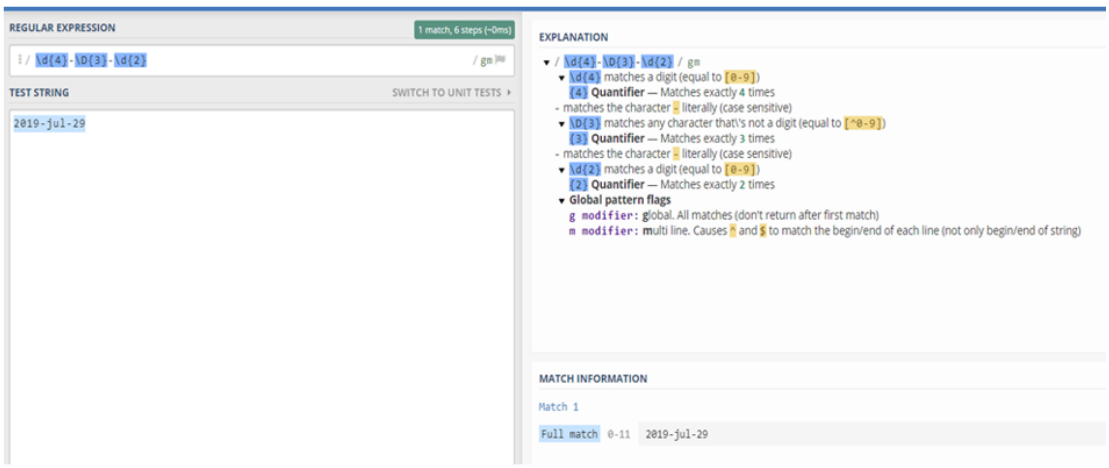
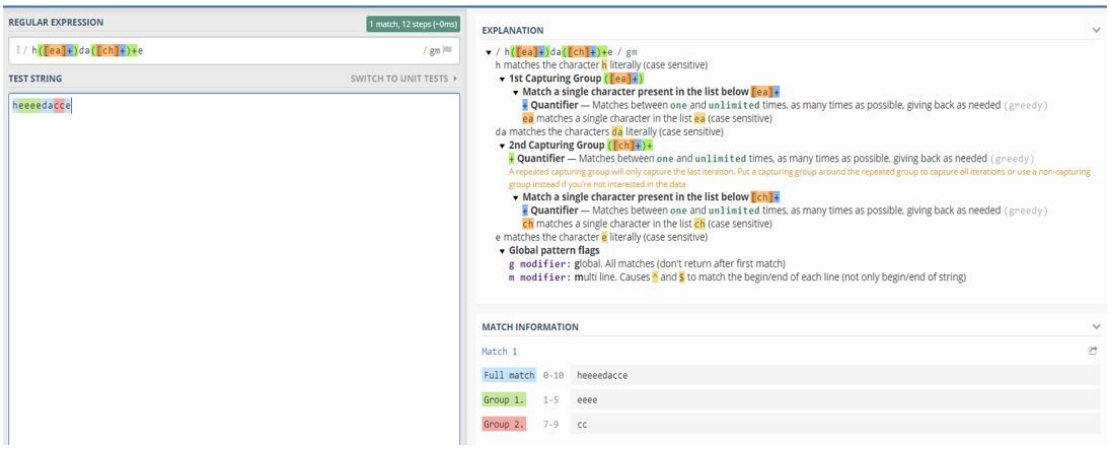


Figure 2. A string pattern testing with regex101 online tool



STEP 3 PRXFUNCTION VS OTHER STRING FUNCTIONS

SEARCHING TEXT –PRXMATCH vs. INDEX & FINDC

PRXMATCH is the most fundamental of the PRX functions and may be used interchangeably with the INDEX function. In other words, the capability of INDEX is a subset of the capability of PRXMATCH.

```
prxmatch(<perl regular expression in quotes>,<source variable>);
```

returns the start position of the matching string

- Basic usage between Index and PRXMATCH

```
if index(var,"High")
if prxmatch('/High/',var);
```

- Multiple searching between Index and PRXMATCH

```
if index(var,"High") and index(var,"Low");
if prxmatch('/(High|Low)/',var);
```

- Multiple searching with upcase letters between Index and PRXMATCH

```
if index(upcase(var),"HIGH") and index(upcase(var),"LOW");
if prxmatch('/(High|Low)/i',var);
```

- Advanced searching with starting or end indicators using PRXMATCH

```
if prxmatch('/^High/',var);
if prxmatch('/Low$/',var);
```

- Basic usage between PRXMATCH and FINDC

```
if findc('01235', , 'd');
if prxmatch('/\d/', '01235');
```

REPLACING TEXT – PRXCHANGE vs. SUBSTR (left of)

PRXCHANGE matches a string by the regular expression that is passed to it and returns a string that has the matching text replaced. Regular expressions that are used in PRXCHANGE are different from PRXMATCH. First of all, they must start with an “s”, before the first delimiter. Then comes the regular expression and after the second delimiter comes the replacement text, that is also followed by a delimiter.

```
newvar=prxchange(<perl regular expression in quotes>,<times>,oldvar);
                          ↓
                 's/<find>/<replace>/'
```

data _null_; date='2019-JUL-31'; substr(date,6,3)='Oct'; put date=; run; date=2019-Oct-31	
data work.test; infile datalines dsd; length line $50.; input line $; datalines; Subject suffered from headache and nause, This Heeadache probably caused by treatment. After two weeks no more hedahce. ; run; data _null_; set work.test; line1=prxchange('s/h([ea]+)da([ch]+)+e/headache/i',-1 ,line); put line=; put line1=; un; line=Subject suffered from headache and nause line1=Subject suffered from headache and nause line=This Heeadache probably caused by treatment. line1=This headache probably caused by treatment. line=After two weeks no more hedahce. line1=After two weeks no more headache. *Code refer to Jules van der Zalm et al. Phuse 2009*	Regex101 explanation /h([ea]+)da([ch]+)+e/ h matches the character h literally (case sensitive) **1st Capturing Group** ([ea]+) **Match a single character present in the list below** [ea]+ **+ Quantifier** — Matches between **one** and **unlimited** times, as many times as possible, giving back as needed (greedy) ea matches a single character in the list ea (case sensitive) da matches the characters da literally (case sensitive) **2nd Capturing Group** ([ch]+)+ **+ Quantifier** — Matches between **one** and **unlimited** times, as many times as possible, giving back as needed (greedy) A repeated capturing group will only capture the last iteration. Put a capturing group around the repeated group to capture all iterations or use a non-capturing group instead if you're not interested in the data **Match a single character present in the list below** [ch]+

	+ Quantifier — Matches between **one** and **unlimited** times, as many times as possible, giving back as needed (greedy) ch matches a single character in the list ch (case sensitive) e matches the character e literally (case sensitive)

STEP4 DATE IMPUTATION PRACTICE WITH PRX FUNCTION

Suppose we are going to do a date imputation of Medical History start date with the following rule: if the year is missing, no imputation, set to null; if the month is missing, set to “07”; if the day is missing, set to “01”.

Source date format: MM/DD/YYYY or UNK/UNK/YYYY

- Both Month and Day are missing

```
If lengthn(MHSTDAT)=12 and prxmatch('/UNK\/UNK\/\d{4}/', MHSTDAT)
then MHSTDTC=catx("-",put(scan(MHSTDAT,3,"/"),4.), "07","01");
```

- Month is missing

```
if lengthn(MHSTDAT)=11 and prxmatch('/UNK\/\d{2}\/\d{4}/', MHSTDAT)
then MHSTDTC=catx("-",put(scan(MHSTDAT,3,"/"),4.),
"07",put(scan(MHSTDAT,2,"/"),2.));
```

- Day is missing

```
if lengthn(MHSTDAT)=11 and prxmatch('/\d{2}\/UNK\/\d{4}/', MHSTDAT)
then MHSTDTC=catx("-",put(scan(MHSTDAT,3,"/"),4.),
put(scan(MHSTDAT,1,"/"),2.),"01");
```

Obs	MHSTDAT	MHSTDTC
1	UNK/UNK/2017	2017-07-01
2	04/UNK/2017	2017-04-01
3	UNK/UNK/2015	2015-07-01
4	UNK/UNK/2008	2008-07-01
5	UNK/UNK/2016	2016-07-01
6	UNK/UNK/2013	2013-07-01
7	UNK/UNK/2011	2011-07-01

Regex101 explanation

/UNK\/UNK\/\d{4}/

UNK matches the characters UNK literally (case sensitive)

\/ matches the character / literally (case sensitive)

UNK matches the characters UNK literally (case sensitive)

\/ matches the character / literally (case sensitive)

\d{4}

matches a digit (equal to [0-9])

{4} **Quantifier** — Matches exactly **4** times

CONCLUSION

In this paper, we focused on incorporating Perl regular expressions into our SAS programming. We began with a few metacharacters, then introduced an online tool for understanding and testing the regex. We compared how PRXMATCH and PRXCHANGE function in similar ways to INDEX, FINDC or SUBSTR. The last example of date imputation demonstrated a practical way to using Perl expression in pattern matching. Readers of this paper can move on to more advanced methods of incorporationg Perl regular expressions into their SAS programming toolbox by referencing the documents in the reference section.

REFERENCES

Jules van der Zalm, OCS Consulting. “Optimized Lookup Using Regular Expressions”. the Netherlands - PhUSE 2009.

Mary F. O. Rosenbloom, Jennifer Mares, Trina Patel ,Edwards Lifesciences LLC, Irvine, CA. “Conjunction, What’s Your Function?: How PRXCHANGE Saved The Day“ . Western Users of SAS® Software, 2014.

Rho®, Inc., Chapel Hill, NC. “Express Yourself! Regular Expressions vs SAS Text String Functions Spencer Childress”. PharmaSUG 2014 - Paper BB08

Joel Campbell, Advanced Analytics, Wilmington, NC. “Perl Regular Expressions in SAS® 9.1+ - Practical Applications”. PharmaSUG 2012 - Paper TA08

http://support.sas.com/kb/38/719.html

https://support.sas.com/rnd/base/datastep/perl_regexp/regexp-tip-sheet.pdf

CONTACT INFORMATION

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

Qin Ni
Everest Clinical Research, Shanghai, China
nancy.ni@ecrscorp.com

Paul Burmenko
Everest Clinical Research, US
paul.burmenko@ecrscorp.com